S. Barry Cooper
Benedikt Löwe
Andrea Sorbi (Eds.)

LNCS 4497

Computation and Logic in the Real World

Third Conference on Computability in Europe, CiE 2007
Siena, Italy, June 2007
Proceedings

Springer

S. Barry Cooper Benedikt Löwe
Andrea Sorbi (Eds.)

Computation and Logic in the Real World

Third Conference on Computability in Europe, CiE 2007
Siena, Italy, June 18-23, 2007
Proceedings

Springer

Volume Editors

S. Barry Cooper
University of Leeds
Dept. of Pure Mathematics
Leeds LS2 9JT, UK
E-mail: pmt6sbc@maths.leeds.ac.uk

Benedikt Löwe
Universiteit van Amsterdam
Institue for Logic, Language and Computation (ILLC)
Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands
E-mail: bloewe@science.uva.nl

Andrea Sorbi
University of Siena
Dipartimento di Scienze Matematiche ed Informatiche "R. Magari"
53100 Siena, Italy
E-mail: sorbi@unisi.it

Library of Congress Control Number: 2007928471

CR Subject Classification (1998): F.1, F.2.1-2, F.4.1, G.1.0, I.2.6, J.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-73000-1 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-73000-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12075188 06/3180 5 4 3 2 1 0

Preface

CiE 2007: Computation and Logic in the Real World
Siena, Italy, June 18–23, 2007

Computability in Europe (CiE) is an informal network of European scientists working on computability theory, including its foundations, technical development, and applications. Among the aims of the network is to advance our theoretical understanding of what can and cannot be computed, by *any* means of computation. Its scientific vision is broad: computations may be performed with discrete or continuous data by all kinds of algorithms, programs, and machines. Computations may be made by experimenting with any sort of physical system obeying the laws of a physical theory such as Newtonian mechanics, quantum theory, or relativity. Computations may be very general, depending upon the foundations of set theory; or very specific, using the combinatorics of finite structures. CiE also works on subjects intimately related to computation, especially theories of data and information, and methods for formal reasoning about computations. The sources of new ideas and methods include practical developments in areas such as neural networks, quantum computation, natural computation, molecular computation, computational learning. Applications are everywhere, especially, in algebra, analysis and geometry, or data types and programming. Within CiE there is general recognition of the underlying relevance of computability to physics and a broad range of other sciences, providing as it does a basic analysis of the causal structure of dynamical systems.

This volume, *Computation and Logic in the Real World*, is the proceedings of the third in a series of conferences of CiE that was held at the Dipartimento di Scienze Matematiche e Informatiche "Roberto Magari," University of Siena, June 18–23, 2007.

The first two meetings of CiE were at the University of Amsterdam, in 2005, and at the University of Wales Swansea in 2006. Their proceedings, edited in 2005 by S. Barry Cooper, Benedikt Löwe and Leen Torenvliet, and in 2006 by Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, were published as *Springer Lecture Notes in Computer Science*, Volumes 3526 and 3988, respectively. As the editors noted in last year's proceedings, CiE and its conferences have changed our perceptions of computability and its interface with other areas of knowledge. The large number of mathematicians and computer scientists attending those conference had their view of computability theory enlarged and transformed: they discovered that its foundations were deeper and more mysterious, its technical development more vigorous, its applications wider and more challenging than they had known. The Siena meeting promised to extend and enrich that process.

The annual CiE conference, based on the *Computability in Europe* network, has become a major event, and is the largest international meeting focused on computability theoretic issues. The series is coordinated by the CiE Conference Series Steering Committee:

Paola Bonizzoni (Milan)
Barry Cooper (Leeds)
Benedikt Löwe (Amsterdam, Chair)
Elvira Mayordomo (Zaragoza)
Dag Normann (Oslo)
Andrea Sorbi (Siena)
Peter van Emde Boas (Amsterdam).

We will reconvene in 2008 in Athens, 2009 in Heidelberg, and 2010 in Lisbon.

Structure and Programme of the Conference

The conference was based on invited tutorials and lectures, and a set of special sessions on a range of subjects; there were also many contributed papers and informal presentations. This volume contains 36 of the invited lectures and 29.9% of the submitted contributed papers, all of which have been refereed. There will be a number of post-proceedings publications, including special issues of *Theoretical Computer Science*, *Theory of Computing Systems*, *Annals of Pure and Applied Logic*, and *Journal of Logic and Computation.*

Tutorials

Pieter Adriaans (Amsterdam), *Learning as Data Compression*
Yaakov Benenson (Cambridge, Massachusetts), *Biological Computing*

Invited Plenary Talks

Anne Condon (Vancouver), *Computational Challenges in Prediction and Design of Nucleic Acid Structure*
Stephen Cook (Toronto), *Low Level Reverse Mathematics*
Yuri Ershov (Novosibirsk), *HF-Computability*
Sophie Laplante (Paris), *Using Kolmogorov Complexity to Define Individual Security of Cryptographic Systems*
Wolfgang Maass (Graz), *Liquid Computing*

Anil Nerode (Cornell), *Logic and Control*
Piergiorgio Odifreddi (Turin), *Conference Introductory Lecture*
Roger Penrose (Oxford), A talk on *Aspects of Physics and Mathematics*
Michael Rathjen (Leeds), *Theories and Ordinals in Proof Theory*
Dana Scott (Pittsburgh), *Two Categories for Computability* (Lecture sponsored by the European Association for Computer Science Logic.)
Robert I. Soare (Chicago), *Computability and Incomputability*
Philip Welch (Bristol), *Turing Unbound: Transfinite Computation*

Special Sessions

Doing Without Turing Machines: Constructivism and Formal Topology, organised by Giovanni Sambin and Dieter Spreen

Givanni Sambin (Padova) *Doing Without Turing Machines: Constructivism and Formal Topology*
Andrej Bauer (Ljubljana), *RZ: A Tool for Bringing Constructive and Computable Mathematics Closer to Programming Practice*
Douglas Bridges (Canterbury, NZ), *Apartness on Lattices*
Thierry Coquand (Göteborg), *A Constructive Version of Riesz Representation Theorem*
Maria Emilia Maietti (Padova), *Constructive Foundation for Mathematics as a Two Level Theory: An Example*

Approaches to Computational Learning, organised by Marco Gori and Franco Montagna

John Case (Newark, Delaware), *Resource Restricted Computability Theoretic Learning: Illustrative Topics and Problems*
Klaus Meer (Odense), *Some Aspects of a Complexity Theory for Continuous Time Systems*
Frank Stephan (Singapore), *Input-Dependence in Function-Learning*
Osamu Watanabe (Tokyo), *Finding Most Likely Solutions*

Real Computation, organised by Vasco Brattka and Pietro Di Gianantonio

Pieter Collins (Amsterdam), *Effective Computation for Nonlinear Systems*
Abbas Edalat (London), *A Continuous Derivative for Real-Valued Functions*
Hajime Ishihara (Tokyo), *Unique Existence and Computability in Constructive Reverse Mathematics*
Robert Rettinger (Hagen), *Computable Riemann Surfaces*
Martin Ziegler (Paderborn), *Real Hypercomputation*

Computability and Mathematical Structure, organised by Serikzhan Badaev and Marat Arslanov

Vasco Brattka (Cape Town), *Computable Compactness*
Barbara F. Csima (Waterloo), *Properties of the Settling-Time Reducibility Ordering*

Sergey S. Goncharov (Novosibirsk), *Computable Numberings Relative to Hierarchies*
Jiří Wiedermann (Prague), *Complexity Issues in Amorphous Computing*
Chi Tat Chong (Singapore), *Maximal Antichains in the Turing Degrees*

Complexity of Algorithms and Proofs, organised by Elvira Mayordomo and Jan Johannsen

Eric Allender (Piscataway, New Jersey), *Reachability Problems: An Update*
Jörg Flum (Freiburg), *Parameterized Complexity and Logic*
Michal Koucký (Prague), *Circuit Complexity of Regular Languages*
Neil Thapen (Prague), *The Polynomial and Linear Hierarchies in Weak Theories of Bounded Arithmetic*
Heribert Vollmer (Hannover), *Computational Complexity of Constraint Satisfaction*

Logic and New Paradigms of Computability, organised by Paola Bonizzoni and Olivier Bournez

Felix Costa (Lisbon), *The New Promise of Analog Computation*
Natasha Jonoska (Tampa, Florida), *Computing by Self-Assembly*
Giancarlo Mauri (Milan), *Membrane Systems and Their Applications to Systems Biology*
Grzegorz Rozenberg (Leiden), *Biochemical Reactions as Computations*
Damien Woods (Cork), (with Turlough Neary) *The Complexity of Small Universal Turing Machines*

Computational Foundations of Physics and Biology, organised by Guglielmo Tamburrini and Christopher Timpson

James Ladyman (Bristol), *Physics and Computation: The Status of Landauer's Principle*
Itamar Pitowsky (Jerusalem), *From Logic to Physics: How the Meaning of Computation Changed Over Time*
Grzegorz Rozenberg (Leiden), *Natural Computing: A Natural and Timely Trend for Natural Sciences and Science of Computation*
Christopher Timpson (Leeds), *What's the Lesson of Quantum Computing?*
Giuseppe Trautteur (Naples), *Does the Cell Compute?*

Women in Computability Workshop, organised by Paola Bonizzoni and Elvira Mayordomo

A new initiative at CiE 2007 was the adding of the Women in Computability workshop to the programme. Women in computer science and mathematics face particular challenges in pursuing and maintaining academic and scientific careers. The Women in Computability workshop brought together women in computing and mathematical research to present and exchange their academic and scientific experiences with young researchers. The speakers were:

Anne Condon (British Columbia)
Natasha Jonoska (Tampa, Florida)
Carmen Leccardi (Milan)
Andrea Cerroni (Milan)

Organisation and Acknowledgements

The CiE 2007 conference was organised by the logicians at Siena: Andrea Sorbi, Thomas Kent, Franco Montagna, Tommaso Flaminio, Luca Spada, Andrew Lewis, Maria Libera Affatato and Guido Gherardi; and with the help of Leeds computability theorists: S Barry Cooper, Charles Harris and George Barmpalias; and Benedikt Löwe (Amsterdam). The CiE CS Steering Committee also played an essential role.

The Programme Committee was chaired by Andrea Sorbi and Barry Cooper and consisted of:

Manindra Agrawal (Kanpur)
Marat M. Arslanov (Kazan)
Giorgio Ausiello (Rome)
Andrej Bauer (Ljubljana)
Arnold Beckmann (Swansea)
Ulrich Berger (Swansea)
Paola Bonizzoni (Milan)
Andrea Cantini (Florence)
S. Barry Cooper (Leeds, Co-chair)
Laura Crosilla (Leeds)
Josep Diaz (Barcelona)
Costas Dimitracopoulos (Athens)
Fernando Ferreira (Lisbon)
Sergei S. Goncharov (Novosibirsk)
Peter Grünwald (Amsterdam)
David Harel (Jerusalem)
Andrew Hodges (Oxford)
Julia Kempe (Paris)
Giuseppe Longo (Paris)
Benedikt Löwe (Amsterdam)
Johann A. Makowsky (Haifa)
Elvira Mayordomo Cámara (Zaragoza)
Wolfgang Merkle (Heidelberg)
Franco Montagna (Siena)
Dag Normann (Oslo)
Thanases C. Pheidas (Iraklion, Crete)
Grzegorz Rozenberg (Leiden)
Giovanni Sambin (Padova)
Helmut Schwichtenberg (Munich)
Wilfried Sieg (Pittsburgh)
Andrea Sorbi (Siena, Co-chair)
Ivan N. Soskov (Sofia)
Peter van Emde Boas (Amsterdam)

We are delighted to acknowledge and thank the following for their essential financial support: the Department of Mathematics and Computer Science “Roberto Magari” at Siena; the Fondazione del Monte dei Paschi di Siena; the Istituto Nazionale di Alta Matematica: Gruppo Nazionale per le Strutture Algebriche, Geometriche e le loro Applicazioni (INDAM-GNSAGA); the University of Siena; the Associazione Italiana di Logica e sue Applicazioni (AILA); the Association for Symbolic Logic (ASL); the European Association for Computer Science Logic (EACSL). We would also like to thanks our sponsors: the European Association for Theoretical Computer Science (EATCS); the Association of Logic, Language and Information (FoLLI); the Committee on the Status of Women in

Computing Research (CRA-W). We are pleased to thank our colleagues on the Organising Committee for their many contributions and our research students for practical help at the conference. Special thanks are due to Thomas Kent, Tommaso Flaminio, Luca Spada, Andy Lewis, and Franco Montagna for their precious collaboration, and the Congress Service of the University of Siena for the administrative aspects of the conference.

The high scientific quality of the conference was possible through the conscientious work of the Programme Committee, the Special Session organisers, and the referees. We are grateful to all members of the Programme Committee for their efficient evaluations and extensive debates, which established the final programme. We also thank the following referees:

Klaus Aehlig
Pilar Albert
Klaus Ambos-Spies
Andrea Asperti
Luís Antunes
Albert Atserias
George Barmpalias
Freiric Barral
Sebastian Bauer
Almut Beige
Josef Berger
Luca Bernardinello
Daniela Besozzi
Laurent Bienvenu
Christian Blum
Markus Bläser
Thomas Bolander
Roberto Bonato
Lars Borner
Abraham P. Bos
Malte Braack
Vasco Brattka
Andries E. Brouwer
Joshua Buresh-Oppenheim
Nadia Busi
Cristian S. Calude
Riccardo Camerlo
John Case
Orestes Cerdeira
Yijia Chen
Luca Chiarabini
Jose Félix Costa
Ronald Cramer
Paola D'Aquino
Ugo Dal Lago
Victor Dalmau
Tijmen Daniëls
Anuj Dawar
Barnaby Dawson
Ronald de Wolf
José del Campo
Karim Djemame
David Doty
Rod Downey
Martin Escardo
Antonio Fernandes
Claudio Ferretti
Eric Filiol
Eldar Fischer
Daniel Garca
Parmenides Garcia Cornejo
William Gasarch
Ricard Gavaldà
Giangiacomo Gerla
Eugene Goldberg
Massimiliano Goldwurm
Johan Granström
Phil Grant
Dima Grigoriev
Barbara Hammer
Tero Harju
Montserrat Hermo
Peter Hertling
Thomas Hildebrandt
John Hitchcock
Pascal Hitzler
Steffen Hölldobler
Mathieu Hoyrup
Simon Huber
Jim Hurford
Carl Jockusch
Jan Johannsen
Michael Kaminski
Vladimir Kanovei
Basil Karadais
Vassilios Karakostas
Iztok Kavkler
Thomas Kent
Hans Kleine Büning
Sven Kosub
Bogomil Kovachev
Evangelos Kranakis
S. N. Krishna
Oleg Kudinov
Petr Kurka
Eyal Kushilevitz
Akhlesh Lakhtakia
Jérôme Lang
Hans Leiss
Stephane Lengrand
Alberto Leporati
Andy Lewis
Maria Lopez-Valdes
Michele Loreti
Alejandro Maass
Vincenzo Manca
Edwin Mares

Luciano Margara	Dirk Pattinson	Neil Thapen
Maurice Margenstern	George Paun	Klaus Thomsen
Simone Martini	Andrea Pietracaprina	Christopher Timpson
Ralph Matthes	Sergey Podzorov	Michael Tiomkin
Andrea Maurino	Chris Pollett	Edmondo Trentin
Klaus Meer	Pavel Pudlak	Trifon Trifonov
Nenad Mihailovic	Diana Ratiu	José Triviño-Rodriguez
Russell Miller	Jan Reimann	Reut Tsarfaty
Pierluigi Minari	Paul Ruet	John Tucker
Nikola Mitrovic	Markus Sauerman	Sara Uckelman
Tal Mor	Stefan Schimanski	Christian Urban
Philippe Moser	Wolfgang Schönfeld	Tullio Vardanega
Mioara Mugu-Schachter	Jeremy Seligman	Sergey Verlan
Thomas Müller	Peter Selinger	Thomas Vidick
Nguyen Hoang Nga	Mariya Soskova	Heribert Vollmer
Ray Nickson	Bas Spitters	Rebecca Weber
Karl-Heinz Niggl	Frank Stephan	Philip Welch
Martin Otto	Mario Szegedy	Guohua Wu
Jiannis Pachos	Wouter Teepe	Reem Yassawi
Aris Pagourtzis	Balder ten Cate	Martin Ziegler
Dimitrii Palchunov	Sebastiaan Terwijn	Jeffery Zucker
Francesco Paoli	Christof Teuscher	Dragisa Zunic

We thank Andrej Voronkov for his Easy Chair system which facilitated the work of the Programme Committee and the editors considerably.

Finally, we wish to thank once again Thomas Kent of the Organising Committee, for typesetting this volume virtually by himself.

April 2007

Andrea Sorbi
S. Barry Cooper
Benedikt Löwe

Table of Contents

Shifting and Lifting of Cellular Automata[⋆]

Luigi Acerbi[1], Alberto Dennunzio[1], and Enrico Formenti[2,⋆⋆]

[1] Università degli Studi di Milano–Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione,
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
dennunzio@disco.unimib.it, luigi.acerbi@gmail.com
[2] Université de Nice-Sophia Antipolis
Laboratoire I3S,
2000 Route des Colles, 06903 Sophia Antipolis, France
enrico.formenti@unice.fr

Abstract. We consider the family of all the Cellular Automata (CA) sharing the same local rule but have different memory. This family contains also all the CA with memory $m \leq 0$ (one-sided CA) which can act both on $A^{\mathbb{Z}}$ and on $A^{\mathbb{N}}$. We study several set theoretical and topological properties for these classes. In particular we investigate if the properties of a given CA are preserved when we consider the CA obtained by changing the memory of the original one (shifting operation). Furthermore we focus our attention to the one-sided CA acting on $A^{\mathbb{Z}}$ starting from the one-sided CA acting on $A^{\mathbb{N}}$ and having the same local rule (lifting operation). As a particular consequence of these investigations, we prove that the long-standing conjecture [Surjectivity $\Rightarrow$ Density of the Periodic Orbits (DPO)] is equivalent to the conjecture [Topological Mixing $\Rightarrow$ DPO].

Keywords: discrete time dynamical systems, cellular automata, topological dynamics, deterministic chaos.

1 Introduction and Motivations

Cellular automata (CA) are a simple formal model for complex systems. Their are used in many scientific fields ranging from biology to chemistry or from physics to computer science.

A CA is made of an infinite set of finite automata distributed over a regular lattice $\mathcal{L}$. All finite automata are identical. Each automaton assumes a *state*, chosen from a finite set A, called the *set of states* or the *alphabet*. A *configuration* is a snapshot of all states of the automata *i.e.* a function from $\mathcal{L}$ to A. In the present paper, $\mathcal{L} = \mathbb{Z}$ or $\mathcal{L} = \mathbb{N}$.

[⋆] This work has been supported by the Interlink/MIUR project "Cellular Automata: Topological Properties, Chaos and Associated Formal Languages", by the ANR Blanc "Projet Sycomore" and by the PRIN/MIUR project "Formal Languages and Automata: Mathematical and Applicative Aspects".

[⋆⋆] Corresponding author.

A *local rule* updates the state of an automaton on the basis of its current state and the ones of a fixed set of neighboring automata which are individuated by the neighborhood frame $N = \{-m, -m+1, \dots, -m+d\}$, where $m \in \mathbb{Z}$, $d \in \mathbb{N}$, and $r = \max\{m, d-m\}$ are the *memory*, the *diameter*, and the *radius* of the CA, respectively. Formally, the local rule is a function $f : A^{d+1} \to A$.

All the automata of the lattice are updated synchronously. In other words, the local rule f induces a *global* rule $F_m : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ describing the evolution of the whole system from time t to $t+1$:

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad F_m(x)_i = f(x_{i-m}, \dots, x_{i-m+d}) \enspace .$$

The *shift map* σ is one among the simplest examples of CA and it is induced by the rule $f : A \to A$, defined as $\forall a \in A,\ f(a) = a$, with $m = -1$, $d = 0$. Remark that σ can also be induced by the rule $f : A^2 \to A$ defined as $\forall a, b \in A,\ f(a, b) = b$ with $m = 0$, $d = 1$. We prefer to use the former representation since it minimizes the neighborhood size.

For any CA on $A^{\mathbb{Z}}$, the structure $\langle A^{\mathbb{Z}}, F_m \rangle$ is a (discrete time) dynamical system. From now on, for the sake of simplicity, we identify a CA with the dynamical system induced by itself or even with its global rule F_m.

The local rule of a CA can be convenient represented by a look-up table. Anyway, the look-up table does not uniquely define the CA. Indeed, for each value of m we have a different CA. It is therefore natural to wonder what dynamical properties are conserved by the CA obtained by changing the value of m but keeping the same look-up table for the local rule. This paper tries to answer this question.

Remark that the solution to the problem is absolutely not trivial. For instance, a periodic Coven automaton is a CA defined by the following local rule ($A = \{0, 1\}$, $m = 0$ and $d \in \mathbb{N}$): $f(a_0, a_1, \dots, a_d) = a_0 \oplus \prod_{k=1}^{d} (a_k \oplus w_k \oplus 1)$, where $\oplus$ is the usual xor operation and $w = w_1 w_2 \dots w_d \in \{0, 1\}^d$ is a periodic word[1]. Despite of the fact that for aperiodic Coven automata almost everything is known [2], very little is known about the periodic case even for the simplest example *i.e.* when $w = 11$. Call $\mathcal{A}$ this last automaton and F_0 its global rule. Taking the same look-up table as $\mathcal{A}$ but $m = -1$, we obtain the elementary CA rule called $ECA120$. Most of the dynamical properties of $ECA120$ are well-known (see [7], for instance). More formally, one can write that $ECA120 = \sigma \circ F_0$. For this reason we say that $ECA120$ is a *shifted version* of F_0.

In [13], Sablik studies the behavior of the shift operation over look-up tables *w.r.t.* the equicontinuity property and gives precise bounds for conservation and non-conservation. In this paper, we focus on the periodic behavior. We show that the proof of an old-standing conjecture about denseness of periodic orbits (DPO) can be reduced to the study of the class of topologically mixing CA *i.e.* a very small class with very special dynamical behavior. Maybe this would simplify the task of proving the conjecture. The result is obtained as a by-product of our

[1] A word $w \in \{0, 1\}^d$ is *periodic* if there exists $1 \leq p \leq d-1$ such that $w_i = w_{i+p}$ for $1 \leq i \leq d-p$. A word is *aperiodic* if it is not periodic.

results about the conservation of other interesting properties like surjectivity, left (or right) closingness *etc.*

Any CA with memory $m \leq 0$ is well defined both on $A^{\mathbb{Z}}$ and on $A^{\mathbb{N}}$. In the $A^{\mathbb{N}}$ we prefer to use the slightly different notation Φ_m in order to avoid confusion with the $A^{\mathbb{Z}}$ case. The mapping $\Phi_m : A^{\mathbb{N}} \mapsto A^{\mathbb{N}}$ acts on any configuration $x \in A^{\mathbb{N}}$ as follows

$$\forall i \in \mathbb{N}, \quad \Phi_m(x)_i = f(x_{i-m}, \ldots, x_{i-m+d}) \ .$$

Along the same line of thoughts as before, one can wonder which properties are conserved when passing from $A^{\mathbb{N}}$ to $A^{\mathbb{Z}}$ using the same local rule (with memory $m \leq 0$). The opposite case, *i.e.*, when passing from $A^{\mathbb{Z}}$ to $A^{\mathbb{N}}$ is trivial.

In [3], Blanchard and Maass show a deep combinatorial characterization of expansive CA on $A^{\mathbb{N}}$. These results were successively extended by Boyle and Fiebig [5]. Unfortunately, both the constructions are not of help for the $A^{\mathbb{Z}}$ case.

In this paper we show that most of the interesting properties are conserved when passing from $A^{\mathbb{N}}$ to $A^{\mathbb{Z}}$. If the same holds for DPO is still an open question.

2 Topology and Dynamical Properties

In order to study the dynamical properties of CA, $A^{\mathbb{Z}}$ is usually equipped with the Thychonoff metric d defined as follows

$$\forall x, y \in A^{\mathbb{Z}},\ d(x,y) = 2^{-n},\ \text{where}\ n = \min\{i \geq 0 \ :\ x_i \neq y_i \text{ or } x_{-i} \neq y_{-i}\} \ .$$

Then $A^{\mathbb{Z}}$ is a compact, totally disconnected and perfect topological space. For any pair $i, j \in \mathbb{Z}$, with $i \leq j$, we denote by $x_{[i,j]}$ the word $x_i \cdots x_j \in A^{j-i+1}$, *i.e.*, the portion of the configuration $x \in A^{\mathbb{Z}}$ inside the integer interval $[i,j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$. A *cylinder* of block $u \in A^k$ and position $i \in \mathbb{Z}$ is the set $C_i(u) = \{x \in A^{\mathbb{Z}} : x_{[i,i+k-1]} = u\}$. Cylinders are clopen sets *w.r.t.* the Thyconoff metric.

Given a CA F_m, a configuration $x \in A^{\mathbb{Z}}$ is a *periodic* point of F_m if there exists an integer $p > 0$ such that $F_m^p(x) = x$. The minimum p for which $F_m^p(x) = x$ holds is called *period* of x. If the set of all periodic points of F_m is dense in $A^{\mathbb{Z}}$, we say that the CA has the *denseness of periodic orbits* (DPO). A CA F_m has the *joint denseness of periodic orbits* (JDPO) if it has a dense set of points which are periodic both for F_m and σ.

The study of the chaotic behavior of CA (and more in general of discrete dynamical systems) is interesting and it captured the attention of researchers in the last twenty years. Although there is not a universally accepted definition of chaos, the notion introduced by Devaney is the most popular one [9]. It is characterized by three properties: sensitivity to the initial conditions, DPO and transitivity.

Recall that a CA F_m is *sensitive to the initial conditions* (or simply *sensitive*) if there exists a constant $\varepsilon > 0$ such that for any configuration $x \in A^{\mathbb{Z}}$ and any $\delta > 0$ there is a configuration $y \in A^{\mathbb{Z}}$ such that $d(y,x) < \delta$ and $d(F_m^n(y), F_m^n(x)) > \varepsilon$ for some $n \in \mathbb{N}$. A CA F_m is *(topologically) transitive* if for any pair of non-empty open sets $U, V \subseteq A^{\mathbb{Z}}$ there exists an integer $n \in \mathbb{N}$ such that $F_m^n(U) \cap V \neq \emptyset$. All the transitive CA are sensitive [8]. A CA is *(topologically) mixing* if for any pair of non-empty open sets $U, V \subseteq A^{\mathbb{Z}}$ there exists an integer $n \in \mathbb{N}$ such that for any $t \geq n$ we have $F_m^t(U) \cap V \neq \emptyset$. Trivially, any mixing CA is also transitive.

Let F_m be a CA. A configuration $x \in A^{\mathbb{Z}}$ is an *equicontinuous point* for F_m if $\forall \varepsilon > 0$ there exists $\delta > 0$ such that for all $y \in A^{\mathbb{Z}}$, $d(y, x) < \delta$ implies that $\forall n \in \mathbb{N}$, $d(F_m^n(y), F_m^n(x)) < \varepsilon$. A CA is said to be *equicontinuous* if the set E of all its equicontinuous points is the whole $A^{\mathbb{Z}}$, while it is said to be *almost equicontinuous* if E is residual (*i.e.*, E can be obtained by a infinite intersection of dense open subsets). In [11], Kůrka proved that a CA is almost equicontinuous iff it is non-sensitive iff it admits a blocking word[2].

All the above definitions can be easily adapted to work on $A^{\mathbb{N}}$.

3 Shifting

Let $\langle A^{\mathbb{Z}}, F_m \rangle$ be a CA. For a fixed $h \in \mathbb{Z}$, we consider the CA $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$. Since $F_{m+h} = \sigma^h \circ F_m$, we say that the CA $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ is obtained by a *shifting operation* which move the memory of the originally given CA from m to $m + h$. In this section we study which properties are preserved by the shifting operation.

A CA $\langle A^{\mathbb{Z}}, F_m \rangle$ is *surjective* (resp., *injective*) if F_m is surjective (resp., injective). It is *right* (resp., *left*) *closing* iff $F_m(x) \neq F_m(y)$ for any pair $x, y \in A^{\mathbb{Z}}$ of distinct left (resp., right) asymptotic configurations, *i.e.*, $x_{(-\infty,n]} = y_{(-\infty,n]}$ (resp., $x_{[n,\infty)} = y_{[n,\infty)}$) for some $n \in \mathbb{Z}$, where $z_{(-\infty,n]}$ (resp., $z_{[n,\infty)}$) denotes the portion of a configuration z inside the infinite integer interval $(-\infty, n]$ (resp., $[n, \infty)$). A CA is said to be *closing* if it is either left or right closing. Recall that a closing CA has JDPO [6] and that a CA is open iff it is both left and right closing [6].

Proposition 1. *Let $\langle A^{\mathbb{Z}}, F_m \rangle$ be a CA. For any $h \in \mathbb{Z}$, $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ is surjective (resp., injective, right-closing, left-closing, has JDPO) iff $\langle A^{\mathbb{Z}}, F_m \rangle$ is surjective (resp., injective, right-closing, left-closing, has JDPO).*

Proof. All the statements follow immediately from the definition of F_{m+h}. □

The following theorem establishes the behavior of the shift operation *w.r.t.* sensitivity, equicontinuity and almost equicontinuity. Its proof is essentially contained in [13].

Theorem 1. *For any CA $\langle A^{\mathbb{Z}}, F_m \rangle$ one and only one of the following statements holds:*

$\mathcal{S}_0$: *the CA $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ is nilpotent[3] (and then equicontinuous) for any $h \in \mathbb{Z}$;*

[2] A word $u \in A^k$ is s-blocking ($s \leq k$) for a CA F_m if there exists an offset $j \in [0, k-s]$ such that for any $x, y \in C_0(u)$ and any $n \in \mathbb{N}$, $F_m(x)_{[j,j+s-1]} = F_m(y)_{[j,j+s-1]}$.

[3] A CA F_m is nilpotent if there is a symbol $a \in A$ and an integer $n > 0$ such that for any configuration $x \in A^{\mathbb{Z}}$ we have $F_m^n(x) = (a)^\infty$ (infinite concatenation of a with itself).

$\mathcal{S}_1$: *there exists an integer* $\bar{h}$ *with* $\bar{h}+m \in [-d,d]$ *such that the CA* $\langle A^{\mathbb{Z}}, F_{m+h}\rangle$ *is equicontinuous for* $h=\bar{h}$ *and it is sensitive for any* $h \neq \bar{h}$*;*

$\mathcal{S}_2$: *there is a finite interval* $I \subset \mathbb{Z}$*, with* $I+m \subseteq [-d,d]$*, such that the CA* $\langle A^{\mathbb{Z}}, F_{m+h}\rangle$ *is strictly almost equicontinuous but not equicontinuous iff* $h \in I$ *(and then it is sensitive for any other* $h \in \mathbb{Z}\setminus I$*);*

$\mathcal{S}_3$: *the CA* $\langle A^{\mathbb{Z}}, F_{m+h}\rangle$ *is sensitive (*ever-sensitivity*) for any* $h \in \mathbb{Z}$*.*

In the case of surjective CA, Theorem 1 can be restated as follows.

Theorem 2. *For any surjective CA* $\langle A^{\mathbb{Z}}, F_m\rangle$ *one and only one of the following statements holds:*

$\mathcal{S}'_1$: *there exists an integer* h'*, with* $h'+m \in [-d,d]$*, such that the CA* F_{m+h} *is equicontinuous for* $h=h'$ *and it is mixing for* $h \neq h'$*;*

$\mathcal{S}'_2$: *there exists an integer* h'*, with* $h'+m \in [-d,d]$*, such that the CA* F_{m+h} *is strictly almost equicontinuous but not equicontinuous for* $h=h'$ *and it is mixing for* $h \neq h'$*;*

$\mathcal{S}'_3$: *there is at most a finite set* $I \subset \mathbb{Z}$*, with* $I+m \subseteq [-d,d]$*, such that if* $h \in I$ *then the CA* F_{m+h} *is sensitive but not mixing, while it is mixing if* $h \in \mathbb{Z}\setminus I$*.*

Proof. By Theorem 1, it is enough to prove that if a surjective CA F_m is almost equicontinuous then for any $h \neq 0$ the CA F_{m+h} is mixing. We give the proof for $h>0$, the other case is similar. Let $u \in A^k$ and $v \in A^q$ be two arbitrary blocks and let $w \in A^s$ be a r-blocking word with offset j where r is the radius of the CA. The word wvw is a l-blocking ($l=s+q+r$) with offset j and the configuration $y=(wv)^{\infty}$ is periodic for F_m (see, for instance, the proof of Theorem 5.24 in [12]). Let $p>0$ be the period of y. Then for any configuration $x \in C_0(wvw)$ and any $n \in \mathbb{N}$ we have that $F_m^{p+n}(x)_{[j,j+l-1]} = F_m^n(x)_{[j,j+l-1]}$, in particular $F_m^p(x)_{[s,s+q-1]} = x_{[s,s+q-1]} = v$. Let $t_0>0$ be a multiple of p such that $ht_0-s \geq k$ and $ht_0-s+(p-1)(h-r) \geq k$. For any integer $t \geq t_0$, let us consider a configuration $z \in C_0(u) \cap C_{ht-s+a(h-r)}(v')$ where $a=t \mod p$ and $v' \in f^{-a}(wvw)$ is an a-preimage block of wvw. In this way we are sure that $F_{m+h}^t(z) \in C_0(v)$ and thus the CA F_{m+h} is mixing. □

We recall that a CA F_m is *positively expansive* if there exists a constant $\varepsilon>0$ such that for any pair of distinct configurations x,y we have $d(F_m^n(y),F_m^n(x)) \geq \varepsilon$ for some $n \in \mathbb{N}$. The next proposition assures that (positively) expansive CA are in class $\mathcal{S}'_3$, in particular they are ever-sensitive.

Proposition 2. *If* $\langle A^{\mathbb{Z}}, F_m\rangle$ *is a positively expansive CA, then for any* $h \in \mathbb{Z}$ *the CA* $\langle A^{\mathbb{Z}}, F_{m+h}\rangle$ *is sensitive.*

Proof. For $h=0$ the thesis immediately follows by perfectness of $A^{\mathbb{Z}}$. We give the proof for $h>0$, the case $h<0$ is similar. Let $q=\max\{r,h,s\}+1$, where r is the radius of F_m and $s \in \mathbb{N}$ is an integer such that $\frac{1}{2^s}$ is less than the expansivity constant of the given CA. We show that $\langle A^{\mathbb{Z}}, F_{m+h}\rangle$ is sensitive with sensitivity constant $\epsilon=\frac{1}{2^q}$. Chosen an arbitrary $k \in \mathbb{N}$ and a configuration $x \in A^{\mathbb{Z}}$, consider a configuration $y \in A^{\mathbb{Z}}$ such that $y_{(-\infty,k]} = x_{(-\infty,k]}$ with $y_{k+1} \neq x_{k+1}$. By the expansivity of F_m, the sequence $\{j_n\}_{n\in\mathbb{N}} = \min\{i \in \mathbb{Z} : F_m^n(y)_i \neq F_m^n(x)_i\}$ is well-defined and for any $n \in \mathbb{N}$ we have $j_{n+1} - j_n \geq -r$. We now prove the existence of an integer $t \in \mathbb{N}$ such that $F_m^t(y)_{[-q+ht,q+ht]} \neq F_m^t(x)_{[-q+ht,q+ht]}$ (equivalent to $d(F_{m+h}^t(y), F_{m+h}^t(x)) \geq \epsilon$). By contradiction, assume that no integer satisfies this condition. Thus, for any $t \in \mathbb{N}$, we have $j_t \notin [-q+ht, q+ht]$. If for all $t \in \mathbb{N}$, $j_t > q + ht$ we obtain a contradiction since the original CA is positively expansive. Otherwise, there is an integer $t \in \mathbb{N}$, such that $j_t < -q+ht$ and $j_{t-1} > q+h(t-1)$. In this way, we have $j_t - j_{t-1} < -2q+h < -r$, obtaining again a contradiction. □

The following is a long-standing conjecture in CA theory which dates back at least to [4].

Conjecture 1. Any Surjective CA has DPO.

By Proposition 1, we have that the shift operation conserves surjectivity. Therefore, Conjecture 1 leads naturally to the following.

Conjecture 2. For any CA $\langle A^{\mathbb{Z}}, F_m \rangle$ and any $h \in \mathbb{Z}$, $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ has DPO iff $\langle A^{\mathbb{Z}}, F_m \rangle$ has DPO.

Recall that surjective almost equicontinuous CA have JDPO [4] and that closing CA have JDPO [6]. By Proposition 1, JDPO is preserved by the shifting operation, so all the CA in the classes $\mathcal{S}'_1$ and $\mathcal{S}'_2$ have JDPO. We conjecture that the same holds for (non closing) CA in $\mathcal{S}'_3$:

Conjecture 3. A CA has DPO if it has JDPO.

We want show the equivalence between Conjectures 2 and 3 but before we need the following notion. A CA $\langle A^{\mathbb{Z}}, F_m \rangle$ is *strictly right* (resp., *strictly left*) if $m < 0$ (resp., $d - m < 0$).

Proposition 3. *A surjective strictly right (or strictly left) CA is mixing.*

Proof. We give the proof for a strictly right CA, the case of strictly left CA is similar. Consider a strictly right CA with memory m and diameter d. For any $u, v \in A^\star$ and $i, j \in \mathbb{Z}$, consider the two cylinders $C_i(u)$ and $C_j(v)$. Fix $t \in \mathbb{N}$ such that $i - (d-m)t + |u| < j$. Let $x \in C_i(u)$. The value of x in $[i, i+|u|]$ depends only on the value of $F_m^{-t}(x)$ in $[i - mt, i - (d-m)t + |u| - 1]$. Therefore build $y \in A^{\mathbb{Z}}$ such that $\forall k \in \mathbb{Z}, y_k = v_{k-j+1}$ if $j \leq k \leq j + |v|$ and $y_k = x_k$, otherwise. Then $F_m^t(y) \in C_i(u)$ and $y \in C_j(v)$. □

Proposition 4 (Theorem 3.2 in [7]). *Let $\langle A^{\mathbb{Z}}, F_m \rangle$ be a strictly right CA. Any periodic configuration for the CA is also periodic for σ.*

The following corollary is a trivial consequence of the previous proposition.

Corollary 1. *Consider a strictly right CA. If it has DPO then it has JDPO too.*

Proposition 5. *The following statements are equivalent:*

1. *for any $h \in \mathbb{Z}$, $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ has DPO iff $\langle A^{\mathbb{Z}}, F_m \rangle$ has DPO (Conjecture 2).*
2. *if $\langle A^{\mathbb{Z}}, F_m \rangle$ has DPO, then it also has JDPO (Conjecture 3).*

Proof. ($1 \Rightarrow 2$). Let $\langle A^{\mathbb{Z}}, F_m \rangle$ be a CA with DPO. There exists an integer h such that the $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ is a strictly right CA. Then, by the hypothesis, it has DPO. Corollary 1 and Proposition 1 assure that $\langle A^{\mathbb{Z}}, F_m \rangle$ has JDPO. ($2 \Rightarrow 1$). By the hypothesis, a CA $\langle A^{\mathbb{Z}}, F_m \rangle$ has DPO iff it has JDPO. Proposition 1 concludes the proof. □

As a by-product of our investigations we have the following result.

Theorem 3. *In the CA settings, the following statements are equivalent*

1. *surjectivity implies DPO;*
2. *surjectivity implies JDPO;*
3. *for strictly right CA, topological mixing implies DPO.*

Proof. (1. $\Leftrightarrow$ 3.) It is obvious that 1. implies 3. For the opposite implication assume that 3 is true. Let $\langle A^{\mathbb{Z}}, F_m \rangle$ be a surjective CA. There exists an integer h such that $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ is strictly right. By Propositions 1 and 3, $\langle A^{\mathbb{Z}}, F_{m+h} \rangle$ is a surjective and topologically mixing CA. Then, by the hypothesis, it has DPO. Corollary 1 assures that it also has JDPO. Using Proposition 1 again we conclude the proof. The proof for (1. $\Leftrightarrow$ 2.) can be obtained in a similar way. □

Theorem 3 tells that in order to prove Conjecture 1 one can focus on mixing strictly right CA. Remark that all known examples of topologically mixing CA have DPO. We want to present a result which furthermore support the common feeling that Conjecture 1 is true. First, we need a technical lemma.

Lemma 1. *Any configuration of a CA (on $A^{\mathbb{N}}$ or $A^{\mathbb{Z}}$) which is σ-periodic of period p has a CA image which is σ-periodic with period p' that divides p.*

Proof. Consider a CA F_m on $A^{\mathbb{Z}}$. Let x be a periodic configuration of σ with period p. By the definition of CA, $F_m(x)$ is still a periodic configuration of period q for σ but, in general, $q \leq p$. If $q = p$ then we are done. Otherwise, let m be the largest integer such that $mq < p$. By the Hedlund's theorem we have

$$F_m \circ \sigma^p(x) = \sigma^p \circ F_m(x) \ . \tag{1}$$

Compose both members of (1) with σ^{-mq}. The left-hand side gives $\sigma^{-mq} \circ F_m \circ \sigma^p(x) = \sigma^{-mq} \circ F_m(x) = F_m(\sigma^{-mq}(x)) = F_m(x)$ since x (resp., $F(x)$) is periodic of period p (resp., q) for σ. The right-hand side gives simply $\sigma^{p-mq}(F_m(x))$. And hence (1) can be rewritten as $F(x) = \sigma^{p-mq}(F_m(x))$ which implies that $p - mq = q$ since q is the period of $F(x)$ according to σ. We conclude that $p = (m+1)q$. □

Proposition 6. *Any surjective CA (both on $A^{\mathbb{Z}}$ and on $A^{\mathbb{N}}$) has an infinite set of points which are jointly periodic for the CA and σ.*

Proof. Consider a CA F_m on $A^{\mathbb{Z}}$. By Lemma 1, given a σ-periodic configuration x of period p, $F_m(x)$ is periodic for σ with period p' which divides p. Let Π_n be the set of periodic points of σ of period n. Hence, if p is a prime number and $x \in \Pi_p$ we have that $F_m(x)$ belongs either to Π_p or to Π_1. By a result of Hedlund [10], we know that for surjective CA, each configuration has a finite number of pre-images. In particular, each Π_p has a finite number of pre-images by F_m. Hence, if p is a big enough prime number we have that $F_m(\Pi_p) \cap \Pi_p \neq \emptyset$. This concludes the proof since for any prime number $p' > p$ we must have $F_m(\Pi_p) \subseteq \Pi_p$. □

4 Lifting

For a fixed local rule f, consider the two one-sided CA $\langle A^{\mathbb{Z}}, F_m \rangle$ and $\langle A^{\mathbb{N}}, \Phi_m \rangle$ on $A^{\mathbb{Z}}$ and $A^{\mathbb{N}}$, respectively. They share the same local rule f and the same memory $m \leq 0$. In this section we study the properties that are conserved when passing from $A^{\mathbb{Z}}$ to $A^{\mathbb{N}}$ and *vice-versa*.

Consider the *projection* $P : A^{\mathbb{Z}} \to A^{\mathbb{N}}$ defined as follows: $\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{N}, \quad P(x)_i = x_i$. Then, P is a continuous, open, and surjective function. Moreover, $\Phi_m \circ P = P \circ F_m$. Therefore, the CA $\langle A^{\mathbb{N}}, \Phi_m \rangle$ is a factor of $\langle A^{\mathbb{Z}}, F_m \rangle$. For these reasons, we also say that the CA on $A^{\mathbb{Z}}$ is obtained by a *lifting (up) operation* from the CA on $A^{\mathbb{N}}$ (having the same rule and memory). As an immediate consequence of the fact that $\Phi_m \circ P = P \circ F_m$, the CA on $A^{\mathbb{N}}$ inherits from the CA on $A^{\mathbb{Z}}$ several properties such as surjectivity, left closingness, openess, DPO, transitivity, mixing.

The following proposition shows that the injectivity property is lifted down only under special conditions. Proposition 8 proves that the opposite case (lift up) is verified without further hypothesis.

Proposition 7. *Let $\langle A^{\mathbb{Z}}, F_m \rangle$ be an injective one-sided CA. The CA $\langle A^{\mathbb{Z}}, \Phi_m \rangle$ is injective if and only if $m = 0$ and the left Welch index $L(f) = 1$*[4].

Proof. If Φ_m is injective (and then also surjective) we necessarily have $m = 0$. By contradiction, if $L(f) \geq 2$ there exist blocks u, w, w', v, with $w \neq w'$, such that $f(wu) = f(w'u) = v$. So $\Phi_0(x) = \Phi_0(y)$ for two suitable right-asymptotic configurations x, y obtained by extending to the right the words wu and $w'u$, respectively. Conversely let us assume that $L(f) = 1$ and $m = 0$. If, by contradiction, Φ_0 is not injective then there exist two distinct configurations $x, y \in A^{\mathbb{N}}$ having the same image $z \in A^{\mathbb{N}}$. By surjectivity, we have two cases to trait. In the first one, x and y are right asymptotic. So there is a block with two distinct right extensions which collapse by f in the same word, contrary to the fact that $L(f) = 1$. In the second one, for any i large enough we have $x_{[i,i+d-1]} \neq y_{[i,i+d-1]}$. By a result in [10], $L(f) = 1$ implies that f is leftmost permutive[5]. Thus we

[4] The left Welch index $L(f)$ is an un upper bound for the number of the left possible extensions of a block u which collapse by f in the same word. For a formal definition, see for instance [10]).

[5] A rule $f : A^{d+1} \to A$ is leftmost permutive iff for any $u \in A^d$, $b \in A$ there exists $a \in A$ such that $f(au) = b$.

are able to build two distinct configurations $x', y' \in A^{\mathbb{Z}}$, with $x = P(x')$ and $y = P(y')$, such that $F_0(x') = F_0(y')$, contrary to the hypothesis. $\square$

Proposition 8. *If $\langle A^{\mathbb{N}}, \Phi_m \rangle$ is an injective (resp., surjective) CA, then the lifted CA $\langle A^{\mathbb{Z}}, F_m \rangle$ is injective (resp., surjective).*

Proof. Assume that F_m is not injective. There exist two configurations $x, y \in A^{\mathbb{Z}}$, with $x_i \neq y_i$ for some $i \geq 0$, such that $F_m(x) = F_m(y)$. Therefore $x' = P(x)$ and $y' = P(y)$ are two different configurations in $A^{\mathbb{N}}$ such that $\Phi_m(x') = \Phi_m(y')$.

By a theorem of Hedlund [10], we have that a CA is surjective (on $A^{\mathbb{N}}$ or on $A^{\mathbb{Z}}$) iff for any $u \in A^+, |f^{-1}(u)| = A^d$. $\square$

The following result can be obtained immediately from the definitions.

Proposition 9. *Left-closingness is conserved by the lifting up operation.*

The following results is obtained immediately from the fact that a word is blocking for a CA on $A^{\mathbb{N}}$ iff it is blocking for its lifted CA.

Proposition 10. *A CA $\langle A^{\mathbb{N}}, \Phi_m \rangle$ is equicontinous (resp., almost equicontinuous) (resp., sensitive) iff the CA $\langle A^{\mathbb{N}}, F_m \rangle$ is equicontinous (resp., almost equicontinous) (resp., sensitive).*

Positive expansivity is not preserved by the lifting operation since, by Proposition 11, there are no positively expansive one-sided CA on $A^{\mathbb{Z}}$.

Proposition 11. *No one-sided CA $\langle A^{\mathbb{Z}}, F_m \rangle$ is positively expansive.*

Proof. For the sake of argument let us assume that the CA $\langle A^{\mathbb{Z}}, F_m \rangle$ be a positively expansive one-sided CA with expansivity constant $\epsilon > 0$. Let k be an integer such that $\frac{1}{2^k} < \epsilon$ and let $x, y \in A^{\mathbb{Z}}$ be two different configurations with $x_{[-k,\infty)} = y_{[-k,\infty)}$. We have that for any $t \in \mathbb{N}$, $d(F_m^t(x), F_m^t(y)) < \epsilon$. $\square$

Proposition 12. *If $\langle A^{\mathbb{N}}, \Phi_m \rangle$ is mixing (resp., transitive), then its lifted CA $\langle A^{\mathbb{Z}}, F_m \rangle$ is mixing (resp., transitive).*

Proof. We prove the thesis for a topologically mixing CA. By Proposition 3, it is sufficient to consider the case $m = 0$. Let $u \in A^{2k+1}$, $v \in A^{2h+1}$ be two arbitrary blocks. There exist a sequence of one-sided configurations $x^{(n)} \in C_{l-k}(u)$ and a time $t_0 \in \mathbb{N}$ such that for any $t \geq t_0$, $\Phi_0^t(x^{(t-t_0)}) \in C_{l-h}(v)$ where $l = \max\{h, k\}$. Let $z^{(n)} \in A^{\mathbb{Z}}$ be a sequence of two-sided configurations such that $z^{(n)}_{[-k,\infty)} = x^{(n)}$. We have that for any $t \geq t_0$, $F_0^t(z^{(t-t_0)}) \in C_{-h}(v)$. $\square$

The lifting (up) of DPO remains an open problem even if on the basis of the results of Section 3 we conjecture that it holds.

5 Conclusions and Future Works

In this paper we studied the behavior of two operations on the rule space of CA, namely, the shifting and lifting operations. These investigations helped to shape out a new scenario for the old- standing conjecture about the equivalence between surjectivity and DPO for CA: the study can be restricted to topologically mixing strictly right CA. This enhances a former idea of Blanchard [1]. The work can be continued along several directions: transitivity and stronger variants, languages generated by the involved CA, and attractors. Moreover a generalization of the obtained results and of the forthcoming studies can be considered in terms of directional dynamics introduced in [13]. The authors are currently investigating these subjects.

References

[1] Blanchard, F.: Dense periodic points in cellular automata, `http://www.math.iupui.edu/∼mmisiure/open/`

[2] Blanchard, F., Maass, A.: Dynamical behavior of Coven's aperiodic cellular automata. Theoretical Computer Science 163, 291–302 (1996)

[3] Blanchard, F., Maass, A.: Dynamical properties of expansive one-sided cellular automata, Israel Journal of Mathematics 99 pp. 149–174 (1997)

[4] Blanchard, F., Tisseur, P.: Some properties of cellular automata with equicontinuity points. Ann. Inst. Henri Poincaré, Probabilité et Statistiques 36, 569–582 (2000)

[5] Boyle, M., Fiebig, D., Fiebig, U.: A dimension group for local homeomorphisms and endomorphisms of onesided shifts of finite type. Journal f ur die Reine und Angewandte Mathematik 487, 27–59 (1997)

[6] Boyle, M., Kitchens, B.: Periodic points for cellular automata. Indag. Math. 10, 483–493 (1999)

[7] Cattaneo, G., Finelli, M., Margara, L.: Investigating topological chaos by elementary cellular automata dynamics. Theoretical Computer Science 244, 219–241 (2000)

[8] Codenotti, B., Margara, L.: Transitive cellular automata are sensitive. American Mathematical Monthly 103, 58–62 (1996)

[9] Devaney, R.L.: An introduction to chaotic dynamical systems, 2nd edn. Addison-Wesley, London (1989)

[10] Hedlund, G.A.: Endomorphism and automorphism of the shift dynamical system. Mathematical System Theory 3, 320–375 (1969)

[11] Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Ergod. Th. & Dynam. Sys. 17, 417–433 (1997)

[12] Kůrka, P.: Topological symbolic dynamics, Volume 11 of Cours Spécialisés, Société Mathématique de France (2004)

[13] Sablik, M.: Directional dynamics for cellular automata. a sensitivity to the initial conditions approach. Preprint (2006)

Learning as Data Compression*

Pieter Adriaans

Department of Computer Science
University of Amsterdam,
Kruislaan 419,
1098VA Amsterdam,
The Netherlands
pietera@science.uva.nl

Abstract. In this paper I describe the general principles of learning as data compression. I introduce two-part code optimization and analyze the theoretical background in terms of Kolmogorov complexity. The good news is that the optimal compression theoretically represents the optimal interpretation of the data, the bad news is that such an optimal compression cannot be computed and that an increase in compression not necessarily implies a better theory. I discuss the application of these insights to DFA induction.

Keywords: learning as compression, MDL, two-part code optimization, randomness deficiency, DFA induction.

1 Learning as Compression

Since the beginning of science in antiquity, the idea that the complexity of the world can be explained in terms of some simple first principles has fascinated researchers. In modern methodology of science this notion is studied under various guises: Occams razor [7], the minimal description length (MDL) principle [8], two-part-code optimization [11], learning as data compression [21] etc. Although there has been some debate about this principle with fierce opponents [7] and strong defenders [21], until recently the view of learning as data compression did not seem to have much practical value. Lots of learning algorithms in fact perform some kind of data compression, but this was not a guiding principle of their design [9; 20]. Two developments in the last five years have changed this perspective quite fundamentally : 1) a better understanding of the mathematics behind compression, specifically Kolmogorovs structure function [11; 10] and 2) the application of existing implementations of compression algorithms to approximate the ideal (and uncomputable) Kolmogorov complexity as pioneered by Cilibrasi and Vitanyi [5; 6]. At this moment we have not only a much better

* This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

understanding of the theoretical issues behind data compression, but there is also a wealth of interesting and successful applications. Due to limited space in this paper I will restrict myself to a description of the general principles and a study of the application of MDL to DFA induction [1; 4]. In the tutorial itself I will also describe a number of other applications (e.g. Normalized Compression Distance) [5] and I will touch on some philosophical issues: the relation between data compression, thermodynamics and human cognition [2].

Take a cup of coffee and pour some cream in it (See Figure 1). Take a picture of it with your digital camera. In the beginning the cream will be just an uninteresting blob. Stir slowly and make pictures of various stages that have nice patterns. Continue until the cream has dissolved and your cup has an even brown color. Drink the coffee, then look at the file size of the different pictures. If your camera uses an adequate compression algorithm you will find that the file size has increased up to a certain point and then decreases. The compression algorithm of your camera reflects the complexity of the data set until the moment that the complexity has reached a global equilibrium and is beyond its resolution. In this experiment we have a system that evolves in time, the cup of coffee, and a data set of observations, the pictures. The crux of this experiment is that the size of the individual pictures somehow reflects the 'interestingness' of the system. In the beginning there is a lot of order in the system. This is not very interesting. In the end there is an equilibrium that also has little cognitive appeal.

In general science, in the study of human cognition and even in art we seem to have an interest in systems that have a complexity in the 'sweet spot' between order and chaos, between boredom and noise. The 'interestingness' of these data sets is somehow related to compressibility. It will prove useful to describe these compressions in terms of a so-called two-part-code: a description of a general class of sets, the *model code* and an element or a set of elements of this set, the *data-to-model-code* [11; 10].

Let me give some examples:

– **Symmetry.** This is one of the most fundamental ordering principles in nature. Most living creatures have symmetry: plants, trees, predator, prey. If a data set has symmetry it means that we only have to describe half of it (the data-to-model-code) plus some information about the nature of the symmetry of constant length (the model-code). In the limit such a data set can be compressed to at least half its size. In terms of generating languages symmetry is context free: a symmetric data set can be produced by a simple memoryless central process. Discovering symmetry in a data set can be seen as a very simple learning problem. It can easily be discovered in linear time.
– **Repetition.** In order to describe a repeating pattern I only have to give a description of the generating pattern (the data-to-model-code) and some information about the way the pattern repeats itself (the model-code). Repetition is more complex than symmetry in the sense that it presupposes a generating process with a memory: in terms of languages repetition is context

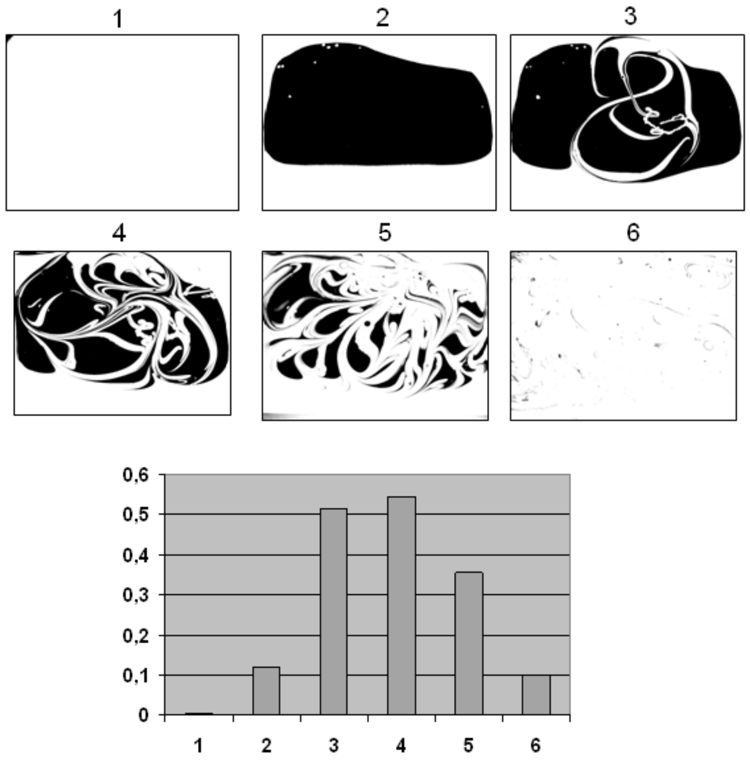


Fig. 1. Facticity scores for mixing black and white paint. The facticity of a data x is the product of the normalized entropy $K(x)/U(x)$ and the normalized randomness deficiency $(U(x) - K(x))/U(x)$. Configuration 4 has the best balance between order and chaos and thus would be the most 'interesting' one. The scores have been calculated using JPEG, followed by RAR compression. Maximal entropy $U(x)$ has been approximated by adding 400 % noise to the images. The standard entropy $K(x)$ is approximated by the file size after compression.

sensitive. Finding repeating patterns in a data set is also a basic learning problem that can be solved in time $n \log n$ [3].

– **Grammar.** A corpus of a language could be described in terms of the grammar G (the model-code) of the language and a set of indexes corresponding to an enumeration of the sentences in the corpus (the data-to-model-code). If the size of the corpus is large enough in relation to the size of the grammar G then this description in terms of two will be shorter than an extensional description of the sentences in the corpus. Finding this description is a well studied learning problem. If the language is regular then the task of approximating the smallest DFA consistent with a set of sentences is NP-hard [14; 4].
– **Program.V** We could ask ourselves, given a certain data set: what would be the shortest program generating this data set in a certain programming

language, or, even more general, we could try to find the shortest combination of a Turing machinenoindent T_i (the model-code) and a program P (the data-to-model-code). In a sense this would be, from a computational point of view, the ultimate compression possible and the Turing machine T_i would be the ultimate 'explanation' of the data set. Needless to say that because of the Halting problem there is no algorithm that will construct this ultimate compression for us. The problem is undecidable. Still, conditional to the programming language we choose, the notion of the shortest program generating a certain data set is well defined. Kolmogorov complexity studies these optimal compressions from the perspective of universal Turing machines [10].

Here I have described four classes of learning problems (varying from very easy, via NP-hard, to undecidable) as compression problems where the task is to find a two-part code compression for a data set. Apparently there is a deep connection between data compression and learning. In this tutorial I will also describe the theory behind these phenomena and explain how they can be used to develop algorithms to analyze data sets and understand the way they work.

2 MDL as Two-Part Code Optimization

It is important to note that two part code optimization is a specific application of MDL. The majority of work on MDL is closer in spirit to the statistical than to the Kolmogorov complexity world. Rather than two-part codes, one uses general universal codes for individual sequences; two-part codes are only a special case. We give the traditional formulation of MDL [9; 8]:

Definition 1. The Minimum Description Length principle*: The best theory to explain a set of data is the one which minimizes the sum of*

- *the length, in bits, of the description of the theory and*
- *the length, in bits, of the data when encoded with the help of the theory*

Let $M \in \mathcal{M}$ be a model in a class of models $\mathcal{M}$, and let D be a data set. The **prior probability** of a hypothesis or model M is $P(M)$. Probability of the data D is $P(D)$. **Posterior probability** of the model given the data is:

$$P(M|D) = \frac{P(M)P(D|M)}{P(D)}$$

The following derivation [9] illustrates the well known equivalence between MDL and the selection of the Maximum A posteriori hypothesis in the context of Shannon's information theory. Selecting the **Maximum A Posteriori hypothesis (MAP)**:

$$M_{MAP} \equiv argmax_{M \in \mathcal{M}}\ P(M|D)$$

$$= argmax_{M \in \mathcal{M}}\ (P(M)P(D|M))/P(D)$$

(since D is constant)

$$\equiv argmax_{M \in \mathcal{M}}\ (P(M)P(D|M))$$

$$\equiv argmax_{M \in \mathcal{M}} \log P(M) + \log P(D|M)$$

$$\equiv argmin_{M \in \mathcal{M}} - \log P(M) - \log P(D|M)$$

where according to Shannon $-\log P(M)$ is the length of the optimal *model-code* in bits and $-\log P(D|M)$ is the length of the optimal *data-to-mode-code* in bits. This implies that the model that is chosen with Bayes' rule is equal to the model that MDL would select:

$$M_{MAP} \equiv M_{MDL}$$

The formula $argmin_{M \in \mathcal{M}} - \log P(M) - \log P(D|M)$ indicates that a model that generates an optimal data compression (i.e. the shortest code) is also the best model. This is true even if $\mathcal{M}$ does not contain the original intended model as was proved by [11]. It also suggests that compression algorithms can be used to approximate an optimal solution in terms of successive steps of incremental compression of the data set D. This is *not* true as was shown by [1]. Yet this illicit use of the principle of MDL is common practice.

In order to understand these results better we must answer two questions 1) What do we mean by the length of optimal or shortest code and 2) what is an independent measure of the quality of a model M given a data set D? The respective answers to these questions are *prefix-free Kolomogorov complexity* and *randomness deficiency*.

2.1 Kolmogorov Complexity

Let $x, y, z \in \mathcal{N}$, where $\mathcal{N}$ denotes the natural numbers and we identify $\mathcal{N}$ and $\{0,1\}^*$ according to the correspondence

$$(0, \epsilon), (1, 0), (2, 1), (3, 00), (4, 01), \ldots$$

Here ϵ denotes the *empty word*. The *length* $|x|$ of x is the number of bits in the binary string x, not to be confused with the *cardinality* $|S|$ of a finite set S. For example, $|010| = 3$ and $|\epsilon| = 0$, while $|\{0,1\}^n| = 2^n$ and $|\emptyset| = 0$. The emphasis is on binary sequences only for convenience; observations in any alphabet can be encoded in a 'theory neutral' way. Below we will use the natural numbers and the binary strings interchangeably. In the rest of the paper we will interpret the set of models $\mathcal{M}$ in the following way:

Definition 2. *Given the correspondence between natural numbers and binary strings, $\mathcal{M}$ consists of an enumeration of all possible self-delimiting programs for a preselected arbitrary universal Turing machine U. Let x be an arbitrary bit* *string. The shortest program that produces x on U is $x^* = argmin_{M \in \mathcal{M}}(U(M) = x)$ and the Kolmogorov complexity of x is $K(x) = |x^*|$. The conditional Kolmogorov complexity of a string x given a string y is $K(x|y)$, this can be interpreted as the length of a program for x given input y. A string is defined to be random if $K(x) \geq |x|$.*

This makes $\mathcal{M}$ one of the most general model classes with a number of very desirable properties: it is universal since all possible programs are enumerated, because the programs are self-delimiting we can concatenate programs at will, in order to create complex objects out of simple ones we can define an a-priori complexity and probability for binary strings. There are also some less desirable properties: $K(x)$ cannot be computed (but it can be approximated) and $K(x)$ is asymptotic, i.e. since it is defined relative to an arbitrary Turing machine U it makes less sense for objects of a size that is close to the size of the definition of U. Details can be checked in [10]. We have:

$$argmin_{M \in \mathcal{M}} - \log P(M) - \log P(D|M) =$$

$$argmin_{M \in \mathcal{M}} K(M) + K(D|M) = M_{MDL} \tag{1}$$

Under this interpretation of $\mathcal{M}$, the length of the optimal code for an object is equivalent to its Kolmogorov complexity.

In this paper I will often use the notions of *typicality* and *incompressibility* of elements of a set, e.g. in those cases where I state that the vast majority of elements of a set have a certain quality. This might at first sight sound a bit inaccurate. To show that this notion actually has an exact definition I give the following theorem due to Li and Vitányi [10] pg. 109):

Theorem 1. *Let c be a positive integer. For each fixed y, every finite set A of cardinality m has at least $m(1 - 2^{-c}) + 1$ elements x with $C(x|y) \geq \log m - c$.*

Proof: The number of programs of length less than $\log m - c$ is

$$\sum_{i=0}^{\log m - c - 1} 2^i = 2^{logm - c} - 1$$

Hence, there are at least $m - m2^{-c} + 1$ elements in A that have no program of length less than $\log m - c$.

This shows that in the limit the number of elements of a set that have low Kolmogorov complexity is a vanishing fraction. In the limit a typical element of a set is a random element. In general the vast majority of elements of a set is not compressible. One of the problems with Kolmogorov complexity is that it specifies the length of a program but tells us nothing about the time complexity of the computation involved. Therefore Kolmogorov complexity can not be used directly to prove lower bounds for the time complexity of problems.

2.2 Randomness Deficiency

It is important to note that objects that are non-random are very rare. To make this more specific: in the limit the density of compressible strings x in the set $\{0,1\}^{\leq k}$ for which we have $K(x) < |x|$ is zero [10]. The overwhelming majority of strings is random. In different words: an element is *typical* for a data set if and only if it is *random* in this data set. In yet different words: if it has maximal entropy in the data set. This insight allows us to formulate a theory independent measure for the quality of models: *randomness deficiency*.

We start by giving some estimates for upper-bounds of conditional complexity. Let $x \in M$ be a string in a finite model M then

$$K(x|M) \leq \log|M| + O(1) \tag{2}$$

i.e. if we know the set M then we only have to specify an index of size $\log|M|$ to identify x in M. Consequently:

$$K(x) \leq K(M) + \log|M| + O(1) \tag{3}$$

The factor $O(1)$ is needed for additional information to reconstruct x from M and the index. Its importance is thus limited for larger data sets. These definitions motivate the famous Kolmogorov structure function:

$$h_x(\alpha) = \min_S\{\log|S| : x \in S, K(S) \leq \alpha\} \tag{4}$$

Here α limits the complexity of the model class S that we construct in order to 'explain' an object x that is identified by an index in S. Let $D \subseteq M$ be a subset of a finite model M. We specify $d = |D|$ and $m = |M|$. Now we have:

$$K(D|M,d) \leq \log\binom{m}{d} + O(1) \tag{5}$$

Here the term $\binom{m}{d}$ specifies the size of the class of possible selections of d elements out of a set of m elements. The term $\log\binom{m}{d}$ gives the length of an index for this set. If we know M and d then this index allows us to reconstruct D.

A crucial insight is that the inequalities 2 and 5 become 'close' to equalities when respectively x and D are *typical* for M, i.e. when they are random in M. This typicality can be interpreted as a measure for the goodness of fit of the model M. A model M for a data set D is optimal if D is random in M, i.e. the randomness deficiency of D in M is minimal. The following definitions formulate this intuition. The *randomness deficiency* of D in M is defined by:

$$\delta(D|M,d) = \log\binom{m}{d} - K(D|M,d), \tag{6}$$

for $D \subseteq M$, and ∞ otherwise. If the randomness deficiency is close to 0, then there are no simple special properties that single D out from the majority of data samples to be drawn from M.

The *minimal randomness deficiency* function is

$$\beta_x(\alpha) = \beta_D(\alpha) = \min_M \{\delta(D|M) : M \supseteq D,\ K(M) \leq \alpha\}, \tag{7}$$

If the randomness deficiency is minimal then the data set is typical for the theory and with high probability future data sets will share the same characteristics, i.e. minimal randomness deficiency is also a good measure for the future performance of models. For a formal proof of this intuition, see [11].

We now turn our attention to incremental compression. Equation 1 gives the length of the optimal *two-part-code*. The length of the two-part-code of an intermediate model M_i is given by:

$$\Lambda(M_i, d) = \log \binom{m_i}{d} + K(M_i) \geq K(D) - O(1) \tag{8}$$

This equation suggests that the optimal solution for a learning problem can be approximated using an incremental compression approach. This is indeed what a lot of learning algorithms seem to be doing: find a lossy compression of the data set finding regularities. This holds for such diverse approaches as nearest neighbor search, decision tree induction, induction of association rules and neural networks. There is a caveat however; [1] have shown that the randomness deficiency not necessarily decreases with the length of the MDL code, i.e. shorter code does not always give smaller randomness deficiency, e.g. a better theory. This leads to the following observations [1]:

- The optimal compression of a data set in terms of model and a data-to-model code always gives the best model approximation "irrespective of whether the 'true' model is in the model class considered or not" [11][1].
- This optimal compression cannot be computed.
- Shorter code does not necessarily mean a better model.

These observations show that the naive use of the MDL principle is quite risky.

3 A Case Study: MDL and DFA Induction

In the domain of machine learning pure applications of MDL are rare, mainly because of the difficulties one encounters trying to define an adequate model code and data-to-model code. The field of grammar induction studies a whole class of algorithms that aims at constructing a grammar by means of incremental compression of the data set represented as a digraph. This digraph can be seen as the maximal theory equivalent with the data set. Every word in the data

[1] This is true only in this specific computational framework of reference. In a probabilistic context, both for Bayesian and MDL inference, the assumption that the true model is in the model class considered can sometimes be crucial - this also explains why in Vapnik-Chervonenkis type approaches, complexity is penalized much more heavily than in MDL [12]).

set is represented as a path in the digraph with the symbols either on the edges or on the nodes. The learning process takes the form of a guided incremental compression of the data set by means of merging or clustering of the nodes in the graph. None of these algorithms explicitly makes an explicit estimate of the MDL code. Instead they use heuristics to guide the model reduction. After a certain time a proposal for a grammar can be constructed from the current state of the compressed graph. Examples of such algorithms are SP [23; 22], EMILE [15; 16], ABL [18], ADIOS [19] and a number of DFA induction algorithms, specifically evidence driven state merging (EDSM), [17; 24]. In this paragraph we present a sound theoretical basis to analyze the performance and idiosyncrasies of DFA induction in an MDL context [4]. We will follow the presentation in [20]. The general methodology for applying two-part-code optimization to a certain learning problem is:

– Design an approximation of the optimal model code. Such a model code should reflect structural changes in the model complexity in an adequate way and should at the same time be computationally feasible. In this case we will use a simple count on the nodes and edges.
– Design an approximation of the optimal data-model-code with the same desiderata, for details see below.
– Select a compression algorithm that is computationally feasible and heuristically adequate. We will use standard evidence driven state merging with MDL as optimization criterion.
– Define a start state for the learning process. This will be the so-called Maximal Canonical Automaton, the graph that exactly generates the data set from one start state.
– Define an adequate stop condition for the compression process. In this case we will simply limit the computation time.

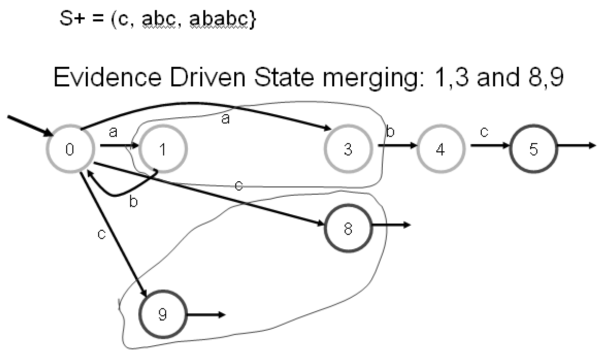


Fig. 2. Compressing a DFA (model) by means of state merging, given some set of positive examples S+

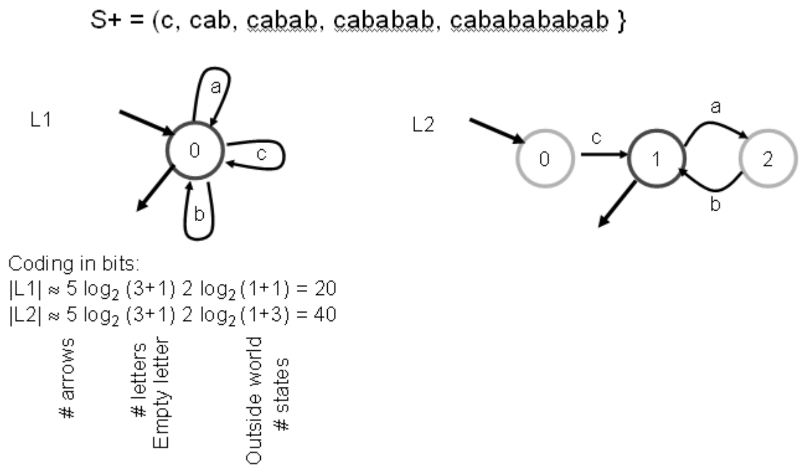

S+ = (c, cab, cabab, cababab, cabababababab }
L1
L2
0
1
2
a
b
c
Coding in bits:
|L1| ≈ 5 log2 (3+1) 2 log2 (1+1) = 20
|L2| ≈ 5 log2 (3+1) 2 log2 (1+3) = 40
# arrows
# letters
Empty letter
Outside world
# states

Fig. 3. Two DFA generating S+. L1 is the shortest model, but L2 generates the shortest MDL code.

We start with some relevant observations. We will restrict ourselves to languages in $\{0.1\}^*$. The class of DFA is equivalent to the class of regular languages. We call the set of positive examples D^+ and the set of negative examples D^-. The complement of a regular language is a regular language. Consequently the task of finding an optimal model given D^+ is symmetric to the task of finding an optimal model given D^-. The task of finding the minimum DFA consistent with a set of positive and negative examples is *decidable*. We can enumerate all DFA's according to their size and test them on the data set. Yet this minimum DFA cannot be approximated within polynomial time [14].

The task of finding the smallest DFA consistent with a set of positive examples is trivial. This is the universal DFA. Yet the universal DFA will in most cases have a poor generalization error. MDL is a possible candidate for a solution here. Suppose that we have a finite positive data set representing an infinite regular language. The task is then to find a DFA with minimum expected generalization error over the set of infinite regular languages consistent with D^+. MDL in theory identifies such a DFA.

Definition 3. *A* partition π *of a set* X *is a set of nonempty subsets of* X *such that every element* x *in* X *is in exactly one of these subsets.* $B(s, \pi) \subseteq X$ *indicates the subset of the partition* π *of which* x *is an element.*

Definition 4. *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be a DFA. The* quotient automaton $A/\pi = (Q', \Sigma, \delta', B(q_0, \pi), F')$ *derived from* A *on the basis of a partition* π *of* Q *is defined as follows:*

- $Q' = Q/\pi = \{B(q,\pi)|q \in Q\}$,
- $F' = \{B \in Q'|B \cap F \neq \emptyset\}$,
- $\delta' : (Q' \times \Sigma) \rightarrow 2^{Q'} : \forall B, B' \in Q', \forall a \in \Sigma, B' \in \delta'(B,a)$ *iff* $\exists q, q' \in Q, q \in B, q' \in B'$ *and* $q' \in \delta(q,a)$.

We say that the states in Q that belong to the same block B are merged.

We give without proof:

Lemma 1. *If an automaton A/π is derived from an automaton A by means of a partition π then $L(A) \subseteq L(A/\pi)$.*

The relevance of these definitions for grammar induction lies in the fact that we can increase or decrease the generality of the automaton and the associated language inclusion hierarchies by means of splitting and merging states. We now develop an adequate data to model code based on the idea that a positive data sample has an entropy in each node of the DFA.

Definition 5. *Let A be a DFA. An* index set *for A is a set that associates a unique natural number with each string that is accepted by A. The* index set relative to certain data set *$D \subseteq L(A)$ is $I_D = \{i|i \in \mathbf{N}, L(A)(i) \in D\}$. The* initial segment *associated with an index set D and $L(A)$ is the set $I_{\leq D} = \{i|i \in \mathbf{N}, \exists j \in I_D : j \geq i\}$, i.e. the set of all natural numbers that are smaller than or equal to an index in I_D. The* maximal entropy *of I_D in $I_{\leq D}$ is $\log \binom{|I_{\leq D}|}{|I_D|}$, where $|I_{\leq D}|$ is a measure for the total number of sentences in the language up to the sentence in D with the highest index and $|I_D|$ is the size of D.*

The notion of an initial segment is introduced to make the argument work for infinite languages. We have $K(D|A) \leq K(I_D) + O(1) \leq \log \binom{|I_{\leq D}|}{|I_D|} + O(1)$. Suppose that f is an accepting state of a DFA A, with index set I and that $D \subseteq L(A)$.

Definition 6. *The maximal* state entropy *of f given D is $I_{\leq D,f} = \log \binom{|I_{\leq D,f}|}{|I_{D,f}|}$, where $I_{\leq D,f}$ and $I_{D,f}$ identify those indexes that are associated with strings that are accepted in f.*

These theoretical definitions can be used to define a nearly optimal data-to-model code.

Suppose A is a DFA suggested as an explanation for a data set D. A has i accepting states and j non-accepting states. Since we use both positive and negative examples A must be functionally complete (i.e. have an outgoing arrow for each element of the lexicon from each state). Suppose l is the maximal length of a string in the data set D. D^+ is the set of positive examples, D^- the set of negative examples, d^+ is the number of positive examples, d^- the number of negative examples. There are $2^{l+1} - 1$ binary strings with length $\leq l$. Call this set N, then n^+ is the number of strings accepted by A and n^- is the number of strings not accepted by A. A partitions N in two sets: N^+ and N^-. N^+ is partitioned in i subsets by the i accepting states of A and N^- is partitioned in j subsets by the j non-accepting states of A. The correct data-to-model code has size:

$$\log(\prod_i \binom{n_i^+}{d_i^+} \times \prod_j \binom{n_j^-}{d_j^-}) = \sum_i (\log \binom{n_i^+}{d_i^+}) + \sum_j (\log \binom{n_j^-}{d_j^-}) \tag{9}$$

One can read this as follows. The formula specifies an index for the data set D given the data set N. There are i pieces of code for the positive states, and j pieces of code for the negative states. If there are states that do not generate elements for D then their contribution to the length of the code is 0. When applying this formula to DFA induction one must estimate the values using the Stirling formula or an integral over $\log n!$[2]. Remember that from an MDL perspective we are only interested in the length of the index, not its specific value. The beautiful thing is that this index can always be used: for positive examples, for complete examples and even for only negative examples.

We have tried this MDL approach on the problem set of the Abbadingo DFA inference competition [17]. We were able to solve problems 1, 2, A, B, C, D, and R. In comparison, standard EDSM can solve all these problems, and also problems 3, 4, 6, and S. So, indeed, it seems that MDL is not a very reliable guide for the compression of a DFA. At least, EDSM is better.

4 Conclusion

I have described the general principles behind two-part code optimization. I have studied MDL in terms of two-part code optimization and randomness deficiency for DFA induction. In this framework we noted that 1) Shorter code does not necessarily lead to better theories, e.g. the randomness deficiency does not decrease monotonically with the MDL code, 2) contrary to what is suggested by the results of [13] there is no fundamental difference between positive and negative data from an MDL perspective, 3) MDL is extremely sensitive to the correct calculation of code length. Using these ideas we have implemented a MDL variant of the EDSM algorithm [17]. The results show that although MDL works well as a global optimization criterion, it falls short of the performance of algorithms that evaluate local features of the problem space. MDL can be described as a global strategy for featureless learning. In the tutorial I will describe recent developments like normalized compression distance (NCD) and also present some philosophical reflection on the data compression and thermodynamics.

[2] The formula $\log \binom{n}{k}$ can be approximated by: $\log \binom{n}{k} \approx \int_{n-k}^{n} logx\ dx - \int_{1}^{k} logx\ dx$, which is easy to compute. Already for $k = 65$ the error is less than 1% and rapidly decreasing.

Bibliography

[1] Adriaans, P., Vitányi, P.: The Power and Perils of MDL, IEEE Trans. Inform. Th. (submitted)
[2] Adriaans, P.W.: The philosophy of learning, Handbook of the philosophy of information. In: Adriaans, P.W., van Benthem, J. (eds.) Handbook of the philosophy of science, Series edited by Gabbay, D. M., Thagard, P., Woods, J. (to appear)
[3] Adriaans, P.W.: Learning Deterministic DEC Grammars Is Learning Rational Numbers. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 320–326. Springer, Heidelberg (2006)
[4] Adriaans, P.W.: Using MDL for Grammar Induction, in Grammatical Inference: Algorithms and Applications. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 293–306. Springer, Heidelberg (2006)
[5] Cilibrasi, R., Vitányi, P.: Clustering by compression, IEEE Trans. Infomat. Th., Submitted. See http://arxiv.org/abs/cs.CV/0312044
[6] Cilibrasi, R., Vitányi, P.M.B.: Automatic Meaning Discovery Using Google (2004), http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0412098
[7] Domingos, P.: The Role of Occam's Razor in Knowledge Discovery. Data. Mining and Knowledge Discovery 3(4), 409–425 (1999)
[8] Barron, A., Rissanen, J., Yu, B.: The minimum description length principle in coding and modeling. IEEE Trans. Information Theory 44(6), 2743–2760 (1998)
[9] Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
[10] Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, New York (1997)
[11] Vereshchagin, N.K., Vitányi, P.M.B.: Kolmogorov's structure functions and model selection. IEEE Trans. Information Theory 50(12), 3265–3290 (2004)
[12] Grünwald, P.D., Langford, J.: Suboptimal behavior of Bayes and MDL in classification under misspecification. Machine Learning (2007)
[13] Gold, E.: Mark, Language Identification in the Limit. Information and Control 10(5), 447–474 (1967)
[14] Pitt, L., Warmuth, M.K.: The Minimum Consistent DFA Problem Cannot be Approximated within any Polynomial. Journal of the ACM 40(1), 95–142 (1993)
[15] Adriaans, P., Vervoort, M.: The EMILE 4.1 grammar induction toolbox. In: Adriaans, P., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 293–295. Springer, Heidelberg (2002)
[16] Vervoort, M.: Games, walks and Grammars, Thesis University of Amsterdam (2000)
[17] Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In: Adriaans, P., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 1–12. Springer, Heidelberg (2002)
[18] van Zaanen, M., Adriaans, P.: Alignment-Based Learning versus EMILE: A Comparison. In: Proceedings of the Belgian-Dutch Conference on Artificial Intelligence (BNAIC), pp. 315–322. Amsterdam, the Netherlands (2001)
[19] Solan, Z., Horn, D., Ruppin, E., Edelman, S.: Unsupervised learning of natural languages. PNAS 102(33), 11629–11634 (2005)
[20] Curnéjols, A., Miclet, L.: Apprentissage artificiel, concepts et algorithmes, Eyrolles (2003)

[21] Gerard Wolff, J.: Unifying Computing And Cognition, The SP Theory and its Applications, CognitionResearch.org.uk (2006)
[22] Wolff, J.G.: Computing As Compression: An Overview of the SP Theory and System. New Generation Comput. 13(2), 187–214 (1995)
[23] Wolff, J.G.: Information Compression by Multiple Alignment, Unification and Search as a Unifying Principle in Computing and Cognition. Journal of Artificial Intelligence Research 19(3), 193–230 (2003)
[24] Proceedings of the Workshop and tutorial de la Higuera, D., Oncina, J., Adriaans, P., van Zaanen, M.: Learning Context-Free Grammars. In: Lavrač, N., Gamberger, D., Todorvski, L., Blockeel, H. (eds.) ECML 2003 and PKDD 2003. LNCS, vols. 2837 and 2838. Springer, Heidelberg (2003)

Reachability Problems: An Update

Eric Allender

Department of Computer Science, Rutgers University, Piscataway, NJ 08855
allender@cs.rutgers.edu

Abstract. There has been a great deal of progress in the fifteen years that have elapsed since Wigderson published his survey on the complexity of the graph connectivity problem [Wig92]. Most significantly, Reingold solved the longstanding question of the complexity of the s-t connectivity problem in undirected graphs, showing that this is complete for logspace (L) [Rei05].

This survey talk will focus on some of the remaining open questions dealing with graph reachability problems. Particular attention will be paid to these topics:

– Reachability in planar directed graphs (and more generally, in graphs of low genus) [ADR05, BTV07].
– Reachability in different classes of grid graphs [ABC$^+$06].
– Reachability in mangroves [AL98].

The problem of finding a path from one vertex to another in a graph is the first problem that was identified as being complete for a natural subclass of P; it was shown to be complete for nondeterministic logspace (NL) by Jones [Jon75]. Restricted versions of this problem were subsequently shown to be complete for other natural complexity classes such as NC^1 and L. More than three decades have passed since the publication of Jones' work, and for most of that time, the outstanding open problem about graph reachability centered on the complexity of the reachability problem in *undirected* graphs. This problem was finally resolved by Reingold [Rei05], who showed that it is complete for L.

There are several other natural graph reachability problems whose complexity remains uncharacterized. The purpose of this lecture is to present some open questions about graph reachability problems, and to survey some recent progress toward understanding these problems.

We make use of reachability problems in order to understand familiar subclasses of NL, such as L (deterministic logspace), AC^0 (the class of problems solvable by constant-depth polynomial-size circuits of unbounded fan-in AND and OR gates), TC^0 (the class of problems solvable by constant-depth threshold circuits of polynomial size), and NC^1 (the class of problems solvable by Boolean formulae of polynomial size). Two other complexity classes turn out to play important roles in our study of reachability problems: UL and RUL. UL (unambiguous logspace) is the class of problems solvable by NL machines with the property that, on every input, they have at most one accepting computation path [AJ93]. Although this seems to be a severe limitation, there is evidence that NL $=$ UL [RA00, ARZ99]. RUL ("Reach" unambiguous logspace) was introduced in [BJLR91] by imposing a more restrictive condition

S.B. Cooper, B. Löwe, and A. Sorbi (Eds.): CiE 2007, LNCS 4497, pp. 25–27, 2007.

on UL machines; no configuration can be reached by two distinct computation paths on any input (even on rejecting computation paths). Thus on a RUL machine, the subgraph of reachable configurations always forms a directed tree rooted at the start configuration.

$$\mathrm{AC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{RUL} \subseteq \mathrm{UL} \subseteq \mathrm{NL}.$$

All of these classes are known to be closed under complement, except UL. All of these classes except UL also contain sets that are complete under AC^0 reductions; this is trivial except for the case of RUL [Lan97]. UL does contain a set that is complete under *nonuniform* AC^0 reductions [RA00].

Planarity is one of the most important and most frequently studied graph-theoretic restrictions, but only very recently has there started to be any evidence that the planar case might be easier than the unrestricted reachability problem. Reachability in planar digraphs is now known to be solvable in UL [BTV07]. Planar reachability is logspace-equivalent to the restricted problem of reachability in *grid graphs*, as well as to the more general problem of determining reachability for graphs embedded on a torus (i.e., genus 1 graphs) [ADR05]. Interestingly, nothing is known about graphs of genus 2; it is possible that computing reachability for genus 2 graphs is hard for NL.

No deterministic algorithm for planar reachability has been found that uses less than $\log^2 n$ space (although a logspace algorithm was presented in [ABC$^+$06] for the special case of planar acyclic digraphs having a single source). One class of digraphs where a better deterministic algorithm has been found is the class of *mangroves*. A graph is a mangrove if, for every vertex v, both the subgraph of vertices *reachable from* v and the subgraph of vertices that *reach* v are trees. (Equivalently, for every pair of vertices (u, v), there is at most one path from u to v.) A deterministic algorithm for reachability on mangroves was presented in [AL98] that uses space $\log^2 n/\log\log n$, and the same paper builds on this to show RUL $\subseteq$ DSPACE$(\log^2 n/\log\log n)$.

Because grid graph reachability is logspace-equivalent to planar reachability, it suffices to concentrate on grid graphs in trying to find a better algorithm for planar reachability. It is interesting to note, however, that whereas planar reachability is hard for L under AC^0 reductions, this is not known to hold for grid graph reachability. A detailed study of grid graph reachability was undertaken in [ABC$^+$06]. There, it was shown that many restricted versions of grid graph reachability (such as the undirected case, the outdegree one case, and the case where both indegree and outdegree are exactly one) are equivalent under AC^0 reductions, thus giving rise to a natural cluster of problems intermediate between L and NC^1 (i.e., known to be hard for NC^1 and lying in L but not known to be hard for L). A very restricted grid graph reachability problem was also shown to be complete for TC^0 under AC^0-Turing reductions.

Acknowledgments

The research of the author is supported in part by NSF Grant CCF-0514155.

References

[ABC$^+$06] Allender, E., Mix Barrington, D., Chakraborty, T., Datta, S., Roy, S.: Grid graph reachability problems. In: IEEE Conference on Computational Complexity, pp. 299–313 (2006)

[ADR05] Allender, E., Datta, S., Roy, S.: The directed planar reachability problem. In: Proc. 25th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS). LNCS, vol. 1373, pp. 238–249. Springer, Heidelberg (2005)

[AJ93] Àlvarez, C., Jenner, B.: A very hard log-space counting class. Theoretical Computer Science 107, 3–30 (1993)

[AL98] Allender, E., Lange, K.-J.: RUSPACE($\log n$) is contained in DSPACE($\log^2 n / \log\log n$). Theory of Computing Systems 31, 539–550 (1998)

[ARZ99] Allender, E., Reinhardt, K., Zhou, S.: Isolation, matching, and counting: Uniform and nonuniform upper bounds. Journal of Computer and System Sciences 59(2), 164–181 (1999)

[BJLR91] Buntrock, G., Jenner, B., Lange, K.-J., Rossmanith, P.: Unambiguity and fewness for logarithmic space. In: Budach, L. (ed.) FCT 1991. LNCS, vol. 529, pp. 168–179. Springer, Heidelberg (1991)

[BTV07] Bourke, C., Tewari, R., Vinodchandran, N.V.: Directed planar reachability is in unambiguous logspace. In: IEEE Conference on Computational Complexity (to appear 2007)

[Jon75] Jones, N.D.: Space bounded reducibility among combinatorial problems. Journal of Computer and System Sciences 11, 68–85 (1975)

[Lan97] Lange, K.-J.: An unambiguous class possessing a complete set. In: 14th International Symposium on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 1200, pp. 339–350. Springer, Heidelberg (1997)

[RA00] Reinhardt, K., Allender, E.: Making nondeterminism unambiguous. SIAM Journal of Computing 29, 1118–1131 (2000)

[Rei05] Reingold, O.: Undirected st-connectivity in log-space. In: Proceedings 37th Symposium on Foundations of Computer Science, pp. 376–385. IEEE Computer Society Press, Washington, DC, USA (2005)

[Wig92] Wigderson, A.: The complexity of graph connectivity. In: Havel, I.M., Koubek, V. (eds.) Symposium on Mathematical Foundations of Computer Science. LNCS, vol. 629, pp. 112–132. Springer, Heidelberg (1992)

RZ: A Tool for Bringing Constructive and Computable Mathematics Closer to Programming Practice

Andrej Bauer[1] and Christopher A. Stone[2]

[1] Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Andrej.Bauer@fmf.uni-lj.si
[2] Computer Science Department, Harvey Mudd College, USA
stone@cs.hmc.edu

Abstract. Realizability theory can produce interfaces for the data structure corresponding to a mathematical theory. Our tool, called RZ, serves as a bridge between constructive mathematics and programming by translating specifications in constructive logic into annotated interface code in Objective Caml. The system supports a rich input language allowing descriptions of complex mathematical structures. RZ does not extract code from proofs, but allows any implementation method, from handwritten code to code extracted from proofs by other tools.

1 Introduction

Given a description of a mathematical structure (constants, functions, relations, and axioms), what should a computer implementation look like?

For simple cases, like groups, the answer is obvious. But for more interesting structures, especially those arising in mathematical analysis, the answer is less clear. How do we implement the real numbers (a Cauchy-complete Archimedean ordered field)? Or choose the operations for a compact metric space or a space of smooth functions? Significant research goes into finding satisfactory representations [1,2,3,4], and implementations of exact real arithmetic [5,6] show that the theory can be put into practice quite successfully.

Realizability theory can be used to produce a description of the data structure (a code interface) directly corresponding to a mathematical specification. But few programmers — even those with strong backgrounds in mathematics and classical logic — are familiar with constructive logic or realizability.

We have therefore implemented a system, called RZ, to serve as a bridge between the logical world and the programming world.[1] RZ translates specifications in constructive logic into standard interface code in a programming language (currently Objective Caml [7], but other languages could be used).

The constructive part of the original specification turns into interface code, listing types and values to be implemented. The rest becomes assertions about

[1] RZ is publicly available for download at http://math.andrej.com/rz/, together with an extended version of this paper.

these types and values. The assertions have no computational content, so their constructive and classical meanings agree, and they can be understood by programmers and mathematicians accustomed to classical logic.

RZ was designed as a lightweight system supporting a rich input language. Although transforming complete proofs into complete code is possible [8], we have not implemented this. Other good systems, including Coq [9] and Minlog [10], can extract programs from proofs. But they work best managing the entire task, from specification to code generation. In contrast, interfaces generated by RZ can be implemented in any fashion as long as the assertions are satisfied. Code can be written by hand, using imperative, concurrent, and other language features rather than a "purely functional" subset. Or, the output can serve as a basis for theorem-proving and code extraction using another system.

An earlier description of RZ work appears in [11]; since then, the input syntax and underlying implementation has been significantly revised and improved, and the support for dependent types and hoisting is completely new.

2 Typed Realizability

RZ is based on *typed realizability* by John Longley [12]. This variant of realizability corresponds most directly to programmers' intuition about implementations.

We approach typed realizability and its relationship to real-world programming by way of example. Suppose we are asked to design a data structure for the set $\mathcal{G}$ of all finite simple directed graphs with vertices labeled by distinct integers. A common representation is a pair of lists (ℓ_V, ℓ_A), where ℓ_V is the list of vertex labels and ℓ_A is the *adjacency list* representing the arrows by pairing the labels of each source and target. Thus we define the datatype of graphs as[2]

$$\texttt{type graph} = \texttt{int list} \quad * \quad (\texttt{int} * \texttt{int})\ \texttt{list}$$

However, this is not a complete description of the representation, as there would be representation invariants and conditions not expressed by the type, e.g., the order in which vertices and arrows are listed is not important, each vertex and arrow must be listed exactly once, and the source and target of each arrow must appear in the list of vertices.

Thus, to implement the mathematical set $\mathcal{G}$, we must not only decide on the underlying datatype `graph`, but also determine what values of that type represent which elements of $\mathcal{G}$. As we shall see next, this can be expressed either using a *realizability relation* or a *partial equivalence relation (per)*.

2.1 Modest Sets and Pers

We now define typed realizability as it applies to OCaml. Other general-purpose programming languages could be used instead.

[2] We use OCaml notation in which t `list` classifies finite lists of elements of type t, and $t_1 * t_2$ classifies pairs containing a value of type t_1 and a value of type t_2.

Let Type be the collection of all (non-parametric) OCaml types. To each type $t \in \mathsf{Type}$ we assign the set $[\![t]\!]$ of values of type t that behave *functionally* in the sense of Longley [13]. Such values are represented by terminating expressions that do not throw exceptions or return different results on different invocations. They may *use* exceptions, store, and other computational effects, provided they appear functional from the outside; a useful example using computational effects is presented in Section 7.4. A functional value of function type may diverge as soon as it is applied. The collection Type with the assignment of functional values $[\![t]\!]$ to each $t \in \mathsf{Type}$ forms a *typed partial combinatory algebra (TPCA).*

Going back to our example, we see that an implementation of directed graphs $\mathcal{G}$ specifies a datatype $|\mathcal{G}| = \mathtt{graph}$ together with a *realizability relation* $\Vdash_{\mathcal{G}}$ between $\mathcal{G}$ and $[\![\mathtt{graph}]\!]$. The meaning of $(\ell_V, \ell_A) \Vdash_{\mathcal{G}} G$ is "OCaml value (ℓ_V, ℓ_A) represents/realizes/implements graph G". Generalizing from this, we define a *modest set* to be a triple $A = (\langle A\rangle, |A|, \Vdash_A)$ where $\langle A\rangle$ is the *underlying set*, $|A| \in \mathsf{Type}$ is the *underlying type*, and $\Vdash_A$ is a *realizability relation* between $[\![|A|]\!]$ and $\langle A\rangle$, satisfying (1) *totality:* for every $x \in \langle A\rangle$ there is $v \in [\![|A|]\!]$ such that $v \Vdash_A x$, and (2) *modesty:* if $u \Vdash_A x$ and $u \Vdash_A y$ then $x = y$. The *support* of A is the set $\|A\| = \{v \in [\![|A|]\!] \mid \exists\, x \in \langle A\rangle \,.\, v \Vdash_A x\}$ of those values that realize something. We define the relation $\approx_A$ on $[\![|A|]\!]$ by

$$u \approx_A v \iff \exists\, x \in \langle A\rangle \,.\, (u \Vdash_A x \wedge v \Vdash_A x) \ .$$

From totality and modesty of $\Vdash_A$ it follows that $\approx_A$ is a per, i.e., symmetric and transitive. Observe that $\|A\| = \{v \in [\![|A|]\!] \mid v \approx_A v\}$, whence $\approx_A$ restricted to $\|A\|$ is an equivalence relation. In fact, we may recover a modest set up to isomorphism from $|A|$ and $\approx_A$ by taking $\langle A\rangle$ to be the set of equivalence classes of $\approx_A$, and $v \Vdash_A x$ to mean $v \in x$.

The two views of implementations, as modest sets $(\langle A\rangle, |A|, \Vdash_A)$, and as pers $(|A|, \approx_A)$, are equivalent.[3] We concentrate on the view of modest sets as pers. They are more convenient to use in RZ because they refer only to types and values, as opposed to arbitrary sets. Nevertheless, it is useful to understand how modest sets and pers arise from natural programming practice.

Pers form a category whose objects are pairs $A = (|A|, \approx_A)$ where $|A| \in \mathsf{Type}$ and $\approx_A$ is a per on $[\![|A|]\!]$. A morphism $A \to B$ is represented by a function $v \in [\![|A| \to |B|]\!]$ such that, for all $u, u' \in \|A\|$, $u \approx_A u' \implies v\ u \approx_B v\ u'$. Two such functions v and v' represent the same morphism if, for all $u, u' \in \|A\|$, $u \approx_A u'$ implies $v\ u \approx_B v'\ u'$.

The category of pers has a very rich structure, namely that of a regular locally cartesian closed category [14]. This suffices for the interpretation of first-order logic and (extensional) dependent types [15].

Not all pers are *decidable*, i.e., there may be no algorithm for deciding when two values are equivalent. Examples include implementations of semigroups with an undecidable word problem [16] and implementations of computable real numbers (which might be realized by infinite Cauchy sequences).

[3] And there is a third view, as a partial surjection $\delta_A : \subseteq [\![|A|]\!] \twoheadrightarrow \langle A\rangle$, with $\delta_A(v) = x$ when $v \Vdash_A x$. This is how realizability is presented in Type Two Effectivity [1].

Underlying types of realizers:

$$\begin{array}{llll}
|\top| & = \mathtt{unit} & |\bot| & = \mathtt{unit} \\
|x = y| & = \mathtt{unit} & |\phi \wedge \psi| & = |\phi| \times |\psi| \\
|\phi \Rightarrow \psi| & = |\phi| \to |\psi| & |\phi \vee \psi| & = \text{`}\mathtt{or}_0\ \mathtt{of}\ |\phi_0| + \text{`}\mathtt{or}_1\ \mathtt{of}\ |\phi_1| \\
|\forall x{:}A.\ \phi| & = |A| \to |\phi| & |\exists x{:}A.\ \phi| & = |A| \times |\phi|
\end{array}$$

Realizers:

$$\begin{array}{lll}
() \Vdash \top & & \\
() \Vdash x = y & \text{iff} & x = y \\
(t_1, t_2) \Vdash \phi \wedge \psi & \text{iff} & t_1 \Vdash \phi \text{ and } t_2 \Vdash \psi \\
t \Vdash \phi \Rightarrow \psi & \text{iff} & \text{for all } u \in |\phi|, \text{ if } u \Vdash \phi \text{ then } t\,u \Vdash \psi \\
\text{`}\mathtt{or}_0\, t \Vdash \phi \vee \psi & \text{iff} & t \Vdash \phi \\
\text{`}\mathtt{or}_1\, t \Vdash \phi \vee \psi & \text{iff} & t \Vdash \psi \\
t \Vdash \forall x{:}A.\ \phi(x) & \text{iff} & \text{for all } u \in |A|, \text{ if } u \Vdash_A x \text{ then } t\,u \Vdash \phi(x) \\
(t_1, t_2) \Vdash \exists x{:}A.\ \phi(x) & \text{iff} & t_1 \Vdash_A x \text{ and } t_2 \Vdash \phi(x)
\end{array}$$

Fig. 1. Realizability interpretation of logic (outline)

2.2 Interpretation of Logic

In the realizability interpretation of logic, each formula ϕ is assigned a set of *realizers*, which can be thought of as computations that witness the validity of ϕ. The situation is somewhat similar, but not equivalent, to the propositions-as-types translation of logic into type theory, where proofs of a proposition correspond to terms of the corresponding type. More precisely, to each formula ϕ we assign an underlying type $|\phi|$ of realizers, but unlike the propositions-as-types translation, not all terms of type $|\phi|$ are necessarily valid realizers for ϕ, and some terms that are realizers may not correspond to any proofs, for example, if they denote partial functions or use computational effects.

It is customary to write $t \Vdash \phi$ when $t \in [\![|\phi|]\!]$ is a realizer for ϕ. The underlying types and the realizability relation $\Vdash$ are defined inductively on the structure of ϕ; an outline is shown in Figure 1. We say that a formula ϕ is *valid* if it has at least one realizer.

In classical mathematics, a predicate on a set X may be viewed as a subset of X or a (possibly non-computable) function $X \to \mathtt{bool}$, where $\mathtt{bool} = \{\bot, \top\}$ is the set of truth values. Accordingly, since in realizability propositions are witnessed by realizers, a predicate ϕ on a per $A = (|A|, \approx_A)$ is a (possibly non-computable) function $\phi : [\![|A|]\!] \times [\![|\phi|]\!] \to \mathtt{bool}$ that is *strict* (if $\phi(u, v)$ then $u \in \|A\|$) and *extensional* (if $\phi(u_1, v)$ and $u_1 \approx_A u_2$ then $\phi(u_2, v)$).

Suppose we have implemented the real numbers $\mathbb{R}$ as a per $R = (\mathtt{real}, \approx_R)$, and consider $\forall a{:}R.\ \forall b{:}R.\ \exists x{:}R.\ x^3 + ax + b = 0$. By computing according to Figure 1, we see that a realizer for this proposition is a value r of type $\mathtt{real} \to \mathtt{real} \to \mathtt{real} \times \mathtt{unit}$ such that, if t realizes $a \in \mathbb{R}$ and u realizes $b \in \mathbb{R}$, then $r\,t\,u = (v, w)$ with v realizing a real number x such that $x^3 + ax + b = 0$, and w is trivial. (This can be "thinned" to a realizer of type $\mathtt{real} \to \mathtt{real} \to \mathtt{real}$ that does not bother to compute w.) In essence, the realizer r computes a root of the cubic equation. Note that r is *not* extensional, i.e., different realizers t and u for the same a and b may result in different roots. To put this in another way, r realizes a *multi-valued* function[4] rather than a per morphism. It is well known in computable mathematics that certain operations, such as equation solving, are only computable if we allow them to be multi-valued. They arise naturally in RZ as translations of $\forall\exists$ statements.

Some propositions, such as equality and negation, have "irrelevant" realizers free of computational content. Sometimes only a part of a realizer is computationally irrelevant. Propositions that are free of computational content are characterized as the $\neg\neg$-*stable propositions*. A proposition ϕ is said to be $\neg\neg$-stable, or just *stable* for short, when $\neg\neg\phi \Rightarrow \phi$ is valid. On input, one can specify whether abstract predicates have computational content. On output, extracted realizers go through a *thinning* phase, which removes irrelevant realizers.

Many structures are naturally viewed as families of sets, or sets depending on parameters, or *dependent types* as they are called in type theory. For example, the n-dimensional Euclidean space $\mathbb{R}^n$ depends on the dimension $n \in \mathbb{N}$, the Banach space $\mathcal{C}([a, b])$ of uniformly continuous real functions on the closed interval $[a, b]$ depends on $a, b \in \mathbb{R}$ such that $a < b$, etc. In general, a family of sets $\{A_i\}_{i \in I}$ is an assignment of a set A_i to each $i \in I$ from an *index set* I.

In the category of pers the appropriate notion is that of a *uniform* family. A uniform family of pers $\{A_i\}_{i \in I}$ indexed by a per I is given by an underlying type $|A|$ and a family of pers $(\approx_{A_i})_{i \in [\![|I|]\!]}$ that is strict (if $u \approx_{A_i} v$ then $i \in \|I\|$) and extensional (if $u \approx_{A_i} v$ and $i \approx_I j$ then $u \approx_{A_j} v$).

We can also form the *sum* $\Sigma_{i \in I} A_i$ or *product* $\Pi_{i \in I} A_i$ of a uniform family, allowing an interpretation of (extensional) dependent type theory.

3 Specifications as Signatures with Assertions

In programming we distinguish between *implementation* and *specification* of a structure. In OCaml these two notions are expressed with modules and module types, respectively. A module defines types and values, while a module type simply lists the types, type definitions, and values provided by a module. For a complete specification, a module type must also be annotated with *assertions* which specify the required properties of declared types and values.

The output of RZ consists of *module specifications*, module types plus assertions about their components. More specifically, a typical specification may contain value declarations, type declarations and definitions, module declarations, specification definitions, proposition declarations, and assertions. RZ only outputs assertions that are free of computational content, and do not require knowledge of constructive mathematics to be understood.

A special construct is the *obligation* `assure` $x{:}\tau,\, p$ `in` e which means "in term e, let x be any element of $[\![\tau]\!]$ that satisfies p". An obligation is equivalent to a combination of Hilbert's indefinite description operator and a local definition,

[4] The multi-valued nature of the realizer comes from the fact that it computes *any one* of many values, not that it computes *all* of the many values.

`let` $x{=}(\varepsilon x{:}\tau.\,p)$ `in` e, where $\varepsilon x{:}\tau.\,p$ means "any $x \in [\![\tau]\!]$ such that p". The alternative form `assure` p `in` e stands for `assure _:unit`, p `in` e.

Obligations arise from the fact that well-formedness of the input language is undecidable; see Section 4. In such cases the system computes a realizability translation, but also produces obligations. The programmer must replace each obligation with a value satisfying the obligation. If such values do not exist, the specification is unimplementable.

4 The Input Language

The input to RZ consists of one or more theories. A RZ *theory* is a generalized logical signature with associated axioms, similar to a Coq module signature. Theories describe *models*, or implementations.

The simplest theory Θ is a list of *theory element*s `thy` $\theta_1 \ldots \theta_n$ `end`. A theory element may specify that a certain set, set element, proposition or predicate, or model must exist (using the Parameter keyword). It may also provide a definition of a set, term, proposition, predicate, or theory (using the Definition keyword). Finally, a theory element can be a named axiom (using the Axiom keyword).

We allow model parameters in theories; typical examples in mathematics include the theory of a vector space parameterized by a field of scalars.

A theory of a parameterized implementation $[m{:}\Theta_1]{\to}\Theta_2$ describes a uniform family of models (i.e., a single implementation; a functor in OCaml) that maps every model m satisfying Θ_1 to a model of Θ_2. In contrast, a theory $\lambda m{:}\Theta_1.\,\Theta_2$ maps models to theories; if T is such a theory, then $T(M_1)$ and $T(M_2)$ are theories whose implementations might be completely unrelated.

Propositions and predicates appearing in theories may use full first-order constructive logic, not just the negative fragment.

The language of sets is rich, going well beyond the type systems of typical programming languages. In addition to any base sets postulated in a theory, one can construct dependent cartesian products and dependent function spaces. We also supports disjoint unions (with labeled tags), quotient spaces (a set modulo a stable equivalence relation), subsets (elements of a set satisfying a predicate). RZ even permits explicit references to sets of realizers.

The term language includes introduction and elimination constructs for the set level. For product sets we have tuples and projections ($\pi_1\,e$, $\pi_2\,e$, $\ldots$), and for function spaces we have lambda abstractions and application. One can inject a term into a tagged union, or do case analyses on the members of a union. We can produce an equivalence class or pick a representative from a equivalence class (as long as what we do with it does not depend on the choice of representative). We can produce a set of realizers or choose a representative from a given set of realizers (as long as what we do with it does not depend on the choice of representative). We can inject a term into a subset (if it satisfies the appropriate predicate), or project an element out of a subset. Finally, the term language also allows local definitions of term variables, and definite descriptions (as long as there is a unique element satisfying the predicate in question).

From the previous paragraph, it is clear that checking the well-formedness of terms is not decidable. RZ checks what it can, but does not attempt serious theorem proving. Uncheckable constraints remain as obligations in the final output, and should be verified by other means before the output can be used.

5 Translation

5.1 Translation of Sets and Terms

A set declaration `Parameter` s : `Set` is translated to

```
type s
predicate (≈_s) : s → s → bool
assertion symmetric_s :  ∀ x:s, y:s, x ≈_s y → y ≈_s x
assertion transitive_s : ∀ x:s, y:s, z:s, x ≈_s y ∧ y ≈_s z → x ≈_s z
predicate ||s|| : s → bool
assertion support_def_s :  ∀ x:s,  x : ||s|| ↔ x ≈_s x
```

This says that the programmer should define a type s and a per $\approx_s$ on $[\![s]\!]$. Here $\approx_s$ is *not* an OCaml value of type $s \to s \to$ `bool`, but an abstract relation on the set $[\![s]\!] \times [\![s]\!]$. The relation may be uncomputable.

The translation of the declaration of a dependent set `Parameter t : s` $\to$ `Set` uses uniform families (Section 2.2). The underlying type `t` is non-dependent, but the per $\approx_{\mathtt{t}}$ receives an additional parameter `x` : $[\![\mathtt{s}]\!]$.

A value declaration `Parameter` $x : s$ is translated to

```
val x : s
assertion x_support : x : ||s||
```

which requires the definition of a value `x` of type `s` which is in the support of `s`.

A value definition `Definition x` $:= e$ where e is an expression denoting an element of `s` is translated to

```
val x : s
assertion x_def : x ≈_s e
```

The assertion does *not* force `x` to be defined as e, only to be equivalent to it with respect to $\approx_{\mathtt{s}}$. This is useful, as often the easiest way to define a value is not the most efficient way to compute it.

Constructions of sets in the input language are translated to corresponding constructions of modest sets. We comment on those that are least familiar.

Subsets. Given a predicate ϕ on a per A, the sub-per $\{x : A \mid \phi\}$ has underlying type $|A| \times |\phi|$ where $(u_1, v_1) \approx_{\{x:A|\phi\}} (u_2, v_2)$ when $u_1 \approx_A u_2$, $v_1 \Vdash \phi(u_1)$ and $v_2 \Vdash \phi(u_2)$. The point is that a realizer for an element of $\{x : A \mid \phi\}$ carries information about *why* the element belongs to the subset.

A type coercion $e : t$ can convert an element of the subset $s = \{x : t \mid \phi(x)\}$ to an element of t. At the level of realizers this is achieved by the first projection, which keeps a realizer for the element but forgets the one for $\phi(e)$. The opposite type coercion $e' : s$ takes an $e' \in t$ and converts it to an element of the subset. This is only well-formed when $\phi(e')$ is valid. Then, if $u \Vdash_t e'$ and $v \Vdash \phi(e')$, a realizer for $e' : s$ is (u, v). However, since RZ cannot in general know a v which validates $\phi(e')$, it emits the pair $(u, (\texttt{assure}\, v{:}|\phi|,\ \phi\, u\, v\, \texttt{in}\, v))$.

Quotients. Even though we may form quotients of pers by arbitrary equivalence relations, only quotients by $\neg\neg$-stable relations behave as expected.[5] A stable equivalence relation on a per A is the same thing as a partial equivalence relation ρ on $|A|$ which satisfies $\rho(x, y) \implies x \approx_A y$. Then the quotient A/ρ is the per with $|A/\rho| = |A|$ and $x \approx_{A/\rho} y \iff \rho(x, y)$.

Luckily, it seems that many equivalence relations occurring in computable mathematics are stable, or can be made stable. For example, the coincidence relation on Cauchy sequences is expressed by a $\forall\exists\forall$ formula, but if we consider *rapid* Cauchy sequences (those sequences a satisfying $\forall\, i \in \mathbb{N}\, .\, |a_{i+1} - a_i| \leq 2^{-i}$), it becomes a (negative) $\forall$ formula. It is interesting that most practical implementations of real numbers follow this line of reasoning and represent real numbers in a way that avoids annotating every sequence with its rate of convergence.

Translation of an equivalence class $[e]_\rho$ is quite simple, since a realizer for e also realizes its equivalence class $[e]_\rho$. The elimination term $\texttt{let}\ [x]_\rho = \xi\ \texttt{in}\ e$, means "let x be any element of ρ-equivalence class ξ in e". It is only well-formed when e does not depend on the choice of x, but this is something RZ cannot check. Therefore, if u realizes ξ, RZ uses u as a realizer for x and emits an obligation saying that the choice of a realizer for x does not affect e.

The underlying set of realizers. Another construction on a per A is the underlying per of realizers $\texttt{rz}\ A$, defined by $|\texttt{rz}\ A| = |A|$ and $u \approx_{\texttt{rz}\ A} vu \in \|A\| \wedge \iff u = v$, where by $u = v$ we mean observational equality of values u and v. An element $r \in \texttt{rz}\ A$ realizes a unique element $\texttt{rz}\ r \in A$. The elimination term $\texttt{let rz}\ x = e_1\ \texttt{in}\ e_2$, which means "let x be any realizer for e_1 in e_2", is only well-formed if e_2 does not depend on the choice of x. This is an uncheckable condition, hence RZ emits a suitable obligation in the output, and uses for x the same realizer as for e_1.

The construction $\texttt{rz}\ A$ validates the Presentation Axiom (see Section 7.3). In the input language it gives us access to realizers, which is useful because many constructions in computable mathematics, such as those in Type Two Effectivity [1], are explicitly expressed in terms of realizers.

5.2 Translation of Propositions

The driving force behind the translation of logic is a theorem [17, 4.4.10] that says that under the realizability interpretation every formula ϕ is equivalent to one that says, informally speaking, "there exists $u \in |\phi|$, such that u realizes ϕ".

[5] The trouble is that from equality of equivalence classes $[x]_\rho = [y]_\rho$ we may conclude only $\neg\neg\rho(x, y)$ rather than the expected $\rho(x, y)$.

Furthermore, the formula "u realizes ϕ" is computationally trivial. The translation of a predicate ϕ then consists of its underlying type $|\phi|$ and the relation $u \Vdash \phi$, expressed as a negative formula.

Thus an axiom `Axiom A` : ϕ in the input is translated to

```
val u : |φ|
assertion A : u ⊩ φ
```

which requires the programmer to validate ϕ by providing a realizer for it. When ϕ is a compound statement RZ computes the meaning as described in Figure 1.

In RZ we avoid the explicit realizer notation $u \Vdash \phi$ in order to make the output easier to read. A basic predicate declaration `Parameter` $p : s \rightarrow$`Prop` is translated to a type declaration `type ty_p` and a predicate declaration `predicate` $p : s \rightarrow$ `ty_p` $\rightarrow$ `bool` together with assertions that p is strict and extensional.

6 Implementation

The RZ implementation consists of several sequential passes.

After the initial parsing, a *type reconstruction* phase checks that the input is well-typed, and if successful produces an annotated result with all variables explicitly tagged with types. The type checking phase uses a system of dependent types, with limited subtyping (implicit coercions) for sum types and subset types.

Next the realizability translation is performed as described in Section 5, producing interface code. The flexibility of the full input language (e.g., n-ary sum types and dependent product types) makes the translation code fairly involved, and so it is performed in a "naive" fashion whenever possible. The immediate result of the translation is not easily readable.

Thus, up to four more passes simplify the output before it is displayed to the user. A *thinning* pass removes all references to trivial realizers produced by stable formulas. An *optimization* pass applies an ad-hoc collection of basic logical and term simplifications in order to make the output more readable. Some redundancy may remain, but in practice the optimization pass helps significantly.

Finally, the user can specify two optional steps occur. RZ can perform a *phase-splitting* pass [18]. This is an experimental implementation of an transformation that can replace a functor (a relatively heavyweight language construct) by parameterized types and/or polymorphic values.

The other optional transformation is a *hoisting* pass which moves obligations in the output to top-level positions. Obligations appear in the output inside assertions, at the point where an uncheckable property was needed. Moving these obligations to the top-level make it easier to see exactly what one is obliged to verify, and can sometimes make them easier to read, at the cost of losing information about why the obligation was required at all.

7 Examples

In this section we look at several examples which demonstrate various points of RZ. Unfortunately, serious examples from computable mathematics take too much space[6] and will have to be presented separately. The main theme is that constructively reasonable axioms yield computationally reasonable operations.

7.1 Decidable Sets

A set S is said to be decidable when, for all $x, y \in S$, $x = y$ or $\neg(x = y)$. In classical mathematics all sets are decidable, but RZ requires an axiom

```
Parameter s : Set.
Axiom eq: ∀ x y : s, x = y ∨ ¬ (x = y).
```

to produce a realizer for equality

```
val eq : s → s → [‘or0 | ‘or1]
assertion eq : ∀ (x:‖s‖, y:‖s‖), (match eq x y with
                                        ‘or0 ⇒ x ≈s y
                                      | ‘or1 ⇒ ¬ (x ≈s y) )
```

We read this as follows: `eq` is a function which takes arguments `x` and `y` of type `s` and returns ‘`or0` or ‘`or1`. If it returns ‘`or0`, then $\mathtt{x} \approx_{\mathtt{s}} \mathtt{y}$, and if it returns ‘`or1`, then $\neg(\mathtt{x} \approx_{\mathtt{s}} \mathtt{y})$. In other words `eq` is a decision procedure.

7.2 Inductive Types

To demonstrate the use of dependent types we show how RZ handles general inductive types, also known as W-types or general trees [19]. Recall that a W-type is a set of well-founded trees, where the branching types of trees are described by a family of sets $B = \{T(x)\}_{x \in S}$. Each node in a tree has a *branching type* $x \in S$, which determines that the successors of the node are labeled by the elements of $T(x)$. Figure 2 shows an RZ axiomatization of W-types. The theory `Branching` describes that a branching type consists of a set `s` and a set `t` depending on `s`. The theory `W` is parameterized by a branching type `B`. It specifies a set `w` of well-founded trees and a tree-forming operation `tree` with a dependent type $\Pi_{\mathtt{x} \in \mathtt{B.s}}(\mathtt{B.t}(\mathtt{x}) \to \mathtt{w}) \to \mathtt{w}$. The inductive nature of `w` is expressed with the axiom `induction`, which states that for every property `M.p`, if `M.p` is an inductive property then every tree satisfies it. A property is said to be *inductive* if a tree `tree x f` satisfies it whenever all its successors satisfy it.

In the translation dependencies at the level of types and terms disappear. A branching type is determined by a pair of non-dependent types `s` and `t` but the per $\approx_{\mathtt{t}}$ depends on $[\![\mathtt{s}]\!]$. The theory `W` turns into a signature for a functor receiving a branching type `B` and returning a type `w`, and an operation `tree` of type $\mathtt{B.s} \to (\mathtt{B.t} \to \mathtt{w}) \to \mathtt{w}$. One can use phase-splitting to translate axiom `induction` into a specification of a polymorphic function

$$\mathtt{induction} : (\mathtt{B.s} \to (\mathtt{B.t} \to \mathtt{w}) \to (\mathtt{B.t} \to \alpha) \to \alpha) \to \mathtt{w} \to \alpha,$$

[6] The most basic structure in analysis (the real numbers) alone requires several operations and a dozen or more axioms.

```
Definition Branching :=
thy
  Parameter s : Set.      (* branching types *)
  Parameter t : s -> Set. (* branch labels *)
end.

Parameter W : [B : Branching] →
thy
  Parameter w : Set.
  Parameter tree : [x : B.s] → (B.t x → w) → w.
  Axiom induction:
    ∀ M : thy Parameter p : w → Prop. end,
    (∀ x : B.s, ∀ f : B.t x → w,
       ((∀ y : B.t x, M.p (f y)) → M.p (tree x f))) →
    ∀ t : w, M.p t.
end.
```

Fig. 2. General inductive types

which is a form of recursion on well-founded trees. Instead of explaining `induction`, we show a surprisingly simple, hand-written implementation of W-types in OCaml. The reader may enjoy figuring out how it works:

```
module W (B : Branching) = struct
  type w = Tree of B.s * (B.t -> w)
  let tree x y = Tree (x, y)
  let rec induction f (Tree (x, g)) =
    f x g (fun y -> induction f (g y))
end
```

7.3 Axiom of Choice

RZ can help explain why a generally accepted axiom is not constructively valid. Consider the Axiom of Choice:

```
Parameter a b : Set.
Parameter r : a → b → Prop.
Axiom ac: (∀ x : a, ∃ y : b, r x y) →
             (∃ c : a → b, ∀ x : a, r x (c x)).
```

The relevant part of the output is

```
val ac : (a → b * ty_r) → (a → b) * (a → ty_r)
assertion ac :
  ∀ f:a → b * ty_r,
    (∀ (x:||a||),  let (p,q) = f x in p : ||b|| ∧ r x p q) →
    let (g,h) = ac f in
      g : ||a → b|| ∧ (∀ (x:||a||),  r x (g x) (h x))
```

This requires a function `ac` which accepts a function `f` and computes a pair of functions (`g`, `h`). The input function `f` takes an `x`:||`a`|| and returns a pair (`p`, `q`) such that `q` realizes the fact that `r x p` holds. The output functions `g` and `h` taking `x:‖a‖` as input must be such that `h x` realizes `r x (g x)`. Crucially, the requirement `g:‖a → b‖` says that `g` must be extensional, i.e., map equivalent realizers to equivalent realizers. We could define `h` as the first component of `f`, but we cannot hope to implement `g` in general because the second component of `f` is not assumed to be extensional.

The *Intensional* Axiom of Choice allows the choice function to depend on the realizers:

```
Axiom iac: (∀ x : a, ∃ y : b, r x y) →
           (∃ c : rz a → b, ∀ x : rz a, r (rz x) (c x)).
```

Now the output is

```
val iac : (a → b * ty_r) → (a → b) * (a → ty_r)
assertion iac :
  ∀ f:a → b * ty_r,
    (∀ (x:‖a‖),  let (p,q) = f x in p : ‖b‖ ∧ r x p q) →
    let (g,h) = iac f in
      (∀ x:a, x : ‖a‖ → g x : ‖b‖) ∧ (∀ (x:‖a‖),  r x (g x) (h x))
```

This is exactly the same as before *except* the troublesome requirement was weakened to ∀x:a. (x:‖a‖ → g x:‖b‖). We can implement `iac` in OCaml as

```
let iac f = (fun x -> fst (f x)), (fun x -> snd (f x))
```

The Intensional Axiom of Choice is in fact just an instance of the usual Axiom of Choice applied to `rz` A and B. Combined with the fact that `rz` A covers A, this establishes the validity of *Presentation Axiom* [20], which states that every set is an image of one satisfying the axiom of choice.

7.4 Modulus of Continuity

As a last example we show how certain constructive principles require the use of computational effects. To keep the example short, we presume that we are already given the set of natural numbers `nat` with the usual structure.

A *type 2 functional* is a map f : (`nat` → `nat`) → `nat`. It is said to be continuous if the output of $f(a)$ depends only on an initial segment of the sequence a. We can express the (non-classical) axiom that all type 2 functionals are continuous in RZ as follows:

```
Axiom continuity: ∀ f : (nat → nat) → nat, ∀ a : nat → nat,
  ∃ k, ∀ b : nat → nat, (∀ m, m ≤ k → a m = b m) → f a = f b.
```

The axiom says that for any `f` and `a` there exists `k` ∈ `nat` such that `f(b)` = `f(a)` when sequences `a` and `b` agree on the first `k` terms. It translate to:

```
val continuity : ((nat → nat) → nat) → (nat → nat) → nat
assertion continuity :
  ∀ (f:‖(nat → nat) → nat‖, a:‖nat → nat‖),
    let p = continuity f a in p : ‖nat‖ ∧
    (∀ (b:‖nat → nat‖),
       (∀ (m:‖nat‖),  m ≤ p → a m ≈nat b m) → f a ≈nat f b)
```

i.e., that `continuity f a` is a number `p` such that $\mathtt{f(a)} = \mathtt{f(b)}$ whenever `a` and `b` agree on the first `p` terms. In other words, `continuity` is a *modulus of continuity* functional. It cannot be implemented in a purely functional language,[7] but with the use of store we can implement it in OCaml as

```
let continuity f a = let p = ref 0 in
                     let a' n = (p := max !p n; a n) in
                       f a' ; !p
```

To compute a modulus for `f` at `a`, the program creates a function $\mathtt{a}'$ which is just like `a` except that it stores in `p` the largest argument at which it has been called. Then $\mathtt{f\ a}'$ is computed, its value is discarded, and the value of `p` is returned. The program works because `f` is assumed to be extensional and must therefore not distinguish between extensionally equal sequences `a` and $\mathtt{a}'$.

8 Related Work

8.1 Coq and Other Tools

Coq provides complete support for theorem-proving and creating trusted code. Often one writes code in Coq's functional language, states and proves theorems that the code behaves correctly, and has Coq extract correct code. In such cases RZ is complementary; it can suggest the appropriate division between code and theorems. We hope RZ will soon be able to produce output in Coq's input syntax.

Komagata and Schmidt [8] describe a system that uses a realizability in a way similar to RZ. Like Coq, it extracts code from proofs. An interesting implementation difference is that the algorithm they use (attributed to John Hatcliff) does thinning as it goes along, rather than making a separate pass as RZ does. Unlike RZ, their system needs full formal proofs as input; it checks the proofs, and generates executable code. RZ also handles a much richer input language (function, subset, quotient, and dependent types; quantification over theories; parameterized theories; etc.) that goes well beyond simple predicate logic over integers and lists.

The idea of annotating ML signatures with assertions is not new (e.g., [22]).

8.2 Other Models of Computability

Many formulations of computable mathematics are based on realizability models [14], even though they were not initially developed, (nor are they usually presented) within the framework of realizability: Recursive Mathematics [23] is based on the original realizability by Turing machines [24]; Type Two Effectivity [1] on function realizability [25] and relative function realizability [26], while topological and domain representations [27,28] are based on realizability over the graph model $\mathcal{P}\omega$ [29]. A common feature is that they use models of computation which are well suited for the theoretical studies of computability.

[7] There are models of λ-calculus which validate the choice principle $AC_{2,0}$, but this contradicts the existence of a modulus of continuity functional, see [21, 9.6.10].

Approaches based on simple programming languages with datatypes for real numbers [30,31] and topological algebras [2], and machines augmented with (suitably chosen subsets of) real numbers [32,33,34] are motivated by issues ranging from theoretical concerns about computability/complexity to practical questions in computational geometry. RZ attempts to improve practicality by using a real-world language, and by providing an input language rich enough for descriptions of mathematical structures going well beyond the real numbers.

Finally, we hope that RZ and, hopefully, its forthcoming applications, give plenty of evidence for the *practical* value of Constructive Mathematics [35].

References

1. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)
2. Tucker, J., Zucker, J.I.: Computable functions and semicomputable sets on many-sorted algebras. In: Abramsky, S., Gabbay, D., Maibaum, T. (eds.) Handbook of Logic in Computer Science, vol. 5, Clarendon Press, Oxford (1998)
3. Blanck, J.: Domain representability of metric spaces. Annals of Pure. and Applied Logic 83, 225–247 (1997)
4. Edalat, A., Lieutier, A.: Domain of differentiable functions. In: Blanck, J., Brattka, V., Hertling, P., Weihrauch, K. (eds.) Computability and Complexity in Analysis CCA2000 Workshop, Swansea, Wales, September 17–19, 2000 (2000)
5. Müller, N.: The iRRAM: Exact arithmetic in C++. In: Blanck, J., Brattka, V., Hertling, P., Weihrauch, K. (eds.) Computability and Complexity in Analysis pp. 319–350 CCA2000 Workshop, Swansea, Wales, September 17–19, 2000 (2000)
6. Lambov, B.: RealLib: an efficient implementation of exact real arithmetic. In: Grubba, T., Hertling, P., Tsuiki, H., Weihrauch, K. (eds.) Computability and Complexity in Analysis Proccedings, Second International Conference, CCA 2005, Kyoto, Japan, August 25–29, 2005 pp. 169–175 (2005)
7. Leroy, X., Doligez, D., Garrigue, J., Rémy, D., Vouillon, J.: The Objective Caml system, documentation and user's manual - release 3.08. Technical report, INRIA (July 2004)
8. Komagata, Y., Schmidt, D.A.: Implementation of intuitionistic type theory and realizability theory. Technical Report TR-CS-95-4, Kansas State University (1995)
9. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Springer, Heidelberg (2004)
10. Benl, H., Berger, U., Schwichtenberg, H., Seisenberger, M., Zuber, W.: Proof theory at work: Program development in the Minlog system. In: Bibel, W., Schmidt, P.H. (eds.) Automated Deduction: A Basis for Applications, Systems and Implementation Techniques, vol. II, Kluwer Academic Publishers, Dordrecht (1998)
11. Bauer, A., Stone, C.A.: Specifications via realizability. In: Proceedings of the Workshop on the Constructive Logic for Automated Software Engineering (CLASE 2005) volume 153 of Electronic Notes in Theoretical Computer Science, pp. 77–92 (2006)
12. Longley, J.: Matching typed and untyped realizability. Electr. Notes Theor. Comput. Sci. 23(1) (1999)
13. Longley, J.: When is a functional program not a functional program? In: International Conference on Functional Programming, pp. 1–7 (1999)
14. Bauer, A.: The Realizability Approach to Computable Analysis and Topology. PhD thesis, Carnegie Mellon University (2000)

15. Jacobs, B.: Categorical Logic and Type Theory. Elsevier, Amsterdam (1999)
16. Post, E.: Recursive unsolvability of a problem of Thue. The Journal of Symbolic Logic 12, 1–11 (1947)
17. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics, An Introduction. In: Studies in Logic and the Foundations of Mathematics, vol. 1(121). North-Holland, Amsterdam (1988)
18. Harper, R., Mitchell, J.C., Moggi, E.: Higher-order Modules and the Phase Distinction. In: Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL '90), pp. 341–354 (1990)
19. Nordström, B., Petersson, K., Smith, J.M.: Programming in Martin-Löf's Type Theory. Oxford University Press, Oxford (1990)
20. Barwise, J.: Admissible Sets and Structures. Springer, Heidelberg (1975)
21. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics, An Introduction. In: Studies in Logic and the Foundations of Mathematics, vol. 2(123), North-Holland, Amsterdam (1988)
22. Kahrs, S., Sannella, D., Tarlecki, A.: The definition of Extended ML: A gentle introduction. Theoretical Computer Science 173(2), 445–484 (1997)
23. Ershov, Y.L., Goncharov, S.S., Nerode, A., Remmel, J.B. (eds.): Handbook of Recursive Mathematics. Elsevier, Amsterdam (1998)
24. Kleene, S.C.: On the interpretation of intuitionistic number theory. Journal of Symbolic Logic 10, 109–124 (1945)
25. Kleene, S.C., Vesley, R.E.: The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions. North-Holland Publishing Company, Amsterdam (1965)
26. Birkedal, L.: Developing Theories of Types and Computability. PhD thesis, School of Computer Science, Carnegie Mellon University (December 1999)
27. Blanck, J.: Computability on topological spaces by effective domain representations. PhD thesis, Uppsala University, Department of Mathematics, Uppsala, Sweden (1997)
28. Bauer, A., Birkedal, L., Scott, D.S.: Equilogical spaces. Theoretical Computer Science 1(315), 35–59 (2004)
29. Scott, D.S.: Data types as lattices. SIAM Journal of Computing 5(3), 522–587 (1976)
30. Escardó, M.H.: PCF extended with real numbers. PhD thesis, Department of Computer Science, University of Edinburgh (December 1997)
31. Marcial-Romero, J.R., Escard M.H., ó.: Semantics of a sequential language for exact real-number computation. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, pp. 426–435 (July 2004)
32. Borodin, A., Monro, J.I.: The computational complexity of algebraic and numeric problems. Elsevier computer science library: Theory of computation series, vol. 1. American Elsevier, New York, London, Amsterdam (1975)
33. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, New York (1998)
34. Yap, C.K.: Theory of real computation according to EGC (2006) To appear in LNCS Volume based on the Dagstuhl Seminar Reliable Implementation of Real Number Algorithms: Theory and Practice, (Jan 8-13, 2006)
35. Bishop, E., Bridges, D.: Constructive Analysis. Grundlehren der math. Wissenschaften, vol. 279. Springer, Heidelberg (1985)

Producer/Consumer in Membrane Systems and Petri Nets

Francesco Bernardini[1], Marian Gheorghe[2], Maurice Margenstern[3], and Sergey Verlan[4]

[1] LIACS, Universiteit Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl

[2] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

[3] Université Paul Verlaine - Metz, UFR MIM, LITA, EA 3097
Ile du Saulcy, 57045 Metz Cédex, France
margens@univ-metz.fr

[4] LACL, Département Informatique, Université Paris XII
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract. The paper investigates different relationships between membrane systems and Petri nets by focusing on modelling variants of the producer/consumer paradigm. Two models of producer/consumer systems based on membrane systems are described, and it is shown how to translate these models into equivalent Petri nets with a corresponding semantics. It is then observed a direct correspondence between the Petri nets representation of the proposed models and standard solutions based on Petri nets already present in the literature.

1 Introduction

Membrane computing is an emerging branch of natural computing which deals with distributed and parallel computing devices of a bio-inspired type, which are called *membrane systems*, or *P systems* (see [8] and also [2] for a comprehensive bibliography of membrane systems). Membrane systems are computing devices which abstract from the structure and functioning of living cells (*cell-like P systems*) and are characterised by a tree-like structure. Other classes of membrane systems, called *tissue P systems*, abstract from the organisation of cells in tissues of multicellular organisms and are characterised by a graph-like structure.

Here we investigate relationships between membrane systems and Petri nets by following the work previously done in [5,7,6] where appropriate constructions are devised to translate membrane systems into Petri nets so to preserve certain structural properties. Specifically, for certain classes of membrane systems, such constructions associate to every P system an *equivalent* Petri net such that: there is a one-to-one correspondence between configurations of the P system and markings of the equivalent Petri net, and the P system transits from one configuration to another one iff, in the equivalent Petri net, the marking corresponding to the latter configuration is reachable from the marking corresponding to the first one. Given this construction, the focus of [7,6] is then on defining a formal framework for describing the behaviour of P systems in terms of causality/concurrency and for reasoning about reachability, conflicts and soundness of these systems by starting from their translation into *place-transitions nets* (PT-nets, for short), a specific class of Petri nets. The PT-net representation of a membrane system is therefore used to define the semantics of these systems in terms of sequences of events which consume some resources (i.e., tokens inside the places) in order to produce some new ones (*process semantics*). This construction, which is standard asynchronous PT-net, is extended in [6,7] to PT-nets operating in a maximally parallel way and to PT-nets with localities operating in a locally-maximal parallel way. PT-nets with localities are a class of PT-nets introduced in [7] where each transition belongs to certain location, in a way that resembles the distribution of the rules over the various regions of a membrane system. For such PT-nets, locally-maximal parallelism means that, in each step, for an arbitrary set of localities, a maximal number of transitions belonging to those localities fire. As well as this, the paper [6] discusses about how to represent in terms of PT-nets other features of membrane systems such as rule creation, promoters/inhibitors, membrane dissolution and variable membrane permeability.

In this paper, we start with membrane systems and we consider the problem of modelling two producer/consumer systems: a system consisting of a producer and a consumer which synchronise through a buffer of capacity one item, and a parallel version of this system where the producer and the consumer have direct access to two separate buffers, both of them having capacity equal to one. For the former system, a model is devised that is based on cell-like P systems with generalised boundary rules, whereas, for the system with two buffers, we use networks of cells which extend boundary rules to pairs of interacting cells. Then, as done in [7,6], we show how to translate our models of membrane systems into equivalent Petri nets with a corresponding semantics. The application of this construction to the two aforementioned producer/consumer systems returns Petri nets representations which are “essentially” the same as the standard Petri nets solutions illustrated in [9].

2 Membrane Systems

The reader is supposed to be familiar with the notation commonly used in membrane computing (e.g., see [8]). We recall that a *multiset* over a given alphabet V is represented as a string $x \in V^*$ where the order of the symbols does not matter and, for every $a \in V$, the *multiplicity* of a in x is denoted by $|x|_a$. We also recall that a *membrane structure* is defined as being a string of pairs of matching square brackets, which are called *membranes*. In particular, the most external membrane is called *skin membrane*, whereas a membrane that does not enclose (i.e., contain) any other membrane is called *elementary*.

Thus, a P system is formally defined as follows.

Definition 1. *A* P system *is a construct* $\Pi = (V, \mu, w_1, w_2, \ldots, w_m, R)$ *where:*

1. V *is a* finite alphabet*;*
2. μ *is a* membrane structure *consisting of* m membranes *that are labelled in a one-to-one manner by* $1, \ldots, m$*;*
3. $w_i \in V^*$*, for each* $1 \leq i \leq m$ *is a* multiset of objects associated with the region i delimited by membrane i*;*
4. R *is a* finite set of generalised boundary rules *of the form* $u\,[_i\, v\,] \rightarrow u'\,[_i\, v'\,]$ *with* $u, v, u', v' \in V^*$ *and* $1 \leq i \leq m$*.*

A P system consists of a set of m hierarchically nested membranes that identify m distinct regions (the membrane structure μ), where each region i has got assigned a multiset of objects w_i. In each region i, we can use the rules in R of the form $u\,[_i\, v\,] \rightarrow u'\,[_i\, v'\,]$ to simultaneously replace a multiset u placed outside region i and a multiset v placed inside region i with a multiset u' and a multiset v' respectively.

The computational power of boundary rules was originally investigated in [3], whereas a discussion about modelling aspects of the present generalisation is reported in [4].

A *configuration* of a P system Π is any tuple $(w'_0, w'_1, \ldots, w'_m)$ where, for all $1 \leq i \leq m$, w'_i is a multiset associated to region i, and w'_0 is a multiset of objects present outside the most external membrane. The *initial configuration* of Π is the tuple $(\lambda, w_1, \ldots, w_m)$. A P system transits from one configuration to another by applying its rules according to a certain strategy. Specifically, by following [7,6], we consider four different strategies: *sequential application* (in each step, only one rule is applied), *free parallelism* (in each step, an arbitrary number of rules is applied in parallel), *maximal parallelism*(in each step, in each membrane, a maximal number of rules is applied), *locally-maximal application* (in each step, for an arbitrary number of membranes, a maximal number of rules is applied).

Here we introduce a more general model of membrane systems which allow us to capture the essential features of most variants of cell-like P systems and tissue P systems.

Definition 2. *A* network of cells *(a* NC *for short) is a construct:*

$$\Gamma = (V, w_1, w_2, \ldots, w_n, R)$$

where: V *is a* finite alphabet*;* $w_i \in V^*$*, for all* $1 \leq i \leq n$*, is the multiset* initially associated to cell i*;* R *is a finite set of* interaction rules *of the forms* $(u, i \rightarrow u', k; v, j \rightarrow v', l)$ *or* $(u, i \rightarrow u', k)$ *with* $u, v, u', v' \in V^*$*,* $1 \leq i, j, k, l \leq n$*.*

A network of cells consists of n cells, numbered from 1 to n, that contain multisets of objects over V, initially cell i contains multiset w_i. Cells can interact with each other by means of the rules in R. An interaction rule of the form $(u, i \rightarrow$

$u', k; v, j \to v', l)$ specifies that an occurrence of multiset u and an occurrence of multiset v can be respectively consumed in cell i and in cell j at the same time by producing a multiset u' inside cell k and an occurrence of multiset v' inside cell l. Similarly, a rule $(u, i \to u', k)$ specifies that a multiset u inside cell i can be "replaced" by a multiset u' inside cell k. Thus, in a NC, interactions are not limited to occur between the inside and the outside of a membrane but they can occur between any two cells in the system. In particular, these cells may be (but they are not necessarily) organised in a graph-like structure as in tissue P system. Moreover, it is clear that P systems of Definition 1 and other variants of membranes systems already considered in the literature (e.g., P systems with symport/antiport [8], generalised communicating P systems [10]) represent just particular cases of NC's, hence NC's are Turing complete. Amongst the classes listed above, generalised communicating P systems [10] are the closest to NC's by their way of synchronising cells and their contents; however, this is a communicative model where objects involved in the rules cannot be transformed, but they are jut moved from a cell to another one.

A *configuration* of a NC Γ is any tuple $(w'_1, \ldots, w'_n)$ where, for all $1 \le i \le m$, w_i is a multiset associated to cell i; the *initial configuration* of Γ is the tuple $(w_1, \ldots, w_n)$. Yet again, we can have different types of behaviours for a NC by considering different strategies for the application of the rules: *sequential application*, *free parallelism*, *maximal parallelism*, and *locally-maximal application*.

3 Producer/Consumer Systems

We consider the problem of modelling two producer/consumer systems: a system consisting of a producer and a consumer which synchronise through a buffer of capacity one item, and a parallel version of this system where the producer and the consumer have direct access to two separate buffers, both of them having capacity equal to one. Specifically, the producer has two states: "ready to produce" and "ready to deliver"; the consumer has two states, "ready to remove" and "ready to consume"; the buffer has two states: "filled" and "empty". In state "ready to produce", the producer executes the operation "produce" and moves to state "ready to deliver"; in state "ready to deliver", if the buffer is "empty", the producer executes the operation "deliver", which fills the buffer, and moves back to state "ready to produce". Similarly, in state "ready to remove", if the buffer is "filled", the consumer executes the operation "remove", which empties the buffer, and moves to state "ready to consume"; in state "ready to consume", the consumer executes the operation "consume" and moves back to state "ready to remove". In the parallel version, the producer, when in state "ready to deliver" can decide to execute either "deliver to buffer 1" or "deliver to buffer 2".

In order to model the aforementioned producer/consumer systems with only one buffer, we consider a P system PC with 3 membranes labelled in a one-to-one manner with values in $\{1, 2, 3\}$: membrane 1 and membrane 2 are elementary membranes, whereas membrane 3 is a non-elementary membrane containing both membrane 1 and membrane 2. Membrane 1 represents the producer, membrane 2 represents the consumer, whereas membrane 3 represents the buffer. Membrane 1 stores an object which specifies the state of the producer; this can be either P ("ready to produce") or D ("ready to deliver"). Similarly, membrane 2 stores an object which specifies the state of the consumer; this can be either R ("ready to remove") or C ("ready to consume"). Membrane 3 instead stores an object representing the state of the buffer, being either F ("filled") or E ("empty"). The initial configuration is the tuple (P, R, E). The desired behaviour is then implemented by considering the following rules:

1. $[_1 P \rightarrow [_1 D$,
2. $E\, [_1 D \rightarrow F\, [_1 P$,
3. $F\, [_2 R \rightarrow E\, [_2 C$,
4. $[_2 C \rightarrow [_2 R$.

Thus, rule 1 represents the action "produce" which makes the producer transit from state P ("ready to produce") to state D ("ready to deliver"). The buffer can then be filled by applying rule 2 which is applicable only when membrane 3 contains object E (i.e., the buffer is empty) and membrane 1 contains an object D (i.e., the producer is in state "ready to deliver"). Similarly, rules 3 and 4 are associated to membrane 2 (the consumer) and they represent the action "remove" and "consume" respectively. Notice that, if maximal parallelism is considered, P system PC is deterministic, whereas, for all the other strategies, P system PC behaves in a non-deterministic way.

Then, in order to model the producer/consumer system with two parallel buffers, we consider a NC NPB consisting of 4 cells, cell 1 $\equiv$ "the producer", cell 2 $\equiv$ "the consumer", cell 3 $\equiv$ "buffer 1" and cell 4 $\equiv$ "buffer 2". Cell 1, as well as cell 2, can directly interact both with cell 3 and with cell 4 but there is no direct interaction between cell 1 and cell 4. As in the previous P system model, cell 1 always stores either an object P ("ready to produce") or an object D ("ready to deliver"); cell 2 always stores either an object R ("ready to remove") or an object C ("ready to consume"); cell 3 always stores either an object F ("filled") or an object E ("empty"); the same applies to cell 4. The initial configuration is given by the tuple (P, R, E, E). In order to implement the desired behaviour, we instead give the following rules:

1. $(P, 1 \rightarrow D, 1)$,
2. $(D, 1 \rightarrow P, 1; E, 3 \rightarrow F, 3)$,
3. $(D, 1 \rightarrow P, 1; E, 4 \rightarrow F, 4)$,
4. $(R, 2 \rightarrow C, 2; F, 3 \rightarrow E, 3)$,
5. $(R, 2 \rightarrow C, 2; F, 4 \rightarrow E, 4)$,
6. $(C, 2 \rightarrow R, 2)$.

Thus, when an object D is present inside cell 1, we can non-deterministically apply either rule 2 or rule 3, if cell 3 and cell 4 both contain an object E. Then, depending on the content of these two cells, we can non-deterministically apply either rule 4 or rule 5. However, rule 2 can never be applied in parallel with rule 3 because cell 1 never contains more than one object D. Also, rule 4 can never be applied in parallel with rule 5 because cell 2 never contains more than one object R. Notice that, in this case, we have a non deterministic behaviour even if we consider maximal parallelism.

4 Petri Net Representation

We recall the notion of PT-nets with localities from [6].

Definition 3. *A* PT-net with localities *is a construct* $N = (P, T, W, M_0, L)$ *where:* P *is a finite set of* places*,* T *is a finite set of* transitions*, with* $P \cap T = \emptyset$*,* $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is the* weight function*,* M_0 *is a multiset over* P *called the* initial marking*, and* L *is a* location mapping*.*

PT-nets are usually represented by diagrams where places are draw as circles, transitions are drawn as squares annotated with their location, and a directed arc (x, y) is added between x and y if $W(x, y) \geq 1$. These arcs are then annotated with their weight if this is 2 or more. Moreover, when localities are not used (i.e., all transitions are associated to the same location), the location mapping is omitted from the definition.

Given a PT-net N, the *pre-* and *post-multiset* of a transition t are respectively the multiset $pre_N(t)$ and the multiset $post_N(t)$ such that, for all $p \in P$, $|p|_{pre_N(t)} = W(p, t)$ and $|p|_{post_N(t)} = W(t, p)$. A configuration of N, which is called a *marking*, is any multiset over P; in particular, for every $p \in P$, $|p|_M$ represents the number of *tokens* present inside place p. A transition t is *enabled* at a marking M if the multiset $pre_N(t)$ is contained in the multiset M. An enabled transition t at marking M can *fire* and produce a new marking M' such that $M' = M - pre_N(t) + post_N(t)$ (i.e., for every place $p \in P$, the firing transition t consumes $|p|_{pre_N(t)}$ tokens and produces $|p|_{post_N(t)}$ tokens). As shown in [6], the notion of enabled transition can then be extended to a multiset of transitions to allow several occurrences of different transitions to fire in parallel at the same time. In particular, a transition t is said to be *enabled* with multiplicity $m \geq 0$ at a marking M if, M contains at most $m \geq 0$ copies of the multiset $|p|_{pre_N(t)}$. Thus, the following semantics are identified in [6]: *sequential semantics* (in each step, only one occurrence of an enabled transition fires); *freely-parallel semantics* (in each step, an arbitrary number of occurrences of enabled transitions fire); *maximally-parallel semantics*; *locally-maximally-parallel semantics* (an arbitrary number of localities is chosen and, for each one of this locality, a maximal number of occurrences of enabled transitions fire). In this latter semantics, transitions associated to the same locality are seen as belonging to some “independent” sub-unit and locally the net operates in a maximally parallel manner, although globally not all localities have necessarily to operate in parallel at the same time.

For the basic model of membrane system, it is then shown in [7,6] how to transform a P system into an equivalent PT-net with localities. Here we extend this construction to the class of P systems and the class of networks of cells introduced in Section 2.

Specifically, let $\Pi = (V, \mu, w_1, w_2, \ldots, w_m, R)$ be a P system and, for all $0 \leq i \leq m$, let $h_i : V^* \to \{(a, i) \mid a \in V\}^*$ be such that $h_i(a) = (a, i)$, for all $a \in V$, and $h_i(uv) = h_i(u)h_i(v)$, for all $u, v \in V^*$. Moreover, let us suppose the rules in R to be labelled in a one-to-one manner with values in $\{1, 2, \ldots, s\}$, for some $s \geq 0$. Let $V' = \{(a, i) \mid a \in V, 0 \leq i \leq m\}$ and let R' be a set of multiset rewriting rules such that $t: \ h_j(u)h_i(v) \to h_j(u')h_i(v') \in R'$ iff, $t: \ u\,[_i\, v\,] \to u'\,[_i\, v'\,] \in R$, and j is the membrane which directly contains membrane j, or $j = 0$ if i is the skin membrane. The PT-net corresponding to Π is the PT-net $\mathcal{N}(\Pi) = (P, T, W, M_0)$ where: $P = V'$, $T = \{1, 2, \ldots, s\}$, and, for all $b \in V'$ and $1 \leq t \leq s$, $W(b, t) = n$ if $t : u \to v \in R'$ and $|b|_u = n$, for all $b \in V'$ and $1 \leq t \leq s$, $W(t, a) = n$ if $t : u \to v \in R'$ and $|b|_v = n$, and $M_0 = h_1(w_1)h_2(w_2)\ldots h_m(w_m)$.

Let $\Gamma = (V, w_1, w_2, \ldots, w_n, R)$ be a NC and for all $1 \leq i \leq n$, let $h_i : V^* \to \{(a, i) \mid a \in V\}^*$ be as above. Moreover, let us suppose the rules in R to be labelled in a one-to-one manner with values in $\{1, 2, \ldots, s\}$, for some $s \geq 0$. Then, let $V' = \{(a, i) \mid a \in V, 1 \leq i \leq n\}$ and let R' be a set of multiset rewriting rules such that $t: \ h_i(u)h_j(v) \to h_k(u')h_l(v') \in R'$ (or $t: \ h_i(u) \to h_k(u') \in R'$) iff, $t: \ (u, i \to u', k; v, j \to v', l) \in R$ (or $t: \ (u, i \to u', k; v, j \to v', l) \in R$). The PT-net corresponding to Γ is the PT-net $\mathcal{N}'(\Gamma) = (P, T, W, M_0)$ where: $P = V'$, $T = \{1, 2, \ldots, s\}$, and, for all $b \in V'$ and $1 \leq t \leq s$, $W(b, t) = k$ if $t : u \to v \in R'$ and $|b|_u = k$, for all $b \in V'$ and $1 \leq t \leq s$, $W(t, a) = n$ if $t : u \to v \in R'$ and $|b|_v = k$, and $M_0 = h_1(w_1)h_2(w_2)\ldots h_n(w_n)$.

Thus, as observed in [7,6], we have that, for a sequential semantics, a freely-parallel semantics and a maximally-parallel semantics, every P system Π behaves the same as the PT-net $\mathcal{N}(\Pi))$ and every NC Γ behaves the same as the PT-net $\mathcal{N}'(\Gamma))$. For a P system Π, "behaves the same" means that Π transits from a configuration $(w_0', w_1', \ldots, w_m')$ to a configuration $(w_0'', w_1'', \ldots, w_m'')$ if and only if, in the PT-net $\mathcal{N}((Pi))$, the marking $h_0(w_0'')h_1(w_1'')\ldots h_m(w_m'')$ is obtained from the marking $h_0(w_0')h_1(w_1')\ldots h_m(w_m')$. The same applies to a NC Γ. If we instead consider a locally-maximally-parallel semantics, then Π behaves the same as the PT-net $\mathcal{N}(\Pi))$ when localities are introduced in such a way that every transition corresponding to a rule $u\,[_i\, v\,] \to u'\,[_i\, v'\,] \in R$ is assigned to locality i. This is not the case for a NC Γ, for a locally-maximally-parallel semantics, we cannot directly transform a NC Γ into an equivalent PT-net with localities because an interaction rule $(u, i \to u', k; v, j \to v', l)$ is symmetric and, in a sense, it belongs both to cell i and cell j (i.e., it is dynamically assigned either to cell i or to cell j or to both).

Now, let us consider the two producer/consumer systems of Section 3. PT-net models of these two systems are given in [9] and they are reported in Figure 1 and Figure 2.

For such PT-nets, we have that: if we construct PT-net $\mathcal{N}(PC)$, with PC being the P system of Section 3, then $\mathcal{N}(PC)$ is the same as the PT-net of Figure 1 except for the names of places and transitions (i.e., they identify two isomorphic graphs); a similar consideration holds for PT-net $\mathcal{N}'(NPB)$, with NPB being the network of cells of Section 3, with respect to the PT-net of Figure 2. In other words, we observe a "direct" correspondence between the

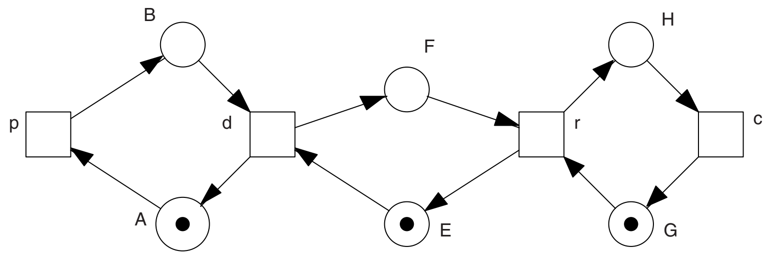


Fig. 1. PT-net model of a producer/consumer system

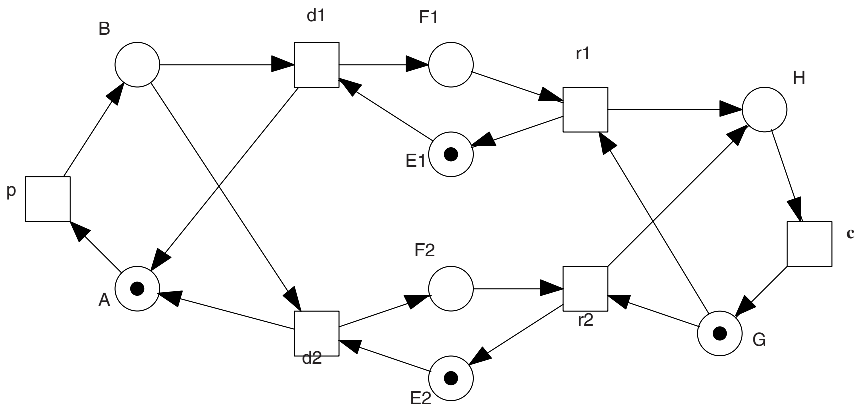


Fig. 2. PT-net model of a producer/consumer system with two parallel buffers

membrane systems representation and the Petri nets representation: every rule in the membrane system model corresponds to a transition in the Petri net models, and every object that appears inside the membrane system corresponds to a token that appears inside a certain place in the PT-net.

On the other hand, if we start with the PT-net of Figure 1 (or with the PT-net of Figure 2), we can obtain different "interpretations" in terms of membrane systems. For instance, we can consider a one-membrane P system containing all the rules corresponding to all the transitions of the PT-net without any distinction between producer, consumer and buffer. Alternatively, we can see the four transitions in the PT-net of Figure 1 as four rules involving interactions between four distinct cells where the passage token between places corresponds to objects which are exchanged between these membranes. In other words, for a locally-maximally-parallel semantics, in order to obtain the same type of behaviour, we have to consider different membrane structures according to the particular choice of location mapping.

5 Discussion

In this paper we have introduced the notion of network of cells as a model of membrane systems suitable to capture the essential features of most variants of cell-like P systems and tissue P systems. Specifically, for the case of cell-like P systems, network of cells corresponds to P systems with generalised boundary rules (Definition 1). We have then illustrated two models of producer/consumer systems based on these variants of membrane systems.

Next, as done in [7,6], we have shown how to translate our membrane system models into equivalent Petri nets with a corresponding semantics. In particular, for the two models of producer/consumer systems, we have observed that the solutions based on membrane systems proposed here are "essentially" equivalent to the standard Petri nets representations described in [9]. In turn, this allows us to transfer to the membrane system models the structural properties which are formally proved in [9] and are thoroughly managed through a plethora of tools [1] for the corresponding classes of Petri nets.

However, we want to remark some differences between membrane systems and Petri nets, and between their relative modelling approaches. In membrane systems, one naturally reasons about components (e.g., producer, consumer, buffer) and these are usually seen as being separate membranes (or cells). Also, one naturally distinguishes between operations affecting the inner state of a membrane and the operations involving interactions between different membranes. Moreover, in membrane systems, the inner state of a membrane can be given by an arbitrarily large multiset; this allows us to combine cooperation at the level of objects with interaction between different cells. Petri nets, with their graphical notation, are centred around the idea of resources which have to be acquired (tokens inside places) before certain actions can be taken; this facilitates the reasoning about properties like causality (the execution of certain actions depends on the execution of some others), concurrency (certain group of action can always be executed in parallel), and conflicts (certain actions compete with some others for the usage of certain resources); in membrane systems, these structural properties instead remains somehow hidden in the formalism used for representing the rules. Finally, we notice that the concept of localities in PT-nets do not necessarily corresponds to the concept of cells (or membranes) in membrane systems. Localities are mainly used to identify groups of transitions which locally may be executed in a maximally parallel way; from a membrane systems point of view, transitions in the same groups may correspond to interactions involving many different cells. In fact in the PT-net model, transitions represent both transformations occurring inside the cell and interactions with other cells, hence the locality of a transition may not correspond to the concept of "a transformation taking place locally in a certain cell".

Acknowledgements

Marian Gheorghe and Francesco Bernardini are supported by the British Council research contract PN 05.014/2006. Maurice Margenstern and Sergey Verlan are funded by the Égide programme Alliance 0881UG. Francesco Bernardini is also supported by NWO, Organisation for Scientific Research of The Netherlands, project 635.100.006 “VIEWS”.

References

1. Petri Nets World - Petri Nets Tools Database: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
2. The P Systems Web Page: http://psystems.disco.unimib.it
3. Bernardini, F., Manca, V.: P systems with boundary rules. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC-CdeA 02, LNCS, vol. 2597, pp. 107–118. Springer, Heidelberg (2003)
4. Bernardini, F., Romero-Campero, F., Gheorghe, M., Pérez-Jiménez, M.: A Modelling Approach Based on P Systems with Bounded Parallelism. In: Hoogeboom, H., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 49–65. Springer, Heidelberg (2007)
5. Dal Zilio, S., Formenti, E.: On the Dynamics of PB Systems: A Petri Net View. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003, LNCS, vol. 2933, pp. 153–167. Springer, Heidelberg (2004)
6. Kleijn, J., Koutny, M.: Synchrony and Asynchrony in Membrane Systems. In: Hoogeboom, H., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 66–85. Springer, Heidelberg (2007)
7. Kleijn, J., Koutny, M., Rozenberg, G.: Towards a Petri Net Semantics for Membrane Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 292–309. Springer, Heidelberg (2006)
8. Păun, Gh.: Membrane Computing. An Introduction, Springer, Heidelberg (2002)
9. Reisig, W.: Elements of Distributed Algorithms. Modelling and Analysis with Petri Nets, Springer, Heidelberg (1998)
10. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: Generalized Communicating P Systems. (Submitted 2007)

A Minimal Pair in the Quotient Structure $M/NCup$

Rongfang Bie[1,⋆] and Guohua Wu[2,⋆⋆]

[1] School of Information Science and Technology
Beijing Normal University
Beijing 100875, People's Republic of China
rfbie@bnu.edu.cn
[2] Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University
Singapore 639798, Republic of Singapore
guohua@ntu.edu.sg

Abstract. In this paper, we prove the existence of a minimal pair of c.e. degrees $\mathbf{a}$ and $\mathbf{b}$ such that both of them are cuppable, and no incomplete c.e. degree can cup both of them to $\mathbf{0}'$. As a consequence, $[\mathbf{a}]$ and $[\mathbf{b}]$ form a minimal pair in $M/NCup$, the quotient structure of the cappable degrees modulo noncuppable degrees. We also prove that the dual of Lempp's conjecture is true.

Keywords: Computably enumerable degrees, quotient structure, minimal pairs.

1 Introduction

Friedberg (1956) and Muchnik (1957) proved independently that there are two incomparable c.e. degrees. This answers Post's question positively. Improving this, Sacks [13,14] showed that every nonzero c.e. degree $\mathbf{a}$ is the joint of two incomparable c.e. degrees and that the computably enumerable degrees are dense. After seeing this, in 1965, Shoenfield conjectured that for any finite partial orderings $P \subseteq Q$, with the least element 0 and the greatest element 1, any embedding of P into $\mathcal{R}$ (the set of all c.e. degrees) can be extended to an embedding of Q into $\mathcal{R}$. Shoenfield also listed two consequences of this conjecture:

C1. There are no incomparable c.e. degrees $\mathbf{a}, \mathbf{b}$ such that $\mathbf{a} \cap \mathbf{b}$ (the infimum of $\mathbf{a}, \mathbf{b}$) exists;
C2. For any c.e. degrees $\mathbf{0} < \mathbf{c} < \mathbf{a}$, there is a c.e. degree $\mathbf{b} < \mathbf{a}$ such that $\mathbf{b} \cup \mathbf{c} = \mathbf{a}$.

C1 is refuted by the existence of minimal pairs (Lachlan [7], Yates [20], independently). Therefore, Shoenfield's conjecture cannot be true. Here we say that two nonzero c.e. degrees $\mathbf{a}, \mathbf{b}$ form a *minimal pair* if $\mathbf{a}, \mathbf{b}$ have infimum $\mathbf{0}$.

Say that a degree $\mathbf{a}$ is *cappable* if $\mathbf{a}$ is $\mathbf{0}$ or a half of a minimal pair. A degree is *noncappable* if it is not cappable. The dual notion of the cappable degrees is the cuppable

⋆ Bie is supported by NNSF Grant No. 19931020, No. 10001006 and No. 60273015 of China.
⋆⋆ Correspondence author. Wu is partially supported by a a start-up grant No. M48110008 and a research grant No. RG58/06 from NTU.

degrees. That is, a c.e. degree $\mathbf{c}$ is cuppable if there is an incomplete c.e. degree $\mathbf{b}$ such that $\mathbf{c} \cup \mathbf{b} = \mathbf{0}'$. C2 implies that all the nonzero c.e. degrees are cuppable, which turns out to be wrong, because Yates and Cooper (see [3]) proved the existence of a nonzero c.e. degree cupping no incomplete c.e. degree to $\mathbf{0}'$. In [1], Ambos-Spies, Jockusch, Shore and Soare proved that a c.e. degree is noncappable if and only if it can be cupped to $\mathbf{0}'$ via a low c.e. degree. An immediate consequence of this is that all the noncuppable degrees are cappable, and hence each c.e. degree is either cappable or cuppable, which was first proved by Harrington.

Note that all the cappable degrees and all the noncuppable degrees form ideals in $\mathcal{R}$, M and $NCup$ respectively. It becomes interesting to study the corresponding quotient structures: $\mathcal{R}/M$, $\mathcal{R}/NCup$. Schwarz provided in [15] several structural properties of $\mathcal{R}/M$. Particularly, Schwarz pointed out that Sacks splitting is true in $\mathcal{R}/M$, but there is no minimal pair in this structure. Sui and Zhang proved in [19] that C2 listed above is true in $\mathcal{R}/M$. Lempp asked in [17] whether the Shoenfield conjecture holds in $\mathcal{R}/M$. This problem was solved by Yi in [22] who claims that Shoenfield conjecture is also not true in $\mathcal{R}/M$.

Both $\mathcal{R}/M$ and $\mathcal{R}/NCup$ have the least and the greatest elements. In $\mathcal{R}/NCup$, the least element is the set of all noncuppable degrees, and the greatest element contains only one element, $\mathbf{0}'$. It is also easy to see that in $\mathcal{R}/NCup$, every nonzero element is cuppable to the greatest element. In [10], Li, Wu and Yang proved that there is a minimal pair in $\mathcal{R}/NCup$, and hence Shoenfield conjecture does not hold in $\mathcal{R}/NCup$ neither. Recently, in [11], Li, Wu and Yang prove that the diamond lattice can be embedded into $\mathcal{R}/NCup$ preserving 0 and 1. The construction involves several new features.

Theorem 1. *There are two c.e. degrees* $\mathbf{a}$ *and* $\mathbf{b}$ *such that* $\mathbf{a} \cup \mathbf{b} = \mathbf{0}'$ *(hence* $[\mathbf{a}] \cup [\mathbf{b}] = [\mathbf{0}']$*), and* $[\mathbf{a}] \cap [\mathbf{b}] = [\mathbf{0}]$*, where* $[\mathbf{a}], [\mathbf{b}], [\mathbf{0}], [\mathbf{0}']$ *are the equivalence classes of* $\mathbf{a}, \mathbf{b}, \mathbf{0}, \mathbf{0}'$ *in the quotient structure* $\mathcal{R}/NCup$*.*

There are several fundamental questions left open, like whether Sacks splitting is true in $\mathcal{R}/NCup$ (it is true in $\mathcal{R}/M$, as proved in [15]) or whether C2 is true in $\mathcal{R}/NCup$.

Since $NCup$ is also an ideal of M, and M is a upper semi-lattice, we ask what the quotient structure $M/NCup$ looks like. It has a least element, but it seems that there is no greatest element in this structure. If $\mathbf{a}$ is a c.e. degree, from now on, we use $[\mathbf{a}]$ to denote the equivalence class in $M/NCup$ containing $\mathbf{a}$. Thus, $[\mathbf{0}]$ is the set of all noncuppable degrees. In this paper, we first prove that there is a minimal pair in $M/NCup$.

Theorem 2. *There are two cuppable c.e. degrees* $\mathbf{a}$ *and* $\mathbf{b}$ *such that* $\mathbf{a}$ *and* $\mathbf{b}$ *form a minimal pair in the c.e. degrees and* $[\mathbf{a}]$*,* $[\mathbf{b}]$ *form a minimal pair in* $M/NCup$*.*

Thus, Shoenfield Conjecture is not true in $M/NCup$. The following is the crucial step to prove Theorem 2.

Theorem 3. *There are two cuppable degrees* $\mathbf{a}$ *and* $\mathbf{b}$ *such that* $\mathbf{a}$ *and* $\mathbf{b}$ *form a minimal pair and that no incomplete c.e. degree can cup both* $\mathbf{a}$ *and* $\mathbf{b}$ *to* $\mathbf{0}'$*.*

The strategy to ensure that no incomplete c.e. degree can cup both $\mathbf{a}$ and $\mathbf{b}$ to $\mathbf{0}'$ is different from the one provided in [10]. We will outline the proof of Theorem 3 in Section 2.

We prove now that $\mathbf{a}$ and $\mathbf{b}$ degrees provided in Theorem 3 are exactly the ones we want in Theorem 2. Since $\mathbf{a}$ and $\mathbf{b}$ are both cuppable and cappable, $[\mathbf{a}], [\mathbf{b}]$ are nonzero elements in $M/NCup$. To prove that $[\mathbf{a}] \cap [\mathbf{b}] = [\mathbf{0}]$, suppose for a contradiction that there is a c.e. degree $\mathbf{c}$ such that $[\mathbf{0}] < [\mathbf{c}] \leq [\mathbf{a}], [\mathbf{b}]$. Then $\mathbf{c}$ is cuppable, and we assume that $\mathbf{c} \cup \mathbf{w} = \mathbf{0}'$ with $\mathbf{w} < \mathbf{0}'$. Since $[\mathbf{c}] \leq [\mathbf{a}], [\mathbf{b}]$, there are noncuppable degrees $\mathbf{m}_1$, $\mathbf{m}_2$ such that $\mathbf{c} \leq \mathbf{a} \cup \mathbf{m}_1$, $\mathbf{c} \leq \mathbf{b} \cup \mathbf{m}_2$, and hence, $\mathbf{a} \cup \mathbf{m}_1 \cup \mathbf{m}_2 \cup \mathbf{w} = \mathbf{0}'$, $\mathbf{b} \cup \mathbf{m}_1 \cup \mathbf{m}_2 \cup \mathbf{w} = \mathbf{0}'$. Let $\mathbf{v} = \mathbf{m}_1 \cup \mathbf{m}_2 \cup \mathbf{w}$. Then $\mathbf{v}$ cups both $\mathbf{a}$ and $\mathbf{b}$ to $\mathbf{0}'$. According to Theorem 3, $\mathbf{v} = \mathbf{m}_1 \cup \mathbf{m}_2 \cup \mathbf{w}$ is complete. Since both $\mathbf{m}_1, \mathbf{m}_2$ are noncuppable, $\mathbf{w} = \mathbf{0}'$. A contradiction. This completes the proof of Theorem 2.

The existence of noncappable degrees was first proved by Yates in [21], and this existence enables us to prove that any nonzero c.e. degree $\mathbf{c}$ bounds a cappable degree. To see this, we consider two cases. If $\mathbf{c}$ itself is cappable, then we are done since cappable degrees are downwards closed. Otherwise, let $\mathbf{a}$ be any cappable degree. Since $\mathbf{c}$ is assumed to be noncappable, there is a nonzero c.e. degree $\mathbf{b}$ below both $\mathbf{a}$ and $\mathbf{c}$, and hence $\mathbf{b}$ is cappable. An almost the same argument proves that any nonzero noncappable degree bounds a minimal pair. Hence, Lachlan's nonbounding degrees are all cappable.

Li and Wang (see Li [9]) proved that it is impossible to take the nonuniformity in the previous argument away. The nature behind this nonuniformity is that the direct permitting method and the minimal pair construction are not consistent. In 1996, Lempp raised the following conjecture:

Conjecture 1. (Lempp, see Slaman [17]) For any c.e. degrees $\mathbf{a}, \mathbf{b}$ with $\mathbf{a} \not\leq \mathbf{b}$, there is a cappable degree $\mathbf{c} \leq \mathbf{a}$ such that $\mathbf{c} \not\leq \mathbf{b}$.

Li refuted Lempp's conjecture in [9] by constructing c.e. degrees $\mathbf{a}, \mathbf{b}$ with $\mathbf{a} \not\leq \mathbf{b}$ such that any cappable degree below $\mathbf{a}$ is also below $\mathbf{b}$.

Unlike the cappable degrees, not every nonzero c.e. degree bounds noncuppable degrees and such degrees are called *plus-cupping* degrees[1]. However, it is true that above any incomplete c.e. degree, there is an incomplete cuppable degree. We can prove this as follows: let $\mathbf{c}$ be a given incomplete c.e. degree. The case when $\mathbf{c}$ itself is cuppable is trivial. If $\mathbf{c}$ is noncuppable, then let $\mathbf{a}$ be any incomplete cuppable degree, then $\mathbf{a} \cup \mathbf{c}$ is also incomplete and cuppable. We will provide a uniform construction in Theorem 4.

In [5], Downey and Lempp considered the dual notion of the plus-cupping degrees, the plus-capping degrees and proved that no plus-capping degrees exist. In this paper, we prove that the dual of Lempp conjecture is true.

Theorem 4. *For any incomplete c.e. degrees* $\mathbf{a}, \mathbf{b}$ *with* $\mathbf{a} \not\geq \mathbf{b}$*, there is an incomplete cuppable degree* $\mathbf{c} > \mathbf{a}$ *such that* $\mathbf{c} \not\geq \mathbf{b}$.

The proof of Theorem 4 employs Sacks coding strategy, Sacks preservation strategy, and splitting $\mathbf{0}'$ into $\mathbf{c}$ and $\mathbf{e}$. While we will make $\mathbf{c} > \mathbf{a}$, we will not require that $\mathbf{e}$ is above $\mathbf{a}$ (Harrington's nonsplitting theorem says that we cannot do this), which leaves

[1] Harrington's original notion of plus-cupping degrees is even stronger: $\mathbf{a}$ is plus-cupping, in the sense of Harrington, if for any c.e. degrees $\mathbf{b}$, $\mathbf{c}$, if $\mathbf{0} < \mathbf{b} \leq \mathbf{a} \leq \mathbf{c}$, there is a c.e. degree $\mathbf{e}$ below $\mathbf{c}$ cupping $\mathbf{b}$ to $\mathbf{c}$. The notion given in this paper was given by Fejer and Soare in [6].

enough space for us to combine the splitting strategy with the Sacks coding strategy. We will outline the proof of Theorem 4 in Section 3.

Our notation and terminology are standard and generally follow Cooper [4] and Soare [18].

2 Proof of Theorem 3

In this section, we give the outline of the proof of Theorem 3. We will construct incomplete c.e. sets A, B, C, D, E, F and p.c. functionals Γ, Δ to satisfy the following requirements:

$G : K = \Gamma^{A,C}$;
$H : K = \Delta^{B,D}$;
$P_e : E \neq \Phi_e^C$;
$Q_e : E \neq \Phi_e^D$;
$R_e : \Phi_e^A = \Phi_e^B = f$ is total $\Rightarrow f$ is computable;
$S_e : \Phi_e^{A,W} = \Psi_e^{B,W} = K \oplus F \Rightarrow W \geq_T K$;

where $e \in \omega$, $\{(\Phi_e, \Psi_e, W_e) : e \in \omega\}$ is an effective enumeration of triples (Φ, Ψ, W), Φ, Ψ p.c. functionals, W a c.e. set. K is a fixed creative set.

It is easy to see that the P, Q-requirements ensure that C and D are incomplete, thus, by the G, H-requirements, both A and B are cuppable. The R-requirements ensure that A and B form a minimal pair, and the S-requirements ensure that no incomplete c.e. set can cup both A and B to K. Therefore, A and B are exactly the sets Theorem 3 requires.

2.1 The G and H-Strategies

The G and H-strategies will be dedicated to the construction of the functionals Γ, Δ respectively, which will reduce K to $A \oplus C$ and $B \oplus D$. As the constructions of Δ and Γ are the same, we only describe the construction of Γ, which will be defined such that for any x, $\Gamma^{A,C}(x)$ is defined and equals to $K(x)$.

Let $\{K_s\}_{s \in \omega}$ be a recursive enumeration of K. Γ will be defined by stages as follows: at stage s,

1. If there are xs such that $\Gamma^{A,C}(x)[s] \downarrow \neq K_s(x)$, then let k be the least such x, enumerate $\gamma(k)$ into C, and let $\Gamma^{A,C}(x)$ be undefined for all $x \geq k$.
2. Otherwise, let k be the least number x such that $\Gamma^{A,C}(x)[s]$ is not defined, then define $\Gamma^{A,C}(k)[s] = K_s(k)$ with $\gamma(k)[s]$ a fresh number.

The G-strategy (and H-strategy) has the highest priority, in the sense that at any time, a number x enters K will require us to put a number $\leq \gamma(x)$ ($\leq \delta(x)$) into C (D respectively) immediately, and no other strategies can stop, or even delay, such actions.

We note that the G-strategy itself never enumerates any element into C. In the construction, from time to time, we need to enumerate certain γ-markers into A to lift the γ-uses, in order to prevent the P-strategies from being injured by the G-strategy.

Returning to the construction of Γ, we will ensure that the corresponding γ-use function to have the following basic properties:

1. For any k, s, if $\Gamma^{A,C}(k)[s]$ is defined, then $\gamma(k)[s] \notin A_s \cup C_s$;
2. For any x, y, if $x < y$, and $\gamma(y)[s]$ is defined, then $\gamma(x)$ is also defined at this stage, and $\gamma(x)[s] < \gamma(y)[s]$;
3. Whenever we define $\gamma(k)$, we define it as a fresh number, the least number bigger than any number being used so far;
4. $\Gamma^{A,C}(x)$ is undefined at stage s iff at this stage, there is an $y \leq x$ such that $\gamma(y)$ is enumerated into A or C.

If Γ is constructed as total, the (1) – (4) above will ensure that $\Gamma^{A,C} = K$ and G is satisfied.

2.2 The *P* and *Q*-Strategies

All the P and Q-strategies will ensure that C and D are not complete. Again, since the Q-strategies are the same as the P-strategies, we only describe how the P-strategies are satisfied.

A single P-strategy, P_e say, will be devoted to find an x such that $C(x) \neq \Phi_e^E(x)$. It is a variant of the Friedberg-Muchnik strategy, modified to cooperate with the G-strategy (later in the S-strategies, we will see how to modify a P-strategy to work consistently with the S-strategies). Recall that the G-strategy always enumerates the γ-markers into C, but it may happen that a P-strategy wants to preserve a computation $\Phi_e^C(x)$, but the G-strategy wants to put a small number into C or A to code K. If we enumerate such a number into C, this enumeration can change the computation $\Phi_e^C(x)$. With this in mind, when a P-strategy wants to preserve a computation $\Phi_e^C(x)$, we enumerate a small number into A first, to lift up the γ-uses, to make sure that the computation $\Phi_e^C(x)$ will not be changed by the G-strategy. A P-strategy works as follows:

1. Choose k, as a fresh number. Whenever a number $n \leq k$ enters K, go to 2.
2. Appoint a witness, $x > k$ say as a fresh number.
3. Wait for a stage s at which $\Phi_e^C(x)[s] \downarrow = 0$.
4. Enumerate $\gamma(k)[s]$ into A and x into E, and stop.

By $x > k$, the enumeration of $\gamma(k)[s]$ into A lifts all $\gamma(n)$, $(n \geq k)$, to big numbers and so, if after stage s, no $n \leq k$ enters K, then since every $\gamma(n)$ with $n \geq k$ is defined as big numbers after stage s, $\Phi_e^C(x)$ is protected from the enumeration of the G-strategy. Therefore,

$$\Phi_e^C(x) = \Phi_e^C(x)[s] = 0 \neq 1 = E_{s+1}(x) = E(x).$$

P_e is satisfied.

Now consider the case when some $n \leq k$ enters K, at stage $s' > s$ say. Then at this stage, $\gamma(n)[s']$, which may be less than $\varphi_e(x)[s]$, is enumerated into C, according to the G-strategy. This enumeration can change the computation $\Phi_e^C(x)$. If so, we go to 2 by choosing another witness for P_e, since $\Phi_e^C(x)$ may converge later to 1, and we cannot obtain a disagreement between E and Φ_e^C at x. Such a process can happen at most k many times, and after the last time, when $\Phi_e^C(x')$ converges to 0 again, we enumerate $\gamma(k)[s]$ and x' into C, and the computation of $\Phi_e^C(x')$ can never be injured by the G-strategy afterwords, and P_e is satisfied forever.

As usual, we call the parameter k above the "killing point" of this P-strategy. When a number $\leq k$ enters K, then we reset this strategy by invalidating all of the parameters we have defined, except k. As discussed above, since k is fixed, this strategy can be reset at most $k+1$ many times.

2.3 The R-Strategies

There is no conflict between the G, H-strategies (coding K into $A \oplus C$ and $B \oplus D$) and the R-strategies since in general, we only put the γ, δ-markers into C and D to rectify Γ and Δ, and we only put numbers into A or B when a P strategy or Q-strategy acts. This is consistent with the minimal pair construction.

2.4 The S-Strategies

In the following, we describe how to make the P, Q-strategies work consistently with the G, H and the S-strategies. First, an S-strategy will construct a p.c. function Θ such that if $\Phi_e^{A,W} = \Psi_e^{B,W} = K \oplus F$ is true, then Θ^W will be totally defined and compute K correct.

Let α be an S_e strategy. First we define the length agreement function as follows:

Definition 1. *(1)* $\ell(\alpha, s) = \max\{x :$ *for all* $y < x$, $\Phi_e^{A,W}(y)[s] = \Psi_e^{B,W}(y)[s] = K_s \oplus F_s(y)\}$; *(2)* $m(\alpha, s) = \max\{0, \ell(\alpha, t) : t < s$ *and* t *is an* α*-stage*$\}$.

Say that s is α-expansionary if $s = 0$ or $\ell(\alpha, s) > m(\alpha, s)$ and s is an α-stage. Θ is defined at the α-expansionary stages. That is, for a particular n, if $\Theta^W(n)$ is not defined at stage s, and $\ell(\alpha, s) > 2n$, then we define $\Theta^W(n)[s] = K_s(n)$ with use $\theta_s(n) = s$. After $\Theta^W(n)$ is defined, only W's changes below $\varphi_e(n)[s]$ or $\psi_e(n)[s]$ can redefine $\Theta^W(n)$ with a new use.

The trouble is that we can enumerate numbers into A and B, by P and Q-strategies, respectively, and can lift the uses $\varphi_e(n)$ and $\psi_e(n)$ to bigger numbers (bigger than s). Now W may change below these new uses (but above s), and at the next α-expansionary stage, we can see that n enters K, and both $\Phi_e^{A,W}(2n)$ and $\Psi_e^{B,W}(2n)$ converge and equal to $K(n) = 1$. However, since W has not change below s, $\Theta^W(n)$ is kept defined as 0. Thus Θ^W is wrong at n, and we should avoid such a scenario.

We apply a strategy of constructing the noncuppable degrees to get around of this problem. That is, before we enumerate a number into A or B, we first enumerate appropriate numbers into F to force W to change on small numbers, so that the trouble situation we described above will never happen. In the construction of noncuppable degrees, we need to satisfy the following requirements:

$$\Phi^{A,W} = K \oplus F \;\Rightarrow\; \exists\Gamma(\Gamma^A = K).$$

To be consistent with this kind of the requirements, when a P-strategy chooses x as an attacker, at stage s_0 say, to make A not computable, it also chooses z, and at the next expansionary stage *(we measure the length of agreement between* $\Phi^{A,W}$ *and* $K \oplus F$

at each stage, and we say a stage is expansionary, if the length of agreement is bigger than previous ones and also bigger than $2z + 1$), we allow the definition of Γ to be extended. Now suppose we want to put a number x into A, what we do first is to put z into F, and wait for the next expansionary stage, which will provides W-changes on numbers small enough to undefine all $\Gamma(m)$ defined after stage s_0. We only put x into A after W has such changes, that is, after we say the next expansionary stage.

In our argument, we will do almost the same thing. Let β be a P-strategy (for Q-strategies, the same, except that we will put $\delta(k)$ into B), and suppose that $\beta \supset \alpha_n \supseteq \alpha_{n-1} \supseteq \cdots \alpha_1 \supseteq \alpha_0$, where $\alpha_n, \alpha_{n-1}, \cdots, \alpha_1, \alpha_0$ are the S-strategy with priority higher than β. Then, β first defines its "killing point" k and then its attackers $x_0, x_1, \cdots, x_k$, and an auxiliary number $z_0, z_1, \cdots, z_k$. Suppose β does this at stage s_0. Now, for each $i \leq n$, we say that a stage s is α_i-expansionary if the length of agreement between $\Phi_{\alpha_i}^{A,W_{\alpha_i}}, \Psi_{\alpha_i}^{B,W_{\alpha_i}}$ and $K \oplus F$ is greater than $2z_j + 1$ for each j with $0 \leq j \leq k$.

When β finds that $\Phi_\beta^C(x_k)$ converges to 0 for the first time, at stage s_1 say, then β puts x_k into E, $\gamma(k)$ into A immediately, to lift $\gamma(k)$ to a big number, and puts z_k into F, and for each i, wait for the next α_i-expansionary stage. β itself is satisfied, unless β is reset because of changes of K below k (if so, we wait for such a stage s_1 again, but with x_k and z_k replaced with x_{k-1} and z_{k-1} respectively). Now consider those α_i-strategies. Fix i, and assume that s_2 is the next α_i-expansionary stage. Then between stages s_1 and s_2, no small numbers are enumerated into B, and hence we must have a change of the corresponding W_{α_i} on some small numbers. It means that between stages s_1 and s_2, all of the Θ_{α_i} defined by α_i after stage s_0 are undefined, and therefore, Θ can be redefined correctly. βs action is consistent with these α_i strategies.

This completes the basic ideas of proof of Theorem 3. We can now implement the whole construction by a tree argument.

3 Proof of Theorem 4

Given c.e. sets A, B with $A \not\geq_T B$, we will construct c.e. sets C and E, and a partial computable functional Γ, to satisfy the following requirements:

$$G : K = \Gamma^{C,E};$$
$$P_e : C \neq \Phi_e^E;$$
$$Q_e : C \neq \Phi_e^A;$$
$$R_e : B \neq \Phi_e^{A,C};$$

where $e \in \omega$, $\{\Phi_e : e \in \omega\}$ is an effective enumeration of p.c. functionals Φ, and K is a fixed creative set.

Note that the G and P-strategies ensure that C and hence $A \oplus C$ is cuppable, while the Q-strategies ensure that $A \oplus C$ is strictly above A, and the R-strategies ensure that $A \oplus C$ is incomplete.

We have seen in Section 2 how to make the P, Q-strategies consistent with the G, H-strategies respectively.

3.1 The *Q*-Strategies

All the Q-strategies ensure that C is not reducible to A, and a single Q-strategy is exactly the Sacks coding strategy. That is, a single Q-strategy will run (infinitely) many cycles, i, each of which will choose a witness x_i, with the purpose of making $C(x_i) \neq \Phi_e^A(x_i)$, and all these cycles will define a p.c. functional Δ to threaten the assumption that A is incomplete. If a cycle i fails to make $C(x_i) \neq \Phi_e^A(x_i)$, then this cycle will code $K(i)$ into A via Δ. A cycle i works as follows:

1. Choose x_i as a fresh number.
2. Wait for a stage s such that $\Phi_e^A(x_i)[s]$ converges to 0.
3. Define $\Delta^A(i)[s] = K_s(i)$ with use $\delta(i) = \varphi_e(i)[s]$. If A changes below $\delta(i)$ before 5, then go back to 2.
4. Start cycle $i+1$ and wait for $K(i)$ to change.
5. Enumerate x_i into C.
6. Wait for A to change below $\delta(i)$.
7. Define $\Delta^A(i) = K(i) = 1$ with use $\delta(i) = -1$ and start cycle $i+1$. In this case, the A-changes will undefine $\Delta^A(j)$ for each $j \geq i$.

If cycle i waits at 2 forever, then $\Phi_e^A(x_i)$ does not converge to 0 and hence $C(x_i) = 0 \neq \Phi_e^A(x_i)$ and Q_e is satisfied. In this case, cycle does not care whether $\Delta^A(i)$ is defined, since this cycle can satisfy the Q_e requirement directly, in stead of relying on Δ^A.

If cycle i waits at 6 forever, then $\Phi_e^A(x_i)$ does converge to 0, $C(x_i) = 1 \neq 0 = \Phi_e^A(x_i)$ and hence Q_e is again satisfied. In this case, cycle i does not care whether $\Delta^A(i)$ computes $K(i)$ correctly neither.

If cycle i waits at 4 forever or 7 happens, then $\Delta^A(i)$ is defined and equals to $K(i)$. In these two cases, cycle i cannot satisfy Q_e, but succeed in defining $\Delta^A(i) = K(i)$.

Without loss of generality, suppose that no cycle waits at 2 or 6 forever. If every cycle eventually waits at 4 or 7 permanently, then for each i, $\Delta^A(i)$ is defined and equals to $K(i)$, and hence $K = \Delta^A$, and A is complete. A contradiction. Therefore, there are cycles going from 4 back to 2 infinitely often, which makes $\Delta^A(i)$ undefined. However, in this case, $\Phi_e^A(x_i)$ diverges, which shows that $\Phi_e^A(x_i) \neq C(x_i)$, and Q_e is satisfied again.

3.2 The *R*-Strategies

All the R-strategies will ensure that B is not reducible to $A \oplus C$. From this we can see that $A \oplus C$ is incomplete, and hence, the Q and the R strategies will ensure that $A \oplus C$ is strictly between A and K.

A single Q-strategy is simply the Sacks preservation strategy, which also runs (infinitely) many cycles, to define a partial function Θ to threaten $B \not\leq_T A$. Fix i. Cycle i works as follows:

1. Wait for a stage s such that $\Phi_e^{A,C}(i)[s] \downarrow = B_s(i)$.
2. Put a restraint on C to prevent small numbers being enumerated into C, to preserve the computation $\Phi_e^{A,C}(i)$. Define $\Theta^A(i)[s] = B_s(i)$ with use $\delta_s(i) = \varphi_{e,s}(i)$. Start cycle $i+1$.
3. Wait for A to change below $\varphi_{e,s}(i)$ or B to change at i.
4. If A changes first, then go back to 1. In this case, $\Theta^A(i)$ is undefined, and all cycles after i are canceled.

5. If B changes at i first, then we get a temporary disagreement between $\Phi_e^{A,C}(i)$ and $B(i)$. Again, wait for A to change below $\varphi_{e,s}(i)$.
   - 5a. If A never changes, then we will have $\Phi_e^{A,C}(i) = 0 \neq 1 = B(i)$, and R_e is satisfied.
   - 5b. Otherwise, go back to 1. In this case, the A-changes also undefine $\Theta^A(j)$ for all $j \geq i$.

If cycle i goes back from 4 or 5b to 1 infinitely often, then $\Theta^A(i)$ is not defined. However, in this case, $\Phi_e^{A,C}(i)$ diverges and hence, $\Phi_e^{A,C}(i) \neq B(i)$. If cycle i waits at 1 or 5a forever, then $\Phi_e^{A,C}(i) \neq B_s(i)$ is again true. In any case, R_e is satisfied. If cycle i waits at 3 forever from some stage on, then $\Theta^A(i)$ is defined and equals to $B(i)$.

Because $B \not\leq_T A$, it cannot be true that every cycle would wait at 3 forever. Suppose i is the least such cycle. Then as discussed above, R_e is satisfied since $\Phi_e^{A,C}(i) \neq B(i)$.

This completes the description of the basic strategies to prove Theorem 4. The whole construction can proceed on a priority tree.

References

1. Ambos-Spies, K., Jockusch Jr., C.G., Shore, R.A., Soare, R.I.: An algebraic decomposition of the recursively enumerable degrees and the coincidence of several degree classes with the promptly simple degrees. Trans. Amer. Math. Soc. 281, 109–128 (1984)
2. Cooper, S.B.: Minimal pairs and high recursively enumerable degrees. J. Symbolic Logic 39, 655–660 (1974)
3. Cooper, S.B.: On a theorem of C. E. M. Yates. Handwritten notes (1974)
4. Cooper, S.B.: Computability Theory, Chapman & Hall/CRC Mathematics, Boca Raton, FL, New York, London (2004)
5. Downey, R., Lempp, S.: There is no plus-capping degree. Arch. Math. Logic 33, 109–119 (1994)
6. Fejer, P.A., Soare, R.I.: The plus-cupping theorem for the recursively enumerable degrees. In: Logic Year 1979–80: University of Connecticut, pp. 49–62 (1981)
7. Lachlan, A.H.: Lower bounds for pairs of recursively enumerable degrees. Proc. London Math. Soc. 16, 537–569 (1966)
8. Lachlan, A.H.: Bounding minimal pairs. J. Symb. Logic 44, 626–642 (1979)
9. Li, A.: On a conjecture of Lempp. Arch. Math. Logic 39, 281–309 (2000)
10. Li, A., Wu, G., Yang, Y.: On the quotient structure of computably enumerable degrees modulo the noncuppable ideal. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 731–736. Springer, Heidelberg (2006)
11. Li, A., Wu, G., Yang, Y.: Embed the diamond lattice into R/NCup preserving 0 and 1. In preparation
12. Miller, D.: High recursively enumerable degrees and the anticupping property. In: Logic Year 1979–80: University of Connecticut, pp. 230–245 (1981)
13. Sacks, G.E.: On the degrees less than 0̇. Ann. of Math. 77, 211–231 (1963)
14. Sacks, G.E.: The recursively enumerable degrees are dense. Ann. of Math. 80, 300–312 (1964)
15. Schwarz, S.: The quotient semilattice of the recursively enumerable degrees modulo the cappable degrees. Trans. Amer. Math. Soc. 283, 315–328 (1984)
16. Shoenfield, J.R.: Applications of model theory to degrees of unsolvability. In: Addison, J.W., Henkin, L., Tarski, A. (eds.) The Theory of Models, Proceedings of the 1963 International Symposium at Berkeley, Studies in Logic and the Foundations of Mathematics, pp. 359–363. Holland Publishing, Amsterdam (1965)

17. Slaman, T.: Questions in recursion theory. In: Cooper, S.B., Slaman, T.A., Wainer, S.S. (eds.) Computability, enumerability, unsolvability. Directions in recursion theory. London Mathematical Society Lecture Note Series, vol. 224, pp. 333–347. Cambridge University Press, Cambridge (1996)
18. Soare, R.I.: Recursively Enumerable Sets and Degrees. Perspective in Mathematical Logic, Springer-Verlag, Berlin, Heidelberg, New York
19. Sui, Y., Zhang, Z.: The cupping theorem in R/ M. J. Symbolic Logic 64, 643–650 (1999)
20. Yates, C.E.M.: A minimal pair of recursively enumerable degrees. J. Symbolic Logic 31, 159–168 (1966)
21. Yates, C.E.M.: On the degrees of index sets. Trans. Amer. Math. Soc. 121, 309–328 (1966)
22. Yi, X.: Extension of embeddings on the recursively enumerable degrees modulo the capable degrees. In: Computability, enumerability, unsolvability, Directions in recursion theory (eds. Cooper, Slaman, Wainer), London Mathematical Society Lecture Note Series 224, pp. 313–331

Constructive Dimension and Weak Truth-Table Degrees

Laurent Bienvenu[1], David Doty[2], and Frank Stephan[3,⋆]

[1] Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence, 39 rue Joliot-Curie, 13453 Marseille Cedex 13, France
laurent.bienvenu@lif.univ-mrs.fr

[2] Department of Computer Science, Iowa State University, Ames, IA 50011, USA
ddoty@iastate.edu

[3] School of Computing and Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543, Republic of Singapore
fstephan@comp.nus.edu.sng

Abstract. This paper examines the constructive Hausdorff and packing dimensions of weak truth-table degrees. The main result is that every infinite sequence S with constructive Hausdorff dimension $\dim_{\mathrm{H}}(S)$ and constructive packing dimension $\dim_{\mathrm{P}}(S)$ is weak truth-table equivalent to a sequence R with $\dim_{\mathrm{H}}(R) \geq \dim_{\mathrm{H}}(S)/\dim_{\mathrm{P}}(S) - \epsilon$, for arbitrary $\epsilon > 0$. Furthermore, if $\dim_{\mathrm{P}}(S) > 0$, then $\dim_{\mathrm{P}}(R) \geq 1-\epsilon$. The reduction thus serves as a *randomness extractor* that increases the algorithmic randomness of S, as measured by constructive dimension.

A number of applications of this result shed new light on the constructive dimensions of wtt degrees (and, by extension, Turing degrees). A lower bound of $\dim_{\mathrm{H}}(S)/\dim_{\mathrm{P}}(S)$ is shown to hold for the wtt degree of any sequence S. A new proof is given of a previously-known zero-one law for the constructive packing dimension of wtt degrees. It is also shown that, for any *regular* sequence S (that is, $\dim_{\mathrm{H}}(S) = \dim_{\mathrm{P}}(S)$) such that $\dim_{\mathrm{H}}(S) > 0$, the wtt degree of S has constructive Hausdorff and packing dimension equal to 1.

Finally, it is shown that no single Turing reduction can be a *universal* constructive Hausdorff dimension extractor.

Keywords: constructive dimension, weak truth-table, extractor, degree, randomness.

1 Introduction

Hausdorff [5] initiated the study of dimension as a general framework to define the size of subsets of metric spaces. Recently this framework had been effectivized; Lutz [9] gives an overview of this historical development. Furthermore, Lutz [8, Section 6] reviews early results that anticipated the effectivization of Hausdorff dimension. *Constructive Hausdorff dimension* was defined by Lutz [8] to study effective dimension at the level of computability theory. Intuitively,

⋆ Corresponding author.

given an infinite binary sequence S – interpreted as a language or decision problem – the constructive Hausdorff dimension $\dim_{\mathrm{H}}(S)$ of S is a real number in the interval [0,1] indicating the density of algorithmic randomness of the sequence. The constructive Hausdorff dimension of a class $\mathcal{C}$ of sequences is the supremum of the dimensions of individual sequences in $\mathcal{C}$. For many classes $\mathcal{C}$ of interest in computability theory, the problem of determining the constructive Hausdorff dimension of $\mathcal{C}$ remains open.

Reimann [14] investigated in particular whether there are degrees of fractional constructive Hausdorff dimension. Stated in terms of individual sequences, Reimann asked which reducibilities (such as Turing, many-one, weak truth-table, etc.) are capable of increasing the constructive Hausdorff dimension of a sequence. We call such a reduction a *dimension extractor*, since its purpose bears a resemblance to that of the *randomness extractors* of computational complexity [18], which are algorithms that turn a source of weak randomness (a probabilistic source with low entropy) into a source of strong randomness (a source with high entropy). Viewing a sequence with positive, but still fractional, constructive Hausdorff dimension as a weak source of randomness, Reimann essentially asked whether such randomness can be extracted via a reduction to create a sequence with dimension closer to 1. If such extraction is *not* possible for some sequence S, this indicates that the degree of S under the reduction has fractional dimension.

A number of negative results for dimension extractors are known. Reimann and Terwijn [14, Theorem 3.10] proved that there are many-one and bounded truth-table degrees with constructive Hausdorff dimension strictly between 0 and 1. Later Reimann and Slaman [15] extended this result to truth-table degrees. Stephan [20] showed that there is a relativized world in which there exists a wtt degree of constructive Hausdorff dimension between $\frac{1}{4}$ and $\frac{1}{2}$. Furthermore, Nies and Reimann [11] obtained a non-relativized variant of this result and constructed, for each rational α between 0 and 1, a wtt degree of constructive Hausdorff dimension α.

Doty [3] attempted positive results and considered the interaction between constructive Hausdorff dimension and *constructive packing dimension* [1], a dual quantity that is a constructive effectivization of classical packing dimension [21,22], another widely-studied fractal dimension. The constructive packing dimension $\dim_{\mathrm{P}}(S)$ of a sequence S always obeys

$$0 \leq \dim_{\mathrm{H}}(S) \leq \dim_{\mathrm{P}}(S) \leq 1,$$

with each inequality tight in the strong sense that there are sequences S in which $\dim_{\mathrm{H}}(S)$ and $\dim_{\mathrm{P}}(S)$ may take on any values obeying the stated constraint. Doty showed that every sequence S with $\dim_{\mathrm{H}}(S) > 0$ is Turing equivalent to a sequence R with $\dim_{\mathrm{P}}(R) \geq 1 - \epsilon$, for arbitrary $\epsilon > 0$. This implies that the constructive packing dimension of the Turing degree of any sequence S with $\dim_{\mathrm{H}}(S) > 0$ is equal to 1. Unfortunately, since $\dim_{\mathrm{H}}(R) \leq \dim_{\mathrm{P}}(R)$, this Turing reduction constitutes a weaker example of a dimension extractor than that sought by Reimann and it tells us nothing of the constructive dimensions of arbitrary Turing degrees.

We obtain in the current paper stronger positive results for constructive dimension extractors. Our main result, in section 2, is that, given any infinite sequence S and $\epsilon > 0$, there exists $R \equiv_{\mathrm{wtt}} S$ such that $\dim_{\mathrm{H}}(R) \geq \frac{\dim_{\mathrm{H}}(S)}{\dim_{\mathrm{P}}(S)} - \epsilon$ and, if $\dim_{\mathrm{P}}(S) > 0$, then $\dim_{\mathrm{P}}(R) \geq 1 - \epsilon$. This has immediate consequences for the dimensions of weak truth-table degrees:

– Given any sequence S, $\dim_{\mathrm{H}}(\deg_{\mathrm{wtt}}(S)) \geq \frac{\dim_{\mathrm{H}}(S)}{\dim_{\mathrm{P}}(S)}$.
– If $\dim_{\mathrm{P}}(S) > 0$, then $\dim_{\mathrm{P}}(\deg_{\mathrm{wtt}}(S)) = 1$, implying that *every* wtt degree has constructive packing dimension 0 or 1.
– Given any *regular* sequence S such that $\dim_{\mathrm{H}}(S) > 0$, $\dim_{\mathrm{H}}(\deg_{\mathrm{wtt}}(S)) = 1$, where a sequence S is regular if it satisfies $\dim_{\mathrm{H}}(S) = \dim_{\mathrm{P}}(S)$.

In section 3, we use Theorem 2.1 to show that, for every $\alpha > 0$, there is no *universal* Turing reduction that is guaranteed to extract dimension from all sequences of dimension at least α.

Before going into the details of the results, we introduce the concepts and notations formally.

Notation. We refer the reader to the textbooks of Li and Vitányi [6] for an introduction to Kolmogorov complexity and algorithmic information theory and of Odifreddi [13] and Soare [19] for an introduction to computability theory. Although we follow mainly the notation in these books, we nevertheless want to remind the reader on the following definitions, either for the readers' convenience or because we had to choose between several common ways of denoting the corresponding mathematical objects.

All logarithms are base 2. $\mathbb{N}$ denotes the set $\{0, 1, 2, 3, \ldots\}$ of the natural numbers including 0. $\{0,1\}^*$ denotes the set of all finite, binary *strings*. For all $x \in \{0,1\}^*$, $|x|$ denotes the *length* of x. λ denotes the empty string. $\mathbf{C} = \{0,1\}^\infty$ denotes the *Cantor space*, the set of all infinite, binary *sequences*. For $x \in \{0,1\}^*$ and $y \in \{0,1\}^* \cup \mathbf{C}$, xy denotes the concatenation of x and y, $x \sqsubseteq y$ denotes that x is a *prefix* of y (that is, there exists $u \in \{0,1\}^* \cup \mathbf{C}$ such that $xu = y$) and $x \sqsubset y$ denotes that $x \sqsubseteq y$ and $x \neq y$. For $S \in \{0,1\}^* \cup \mathbf{C}$ and $i, j \in \mathbb{N}$, $S[i]$ denotes the i^{th} bit of S, with $S[0]$ being the leftmost bit, $S[i\,..\,j]$ denotes the substring consisting of the i^{th} through j^{th} bits of S (inclusive), with $S[i\,..\,j] = \lambda$ if $i > j$.

Reductions and Compression. Let M be a Turing machine and $S \in \mathbf{C}$. We say M *computes* S if M on input $n \in \mathbb{N}$ (written $M(n)$), outputs the string $S[0\,..\,n-1]$. We define an *oracle Turing machine* to be a Turing machine M that can make constant-time queries to an oracle sequence and we let OTM denote the set of all oracle Turing machines. For $R \in \mathbf{C}$, we say M operates *with oracle* R if, whenever M makes a query to index $n \in \mathbb{N}$, the bit $R[n]$ is returned. We write M^R to denote the oracle Turing machine M with oracle R.

Let $S, R \in \mathbf{C}$ and $M \in$ OTM. We say S *is Turing reducible to* R *via* M and we write $S \leq_{\mathrm{T}} R$ *via* M, if M^R computes S (that is, if $M^R(n) = S[0\,..\,n-1]$ for all $n \in \mathbb{N}$). In this case, write $R = M(S)$. We say S *is Turing reducible to* R and we write $S \leq_{\mathrm{T}} R$, if there exists $M \in$ OTM such that $S \leq_{\mathrm{T}} R$ via M. We say S is *Turing equivalent* to R, and we write $S \equiv_{\mathrm{T}} R$, if $S \leq_{\mathrm{T}} R$ and $R \leq_{\mathrm{T}} S$. The *Turing lower span* of S is $\mathrm{span}_{\mathrm{T}}(S) = \{ R \in \mathbf{C} \mid R \leq_{\mathrm{T}} S \}$ and the *Turing degree* of S is $\deg_{\mathrm{T}}(S) = \{ R \in \mathbf{C} \mid R \equiv_{\mathrm{T}} S \}$.

Let $S, R \in \mathbf{C}$ and $M \in \mathrm{OTM}$ such that $S \leq_{\mathrm{T}} R$ via M. Let the notion $\#(M^R, S[0\,..\,n-1])$ denote the *query usage of* M^R *on* $S[0\,..\,n-1]$, the number of bits of R queried by M when computing the string $S[0\,..\,n-1]$.[1] We say S *is weak truth-table (wtt) reducible to* R *via* M and we write $S \leq_{\mathrm{wtt}} R$ *via* M, if $S \leq_{\mathrm{T}} R$ via M and there is a computable function $q : \mathbb{N} \to \mathbb{N}$ such that, for all $n \in \mathbb{N}$, $\#(M^R, S[0\,..\,n-1]) \leq q(n)$. Define $S \leq_{\mathrm{wtt}} R$, $S \equiv_{\mathrm{wtt}} R$, $\mathrm{span}_{\mathrm{wtt}}(S)$ and $\deg_{\mathrm{wtt}}(S)$ analogously to their counterparts for Turing reductions. Define

$$\rho^-_M(S,R) = \liminf_{n\to\infty} \frac{\#(M^R, S[0\,..\,n-1])}{n},$$
$$\rho^+_M(S,R) = \limsup_{n\to\infty} \frac{\#(M^R, S[0\,..\,n-1])}{n}.$$

Viewing R as a compressed version of S, $\rho^-_M(S,R)$ and $\rho^+_M(S,R)$ are respectively the best- and worst-case compression ratios as M decompresses R into S. Note that $0 \leq \rho^-_M(S,R) \leq \rho^+_M(S,R) \leq \infty$.

The following lemma is useful when one wants to compose two reductions:

Lemma 1.1. [2] *Let* $S, S', S'' \in \mathbf{C}$ *and* $M_1, M_2 \in \mathrm{OTM}$ *such that* $S' \leq_{\mathrm{T}} S$ *via* M_1 *and* $S'' \leq_{\mathrm{T}} S'$ *via* M_2. *There exists* $M \in \mathrm{OTM}$ *such that* $S'' \leq_{\mathrm{T}} S$ *via* M *and:*

$$\rho^+_M(S'',S) \leq \rho^+_{M_2}(S'',S')\rho^+_{M_1}(S',S).$$
$$\rho^-_M(S'',S) \leq \rho^-_{M_2}(S'',S')\rho^+_{M_1}(S',S).$$
$$\rho^-_M(S'',S) \leq \rho^+_{M_2}(S'',S')\rho^-_{M_1}(S',S).$$

(The last bound is not explicitly stated in [2], but it holds for the same reason as the second one).

For $S \in \mathbf{C}$, the *lower and upper Turing compression ratios of* S are respectively defined as

$$\rho^-(S) = \min_{\substack{R\in\mathbf{C}\\ M\in\mathrm{OTM}}} \left\{ \rho^-_M(S,R) \;\middle|\; S \leq_{\mathrm{T}} R \text{ via } M \right\},$$
$$\rho^+(S) = \min_{\substack{R\in\mathbf{C}\\ M\in\mathrm{OTM}}} \left\{ \rho^+_M(S,R) \;\middle|\; S \leq_{\mathrm{T}} R \text{ via } M \right\}.$$

Doty [2] showed that the above minima exist. Note that $0 \leq \rho^-(S) \leq \rho^+(S) \leq 1$.

Constructive Dimension. Lutz [8] gives an introduction to the theory of constructive dimension. We use Mayordomo's characterization [10] of the constructive

[1] If we instead define $\#(M^R, S[0\,..\,n-1])$ to be the index of the rightmost bit of R queried by M when computing $S[0\,..\,n-1]$, all results of the present paper still hold.

dimensions of sequences. For all $S \in \mathbf{C}$, the *constructive Hausdorff dimension* and the *constructive packing dimension* of S are respectively defined as

$$\dim_{\mathrm{H}}(S) = \liminf_{n\to\infty} \frac{\mathrm{C}(S[0\,..\,n-1])}{n} \text{ and } \dim_{\mathrm{P}}(S) = \limsup_{n\to\infty} \frac{\mathrm{C}(S[0\,..\,n-1])}{n},$$

where $\mathrm{C}(w)$ denotes the *Kolmogorov complexity* of $w \in \{0,1\}^*$ (see [6]). If $\dim_{\mathrm{H}}(S) = \dim_{\mathrm{P}}(S)$, we say S is a *regular* sequence. Doty [2] showed that, for all $S \in \mathbf{C}$, $\rho^-(S) = \dim_{\mathrm{H}}(S)$ and $\rho^+(S) = \dim_{\mathrm{P}}(S)$.

For all $X \subseteq \mathbf{C}$, the *constructive Hausdorff dimension* and the *constructive packing dimension* of X are respectively defined as

$$\dim_{\mathrm{H}}(X) = \sup_{S\in X} \dim_{\mathrm{H}}(S) \text{ and } \dim_{\mathrm{P}}(X) = \sup_{S\in X} \dim_{\mathrm{P}}(S).$$

2 Constructive Dimension Extractors

Nies and Reimann [11] showed that wtt reductions cannot always extract constructive dimension.

Theorem 2.1 (Nies and Reimann [11]). *For every rational number α with $0 < \alpha < 1$, there exists a sequence $S \in \mathbf{C}$ such that, for all wtt reductions M, $\dim_{\mathrm{H}}(M(S)) \le \dim_{\mathrm{H}}(S) = \alpha$.*

Ryabko [16,17] discovered the next theorem.

Theorem 2.2 (Ryabko [16,17]). *For all $S \in \mathbf{C}$ and $\delta > 0$, there exists $R \in \mathbf{C}$ and $N_d \in \mathrm{OTM}$ such that*

1. *$S \le_{\mathrm{T}} R$ via N_d and $R \le_{\mathrm{T}} S$.*
2. *$\rho^-_{N_d}(S,R) \le \dim_{\mathrm{H}}(S) + \delta$.*

The following theorem was shown in [2].

Theorem 2.3 (Doty [2]). *There is an oracle Turing machine M_d such that, for all $S \in \mathbf{C}$, there exists $R \in \mathbf{C}$ such that*

1. *$S \le_{\mathrm{wtt}} R$ via M_d.*
2. *$\rho^-_{M_d}(S,R) = \dim_{\mathrm{H}}(S)$.*
3. *$\rho^+_{M_d}(S,R) = \dim_{\mathrm{P}}(S)$.*

The following theorem, which is similar to Ryabko's Theorem 2.2, shows that the decoding machine M_d of Theorem 2.3 can also be reversed if the compression requirements are weakened.

Theorem 2.4. *Let M_d be the oracle Turing machine from Theorem 2.3. For all $\delta > 0$, there is an oracle Turing machine M_e such that, for all $S \in \mathbf{C}$, there is a sequence $R' \in \mathbf{C}$ such that*

1. *$S \le_{\mathrm{wtt}} R'$ via M_d and $R' \le_{\mathrm{wtt}} S$ via M_e.*
2. *$\rho^+_{M_d}(S,R') \le \dim_{\mathrm{P}}(S) + \delta$.*

Proof. Let $S \in \mathbf{C}$ and choose R for S as in Theorem 2.3. Let $\delta > 0$ and let $D \in (\dim_{\mathrm{P}}(S), \dim_{\mathrm{P}}(S) + \delta)$ be rational. By Theorem 2.3, there exists $n_0 \in \mathbb{N}$ such that, for all $n \geq n_0$, $\#(M_d^R, S[0\,..\,n-1]) < Dn$.

M_e will make use of the oracle Turing machine M_d. The proof of Theorem 2.3 in [2] shows that M_d has the following useful properties. First, write $S = s_1 s_2 s_3 \ldots$ and $R = r_1 r_2 r_3 \ldots$, where each $s_i, r_i \in \{0,1\}^*$ are blocks such that $|s_i| = i$ and $|r_i| \leq |s_i| + o(|s_i|)$.

– M_d computes S from R in stages, where it outputs the block s_i on the i^{th} stage.
– Assuming that M_d has already computed $s_1 \ldots s_i$, M_d uses only the block r_{i+1} and the prefix $s_1 \ldots s_i$ to compute s_{i+1}.

Because of these properties, we can use M_d to search for a sequence R' that satisfies requirements 1 and 2 in the statement of Theorem 2.4. By Theorem 2.3, R satisfies these requirements, so such an R' will exist. By the above two properties of M_d, if we find a string $r' = r'_1 \ldots r'_i$ that satisfies requirements 1 and 2 (in the sense described below), we will always be able to find an extension $r'' = r'_{i+1} \ldots r'_j$ (for some $j > i$) such that $r'r''$ continues to satisfy the requirements. It will not matter if $r' \not\sqsubset R$, since M_d does not use the portion of R coming before block r_{i+1} to compute s_{i+1}. In other words, to reverse the computation of $M_d^{R'}$ and compute R' from S, we don't need to find the R from Theorem 2.3; we need only to find an R' that is "close enough".

Define the oracle Turing machine M_e with oracle $S \in \mathbf{C}$ as follows. Let $i \in \mathbb{N}$ and assume inductively that the prefix $r' = r'_1 \ldots r'_i \sqsubset R'$ has been computed, so that, letting $|s_1 \ldots s_i| = n$,

(a) $M_d^{r'}(n)$ outputs $S[0\,..\,n-1]$,
(b) for all k with $n_0 \leq k \leq n$, $\#(M_d^{r'}, S[0\,..\,k-1]) \leq Dk$.

Let N be the smallest integer greater than 2^n such that $S[0\,..\,N-1] = s_1 \ldots s_{i'}$, for some $i' \in \mathbb{N}$. M_e^S searches all strings $r'' \in \{0,1\}^N$ until it finds one that satisfies

(a) $M_d^{r'r''}(N)$ outputs $S[0\,..\,N-1]$,
(b) for all k with $n_0 \leq k \leq N$, $\#(M_d^{r'r''}, S[0\,..\,k-1]) \leq Dk$.

M_e^S then outputs r'' and saves it for the computation of the next extension of R'. By the existence of R from Theorem 2.3 and a simple induction on the stages of computation that M_e performs and the fact that N is asymptotically larger than n, M_e^S will always be able to find such an r''. Therefore, in the output sequence R', for all but finitely many N, requirement (b) will be satisfied. Therefore the sequence R' will satisfy the two requirements of Theorem 2.4.

Finally, for any n, $M_e(n)$ makes no more than 2^{2n} queries to S and therefore M_e computes a wtt reduction. □

The following theorem is the main result of this paper. It states that constructive packing dimension can be almost optimally extracted from a sequence of positive packing dimension, while at the same time, constructive Hausdorff dimension is *partially* extracted from this sequence, if it has positive Hausdorff dimension and packing dimension less than 1. The machine M_e from Theorem 2.4 serves as the extractor. Intuitively, this works because M_e compresses the sequence S into the sequence R. Since R is a compressed representation of S, R must itself be more incompressible than S. However, because dimension measures the compressibility of a sequence, this means that the constructive dimensions R are greater than those of S.

Theorem 2.5. *For all* $\epsilon > 0$ *and* $S \in \mathbf{C}$ *such that* $\dim_{\mathrm{P}}(S) > 0$, *there exists* $R \equiv_{\mathrm{wtt}} S$ *such that* $\dim_{\mathrm{P}}(R) \geq 1 - \epsilon$ *and* $\dim_{\mathrm{H}}(R) \geq \frac{\dim_{\mathrm{H}}(S)}{\dim_{\mathrm{P}}(S)} - \epsilon$.

Proof. Let $\epsilon > 0$ and $S \in \mathbf{C}$ such that $\dim_{\mathrm{P}}(S) > 0$. Let $\delta > 0$ and R', M_d be as in Theorem 2.4. Let $R'' \in \mathbf{C}$ and $M \in \mathrm{OTM}$ such that $R' \leq_{\mathrm{T}} R''$ via M, $\rho^-_M(R', R'') = \dim_{\mathrm{H}}(R')$ and $\rho^+_M(R', R'') = \dim_{\mathrm{P}}(R')$ (the existence of M and R'' is asserted by Theorem 2.3). By Lemma 1.1, we have

$$\rho^+(S) \leq \rho^+_{M_d}(S, R')\rho^+_M(R', R''),$$

which, by construction of R' and R'' implies $\rho^+(S) \leq (\dim_{\mathrm{P}}(S) + \delta) \dim_{\mathrm{P}}(R')$. Since $\rho^+(S) = \dim_{\mathrm{P}}(S)$,

$$\dim_{\mathrm{P}}(R') \geq \frac{\dim_{\mathrm{P}}(S)}{\dim_{\mathrm{P}}(S) + \delta}.$$

Moreover (by Lemma 1.1 again), $\rho^-(S) \leq \rho^+_{M_d}(S, R')\rho^-_M(R', R'')$, which, by construction of R' and R'', implies $\rho^-(S) \leq (\dim_{\mathrm{P}}(S) + \delta) \dim_{\mathrm{H}}(R')$. Since $\rho^-(S) = \dim_{\mathrm{H}}(S)$,

$$\dim_{\mathrm{H}}(R') \geq \frac{\dim_{\mathrm{H}}(S)}{\dim_{\mathrm{P}}(S) + \delta}.$$

Taking δ small enough, we get by the above inequalities: $\dim_{\mathrm{P}}(R) \geq 1 - \epsilon$ and $\dim_{\mathrm{H}}(R) \geq \frac{\dim_{\mathrm{H}}(S)}{\dim_{\mathrm{P}}(S)} - \epsilon$. □

Theorem 2.5 has a number of applications, stated in the following corollaries, which shed light on the constructive dimensions of sequences, spans and degrees.

Corollary 2.6. *Let* $S \in \mathbf{C}$ *and assume that* $\dim_{\mathrm{H}}(S) > 0$. *Then* $\dim_{\mathrm{H}}(\deg_{\mathrm{T}}(S))$, $\dim_{\mathrm{H}}(\deg_{\mathrm{wtt}}(S))$, $\dim_{\mathrm{H}}(\mathrm{span}_{\mathrm{T}}(S))$ *and* $\dim_{\mathrm{H}}(\mathrm{span}_{\mathrm{wtt}}(S))$ *are all at least* $\frac{\dim_{\mathrm{H}}(S)}{\dim_{\mathrm{P}}(S)}$.

We obtain a zero-one law for the constructive packing dimension of Turing and weak truth-table lower spans and degrees.

Corollary 2.7. *For all* $S \in \mathbf{C}$, *the dimensions* $\dim_{\mathrm{P}}(\deg_{\mathrm{T}}(S))$, $\dim_{\mathrm{P}}(\mathrm{span}_{\mathrm{T}}(S))$, $\dim_{\mathrm{P}}(\deg_{\mathrm{wtt}}(S))$ *and* $\dim_{\mathrm{P}}(\mathrm{span}_{\mathrm{wtt}}(S))$ *are each either 0 or 1.*

Therefore Theorem 2.1, establishing the existence of wtt degrees of fractional constructive Hausdorff dimension, does not extend to constructive packing dimension. Because of Theorem 2.1, we must settle for more conditional results for constructive Hausdorff dimension. We focus attention on regular sequences.

Corollary 2.8. *For all $\epsilon > 0$ and all regular $S \in \mathbf{C}$ such that $\dim_{\mathrm{H}}(S) > 0$, there exists $R \equiv_{\mathrm{wtt}} S$ such that $\dim_{\mathrm{H}}(R) \geq 1 - \epsilon$.*

Corollary 2.9. *For all regular $S \in \mathbf{C}$ such that $\dim_{\mathrm{H}}(S) > 0$,*

$$\begin{array}{rcccl} \dim_{\mathrm{H}}(\mathrm{span}_{\mathrm{wtt}}(S)) &=& \dim_{\mathrm{H}}(\deg_{\mathrm{wtt}}(S)) &=& \dim_{\mathrm{H}}(\mathrm{span}_{\mathrm{T}}(S)) = \\ \dim_{\mathrm{H}}(\deg_{\mathrm{T}}(S)) &=& \dim_{\mathrm{P}}(\mathrm{span}_{\mathrm{wtt}}(S)) &=& \dim_{\mathrm{P}}(\deg_{\mathrm{wtt}}(S)) = \\ \dim_{\mathrm{P}}(\mathrm{span}_{\mathrm{T}}(S)) &=& \dim_{\mathrm{P}}(\deg_{\mathrm{T}}(S)) &=& 1. \end{array}$$

It remains open whether every Turing lower span or degree of positive constructive Hausdorff dimension contains a regular sequence of positive constructive Hausdorff dimension. If so, this would imply a zero-one law for constructive Hausdorff dimension similar to Corollary 2.7.

We note that the zero-one law for the constructive packing dimension of Turing and wtt lower spans and degrees also follows from the following theorem due to Fortnow, Hitchcock, Pavan, Vinodchandran and Wang [4], giving a polynomial-time extractor for constructive packing dimension. For $R, S \in \mathbf{C}$, write $R \leq^{\mathrm{p}}_{\mathrm{T}} S$ if $R \leq_{\mathrm{T}} S$ via an OTM that, on input n, runs in time polynomial in n, and similarly for $\equiv^{\mathrm{p}}_{\mathrm{T}}$.

Theorem 2.10 ([4]). *For all $\epsilon > 0$ and all $S \in \mathbf{C}$ such that $\dim_{\mathrm{P}}(S) > 0$, there exists $R \equiv^{\mathrm{p}}_{\mathrm{T}} S$ such that $\dim_{\mathrm{P}}(R) \geq 1 - \epsilon$.*

In fact, Theorem 2.10 holds for any resource-bounded packing dimension [7] defined by Turing machines allowed at least polynomial space, which includes constructive packing dimension as a special case, thus proving a spectrum of zero-one packing dimension laws for various dimensions above polynomial space and degrees and lower spans that are at least polynomial-time computable.

3 Nonexistence of Universal Extractors

The wtt reduction in the proof of Theorem 2.5 is uniform in the sense that, for all $\epsilon > 0$, there is a *single* wtt reduction M, universal for ϵ and all sequences S, such that $\dim_{\mathrm{H}}(M(S)) \geq \dim_{\mathrm{H}}(S)/\dim_{\mathrm{P}}(S) - \epsilon$.

While it remains open whether Turing reductions can extract constructive Hausdorff dimension, we can show that there is no *universal* Turing reduction that is guaranteed to increase – to a fixed amount – the dimension of all sequences of sufficiently large dimension.

Theorem 3.1. *For every Turing reduction M and all reals α, β with $0 < \alpha < \beta < 1$, there exists $S \in \mathbf{C}$ with $\dim_{\mathrm{H}}(S) \geq \alpha$ such that $M(S)$ does not exist or $\dim_{\mathrm{H}}(M(S)) < \beta$.*

Proof. For this proof, it will be convenient to say that $R \leq_{\mathrm{T}} S$ via M if $M^S(n)$ outputs $R[n]$, rather than $R[0 \,..\, n-1]$, bearing in mind that both definitions of a Turing reduction are equivalent.

Suppose for the sake of contradiction that there exist reals α, β with $0 < \alpha < \beta < 1$ and a Turing reduction M such that, for all $S \in \mathbf{C}$ satisfying $\dim_{\mathrm{H}}(S) \geq \alpha$, then $\dim_{\mathrm{H}}(R) \geq \beta$, where $R = M(S)$. Fix rationals α', γ such that $\alpha < \alpha' < \gamma < \beta$. We will convert M into a truth-table reduction N (a reduction that halts on all oracles, which is also a wtt reduction) that guarantees the slightly weaker condition that if $\dim_{\mathrm{H}}(S) > \alpha'$, then $\dim_{\mathrm{H}}(N(S)) \geq \beta$. Then for any $S \in \mathbf{C}$ such that $\dim_{\mathrm{H}}(S) = \gamma > \alpha'$, it follows that $\dim_{\mathrm{H}}(N(S)) \geq \beta > \gamma = \dim_{\mathrm{H}}(S)$, which contradicts Theorem 2.1.

On input $n \in \mathbb{N}$ and with oracle sequence S, $N^S(n)$ simulates $M^S(n)$. In parallel, for all integers $m > n$, N searches for a program of length at most $\alpha' m$ computing $S[0\,..\,m-1]$. If N finds such a program before the simulation of $M^S(n)$ terminates, then N outputs 0. If instead the simulation of $M^S(n)$ halts before such a short program is found, then N outputs $R[n]$, the output bit of $M^S(n)$.

If $\dim_{\mathrm{H}}(S) < \alpha'$, then for infinitely many $m \in \mathbb{N}$, $\mathrm{C}(S[0\,..\,m-1]) \leq \alpha' m$. Therefore N^S halts, although the output sequence $N(S)$ may contain a lot of 0's, which is acceptable because we do not care what N outputs if $\dim_{\mathrm{H}}(S) < \alpha'$.

If $\dim_{\mathrm{H}}(S) \geq \alpha'$, then M^S is guaranteed to halt and to compute R such that $\dim_{\mathrm{H}}(R) \geq \beta$. Therefore N^S halts. If $\dim_{\mathrm{H}}(S) = \alpha'$, then once again, we do not care what N outputs. If $\dim_{\mathrm{H}}(S) > \alpha'$, then only finitely many m satisfy $\mathrm{C}(S[0\,..\,m-1]) \leq \alpha' m$. Therefore the parallel search for short programs will never succeed once N begins checking only prefixes of S of sufficiently large length. This means that from that point on, N will simulate M exactly, computing a sequence R' that is a finite variation of R. Since dimension is unchanged under finite variations, $\dim_{\mathrm{H}}(R') = \dim_{\mathrm{H}}(R) \geq \beta$. □

Theorem 3.1 tells us that, contrary to the proofs of Theorems 2.4 and 2.5, any extractor construction for Turing reductions must make use of some property of the sequence beyond a simple bound on its dimension.

Acknowledgments. We thank Joe Miller for assistance with the proof of Theorem 2.4, as well as John Hitchcock, Jan Reimann and André Nies for their insightful comments. We also thank the American Institute of Mathematics which generously invited us to the Workshop on Effective Randomness; this paper is a result of a workgroup discussing open questions during this workshop. Besides the American Institute of Mathematics, we would also like to thank the organizers Denis Hirschfeldt and Joe Miller of the workshop as well as the participants who discussed this research topic with us.

References

1. Athreya, K., Hitchcock, J., Lutz, J.H., Mayordomo, E.: Effective strong dimension, algorithmic information and computational complexity. SIAM Journal on Computing (To appear)
2. Doty, D.: Every sequence is decompressible from a random one. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 153–162. Springer, Heidelberg (2006)

3. Doty, D.: Dimension extractors and optimal decompression. Theory of Computing Systems. Special issue of selected papers from Computability in Europe 2006 (to appear)
4. Fortnow, L., Hitchcock, J.M., Pavan Aduri, N., Vinodchandran, V., Wang, F.: Extracting Kolmogorov complexity with applications to dimension zero-one laws. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 335–345. Springer, Heidelberg (2006)
5. Hausdorff, F.: Dimension und äusseres Mass. Mathematische Annalen 79, 157–179 (1919)
6. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and its Applications, 2nd edn. Springer, Heidelberg (1997)
7. Lutz, J.H.: Dimension in complexity classes. SIAM Journal on Computing 32, 1236–1259 (2003)
8. Lutz, J.H.: The dimensions of individual strings and sequences. Information and Computation 187, 49–79 (2003)
9. Lutz, J.H.: Effective fractal dimensions (invited lecture at the International Conference on Computability and Complexity in Analysis, Cincinnati, OH, August 28-30, 2003). Mathematical Logic Quarterly 51, 62–72 (2005)
10. Mayordomo, E.: A Kolmogorov complexity characterization of constructive Hausdorff dimension. Information Processing Letters 84(1), 1–3 (2002)
11. Nies, A., Reimann, J.: A lower cone in the wtt degrees of non-integral effective dimension. In: Proceedings of IMS workshop on Computational Prospects of Infinity, Singapore. Earlier version appeared as Technical Report 63, Workgroup Mathematical Logic and Theoretical Computer Science, University of Heidelberg (To appear) (2005)
12. Nies, A., Stephan, F., Terwijn, S.A.: Randomness, relativization and Turing degrees. The. Journal of Symbolic Logic 70, 515–535 (2005)
13. Odifreddi, P.: Classical recursion theory, volume 125 of Studies in Logic and the Foundations of Mathematics. North-Holland (1989)
14. Reimann, J.: Computability and fractal dimension. Doctoral thesis, Heidelberg (2005)
15. Reimann, J., Slaman, T.: Randomness, Entropy and Reducibility. Manuscript (2005)
16. Ryabko, B.Y.: Coding of combinatorial sources and Hausdorff dimension. Soviet Mathematics Doklady 30, 219–222 (1984)
17. Ryabko, B.Y.: Noiseless coding of combinatorial sources. Problems of Information Transmission 22, 170–179 (1986)
18. Shaltiel, R.: Recent developments in explicit constructions of extractors. Bulletin of the EATCS 77, 67–95 (2002)
19. Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, Heidelberg (1987)
20. Stephan, F.: Hausdorff-dimension and weak truth-table reducibility. Technical Report TR52/05, School of Computing, National University of Singapore (2005)
21. Sullivan, D.: Entropy, Hausdorff measures old and new, and limit sets of geometrically finite Kleinian groups. Acta. Mathematica 153, 259–277 (1984)
22. Tricot, C.: Two definitions of fractional dimension. Mathematical Proceedings of the Cambridge Philosophical Society 91, 57–74 (1982)

A Classification of Viruses Through Recursion Theorems

Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion

Nancy-Université - Loria - INPL - Ecole Nationale Supérieure des Mines de Nancy
B.P. 239, 54506 Vandœuvre-lès-Nancy Cédex, France
Guillaume.Bonfante@loria.fr
Matthieu.Kaczmarek@loria.fr
Jean-Yves.Marion@loria.fr

Abstract. We study computer virology from an abstract point of view. Viruses and worms are self-replicating programs, whose constructions are essentially based on Kleene's second recursion theorem. We show that we can classify viruses as solutions of fixed point equations which are obtained from different versions of Kleene's second recursion theorem. This lead us to consider four classes of viruses which various polymorphic features. We propose to use virus distribution in order to deal with mutations.

Topics covered. Computability theoretic aspects of programs, computer virology.

Keywords: Computer viruses, polymorphism, propagation, recursion theorem, iteration theorem.

1 Theoretical Computer Virology

An important information security breach is computer virus infections. Following Filiol's book [9], we do think that theoretical studies should help to design new defenses against computer viruses. The objective of this paper is to pursue a theoretical study of computer viruses initiated in [4]. Since viruses are essentially self-replicating programs, we see that virus programming methods are an attempt to answer to von Neumann's question [20].

> Can an automaton be constructed, i.e., assembled and built from appropriately "raw material", by an other automaton? [. . .] Can the construction of automata by automata progress from simpler types to increasingly complicated types?

Abstract computer virology was initiated in the 80's by the seminal works of Cohen and Adleman [7]. The latter coined the term *virus*. Cohen defined viruses with respect to Turing Machines [8]. Later [1], Adleman took a more abstract point of view in order to have a definition independent from any particular computational model. Then, only a few theoretical studies followed those seminal works. Chess and White refined the mutation model of Cohen in [6]. Zuo and

Zhou formalized polymorphism from Adleman's work [21] and they analyzed the time complexity of viruses [22].

Recently, we tried [3,4] to formalize inside computability the notion of viruses. This formalization captures previous definitions that we have mentioned above. We also characterized two kinds of viruses, blueprint and smith viruses, and we proved constructively their existence. This work proposes to go further, introducing a notion of distribution to take into account polymorphism or metamorphism. We define four kinds of viruses:

1. A blueprint virus is a virus, which reproduces by just duplicating its code.
2. An evolving blueprint virus is a virus, which can mutate when it duplicates by modifying its code. Evolving blueprint viruses are generated by a disbution engine.
3. A smith virus is a blueprint virus which can use its propagation function directly to reproduce.
4. Lastly, we present Smith distribution. A virus generating by a Smith distribution can mutate its code like evolving blueprint viruses, but also mutate its propagation function.

We show that each category is closely linked to a corresponding form of the recursion theorem, given a rational taxonomy of viruses. So recursion theorems play a key role in *constructions* of viruses, which is worth to mention. Indeed, and despite the works [10,11], recursion theorems are used essentially to prove "negative" results such as the constructions of undecidable or inseparable sets, see [17] for a general reference, or such as Blum's speed-up theorem [2].

Lastly, we switch to a simple programming language named WHILE$^+$ to illustrate the fact that our constructions lives in the programming world. Actually, we follow the ideas of the experimentation of the iteration theorem and of the recursion theorem, which are developed in [10,11] by Jones et al. and very recently by Moss in [15].

2 A Virus Definition

2.1 The WHILE$^+$ Language

The domain of computation $\mathbb{D}$ is the set of binary trees generated from an atom nil and a pairing mechanism $\langle\ ,\ \rangle$. The syntax of WHILE$^+$ is given by the following grammar from a set of variables $\mathbb{V}$:

$$\begin{aligned}\text{Expressions: } \mathbb{E} \to{} & \mathbb{V} \mid \mathtt{cons}(\mathbb{E}_1, \mathbb{E}_2) \mid \mathtt{hd}(\mathbb{E}) \mid \mathtt{tl}(\mathbb{E}) \mid \\ & \mathtt{exec}_n(\mathbb{E}_0, \mathbb{E}_1, \dots, \mathbb{E}_n) \mid \mathtt{spec}_n(\mathbb{E}_0, \mathbb{E}_1 \dots, \mathbb{E}_n) \text{ with } n \geq 1 \\ \text{Commands: } \mathbb{C} \to{} & \mathbb{V} := \mathbb{E} \mid \mathbb{C}_1; \mathbb{C}_2 \mid \mathtt{while}(\mathbb{E})\{\mathbb{C}\} \mid \mathtt{if}(\mathbb{E})\{\mathbb{C}_1\}\mathtt{else}\{\mathbb{C}_2\}\end{aligned}$$

A WHILE$^+$ program $\mathbf{p}$ is defined as follows $\mathbf{p}(\mathbb{V}_1, \dots, \mathbb{V}_n)\{\mathbb{C}; \mathtt{return}\ \mathbb{E}; \}$. A program $\mathbf{p}$ computes a function $[\![\mathbf{p}]\!]$ from $\mathbb{D}^n$ to $\mathbb{D}$.

We suppose that we are given a concrete syntax of WHILE$^+$, that is an encoding of programs by binary trees of $\mathbb{D}$. From now on, when the context is clear, we do not make any distinction between a program and its concrete syntax. And we make no distinction between programs and data.

For convenience, we have a built-in self-interpreter $\texttt{exec}_n$ of WHILE$^+$ programs which satisfies:

$$[\![\texttt{exec}_n]\!](\mathbf{p}, x_1, \ldots x_n) = [\![\mathbf{p}]\!](x_1, \ldots x_n)$$

In the above equation, the notation **p** means the concrete syntax of the program **p**.

We also use a built-in specializer $\texttt{spec}_n$ which satisfies:

$$[\![[\![\texttt{spec}_m]\!](\mathbf{p}, x_1, \ldots x_m)]\!](x_{m+1}, \ldots, x_n) = [\![\mathbf{p}]\!](x_1, \ldots x_n)$$

We may omit the subscpript n which indicates the number of arguments of an interpreter or a specializer.

The use of an interpreter and of a specializer is justified by Jones who showed in [12] that programs with these constructions can be simulated up to a linear constant time by programs without them.

If f and g designate the same function, we write $f \approx g$. A function f is *semi-computable* if there is a program **p** such that $[\![\mathbf{p}]\!] \approx f$, moreover, if f is total, we say that f is *computable*.

2.2 A Computer Virus Representation

We propose the following scenario in order to represent viruses. When a program **p** is executed within an environment x, the evaluation of $[\![\mathbf{p}]\!](x)$, if it halts, is a new environment. This process may be then repeated by replacing x by the new computed environment. The entry x is thought of as a finite sequence $\langle x_1, \ldots, x_n\rangle$ which represents files and accessible parameters.

Typically, a program **copy** which duplicates a file satisfies $[\![\mathbf{copy}]\!](\mathbf{p}, x) = \langle \mathbf{p}, \mathbf{p}, x\rangle$. The original environment is $\langle \mathbf{p}, x\rangle$. After the evaluation of **copy**, we have the environment $\langle \mathbf{p}, \mathbf{p}, x\rangle$ in which **p** is copied.

Next consider an example of *parasitic virus*. Parasitic viruses insert themselves into existing files. When an infected host is executed, first the virus infects a new host, then it gives the control back to the original host. For more details we refer to the virus writing manual of Ludwig [14]. A parasistic virus is a program **v** which works on an environment $\langle \mathbf{p}, \mathbf{q}, x\rangle$. The infected form of **p** is $B(\mathbf{v}, \mathbf{p})$ where B is a propagation function which specifies how a virus contaminates a file. Here, the propagation function B can be for example a program code concatenation function. So, we have a first "generic" equation: $[\![\mathbf{v}]\!](\mathbf{p}, \langle \mathbf{q}, x\rangle) = [\![B(\mathbf{v}, \mathbf{p})]\!](\langle \mathbf{q}, x\rangle)$. Following the description of a parasitic virus, **v** computes the infected form $B(\mathbf{v}, \mathbf{q})$ and then executes **p**. This means that the following equation also holds: $[\![\mathbf{v}]\!](\mathbf{q}, x) = [\![\mathbf{p}]\!](B(\mathbf{v}, \mathbf{q}), x)$. A parasistic virus is defined by the two above equations.

More generally, the construction of viruses lies in the resolution of fixed point equations such as the ones above in which $\mathbf{v}$ and B are unknowns. The existence of solutions of such systems is provided by Kleene's recursion theorem. From this observation and following [4], we propose the following virus representation:

Definition 1 (Computer Virus). *Let B be a computable function. A virus w.r.t B is a program $\mathbf{v}$ such that $\forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = [\![B(\mathbf{v}, \mathbf{p})]\!](x)$. Then, B is named a propagation function for the virus $\mathbf{v}$.*

This definition includes the ones of Adleman and Cohen, and it handles more propagation and duplication features than the other models [4]. However, it is worth to notice that the existence of a virus $\mathbf{v}$ w.r.t a given propagation function B is constructive. This is a key difference since it allows to build viruses by applying fixed point constructions given by proofs of recursion theorems.

A motivation behind the choice of WHILE^{+} programming language is the fact that there is no self-referential operator, like \$0 in bash, which returns a copy of the program concrete syntax. Indeed, we present below virus construction without this feature. This shows that even if there is no self-referential operator, there are still viruses. Now, viruses should be more efficient if such operators are present. Of course, a seminal paper on this subject is [19].

3 Blueprint Duplication

3.1 Blueprint Distribution Engine

From [4], a *blueprint virus for a function g* is a program $\mathbf{v}$ which computes g using its own code $\mathbf{v}$ and its environment $\mathbf{p}, x$. The function g can be seen as the virus specification function. A blueprint virus for a function g is a program $\mathbf{v}$ which satisfies

$$\begin{cases} \mathbf{v} \text{ is a virus w.r.t some propagation function} \\ \forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = g(\mathbf{v}, \mathbf{p}, x) \end{cases} \tag{1}$$

Note that a blueprint virus does not use any code of its propagation function, unlike smith viruses that we shall see shortly. The solutions of this system are provided by Kleene's recursion theorem.

Theorem 2 (Kleene's Recursion Theorem [13]). *Let f be a semi-computable function. There is a program $\mathbf{e}$ such that $[\![\mathbf{e}]\!](x) = f(\mathbf{e}, x)$.*

Definition 3 (Distribution engine). *A distribution engine is a program $\mathbf{d_v}$ such that for every virus specification program $\mathbf{g}$, $[\![\mathbf{d_v}]\!](\mathbf{g})$ is a virus w.r.t a fixed and given a propagation function B.*

Theorem 4. *There is a distribution engine $\mathbf{d_v}$ such that for any program $\mathbf{g}$, $[\![\mathbf{d_v}]\!](\mathbf{g})$ is a blueprint virus for $[\![\mathbf{g}]\!]$.*

Proof. We use a construction for the recursion theorem due to Smullyan [18]. It provides a fixpoint which can be directly used as a distribution engine. We define $\mathbf{d_v}$ thanks to the concrete syntax of $\mathbf{dg}$ as follows:

```
dg (z,u,y,x){                          dv (g){
 r := exec(z,spec(u,z,u),y,x);          r := spec(dg,g,dg);
 return r;                              return r;
}                                      }
```

We observe that $[\![[\![\mathbf{d_v}]\!](\mathbf{g})]\!](\mathbf{p}, x) = [\![\mathbf{g}]\!]([\![\mathbf{d_v}]\!](\mathbf{g}), \mathbf{p}, x)$. Moreover, $[\![\mathbf{d_v}]\!](\mathbf{g})$ is clearly a virus w.r.t to the propagation function $[\![\texttt{spec}]\!]$. □

We consider a typical example of blueprint duplication which looks like the real life virus `ILoveYou`. This program arrives as an e-mail attachment. Opening the attachment triggers the attack. The infection first scans the memory for passwords and sends them back to the attacker, then the virus self-duplicates sending itself at every address of the local address book.

To represent this scenario we need to deal with mailing processes. A mail $m = \langle @, y \rangle$ is an association of an address @ and data y. Then, we consider that the environment contains a mailbox $mb = \langle m_1, \ldots, m_n \rangle$ which is a sequence of mails. To send a mail m, we add it to the mailbox, that is $mb := \texttt{cons}(m, mb)$. We suppose that an external process deals with mailing.

In the following, x denotes the local file structure, and $@bk = \langle @_1, \ldots, @_n \rangle$ denotes the local address book, a sequence of addresses. We finally introduce a $\texttt{WHILE}^+$ program **find** which searches its input for passwords and which returns them as its evaluation. The virus behavior for the scenario of `ILoveYou` is given by the following program.

```
g (v,mb,⟨@bk, x⟩) {
 pass := exec(find,x);
 mb := cons(cons(“badguy@dom.com”,pass),mb);
 y := @bk;
 while (y) {
  mb := cons(cons(hd(y),v),mb);
  y := tl(y);
 }
 return mb;
}
```

From the virus specification program $\mathbf{g}$, we generate the blueprint virus $[\![\mathbf{d_v}]\!](\mathbf{g})$.

3.2 Distributions of Evolving Blueprint Viruses

An *evolving blueprint virus* is a virus, which can mutate but the propagation function remains the same. Here, we describe a distribution engine for which the specification of a virus can use the code of its own distribution engine. Thus, we can generate evolved copies of a virus. Formally, given a virus specification function g, a distribution of evolving blueprint viruses is a program $\mathbf{d_v}$ satisfying:

$$\begin{cases} \mathbf{d_v} \text{ is a distribution engine} \\ \forall i, \mathbf{p}, x : [\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p}, x) = g(\mathbf{d_v}, i, \mathbf{p}, x) \end{cases} \tag{2}$$

The existence of blueprint distributions corresponds to a stronger form of the recursion theorem, which was first proved by Case [5].

Theorem 5 (Explicit recursion [4]). *Let f be a semi-computable function. There exists a computable function e such that $\forall x, y : [\![e(x)]\!](y) = f(\mathbf{e}, x, y)$ where $\mathbf{e}$ computes e.*

Definition 6 (Distribution engine builder). *A builder of distribution engine is a program $\mathbf{c_v}$ such that for every virus specification program $\mathbf{g}$, $[\![\mathbf{c_v}]\!](\mathbf{g})$ is a distribution engine.*

Theorem 7. *There is a builder of distribution engine $\mathbf{c_v}$ such that for any program $\mathbf{g}$, $[\![\mathbf{c_v}]\!](\mathbf{g})$ is a distribution of evolving blueprint viruses for some fixed propagation function B.*

Proof. We define

```
edg (z,t,i,y,x) {                    c_v (g){
 e := spec(spec_3,t,z,t);             r := spec(spec_3,edg,g,edg);
 return exec(z,e,i,y,x);              return r;
}                                    }
```

We observe that for any i, $[\![[\![[\![\mathbf{c_v}]\!](\mathbf{g})]\!](i)]\!](\mathbf{p}, x) = g([\![\mathbf{c_v}]\!](\mathbf{g}), i, \mathbf{p}, x)$. Moreover, $[\![[\![\mathbf{c_v}]\!](\mathbf{g})]\!](i)$ is a virus w.r.t $[\![\texttt{spec}]\!]$. □

To illustrate Theorem 7, we come back to the scenario of the virus `ILoveYou`, and we add to it mutation abilities. We introduce a `WHILE`$^+$ program **poly** which is a polymorphic engine. This program takes a program $\mathbf{p}$ and a key i, and it rewrites $\mathbf{p}$ according to i, conserving the semantics of $\mathbf{p}$. That is, **poly** satisfies $[\![\mathbf{poly}]\!](\mathbf{p}, i)$ is one-one on i and $[\![[\![\mathbf{poly}]\!](\mathbf{p}, i)]\!] \approx [\![\mathbf{p}]\!]$.

We build a virus which self-duplicates sending mutated forms of itself. With the notations of the Sect. 3.1, we consider a behavior described by the following `WHILE`$^+$ program.

```
g (dv,i,mb,⟨@bk, x⟩) {
 pass := exec(find,x);
 mb := cons(cons(“badguy@dom.com”,pass),mb);
 next_key := cons(nil,i)
 virus := exec(dv,next_key);
 mutation := exec(poly,virus,i);
 y := @bk;
 while (y) {
  mb := cons(cons(hd(y),mutation),mb);
  y := tl(y);
 }
 return mb;
}
```

We apply Theorem 7 to transform this program into a code of the corresponding distribution engine. So, $[\![[\![\mathbf{c_v}]\!](\mathbf{g})]\!](i)$ is a copy indexed by i of the evolving blueprint virus specified by $\mathbf{g}$.

4 Smith Reproduction

4.1 Smith Viruses

We define a *smith virus* as two programs $\mathbf{v}, \mathbf{B}$ which is defined w.r.t a virus specification function g according to the following system.

$$\begin{cases} \mathbf{v} \text{ is a virus w.r.t } [\![\mathbf{B}]\!] \\ \forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = g(\mathbf{B}, \mathbf{v}, \mathbf{p}, x) \end{cases}$$

The class of smith viruses is obtained by the double recursion theorem due to Smullyan [16] as a solution to the above equations.

Theorem 8 (Double Recursion Theorem [16]). *Let f_1 and f_2 be two semi-computable functions. There are two programs $\mathbf{e}_1$ and $\mathbf{e}_2$ such that*

$$[\![\mathbf{e}_1]\!](x) = f_1(\mathbf{e}_1, \mathbf{e}_2, x) \quad [\![\mathbf{e}_2]\!](x) = f_2(\mathbf{e}_1, \mathbf{e}_2, x)$$

We extend the previous definition of engine distribution to propagation engine as follows.

Definition 9 (Virus Distribution). *A virus distribution is a pair $(\mathbf{d_v}, \mathbf{d_B})$ of programs such that for every virus specification $\mathbf{g}$, $[\![\mathbf{d_v}]\!](\mathbf{g})$ is a virus w.r.t $[\![[\![\mathbf{d_B}]\!](\mathbf{g})]\!]$. As previously, $\mathbf{d_v}$ is named* a distribution engine *and $\mathbf{d_B}$ is named* a propagation engine.

Theorem 10. *There is a virus distribution $(\mathbf{d_v}, \mathbf{d_B})$ such that for any program $\mathbf{g}$, $[\![\mathbf{d_v}]\!](\mathbf{g}), [\![\mathbf{d_B}]\!](\mathbf{g})$ is a smith virus for $[\![\mathbf{g}]\!]$.*

Proof. We define the following programs with a double fixed point.

```
dg1 (z1,z2,y1,y2,y,x) {
 e1 := spec(y1,z1,z2,y1,y2);
 e2 := spec(y2,z1,z2,y1,y2);
 return exec(z1,e1,e2,y,x);
}
```

```
dg2 (z1,z2,y1,y2,y,x) {
 e1 := spec(y1,z1,z2,y1,y2);
 e2 := spec(y2,z1,z2,y1,y2);
 return exec(z2,e1,e2,y,x);
}
```

and

```
pispec (g,B,v,y,p) {
 r := spec(g,B,v,p);
 return r;
}
```

Then, let $\mathbf{d_v}$ and $\mathbf{d_B}$ be the following programs.

```
d_v (g){
 r := spec(pispec,g);
 return spec(dg2,r,g,dg1,dg2);
}
```

```
d_B (g){
 r := spec(pispec,g);
 return spec(dg1,r,g,dg1,dg2);
}
```

We observe that for any program $\mathbf{g}$

$$[\![[\![\mathbf{d_v}]\!](\mathbf{g})]\!](\mathbf{p},x) = [\![[\![[\![\mathbf{d_B}]\!](\mathbf{g})]\!]([\![\mathbf{d_v}]\!](\mathbf{g}),\mathbf{p})]\!](x) = g([\![\mathbf{d_B}]\!](\mathbf{g}),[\![\mathbf{d_v}]\!](\mathbf{g}),\mathbf{p},x) \qquad \square$$

We present how to build the parasitic virus of Sect. 2. The virus specification function $\mathbf{g}$ of the virus is the following.

```
g (B,v,p,⟨q,x⟩) {
 infected_form := exec(B,v,p);
 return exec(p,infected_form,x);
}
```

First, it infects a new host $\mathbf{q}$ with the virus $\mathbf{v}$ using the propagation procedure $\mathbf{B}$. Then, it executes the original host $\mathbf{p}$. This corresponds to the behavior of a parasitic virus. We obtain a smith virus using the builder of Theorem 10.

4.2 Smith Distributions

Smith distributions generate viruses which are able to mutate their code and their propagation mechanism. A *smith distribution* $(\mathbf{d_v},\mathbf{d_B})$ w.r.t the virus specification program g satisfies

$$\begin{cases} (\mathbf{d_v},\mathbf{d_B}) \text{ is a virus distribution} \\ \forall i,\mathbf{p},x : [\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p},x) = g(\mathbf{d_B},\mathbf{d_v},i,\mathbf{p},x) \end{cases}$$

The class of Smith distributions is defined as the solutions of this double recursion theorem.

Theorem 11 (Double explicit Recursion). *Let f_1 and f_2 be two semi-computable functions. There are two computable functions e_1 and e_2 such that for all x and y*

$$[\![e_1(x)]\!](y) = f_1(\mathbf{e_1},\mathbf{e_2},x,y) \quad [\![e_2(x)]\!](y) = f_2(\mathbf{e_1},\mathbf{e_2},x,y)$$

where $\mathbf{e_1}$ and $\mathbf{e_2}$ respectively compute e_1 and e_2.

Definition 12 (Distribution builder). *A Distribution builder is a pair of programs* $\mathbf{c_V}, \mathbf{c_B}$ *such that for every virus specification program* $\mathbf{g}$*,* $([\![\mathbf{c_V}]\!](\mathbf{g}), [\![\mathbf{c_B}]\!](\mathbf{g}))$ *is a virus distribution.*

Theorem 13. *There is a distribution builder* $(\mathbf{c_V}, \mathbf{c_B})$ *such that for any program* $\mathbf{g}$*,* $([\![\mathbf{c_V}]\!](\mathbf{g}), [\![\mathbf{c_B}]\!](\mathbf{g}))$ *is a smith distribution for* $[\![\mathbf{g}]\!]$*.*

Proof. We define the following programs:

```
edg1 (z1,z2,t1,t2,i,y,x) {
 e1 := spec(spec5,t1,z1,z2,t1,t2);
 e2 := spec(spec5,t2,z1,z2,t1,t2);
 return exec(z1,e1,e2,i,y,x);
}
```

```
edg2 (z1,z2,t1,t2,i,y,x) {
 e1 := spec(spec5,t1,z1,z2,t1,t2);
 e2 := spec(spec5,t2,z1,z2,t1,t2);
 return exec(z2,e1,e2,i,y,x);
}
```

and

```
pispec' (g,db,dv,i,y,p) {
 r := spec(g,db,dv,i,p);
 return r;
}
```

Let $\mathbf{c_V}$ and $\mathbf{c_B}$ be the following programs.

```
cV (g){
 r := spec(pispec',g)
 return spec(spec5,edg2,r,g,edg1,edg2);
}
```

```
cB (g){
 r := spec(pispec',g)
 return spec(spec5,edg1,r,g,edg1,edg2);
}
```

We observe that for any program $\mathbf{g}$

$$\begin{aligned}[\![[\![[\![\mathbf{c_V}]\!](\mathbf{g})]\!](i)]\!](\mathbf{p}, x) &= [\![[\![[\![[\![\mathbf{c_B}]\!](\mathbf{g})]\!](i)]\!]([\![[\![\mathbf{c_V}]\!](\mathbf{g})]\!](i), \mathbf{p})]\!](x) \\ &= g([\![\mathbf{c_B}]\!](\mathbf{g}), [\![\mathbf{c_V}]\!](\mathbf{g}), i, \mathbf{p}, x)\end{aligned}$$ □

We enhance the virus of Sect. 4.1, adding some polymorphic abilities. Any virus of generation i infects a new host $\mathbf{q}$ with a virus of next generation using the propagation procedure of generation i. Then it gives the control back to the original host $\mathbf{p}$. This behavior is illustrated by the following program.

```
g (db,dv,i,p,⟨q,x⟩) {
 B := exec(db,i);
 v := exec(dv,cons(i,nil));
 mutation := exec(poly,v,i);
 infected_form := exec(B,mutation,q);
 return exec(p,infected_form,x);
}
```

Then, we obtain the smith distribution by the builder of Theorem 13.

References

1. Adleman, L.: An abstract theory of computer viruses. In: Vulkov, L.G., Waśniewski, J., Yalamov, P. (eds.) NAA 2000. LNCS, vol. 403, Springer, Heidelberg (1988)
2. Blum, M.: A machine-independent theory of the complexity of recursive functions. Journal of the Association for Computing Machinery 14(2), 322–336 (1967)
3. Bonfante, G., Kaczmarek, M., Marion, J.-Y.: Toward an abstract computer virology. In: ICTAC, LNCS, vol. 3722, pp. 579–593 (2005)
4. Bonfante, G., Kaczmarek, M., Marion, J.-Y.: On abstract computer virology from a recursion-theoretic perspective. Journal in Computer Virology, 1(3-4) (2006)
5. Case, J.: Periodicity in generations of automata. Theory of Computing Systems 8(1), 15–32 (1974)
6. Chess, D., White, S.: An undetectable computer virus. Proceedings of the 2000 Virus Bulletin Conference (VB2000) (2000)
7. Cohen, F.: Computer Viruses. PhD thesis, University of Southern California (January 1986)
8. Cohen, F.: On the implications of computer viruses and methods of defense. Computers and Security 7, 167–184 (1988)
9. Filiol, E.: Computer Viruses: from Theory to Applications. Springer, Heidelberg (2005)
10. Hansen, T., Nikolajsen, T., Träff, J., Jones, N.: Experiments with implementations of two theoretical constructions. In: Meyer, A.R., Taitslin, M.A. (eds.) Logic at Botik 1989. LNCS, vol. 363, pp. 119–133. Springer, Heidelberg (1989)
11. Jones, N.: Computer implementation and applications of kleene's S-m-n and recursive theorems. In: Moschovakis, Y.N. (ed.) Lecture Notes in Mathematics, Logic From Computer Science, pp. 243–263. Springer, Heidelberg (1991)
12. Jones, N.: Constant Time Factors Do Matter. MIT Press, Cambridge, MA, USA (1997)
13. Kleene, S.: Introduction to Metamathematics. Van Nostrand (1952)
14. Ludwig, M.: The Giant Black Book of Computer Viruses. American Eagle Publications (1998)
15. Moss, L.: Recursion theorems and self-replication via text register machine programs. In: EATCS bulletin (2006)
16. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York (1967)
17. Smullyan, R.: Recursion Theory for Metamathematics. Oxford University Press, Oxford (1993)
18. Smullyan, R.: Diagonalization and Self-Reference. Oxford University Press, Oxford (1994)
19. Thompson, K.: Reflections on trusting trust. Communications of the Association for Computing Machinery 27(8), 761–763 (1984)
20. von Neumann, J.: Theory of Self-Reproducing Automata (edited and completed by A.W.Burks). University of Illinois Press, Urbana, Illinois (1966)
21. Zuo, Z., Zhou, M.: Some further theoretical results about computer viruses. The Computer Journal 47(6), 627–633 (2004)
22. Zuo, Z., Zhu, Q.-x., Zhou, M.-t.: On the time complexity of computer viruses. IEEE Transactions on information theory 51(8), 2962–2966 (2005)

Borel Complexity of Topological Operations on Computable Metric Spaces

Vasco Brattka[1,⋆] and Guido Gherardi[2]

[1] Laboratory of Foundational Aspects of Computer Science
Department of Mathematics & Applied Mathematics
University of Cape Town, Rondebosch 7701, South Africa
Vasco.Brattka@uct.ac.za
[2] Dipartimento di Scienze Matematiche é
Informatiche Roberto Magari
University of Siena, Italy
gherardi3@unisi.it

Abstract. We study the Borel complexity of topological operations on closed subsets of computable metric spaces. The investigated operations include set theoretic operations as union and intersection, but also typical topological operations such as the closure of the complement, the closure of the interior, the boundary and the derivative of a set. These operations are studied with respect to different computability structures on the hyperspace of closed subsets. These structures include positive or negative information on the represented closed subsets. Topologically, they correspond to the lower or upper Fell topology, respectively, and the induced computability concepts generalize the classical notions of r.e. or co-r.e. subsets, respectively. The operations are classified with respect to effective measurability in the Borel hierarchy and it turns out that most operations can be located in the first three levels of the hierarchy, or they are not even Borel measurable at all. In some cases the effective Borel measurability depends on further properties of the underlying metric spaces, such as effective local compactness and effective local connectedness.

Keywords: Computable analysis, effective descriptive set theory, Borel measurability, hyperspace topologies.

1 Introduction

In this paper we study the Borel complexity of set theoretic and topological operations such as

- Union: $\cup : \mathcal{A}(X) \times \mathcal{A}(X) \to \mathcal{A}(X), (A, B) \mapsto A \cup B$,
- Intersection: $\cap : \mathcal{A}(X) \times \mathcal{A}(X) \to \mathcal{A}(X), (A, B) \mapsto A \cap B$,
- Complement: $c : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto \overline{A^c}$,

⋆ This work has been partially supported by the National Research Foundation (NRF) Grant FA2005033000027 on "Computable Analysis and Quantum Computing".

- Interior: $i : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto \overline{A^\circ}$,
- Boundary: $\partial : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto \partial A$,
- Derivative: $d : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto A'$.

In those cases where the operations do not necessarily map to closed sets, we have to take additional closures (so strictly speaking the operations should be called "closure of the complement" and so on). These operations are typically defined on the hyperspace $\mathcal{A}(X)$ of closed subsets $A \subseteq X$ of some computable metric space X. We represent $\mathcal{A}(X)$ with respect to positive, negative or full information and the question is in which sense these operations are computable or relatively computable. The corresponding representations are denoted by ψ_+, ψ_- and ψ and the induced notions of computability are generalizations of the classical concepts of r.e., co-r.e. and recursive sets. We assume familiarity with basic notions of the representation based approach to computable analysis, as it is presented in [1]. The above mentioned representations are discussed in [2].

In many cases it turns out that the above mentioned operations are not computable and not even continuous. In such a situation it is desirable to classify the degree of non-computability of these operations. The most appropriate way to classify this degree is a classification with respect to the Borel hierarchy. While a continuous function is characterized by the property that preimages of open sets are open, $\mathbf{\Sigma}^0_k$–measurable functions are functions such that preimages of open sets are $\mathbf{\Sigma}^0_k$–sets in the Borel hierarchy. This hierarchy starts with the class of open sets $\mathbf{\Sigma}^0_1$ and proceeds with the class of F_σ–sets, which are countable unions of closed sets. Correspondingly, $\mathbf{\Sigma}^0_{k+1}$–sets are countable unions of complements of $\mathbf{\Sigma}^0_k$–sets in general. These classes and measurability are studied in descriptive set theory as presented in [3,4]. We will also use effective versions of these concepts as introduced in [5] and concepts such as $\mathbf{\Sigma}^0_k$–computability and $\mathbf{\Sigma}^0_k$–completeness are precisely defined in [5].

Measurability questions of set theoretic and topological operations are subject to studies for a long time. In Chapter I §18 of [6] union, intersection and difference of lower and upper semi-continuous maps are discussed. These results correspond to some of our results on union and intersection of closed subsets. In Chapter IV §43 of [7] the boundary and derivative operation are discussed as $\mathbf{\Sigma}^0_3$–continuous maps that are not $\mathbf{\Sigma}^0_2$–continuous. In [8,9,10] Christensen has discussed and partially characterized the Borel measurability of some of these operations. In [11] computability properties of some of these operations has been discussed (restricted to compact subsets of Euclidean space) and finally in [12] the Borel complexity of these operations for closed subsets of Euclidean space has been characterized with respect to some representations. In this paper we attempt to collect these results and to characterize the Borel complexity of these operations more precisely and in a more comprehensive way for the general case of computable metric spaces X.

We close the introduction with a brief survey on the following sections of this paper. In Section 2 we present some basic concepts from the representation approach to computable analysis and we state a version of the Representation Theorem for Borel measurable functions that shows that a function is Borel measurable if and only if it has a Borel measurable realizer. For all undefined concepts from computable analysis we refer the reader to [1]. In Section 3 we briefly review some definitions and results on representations of closed subsets of metric spaces as introduced in [2]. In particular, we present the lattice of such representations with respect to computable (and continuous) reducibility. Further results and proofs can be found in [2]. In Section 4 we study the aforementioned lattice of representations from the somewhat coarser point of view of Borel reducibility. It turns out that most of the considered representations induce the Effros Borel structure and altogether, all the considered representations fall in at most three different equivalence classes. We also consider the special case of effectively locally compact metric spaces X where the number of classes is even smaller. In Section 5 we introduce some notions for computable metric spaces that guarantee that these spaces are rich enough in order to allow certain completeness and hardness results. Mainly proper and perfect spaces will satisfy all our requirements. The following sections are devoted to different operations. In Section 6 we study intersection and union in detail, while Section 7 is devoted to operations that involve complement, closure and interior. In Section 8 we study the boundary operation and the derivative and for the boundary the further aspect of effective local connectedness becomes relevant. This extended abstract version of the paper does not contain any proofs.

2 Borel Representation Theorem

Throughout this paper we will use the representation approach to computable analysis as it has been presented in [13,1]. The basic idea is that the Baire space $\mathbb{N}^{\mathbb{N}}$ is used to represent objects of other spaces X. Formally, a *representation* is a surjective map $\delta :\subseteq \mathbb{N}^{\mathbb{N}} \to X$, where the inclusion symbol indicates that the map is potentially partial. In this situation (X, δ) will be called a *represented space*. For represented spaces concepts from topology, computability theory and descriptive set theory can be transferred from Baire space to the represented spaces via realizers.

Definition 1 (Realizer). *Let (X, δ_X) and (Y, δ_Y) be represented spaces. We say that a function $F :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ is a* realizer *for $f :\subseteq X \to Y$ or* realizes *f, if $\delta_Y F(p) = f\delta_X(p)$ holds for all $p \in \mathrm{dom}(f\delta_X)$.*

Using the concept of a realizer, we transfer all sorts of properties of F to f. For instance, we will say that f is *continuous with respect to* (δ_X, δ_Y), if it has a continuous realizer. Analogously, we will transfer properties such as computability, $\mathbf{\Sigma}^0_k$–measurability, $\mathbf{\Sigma}^0_k$–computability (as defined in [5]) and Borel measurability from F to f with respect to (δ_X, δ_Y). This will not lead to any confusion as long as we are dealing with admissible representations.

A representation δ of a topological space X is called *admissible*, if it is maximal among all continuous representations δ' of X, that is if $\delta' \leq_{\mathrm{t}} \delta$ holds for all continuous δ'. Here $\delta' \leq_{\mathrm{t}} \delta$ means that δ' is *continuously reducible* to δ, i.e. that the identity $\mathrm{id} : X \to X$ is continuous with respect to (δ', δ). By $\equiv_{\mathrm{t}}$ we denote the induced equivalence relation and we write $\leq_{\mathrm{B}}$ and $\leq$ if continuity is replaced by Borel measurability and computability, respectively. The corresponding equivalence relations are denoted by $\equiv_{\mathrm{B}}$ and $\equiv$. We also write $\leq_2$ for the case that there is a $\mathbf{\Sigma}^0_2$–computable reduction. However, the latter is a slight misusage of the symbols, as $\leq_2$ is not transitive and hence not a preorder.

If X, Y are second-countable T_0–spaces with admissible representations δ_X and δ_Y, then it follows by the Representation Theorem of Kreitz and Weihrauch that a function f is continuous with respect to (δ_X, δ_Y) if and only if it is continuous in the ordinary topological sense [14,13,1]. By an extension of this theorem due to Schröder, the same holds even for larger classes of spaces, as long as one replaces ordinary continuity by sequential continuity [15]. For computable metric spaces and total maps f the Representation Theorem has been extended to $\mathbf{\Sigma}^0_k$–measurability in [5]. Here we will formulate yet another simple extension of this theorem to the class of Borel measurable maps f.

Theorem 1 (Borel Representation Theorem). *Let X, Y be second countable T_0–spaces with admissible representations δ_X, δ_Y. Then a map $f :\subseteq X \to Y$ is Borel measurable if and only if it is Borel measurable with respect to (δ_X, δ_Y).*

In case of total maps on computable metric spaces, the Representation Theorem in [5] shows that even the level of measurability in the Borel hierarchy is preserved by realizers. That is $\mathbf{\Sigma}^0_k$–measurability is equivalent to $\mathbf{\Sigma}^0_k$–measurability with respect to (δ_X, δ_Y) in that situation.

3 Representations of Closed Subsets

In this section we briefly recall the definitions of some representations of closed subsets of metric spaces as they have been introduced in [2] (but we use a slightly different notation in order to be as compatible to other references as possible).

In case that X is a separable metric space, we can consider the set $\mathcal{A}(X)$ of closed subsets of X as a second countable T_0–space in various ways. We assume that X is equipped with some fixed numbering α of a dense subset of X and we assume that $\overline{k}$ denotes the rational number enumerated by $k \in \mathbb{N}$ (i.e. we assume that $k \mapsto \overline{k}$ is some standard numbering of $\mathbb{Q}$). Then by $I_{\langle i,j\rangle} := B(\alpha(i), \overline{j})$ and $\widehat{I}_{\langle i,j\rangle} := \overline{B}(\alpha(i), \overline{j})$ we denote some standard numberings of rational open balls and rational closed balls of X. Here $B(x, \varepsilon) := \{y \in X : d(x, y) < \varepsilon\}$ and $\overline{B}(x, \varepsilon) := \{y \in X : d(x, y) \leq \varepsilon\}$ and $\langle i, j\rangle \in \mathbb{N} := \frac{1}{2}(i + j)(i + j + 1) + j$ denotes the value of the Cantor pairing function applied to (i, j). We recall that a separable metric space (X, d) with a numbering α of a dense subset is called *computable metric space*, if $d \circ (\alpha \times \alpha)$ is a computable double sequence of reals. For technical simplicity we always assume that computable metric spaces are non-empty. For a computable metric space the following formal properties are recursively enumerable (r.e.):

(1) (Inclusion) $I_{\langle i_1,j_1\rangle} \prec I_{\langle i_2,j_2\rangle} :\iff d(\alpha(i_1), \alpha(i_2)) + \overline{j_1} < \overline{j_2}$,
(2) (Disjointness) $I_{\langle i_1,j_1\rangle} \bowtie I_{\langle i_2,j_2\rangle} :\iff d(\alpha(i_1), \alpha(i_2)) > \overline{j_1} + \overline{j_2}$.

Strictly speaking, these properties are not properties of the respective balls I_n, but of their numbers n. However, for many spaces, such as Banach spaces, this makes no difference. For these spaces the formal properties are equivalent to their material counterparts: $I_{n_1} \prec I_{n_2} \iff \widehat{I}_{n_1} \subsetneqq I_{n_2} \iff \widehat{I}_{n_1} \subseteq I_{n_2}$ and similarly $I_{n_1} \bowtie I_{n_2} \iff I_{n_1} \cap I_{n_2} = \emptyset$. The following definition now provides two standard ways to equip the space $\mathcal{A}(X)$ with representations.

Definition 2 (Standard representations of closed sets). *Let X be a computable metric space. Let us consider the following two sequences of sets in $\mathcal{A}(X)$:*

(1) $I_n^+ := \{A \in \mathcal{A}(X) : A \cap I_n \neq \emptyset\}$,
(2) $I_n^> := \{A \in \mathcal{A}(X) : A \cap \widehat{I}_n = \emptyset\}$.

Then $(I_n^+)_{n\in\mathbb{N}}$ and $(I_n^>)_{n\in\mathbb{N}}$ are subbases of topologies on $\mathcal{A}(X)$ and the induced standard representations are denoted by ψ_+ and $\psi_>$. The join of both representations is denoted by $\psi_= := \psi_+ \wedge \psi_>$.

The representation ψ_+ uses positive information in order to represent closed subsets, as $\psi_+(p) = A$ if and only if p is a list of all rational open balls I_n that intersect A and any such ball includes a positive information on A. Similarly, $\psi_>$ uses negative information in order to represent closed subsets, as $\psi_>(p) = A$ if and only if p is a list of all rational closed balls $\widehat{I}_n$ that do not intersect A and any such ball includes a piece of negative information on A. The representation $\psi_=$ includes both types of information.

One can show that ψ_+ is admissible with respect to the *lower Fell topology* that is induced by the subbase consisting of the sets $U^+ := \{A \in \mathcal{A}(X) : A \cap U \neq \emptyset\}$ where U ranges over all open sets $U \subseteq X$ (see Propositions 4.4.4(2) and 4.4.7 in [15]).

Now we recall definitions of representations of closed subsets via their distance functions $d_A : M \to \overline{\mathbb{R}}$, where $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, \infty\}$ is the set of extended real numbers. For non-empty closed subsets $A \subseteq M$ we define $d_A(x) := d(x, A) := \inf\{d(x, a) \mid a \in A\}$ and for technical reasons we let $d_\emptyset : M \to \overline{\mathbb{R}}, x \mapsto \infty$. We recall that for any computable metric space X there is a *Cauchy representation* δ_X that represents points in X by rapidly converging sequences in the dense subset given by α.

Definition 3 (Representations of closed sets by distance functions). *Let (X, d) be a computable metric space with Cauchy representation δ. We define representations $\psi_+^{\text{dist}}, \psi_-^{\text{dist}}, \psi^{\text{dist}}$ of $\mathcal{A}(X)$ by*

(1) $\psi_+^{\text{dist}}(p) = A :\iff [\delta \to \overline{\rho_>}](p) = d_A$,
(2) $\psi_-^{\text{dist}}(p) = A :\iff [\delta \to \overline{\rho_<}](p) = d_A$,
(3) $\psi^{\text{dist}}(p) = A :\iff [\delta \to \overline{\rho}](p) = d_A$,

for all $p \in \mathbb{N}^\mathbb{N}$ and $A \in \mathcal{A}(X)$.

Here $[\delta \to \delta']$ denotes the canonical function space representation of the space of (δ, δ')–continuous functions [1] and $\overline{\rho_>}$, $\overline{\rho_<}$ and $\overline{\rho}$ denote certain standard representations of $\overline{\mathbb{R}}$, which roughly speaking represent real numbers by upper, lower rational bounds and rational intervals, respectively [16,1]. The corresponding representations of $\mathbb{R}$ are denoted by $\rho_>, \rho_<, \rho$, respectively. It is clear that $\psi^{\text{dist}} \equiv \psi_+^{\text{dist}} \wedge \psi_-^{\text{dist}}$. The following equivalence has been proved in [2].

Proposition 1. $\psi_+ \equiv \psi_+^{\text{dist}}$ *for any computable metric space* X*.*

We recall some further definitions of representations of closed subsets. For the Sierpiński representation we use the *characteristic function*

$$\text{cf}_A : X \to \mathbb{R}, x \mapsto \begin{cases} 0 \text{ if } x \in A \\ 1 \text{ otherwise} \end{cases},$$

defined for any set $A \subseteq X$.

Definition 4 (Further representations of closed sets). Let X be a computable metric space with Cauchy representation δ_X. We define representations ψ_-, ψ^{fiber}, $\psi^{\text{Sierpiński}}$, ψ^{range} of $\mathcal{A}(X)$ by

(1) $\psi_-(p) = X \setminus \bigcup_{n+1 \in \text{range}(p)} I_n$,
(2) $\psi^{\text{fiber}}(p) = A :\iff [\delta_X \to \rho](p) = f : X \to \mathbb{R}$ and $f^{-1}\{0\} = A$,
(3) $\psi^{\text{Sierpiński}}(p) = A \iff [\delta_X \to \rho_<](p) = \text{cf}_A$,
(4) $\psi^{\text{range}}(p) = A :\iff ((\exists n \in \mathbb{N})(\exists q \in \mathbb{N}^{\mathbb{N}})p = 0^n 1q$ and $[\delta_{\mathbb{N}} \to \delta_X](q) = f$ and $\overline{\text{range}(f)} = A)$ or $(p = 0^\omega$ and $A = \emptyset)$,

for any $p \in \mathbb{N}^{\mathbb{N}}$ and $A \in \mathcal{A}(X)$.

Here $\delta_{\mathbb{N}}$ is supposed to be some standard representation of the natural numbers. The first three representations can be considered as representations by negative information. The representation ψ_- represents a closed set by exhausting its complement by rational open balls. The representation ψ^{fiber} represents closed sets via zero sets of real-valued functions and $\psi^{\text{Sierpiński}}$ represents closed sets via their characteristic function. Finally, ψ^{range} represents closed sets by sequences that are dense in the set and hence it is a representation by positive information. We obtain a representation with respect to full information by $\psi := \psi_+ \wedge \psi_-$. The following equivalences have been proved in [2] (see Theorems 3.8 and 3.10).

Proposition 2. $\psi_- \equiv \psi^{\text{fiber}} \equiv \psi^{\text{Sierpiński}}$ *holds for any computable metric space* X *and if* X*, additionally, is complete then* $\psi_+ \equiv \psi^{\text{range}}$*.*

In case of Euclidean space the first three representations are even equivalent to $\psi_>$ and ψ_-^{dist}. However, in the general case the equivalence class of ψ_- seems to be the most natural one. One can show that ψ_- is admissible with respect to the *upper Fell topology* that is induced by the subbase consisting of the sets $K^- := \{A \in \mathcal{A}(X) : A \cap K = \emptyset\}$ where K ranges over all compact sets $K \subseteq X$ (see Propositions 4.4.1 and 4.4.3 in [15]).

Figure 1 contains a survey on the introduced representations of closed subsets and besides the representations it also mentions the names that are used for the induced computable subsets. Any arrow in the diagram stands for a computable reduction and no arrow can be reversed in general nor can any additional arrows be added (up to transitivity). These results have been provided in [2,17]. The arrows do also indicate the logical relations between the given notions of computability for subsets.

$$\begin{tikzcd}
 & \begin{array}{c}\psi\\ \text{recursive}\end{array} \arrow[dl] \arrow[r] & \begin{array}{c}\psi_- \equiv \psi^{\text{fiber}} \equiv \psi^{\text{Sierpiński}}\\ \text{co-r.e.}\end{array} \\
\begin{array}{c}\psi_+ \equiv \psi_+^{\text{dist}}\\ \text{r.e.=upper semi-located}\end{array} & \begin{array}{c}\psi^{\text{dist}}\\ \text{located}\end{array} \arrow[u] \arrow[l] \arrow[r] & \begin{array}{c}\psi_-^{\text{dist}}\\ \text{lower semi-located}\end{array} \arrow[u] \\
\begin{array}{c}\psi^{\text{range}}\\ \text{effectively separable}\end{array} \arrow[u] & \begin{array}{c}\psi_=\\ \text{strongly recursive}\end{array} \arrow[u] \arrow[r] & \begin{array}{c}\psi_>\\ \text{strongly co-r.e.}\end{array} \arrow[u]
\end{tikzcd}$$

Fig. 1. Representations and notions of computability for closed subsets

The three vertical layers of the diagram correspond to positive, full and negative information on sets (from left to right). In the special situation of the Euclidean space (or, more general, of spaces that satisfy the effective covering property and that have compact closed balls) the three horizontal layers of the diagram collapse to a single layer [2]. Some special conditions might simplify the lattice of representations. We will say that a metric d is *isolated*, if all points in $\text{range}(d) \setminus \{0\} \subseteq \mathbb{R}$ are isolated (with respect to the Euclidean topology). For instance the standard metrics for Cantor space $\{0,1\}^{\mathbb{N}}$ and Baire space $\mathbb{N}^{\mathbb{N}}$ are isolated. We will say that a metric space (X,d) has *nice closed balls*, if any ball $\widehat{I_n}$ is either compact or the entire space. In these cases we obtain the following result (see Theorem 3.9(2) in [2]).

Proposition 3. *If (X,d) is a computable metric space with nice closed balls or with isolated d, then $\psi_-^{\text{dist}} \equiv \psi_>$ and $\psi^{\text{dist}} \equiv \psi_=$.*

We close this section with a result that will be applied in later sections. It shows that computability properties of closed subsets are preserved by embeddings in some sense. We will say that $\iota : X \hookrightarrow Y$ is a *computable embedding* if ι is injective and ι as well as its partial inverse $\iota^{-1} :\subseteq Y \to X$ are computable.

Theorem 2 (Embedding Theorem). *Let X, Y be computable metric spaces and let $\iota : X \hookrightarrow Y$ be a computable embedding and let $\text{range}(\iota)$ be co-r.e. closed in Y. Then the map $J : \mathcal{A}(X) \to \mathcal{A}(Y), A \mapsto \iota(A)$ is computable and admits a partial computable right inverse, both with respect to (ψ, ψ). The same holds true with "continuous" in place of "computable" and "closed" in place of "co-r.e. closed".*

4 The Borel Lattice of Representations

In this section we want to study the lattice of Borel structures that is induced by our representations. For this purpose we have to equip the set $\mathcal{A}(X)$ of closed subsets of a separable metric space X with a Borel structure. In general, if Y is a topological space, then we denote by $\mathbf{B}(Y)$ the class of *Borel subsets* of Y, which is the smallest σ–algebra generated by the open sets in Y.

If X is a metric space, then we equip the set $\mathcal{A}(X)$ of closed subsets of X with the *Effros Borel structure* $\mathbf{B}(\mathcal{A}(X))$, which is the Borel structure induced by the lower Fell topology. In fact, for separable metric spaces X this is the same Borel structure induced by the Fell topology, as the following lemma shows. The *Fell topology* is the join of the lower and upper Fell topology (i.e. a subbase consists of the union of the respective subbases).

Lemma 1. *If X is a separable metric space, then the Effros Borel structure $\mathbf{B}(\mathcal{A}(X))$ is identical to the Borel structure induced by the Fell topology on $\mathcal{A}(X)$.*

It is known that the upper Fell topology induces a different Borel structure in general. It is known that for a metric space X the Fell topology on $\mathcal{A}(X)$ is metrizable if and only if X is hemicompact [18]. However, even if $\mathcal{A}(X)$ is not metrizable, there might be some Polish topology on $\mathcal{A}(X)$ that generates the same Borel sets $\mathbf{B}(\mathcal{A}(X))$. Borel structures $\mathbf{B}(Y)$ for which there exists a Polish topology on Y that generates the same Borel structure are called *standard*. In fact, it is known that for separable metrizable X the Borel space $\mathbf{B}(\mathcal{A}(X))$ is standard if and and only if X is the union of a Polish space and a K_σ–space [19]. In particular, $\mathbf{B}(\mathcal{A}(X))$ is standard if X is a Polish space (see [20] and Theorem 12.6 in [3]). In fact, if X is a Polish space then one can assume without loss of generality that the topology of X is induced by a totally bounded metric d and the Hausdorff metric associated with this metric defines a Polish topology on $\mathcal{A}(X)$ that induces the Effros Borel structure $\mathbf{B}(\mathcal{A}(X))$ (see [20,8]).

The next observation is that we can translate upper bounds into lower bounds with a $\mathbf{\Sigma}^0_2$–computable function and vice versa. That implies that our representations using lower or upper approximations of the distance function can be translated into each other with a $\mathbf{\Sigma}^0_2$–computable function.

Proposition 4. *Let X be a computable metric space. Then $\psi^{\mathrm{dist}}_{+} \leq_2 \psi^{\mathrm{dist}}$, $\psi^{\mathrm{dist}}_{-} \leq_2 \psi^{\mathrm{dist}}$ and consequently $\psi_{>} \leq_2 \psi_{=}$, $\psi_{+} \leq_2 \psi$ and $\psi_{+} \leq_2 \psi_{-}$.*

One should keep in mind that the relation "$\leq_2$" is not transitive and the corresponding relation "$\equiv_2$" is not an equivalence relation. However, if $\delta_1 \leq \delta_2$ and $\delta_2 \leq_2 \delta_3$ and $\delta_3 \leq \delta_4$, then $\delta_1 \leq_2 \delta_4$. In particular, the results above imply $\psi^{\mathrm{dist}}_{+} \equiv_2 \psi^{\mathrm{dist}} \equiv_2 \psi^{\mathrm{dist}}_{-}$ and $\psi^{\mathrm{dist}}_{+} \equiv_2 \psi^{\mathrm{dist}}_{-}$. Now we obtain the following result that shows that in general there are at most three different Borel structures induced by our hyperspace representations.

Corollary 1. *Let X be a computable Polish space. Then we obtain the following three Borel equivalence classes of hyperspace representations:*

(1) $\psi_{+} \equiv \psi^{\mathrm{dist}}_{+} \equiv \psi^{\mathrm{range}} \equiv_{\mathrm{B}} \psi \equiv_{\mathrm{B}} \psi^{\mathrm{dist}} \equiv_{\mathrm{B}} \psi^{\mathrm{dist}}_{-}$,
(2) $\psi_{=} \equiv_{\mathrm{B}} \psi_{>}$,
(3) ψ_{-}.

We will see in the next section that for some spaces the three Borel equivalence classes mentioned in this result are actually distinct. In general, if $\delta \leq_{\mathrm{B}} \delta'$ holds for two representations of X, one can conclude that $\mathbf{B}(X, \delta') \subseteq \mathbf{B}(X, \delta)$, where $\mathbf{B}(X, \delta')$ and $\mathbf{B}(X, \delta)$ are the Borel structures induced by the final topologies of the representations δ' and δ, respectively. In particular, the equivalence $\psi_+ \equiv_{\mathrm{B}} \psi$ can be seen as an effective version of Lemma 1. Those representations that are listed under (1) in the corollary above have final topologies that all induce the Effros Borel structure, while those listed under (2) and (3) induce two further Borel structures in general.

We close this section with a result that shows that in the special situation of effectively locally compact spaces, we can obtain further Borel reductions. In order to formulate this we first define effective local compactness. Therefore, we consider the set $\mathcal{K}(X)$ of non-empty compact subsets of X as computable metric space equipped with the Hausdorff metric (for details see [1] or [2]). Classically, a separable metric space is locally compact, if it has a base consisting of relatively compact sets. We formulate an effective version of this characterization.

Definition 5 (Effectively locally compact). *Let X be a computable metric space. Then X is said to be* effectively locally compact*, if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $(I_{f(n)})_{n\in\mathbb{N}}$ is a basis of X and $(\widehat{I}_{f(n)})_{n\in\mathbb{N}}$ is a computable sequence in the computable metric space $\mathcal{K}(X)$.*

It is clear that in an effectively locally compact metric space for any $x \in X$ we can find some $n \in \mathbb{N}$ such that I_n is a relatively compact neighbourhood of x and such that a name of $\widehat{I}_n \in \mathcal{K}(X)$ can be computed. Effectively locally compact metric spaces X satisfy an effective covering property in the sense that the set $\left\{\langle k, \langle n_0, ..., n_i\rangle\rangle \in \mathbb{N} : \widehat{I}_{f(k)} \subseteq \bigcup_{j=0}^{i} I_{n_j}\right\}$ is r.e. Cantor space $\{0,1\}^{\mathbb{N}}$ and Euclidean space $\mathbb{R}^n$ are examples of effectively locally compact spaces.

Proposition 5. *If X is an effectively locally compact computable metric space, then $\psi_- \leq_2 \psi_+$ and $\psi_- \leq_2 \psi$.*

As a non-effective corollary we obtain the following.

Corollary 2. *If X is a locally compact separable metric space, then the Effros Borel structure $\mathbf{B}(\mathcal{A}(X))$ is identical to the Borel structure induced by the upper Fell topology on $\mathcal{A}(X)$.*

5 Perfect and Proper Spaces

For completeness and hardness results that we want to state, it is helpful to have some notions that expresses that a metric space is non-trivial in some sense. Among others, we will use the following concept.

Definition 6 (Richness). *A computable metric space X is called* rich*, if the Cantor space $\{0,1\}^{\mathbb{N}}$ can be computably embedded into X, i.e. if there is a computable injective map $\iota : \{0,1\}^{\mathbb{N}} \hookrightarrow X$.*

Whenever there is a computable injective $\iota : \{0,1\}^{\mathbb{N}} \hookrightarrow X$, then the partial inverse $\iota^{-1} :\subseteq X \to \{0,1\}^{\mathbb{N}}$ is automatically computable (see Corollary 6.3 in [21]). Thus, ι is a computable embedding and as $\{0,1\}^{\mathbb{N}}$ is recursive compact, range(ι) $= \iota(\{0,1\}^{\mathbb{N}})$ is recursive compact too and, in particular, co-r.e. closed in X. Thus, one can apply the Embedding Theorem 2 to ι. This allows to treat Cantor space as a representative for rich spaces, as far as computability results for closed subsets are concerned that are about operations which are preserved by the embedding (such as intersection). Many typical spaces such as Euclidean space $\mathbb{R}^n$ are rich. The following result shows that a large class of spaces is rich. We recall that a metric space is called *perfect*, if it does not contain any isolated points.

Proposition 6. *Any non-empty perfect computable Polish space X is rich.*

The construction required for this proof can be considered as an effective Cantor scheme (see Theorem 6.2 in [3]). In particular, any non-trivial computable Banach space is rich. Sometimes the only property of a perfect space that is required is captured by the following observation that is based on the previous construction.

Proposition 7. *Let X be a non-empty perfect computable Polish space. Then there exist computable functions $g, h : \mathbb{N} \to \mathbb{N}$ such that $\emptyset \neq \widehat{I}_{g(i)} \subseteq I_{h(i)}$ for all i, the $I_{h(i)}$ are pairwise disjoint and $\overline{\bigcup_{i=0}^{\infty} \widehat{I}_{g(i)}}$ is co-r.e. closed.*

It is not clear whether the set $\overline{\bigcup_{i=0}^{\infty} \widehat{I}_{g(i)}}$ is also recursive closed. In general, the balls $\overline{I_n}$ are r.e. closed and the balls $\widehat{I}_n$ are co-r.e. closed. We will say that a metric space X is *proper*, if closed balls are closures of open balls, i.e. if $\overline{B}(x,\varepsilon) = \overline{B(x,\varepsilon)}$ for any $x \in X$ and $\varepsilon > 0$. That is, for proper spaces $(I_n)_{n\in\mathbb{N}}$ is a ψ–computable sequence and in this situation the set in the previous proposition is clearly also r.e. closed.

Corollary 3. *Let X be a non-empty perfect and proper computable Polish space. Then there exist computable functions $g, h : \mathbb{N} \to \mathbb{N}$ such that $\emptyset \neq \overline{I_{g(i)}} \subseteq I_{h(i)}$ for all i, the $I_{h(i)}$ are pairwise disjoint and $\overline{\bigcup_{i=0}^{\infty} \overline{I_{g(i)}}}$ is recursive closed.*

The condition of this corollary is satisfied, for instance, for all non-empty computable Banach spaces. For simplicity we will formulate all our hardness and completeness results either for perfect or for perfect and proper Polish spaces and we will not attempt to generalize these conditions as far as possible. We also recall that all computable metric spaces are considered as non-empty and we will not mention this condition explicitly in the following. We will say that a function $f :\subseteq X \to Y$ is $\mathbf{\Sigma}_2^0$*–hard* with respect to certain representations, if the function

$$C : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}, C(p)(n) = \begin{cases} 0 \text{ if } (\exists k)\ p(k) = n+1 \\ 1 \text{ if } (\forall k)\ p(k) \neq n+1 \end{cases}$$

can be computably reduced to any realizer F of f with respect to the same representations, i.e. if there are computable functions $A :\subseteq \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and

$B :\subseteq \mathbb{N} \to \mathbb{N}$ such that $C(p) = A(p, FB(p))$ for all $p \in \mathbb{N}^{\mathbb{N}}$. The function C translates enumerations of sets into their characteristic functions. Analogously, we say that f is $\mathbf{\Sigma}^0_3$*–hard*, if the function

$$C_2 : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}, C_2(p)(n) = \begin{cases} 0 \text{ if } (\exists m)(\forall k)\ p\langle n,m,k\rangle \neq 0 \\ 1 \text{ if } (\forall m)(\exists k)\ p\langle n,m,k\rangle = 0 \end{cases}$$

can be reduced to any realizer F of f. If f is $\mathbf{\Sigma}^0_2$–hard and $\mathbf{\Sigma}^0_2$–computable, then it is called $\mathbf{\Sigma}^0_2$*–complete*. Analogously, we define the concept of $\mathbf{\Sigma}^0_3$–completeness. See [5] for a motivation of these definitions (among other things, the functions C and C_2 are complete in their respective classes with respect to the reducibility mentioned above).

6 Intersection and Union

In this section we will continue to study the Borel complexity of basic set theoretic operations as union and intersection. From now on we will concentrate on the representations ψ_+, ψ_-, ψ that are the most important ones from the point of view of applications.

Theorem 3 (Intersection). *Let X be a computable metric space. Then intersection $\cap : \mathcal{A}(X) \times \mathcal{A}(X) \to \mathcal{A}(X), (A,B) \mapsto A \cap B$ is*

(1) computable with respect to (ψ_-, ψ_-, ψ_-),
(2) $\mathbf{\Sigma}^0_2$–computable with respect to (ψ_+, ψ_+, ψ_-),
(3) $\mathbf{\Sigma}^0_2$–computable w.r.t. (ψ_-, ψ_-, ψ), if X is effectively locally compact,
(4) $\mathbf{\Sigma}^0_3$–computable w.r.t. (ψ_+, ψ_+, ψ), if X is effectively locally compact,
(5) $\mathbf{\Sigma}^0_3$–hard with respect to (ψ_+, ψ_+, ψ_+), if X is complete and perfect,
(6) $\mathbf{\Sigma}^0_2$–hard with respect to (ψ, ψ, ψ_+), if X is complete and perfect,
(7) not Borel measurable w.r.t. (ψ, ψ, ψ_+), if X is complete but not K_σ.

We mention that all our computability, completeness and hardness results have non-uniform implications. By the Invariance Theorem 8.3 in [5] any $\mathbf{\Sigma}^0_k$–computable function $f : X \to Y$ maps computable inputs x to Δ^0_k–computable outputs $f(x)$ with respect to the arithmetical hierarchy. If f is $\mathbf{\Sigma}^0_{k+1}$–hard, then there exists a computable input x that is mapped to an output $f(x)$ that is not Δ^0_k–computable with respect to the arithmetical hierarchy. Being Δ^0_{k+1}–computable is the same as being computable in the k–th jump $\emptyset^{(k)}$. We obtain the following corollary of the previous theorem (we just formulate a selection of consequences).

Corollary 4. *Let X be a computable and perfect Polish space. Then there exist recursive closed sets $A, B \subseteq X$ such that $A \cap B$ is not r.e. closed and there exist r.e. closed sets $A, B \subseteq X$ such that $A \cap B$ is not even r.e. closed in the halting problem $\emptyset'$. The intersection of r.e. closed sets is always co-r.e. closed in the halting problem $\emptyset'$ and the intersection of co-r.e. closed sets is always co-r.e. closed.*

Compared to intersection, union is a very well-behaved operation. It turns out to be computable with respect to any of the considered representations.

Theorem 4 (Union). *Let X be a computable metric space. Then union $\cup : \mathcal{A}(X) \times \mathcal{A}(X) \to \mathcal{A}(X), (A, B) \mapsto A \cup B$ is computable with respect to (δ, δ, δ) for any choice of δ among ψ_+, $\psi_+^{\rm dist}$, $\psi^{\rm range}$, ψ, $\psi^{\rm dist}$, ψ_-, $\psi_-^{\rm dist}$ and $\psi_>, \psi_=$.*

7 Complement, Interior and Closure

In this section we study the operators that map closed sets to the closure of their complement and to the closure of their interior, respectively.

Theorem 5 (Closure of the complement). *Let (X, d) be a computable metric space. Then the closure of the complement $c : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto \overline{A^{\rm c}}$ is*

(1) computable with respect to (ψ_-, ψ_+),
(2) $\mathbf{\Sigma}_2^0$–computable with respect to (ψ_+, ψ_+) and (ψ_-, ψ),
(3) $\mathbf{\Sigma}_2^0$–complete with respect to (ψ_+, ψ_+), if X is complete and perfect,
(4) $\mathbf{\Sigma}_2^0$–complete with respect to (ψ, ψ_-), if X is complete, perfect and proper.

As $A^\circ = \overline{A^{\rm c}}^{\rm c}$, we can interpret the closure of the complement operation also as interior operation on closed sets or, dually, as closure operation on open sets. We formulate a number of non-uniform consequences of the previous result.

Corollary 5. *Let X be a computable, perfect and proper Polish space. Then there exists a recursive closed $A \subseteq X$ such that $\overline{A^{\rm c}}$ is not co-r.e. closed, but $\overline{A^{\rm c}}$ is always co-r.e. closed in the halting problem $\emptyset'$. There exists a r.e. closed $A \subseteq X$ such that $\overline{A^{\rm c}}$ is not r.e. closed, but $\overline{A^{\rm c}}$ is always r.e. closed in the halting problem $\emptyset'$.*

Now we discuss the closure of the interior operator.

Theorem 6 (Closure of the interior). *Let X be a computable metric space. Then the closure of the interior $i : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto \overline{A^\circ}$ is*

(1) $\mathbf{\Sigma}_2^0$–computable with respect to (ψ_-, ψ_+),
(2) $\mathbf{\Sigma}_3^0$–computable with respect to (ψ_+, ψ_+) and (ψ_-, ψ),
(3) $\mathbf{\Sigma}_3^0$–complete with respect to (ψ_+, ψ_+), if X is complete and perfect,
(4) $\mathbf{\Sigma}_3^0$–complete with respect to (ψ, ψ_-), if X is complete, perfect and proper,
(5) $\mathbf{\Sigma}_2^0$–complete with respect to (ψ, ψ_+), if X is complete, perfect and proper.

One could assume that further iterations of the operation c allow to climb up the Borel hierarchy. However, as $ccc = c$, this is not possible. We mention some non-uniform consequences of the previous theorem.

Corollary 6. *Let X be a computable, perfect and proper Polish space. Then there exists a recursive closed $A \subseteq X$ such that $\overline{A^\circ}$ is not r.e. closed, but $\overline{A^\circ}$ is always r.e. closed in the halting problem $\emptyset'$. There exists a recursive closed $A \subseteq X$ such that $\overline{A^\circ}$ is not even co-r.e. closed in the halting problem $\emptyset'$, but $\overline{A^\circ}$ is always co-r.e. closed in $\emptyset''$.*

8 Boundary and Derivative

In this section we mainly want to study computability properties of the boundary operation and the derived set operation (i.e. the derivative). It turns out that for the boundary operation another underlying topological property of the space is helpful. We use the following notion of effective local connectedness (that is equivalent to the one in [21]). We use the representation ϑ of open subset, defined by $\vartheta(p) := X \setminus \psi_-(p)$.

Definition 7 (Effective local connectedness). *A computable metric space X is called* effectively locally connected *if there is a sequence $(U_{k,n})_{\langle k,n\rangle \in \mathbb{N}}$ of open connected sets that is computable with respect to ϑ and such that for any fixed $n \in \mathbb{N}$ the set $\{U_{k,n} : k \in \mathbb{N}\}$ contains neigbourhoods of any $x \in I_n$ and $U_{k,n} \subseteq I_n$ for all $k \in \mathbb{N}$.*

We recall that a point $x \in X$ is called a *boundary point* of a set $A \subseteq X$ if any neigbourhood of x contains a point of A and a point of $X \setminus A$. By ∂A we denote the *boundary* of A, i.e. the set of boundary points of A. Here and in the following we exploit that $\partial A = \overline{A} \cap \overline{A^c}$. It is clear that the boundary of a set is always closed. Now we are prepared to formulate the following result on the boundary operator.

Theorem 7 (Boundary). *Let X be a computable metric space. Then the boundary $\partial : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto \partial A$ is*

(1) computable with respect to (ψ, ψ_+), if X is effectively locally connected,
(2) $\mathbf{\Sigma}^0_2$–computable with respect to (ψ_+, ψ_+) and (ψ, ψ), if X is effectively locally connected,
(3) $\mathbf{\Sigma}^0_2$–computable with respect to (ψ_-, ψ_-),
(4) $\mathbf{\Sigma}^0_3$–computable with respect to (ψ_-, ψ), if X is effectively locally compact,
(5) $\mathbf{\Sigma}^0_2$–computable with respect to (ψ_-, ψ), if X is effectively locally connected and effectively locally compact,
(6) $\mathbf{\Sigma}^0_2$–complete with respect to (ψ, ψ_-), if X is complete, perfect and proper,
(7) $\mathbf{\Sigma}^0_3$–complete with respect to (ψ, ψ_+), if $X = \{0,1\}^{\mathbb{N}}$,
(8) not Borel measurable with respect to (ψ, ψ_+), if $X = \mathbb{N}^{\mathbb{N}}$.

By iteration of the boundary operator nothing new can be obtained as $\partial\partial A = \partial A$ for closed $A \subseteq X$. We recall that a point $x \in X$ in a topological space X is called an *accumulation point* of a subset $A \subseteq X$ if any only if any neigbourhood of X contains a point of A other than x. By A' we denote the *derived set* of A, which is the set of accumulation points of A. It is clear that the derived set of a set A is always closed. Now we can formulate our result on the derivative.

Theorem 8 (Derivative). *Let X be a computable metric space. Then the derivative $d : \mathcal{A}(X) \to \mathcal{A}(X), A \mapsto A'$ is*

(1) $\mathbf{\Sigma}^0_2$–computable with respect to (ψ_+, ψ_-),
(2) $\mathbf{\Sigma}^0_3$–computable with respect to (ψ_+, ψ) and (ψ_-, ψ_-), if X is effectively locally compact,

(3) $\mathbf{\Sigma}^0_2$*–complete with respect to* (ψ, ψ_-)*, if* X *is complete and perfect,*
(4) $\mathbf{\Sigma}^0_3$*–hard with respect to* (ψ_-, ψ_-)*, if* X *is complete and perfect,*
(5) $\mathbf{\Sigma}^0_3$*–hard with respect to* (ψ, ψ_+)*, if* X *is complete and perfect,*
(6) not Borel measurable with respect to (ψ, ψ_+)*, if* X *is complete but not* K_σ*.*

We formulate a typical non-uniform corollary.

Corollary 7. *Let* X *be a computable and perfect Polish space. Then there exists a recursive closed* $A \subseteq X$ *such that* A' *is not r.e. closed in the halting problem* $\emptyset'$*, but any such* A' *is co-r.e. closed in the halting problem* $\emptyset'$*.*

Further results regarding the derivative and its iteration can be found in [22,23]. In particular, it is proved that d^k is not $\mathbf{\Sigma}^0_{2k}$–measurable with respect to (ψ, ψ) on Cantor space $\{0,1\}^{\mathbb{N}}$ (restricted to compact sets). It is an interesting question whether we could establish a corresponding completeness result.

9 Conclusions

The table in Figure 2 shows degrees of computability of topological operations $f : \mathcal{A}(X)^i \to \mathcal{A}(X)$ with $i \in \{1, 2\}$ with respect to (ψ^i, ψ) and some common spaces X. The columns indicate the space X under consideration and the rows indicate which operation is treated. Any number $k \in \mathbb{N}$ in the table indicates $\mathbf{\Sigma}^0_k$–computability, typically even $\mathbf{\Sigma}^0_k$–completeness and ∞ stands for operations which are not even Borel measurable.

	$\mathbb{N}$	$\{0,1\}^{\mathbb{N}}$	$\mathbb{N}^{\mathbb{N}}$	$[0,1]$	$[0,1]^{\mathbb{N}}$	$\mathbb{R}^n$	$\mathbb{R}^{\mathbb{N}}$	ℓ_2	$\mathcal{C}[0,1]$
$A \cup B$	1	1	1	1	1	1	1	1	1
$A \cap B$	1	2	∞	2	2	2	∞	∞	∞
$\overline{A^c}$	1	2	2	2	2	2	2	2	2
$\overline{A^\circ}$	1	3	3	3	3	3	3	3	3
∂A	1	3	∞	2	2	2	2	2	2
A'	1	3	∞	3	3	3	∞	∞	∞

Fig. 2. Degrees of computability with respect to ψ

References

1. Weihrauch, K.: Computable Analysis. Springer, Heidelberg (2000)
2. Brattka, V., Presser, G.: Computability on subsets of metric spaces. Theoretical Computer Science 305, 43–76 (2003)
3. Kechris, A.S.: Classical Descriptive Set Theory. Volume 156 of Graduate Texts in Mathematics. Springer, Heidelberg (1995)
4. Moschovakis, Y.N.: Descriptive Set Theory. Volume 100 of Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam (1980)
5. Brattka, V.: Effective Borel measurability and reducibility of functions. Mathematical Logic Quarterly 51, 19–44 (2005)

6. Kuratowski, K.: Topology, vol. 1. Academic Press, London (1966)
7. Kuratowski, K.: Topology, vol. 2. Academic Press, London (1968)
8. Christensen, J.P.R.: On some properties of Effros Borel structure on spaces of closed subsets. Mathematische Annalen 195, 17–23 (1971)
9. Christensen, J.P.R.: Necessary and sufficient conditions for the measurability of certain sets of closed subsets. Mathematische Annalen 200, 189–193 (1973)
10. Christensen, J.P.R.: Topology and Borel Structure. North-Holland, Amsterdam (1974)
11. Brattka, V.: Computable invariance. Theoretical Computer Science 210, 3–20 (1999)
12. Gherardi, G.: Effective Borel degrees of some topological functions. Mathematical Logic Quarterly 52, 625–642 (2006)
13. Weihrauch, K.: Computability. Volume 9 of EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg (1987)
14. Kreitz, C., Weihrauch, K.: Theory of representations. Theoretical Computer Science 38, 35–53 (1985)
15. Schröder, M.: Extended admissibility. Theoretical Computer Science 284, 519–538 (2002)
16. Brattka, V., Weihrauch, K.: Computability on subsets of Euclidean space I: Closed and compact subsets. Theoretical Computer Science 219, 65–93 (1999)
17. Brattka, V.: Random numbers and an incomplete immune recursive set. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 8–13. Springer, Heidelberg (2002)
18. Holá, L., Pelant, J., Zsilinszky, L.: Developable hyperspaces are metrizable. Applied General Topology 4, 351–360 (2003)
19. Raymond, J.S.: La structure borélienne d'Effros est-elle standard? Fundamenta Mathematicae 100, 201–210 (1978)
20. Effros, E.G.: Convergence of closed subsets in a topological space. Proceedings of the American Mathematical Society 16, 929–931 (1965)
21. Brattka, V.: Plottable real number functions. In: Daumas, M., et al. (eds.) RNC'5 Real Numbers and Computers, INRIA, Institut National de Recherche en Informatique et en Automatique 13–30 Lyon, September 3–5, 2003 (2003)
22. Cenzer, D., Mauldin, R.D.: On the Borel class of the derived set operator. Bull. Soc. Math. France 110, 357–380 (1982)
23. Cenzer, D., Mauldin, R.D.: On the Borel class of the derived set operator: II. Bull. Soc. Math. France 111, 367–372 (1983)
24. Kuratowski, K.: Some remarks on the relation of classical set-valued mappings to the Baire classification. Colloquium Mathematicum 42, 273–277 (1979)

Colocatedness and Lebesgue Integrability

Douglas S. Bridges

Department of Mathematics & Statistics
University of Canterbury, Private Bag 4800
Christchurch, New Zealand
d.bridges@math.canterbury.ac.nz

Abstract. With reference to Mandelkern's characterisation of colocated subsets of the line in constructive analysis, we introduce the notion of "strongly colocated set" and find conditions under which such a set is Lebesgue integrable.

The work of this paper is motivated by the question, "When is an open set $S \subset \mathbb{R}$ Lebesgue measurable?" Classically, the answer to this question is "always"; but in constructive mathematics—mathematics with intuitionistic logic and an appropriate set theoretic foundation such as that in [1] — we may not be able to prove S Lebesgue measurable even if it is expressed as a disjoint union of open intervals.

We assume that the reader has access to an account of constructive measure theory such as that in Chapter 6 of [4]; for background in constructive analysis, see [3,4,6] or the more recent [5].

A subset S of $\mathbb{R}$ is said to be

– **fixative** if we can decide whether it is empty or inhabited;[1]
– **located** if
$$\rho(x,S) = \inf\{|x-s| : s \in S\}$$
exists for each $x \in \mathbb{R}$;
– **colocated** if there exists a located subset T of $\mathbb{R}$ such that
$$S = \mathbb{R} - T = \{x \in \mathbb{R} : \rho(x,T) > 0\}.$$

Mandelkern [7] established the following characterisation of colocated subsets of $\mathbb{R}$.

Theorem 1. *The union of a sequence $(I_n)_{n\geqslant 1}$ of pairwise-disjoint, fixative open intervals in $\mathbb{R}$ is colocated if and only if there exists a strictly increasing sequence $(n_k)_{k\geqslant 1}$ of positive integers such that for all positive integers n and k, if I_n intersects $(-k,k)$ and $|I_n| > k^{-1}$, then $n \leqslant n_k$.*

[1] A set S is **inhabited** if we can construct a point in it. Such a construction tells us more than merely that it is impossible for S to be empty.

Let $S = \bigcup_{n \geqslant 1} I_n$ be a countable union of pairwise-disjoint, fixative open intervals. Strengthening the condition in Mandelkern's theorem, we say that S is **strongly colocated** if there exists a strictly increasing sequence $(n_k)_{k \geqslant 1}$ of positive integers satisfying the following condition:

(*) *For each $k \in \mathbb{N}^+$ (the set of positive integers), if F is a finite subset of $\mathbb{N}^+$ such that $\sum_{n \in F} |I_n| > k^{-1}$ and I_n meets $(-k, k)$ for each $n \in F$, then there exists $n \in F$ such that $n \leqslant n_k$.*

If S is strongly colocated, then, by Theorem 1, it is colocated.

Proposition 1. *Let $S = \bigcup_{n \geqslant 1} I_n$, where $(I_n)_{n \geqslant 1}$ is a sequence of pairwise-disjoint, fixative open intervals, and suppose that S is colocated. Let $(J_n)_{n \geqslant 1}$ also be a sequence of pairwise-disjoint, fixative open intervals whose union is S. Then there exists a strictly increasing sequence $(p_k)_{k \geqslant 1}$ of positive integers such that for each $k \in \mathbb{N}^+$, if F is a finite subset of $\mathbb{N}^+$ such that $\sum_{n \in F} |J_n| > k^{-1}$ and J_n meets $(-k, k)$ for each $n \in F$, then there exists $n \in F$ such that $n \leqslant p_k$.*

Proof. There exists a strictly increasing sequence $(n_k)_{k \geqslant 1}$ of positive integers with the property (*). We may assume that I_{n_1} is inhabited. First note that if J_n is inhabited, then there exists a unique m such that $J_n = I_m$. To see this, let $x, x' \in J_n$, and pick m, m' such that $x \in I_m$ and $x' \in I_{m'}$. Suppose that $m \neq m'$. Then without loss of generality we may assume that $x < x'$. Since $I_m \cap I_{m'} = \varnothing$, the right endpoint of I_m must lie between x and x' and hence in J_n; but this is absurd, since that endpoint cannot belong to S. Hence $m = m'$ and $J_n \subset I_m$. Interchanging the roles of J_n and I_m, we see also that $I_m \subset J_n$ and hence that $J_n = I_m$.

We now define the desired strictly increasing sequence $(p_k)_{k \geqslant 1}$ of positive integers recursively. We first set

$$p_1 = \max \{m : \exists_{n \leqslant n_1} (J_m = I_n)\}.$$

Having computed p_k, we set

$$p_{k+1} = \max \left\{1 + p_k, \max \left\{m : \exists_{n \leqslant n_{k+1}} (J_m = I_n)\right\}\right\}.$$

Let F be a finite subset of $\mathbb{N}^+$ such that $\sum_{n \in F} |J_n| > k^{-1}$ and J_n meets $(-k, k)$ for each $n \in F$. For such n, since J_n meets $(-k, k)$ and is therefore inhabited, there exists a unique m_n such that $J_n = I_{m_n}$. Thus $\sum_{n \in F} |I_{m_n}| > k^{-1}$ and I_{m_n} meets $(-k, k)$ for each $n \in F$. It follows that there exists $n \in F$ such that $m_n \leqslant n_k$; since $J_n = I_{m_n}$, we have $n \leqslant p_k$.

Theorem 2. *Let $S = \bigcup_{n \geqslant 1} I_n$ be a countable union of pairwise-disjoint, fixative open intervals. Then the following conditions are equivalent.*

(i) *S is strongly colocated.*
(ii) *The series $\sum_{n=1}^{\infty} |I_n \cap (-N, N)|$ converges in $\mathbb{R}$ for each positive integer N.*

Proof. Assuming (i), let $(n_k)_{k\geqslant 1}$ be as in the definition of *strongly colocated*, and let N be a positive integer. Let $0 \leqslant \alpha < \beta$, and choose a positive integer

$$k > \max\left\{N, \frac{2}{\beta-\alpha}\right\}.$$

Consider any finite subset F of $\mathbb{N}^+ \cap (n_k, \infty)$, and let $0 < \varepsilon < 1$. A tedious elementary argument by cases shows that for each n, either $I_n \cap (-N, N)$ is inhabited or else $I_n \cap (-N+\varepsilon, N-\varepsilon) = \varnothing$; so we can write $\mathbb{N}^+ \cap (n_k, \infty)$ as a union of subsets P, Q such that

- ▶ if $n \in P$, then I_n intersects $(-N, N)$;
- ▶ if $n \in Q$, then $I_n \cap (-N+\varepsilon, N-\varepsilon) = \varnothing$.

We lose no generality by assuming that both P and Q are inhabited. By the choice of n_k, we have $\sum_{n\in F\cap P} |I_n| \leqslant k^{-1}$ and therefore

$$\sum_{n\in F\cap P} |I_n \cap (-N, N)| \leqslant k^{-1}.$$

On the other hand, if $n \in F \cap Q$, then $|I_n \cap (-N, N)| \leqslant 2\varepsilon$. So

$$\sum_{n\in F\cap Q} |I_n \cap (-N, N)| \leqslant 2\,|F \cap Q|\,\varepsilon \leqslant 2\,|F|\,\varepsilon.$$

Hence

$$\sum_{n\in F} |I_n \cap (-N, N)| \leqslant k^{-1} + 2\,|F|\,\varepsilon.$$

Since ε is arbitrary, we see that $\sum_{n\in F} |I_n \cap (-N, N)| \leqslant k^{-1}$ for every finite subset F of $\mathbb{N}^+ \cap (n_k, \infty)$.

Either $\sum_{n=1}^{n_k} |I_n \cap (-N, N)| > \alpha$ or $\sum_{n=1}^{n_k} |I_n \cap (-N, N)| < \beta - k^{-1}$. In the latter case, for each $m > n_k$ we have

$$\sum_{n=1}^{m} |I_n \cap (-N, N)| = \sum_{n=1}^{n_k} |I_n \cap (-N, N)| + \sum_{n=n_{k+1}}^{m} |I_n \cap (-N, N)| < \beta,$$

by the first part of the proof. Since α, β are arbitrary, it follows from the constructive least-upper-bound principle ([5], Theorem 2.1.18) that

$$\sum_{n=1}^{\infty} |I_n \cap (-N, N)| = \sup_{m\geqslant 1} \sum_{n=1}^{m} |I_n \cap (-N, N)|$$

exists in $\mathbb{R}$.

Now suppose, conversely, that (ii) holds. Then there exists a strictly increasing sequence $(\nu_k)_{k\geqslant 1}$ of positive integers such that

$$\sum_{j=\nu_k+1}^{\infty} |I_j \cap (-k-1, k+1)| < k^{-1} \qquad (k \geqslant 1)\,.$$

Given a finite subset F of $\mathbb{N}^+ \cap (\nu_{k+1}, \infty)$, suppose that $\sum_{n\in F} |I_n| > k^{-1}$ and that I_n intersects $(-k, k)$ for each $n \in F$. Since

$$\begin{aligned}\mu\left(\bigcup_{n\in F}(I_n \cap (-k-1, k+1))\right) &= \sum_{n\in F} |I_n \cap (-k-1, k+1)| \\ &\leqslant \sum_{n=\nu_k+1}^{\infty} |I_n \cap (-k-1, k+1)| \\ &< k^{-1} < \sum_{n\in F} |I_n|,\end{aligned}$$

there exist $\nu \in F$ and a point $x \in I_\nu$ such that $x \notin I_\nu \cap (-k-1, k+1)$; then $|x| \geqslant k+1$. Since I_ν intersects $(-k, k)$ and is an interval, it follows that either $(-k-1, -k) \subset I_\nu$ or else $(k, k+1) \subset I_\nu$; whence

$$\sum_{n\in F} |I_n \cap (-k-2, k+2)| \geqslant |I_\nu \cap (-k-1, k+1)| \geqslant 1 > (k+1)^{-1},$$

which is absurd as $F \subset (\nu_{k+1}, \infty)$. We conclude that if F is a finite set of positive integers such that $\sum_{n\in F} |I_n| > k^{-1}$ and I_n intersects $(-k, k)$ for each $n \in F$, then there exists $n \in F$ with $n \leqslant \nu_{k+1}$. To complete the proof that S is strongly colocated, we need only take $n_k = \nu_{k+1}$.

Theorem 3. *Let $S = \bigcup_{n\geqslant 1} I_n$ be a countable union of pairwise-disjoint, fixative open intervals that is strongly colocated. Then S is Lebesgue integrable if and only if $\sum_{n=1}^{\infty} |I_n|$ converges, in which case the series converges to the Lebesgue measure of S.*

Proof. Assume first that S is Lebesgue integrable. Given $\varepsilon > 0$, we can find a positive integer N such that $\mu(S - [-N, N]) < \varepsilon$. By Theorem 2, there exists κ such that $\sum_{n=\kappa+1}^{\infty} |I_n \cap [-N, N]| < \varepsilon$. For every $k \geqslant \kappa$ we have

$$\begin{aligned}\mu(S) &= \mu(S - [-N, N]) + \mu(S \cap [-N, N]) \\ &< \varepsilon + \sum_{n=1}^{\infty} |I_n \cap [-N, N]| \\ &\leqslant \varepsilon + \sum_{n=1}^{k} |I_n| + \sum_{n=k+1}^{\infty} |I_n \cap [-N, N]| \\ &< \sum_{n=1}^{k} |I_n| + 2\varepsilon\end{aligned}$$

and therefore $\mu(S) - \sum_{n=1}^{k} |I_n| < 2\varepsilon$. Since $\varepsilon > 0$ is arbitrary, we conclude that $\sum_{n=1}^{\infty} |I_n|$ converges to $\mu(S)$.

If, conversely, $\sum_{n=1}^{\infty} |I_n|$ converges, then S is Lebesgue integrable, by (3.10) on page 235 of [4].

The following recursive counterexample shows that we cannot prove constructively that the union of any sequence of pairwise-disjoint, fixative open intervals is Lebesgue measurable. Assuming the Church–Markov–Turing thesis, we can construct a Specker sequence in $[0,1]$: that is, a strictly increasing sequence $(r_n)_{n\geqslant 1}$ in $[0,1]$ that is eventually bounded away from each real number. (For more information about Specker sequences see [8] or Chapter 3 of [6].) Suppose that the set

$$S=\bigcup_{n\geqslant 1}(r_n,r_{n+1})\,,$$

a countable union of pairwise-disjoint, inhabited open subintervals of $(0,1)$, is Lebesgue integrable. By Theorem (6.7) on page 257 of [4], for each $\varepsilon>0$ there exists a compact subset K of S such that $\mu(S-K)<\varepsilon$. Let $\xi=\sup K\in[0,1]$. By the Specker property of the sequence $(r_n)_{n\geqslant 1}$, there exist $\delta>0$ and a positive integer ν_1 such that $|\xi-r_n|>\delta$ for all $n\geqslant\nu_1$. Pick $\eta\in K\subset S$ such that $\eta>\xi-\delta$. There exists a unique ν_2 such that $\eta\in(r_{\nu_2},r_{\nu_2+1})$. Either $r_{\nu_2+1}>\xi$ or $r_{\nu_2+1}<\xi+\delta$. In the first case, $r_n>\xi$ for all $n\geqslant\nu_2$. In the second case, since $\xi-\delta<\eta<r_{\nu_2+1}<\xi+\delta$, we must have $\nu_2+1<\nu_1$; whence $r_{\nu_1}>r_{\nu_2+1}>\xi-\delta$ and therefore $r_n>r_{\nu_1}>\xi+\delta$ for all $n\geqslant\nu_1$. Thus, setting $N=\max\{\nu_1,\nu_2\}$, in either case we have $r_n>\xi+\delta$, and therefore $(r_n,r_{n+1})\in S-K$, for all $n\geqslant N$. It follows that

$$\sum_{k=N}^{n}|I_k|=\mu\left(\bigcup_{k=N}^{n}I_k\right)\leqslant\mu(S-K)<\varepsilon\quad(n\geqslant N)\,.$$

Since $\varepsilon>0$ is arbitrary, the series $\sum_{k=1}^{\infty}|I_k|$ converges. But for $n\geqslant 2$,

$$\begin{aligned}r_n&=r_1+\sum_{k=2}^{n}(r_k-r_{k-1})=r_1+\sum_{k=2}^{n}|I_k|\\&\to r_1+\sum_{k=1}^{\infty}|I_k|-|I_1|\ \text{ as } n\to\infty,\end{aligned}$$

so the Specker sequence $(r_n)_{n\geqslant 1}$ converges. This contradicts the defining property of a Specker sequence. Hence S is not Lebesgue integrable.

We next show that this set S is not colocated. Supposing that $S=-T$, where T is located, we prove that $\sup_{n\geqslant 1}r_n$ exists. In view of the constructive least-upper-bound principle ([5], Theorem 2.1.18), it will suffice to prove that for each $x\in\mathbb{R}$,

$$\forall_{n\geqslant 1}(r_n<x)\vee\exists_n(r_n>x)\,. \tag{1}$$

To that end, we compute $\delta>0$ and a positive integer N such that $|x-r_n|>\delta$ for all $n\geqslant N$. If $\rho(x,T)>0$, then there exists ν such that $x\in(r_\nu,r_{\nu+1})$ and therefore $r_{\nu+1}>x$. We may therefore assume that $\rho(x,T)<\delta$. Pick $t\in T$ with $|x-t|<\delta$. Since either $r_N>x$ or $r_N<x-\delta$, we may further assume the latter. If there exists $n>N$ such that $r_n>x$ and therefore $r_n>x+\delta$, then

$$t\in(x-\delta,x+\delta)\subset\overline{\bigcup_{k=N}^{n-1}(r_k,r_{k+1})}$$

and $t \neq y$ for all $y \in \bigcup_{k=N}^{n-1} (r_k, r_{k+1})$; whence $t = r_k$ for some $k \geqslant N$, which is impossible by our choice of δ. Hence $r_n \leqslant x$, and therefore $r_n < x - \delta$, for all $n \geqslant N$. By testing $r_1, \ldots, r_{N-1}$, we can now complete the proof of (1). Hence $\lim_{n\to\infty} r_n = \sup_{n\geqslant 1} r_n$ exists, which contradicts the Specker property of $(r_n)_{n\geqslant 1}$.

It follows immediately that S cannot be strongly colocated. We can reach this conclusion by a different route as follows. Suppose that S is strongly colocated. Then by Proposition 1, there exists a strictly increasing sequence $(n_k)_{k\geqslant 1}$ of positive integers with the property (*). For each positive integer k, since every I_n meets $(-k, k)$, we have $\sum_{n=n_k+1}^{m} |I_n| \leqslant k^{-1}$ whenever $m > n_k$. Hence the series $\sum_{n=1}^{\infty} |I_n|$ converges. It follows from Proposition (3.10) in Chapter 6 of [4] that S is Lebesgue integrable, which contradicts the work of the second-to-last paragraph.

These recursive considerations lead us to the question:

> If a union of pairwise-disjoint, fixative open intervals is both colocated and integrable, is it strongly colocated?

Here is a first step towards an affirmative answer.

Proposition 2. *Let S be the colocated union of a sequence $(I_n)_{n\geqslant 1}$ of pairwise-disjoint, fixative open intervals. If S is integrable, then $|I_n| \to 0$ as $n \to \infty$.*

Proof. Choose $(n_k)_{k\geqslant 1}$ as in Mandelkern's theorem. Given $\varepsilon > 0$, choose an integer $N > 2$ such that $\mu(S - [-N+1, N-1]) < \varepsilon$. Write $\mathbb{N}^+$ as a union of subsets P, Q such that

– if $n \in P$, then I_n intersects $(-N, N)$;
– if $n \in Q$, then $I_n \cap \left[-N + \frac{1}{2}, N - \frac{1}{2}\right] = \varnothing$.

Let k be any positive integer $> \max\left\{\frac{1}{\varepsilon}, N\right\}$, and consider any $n > n_k$. If $n \in Q$, then

$$I_n \subset S - [-N+1, N-1],$$

so $|I_n| < \varepsilon$. If $n \in P$ and $|I_n| > 1/k$, then as $I_n \cap (-k, k) \supset I_n \cap (-N, N)$, we have $n \leqslant n_k$, a contradiction; so $|I_n| \leqslant 1/k < \varepsilon$. It follows that $|I_n| < \varepsilon$ for all $n > n_k$.

We can strengthen the conclusion of Proposition 2 to the strong colocatedness of S provided we accept the following weak Heine–Borel property (a variant of which is discussed in [2]).

> **HB$_i$:** *For each compact $K \subset \mathbb{R}$, if $(I_n)_{n\geqslant 1}$ is a sequence of pairwise-disjoint inhabited open intervals, and $K \subset \bigcup_{n\geqslant 1} I_n$, then there exists N such that $K \subset \bigcup_{n=1}^{N} I_n$.*

Theorem 4. ***HB**$_i$* $\vdash$ *Every colocated, integrable union of a sequence of pairwise-disjoint, fixative open intervals is strongly colocated.*

Proof. Let S be a colocated integrable set, and, using Mandelkern's theorem, write S as the union of a sequence $(I_n)_{n\geqslant 1}$ of pairwise-disjoint, fixative open intervals. Without loss of generality, we may assume that each I_n is inhabited. By Theorem 2, in order to establish our desired result, it is enough to prove that the series $\sum_{n=1}^{\infty} |I_n|$, whose partial sums are bounded by $\mu(S)$, converges. Given $\varepsilon > 0$, we have either $\mu(S) < \varepsilon$, in which case $\sum_{n=j+1}^{k} |I_n| < \varepsilon$ whenever $k > j$; or else $\mu(S) > 0$. In the latter case, using Theorem (6.7) on page 257 of [4], we can find a strongly integrable compact set $K \subset S$ such that $\mu(S - K) < \varepsilon$. Then $K \subset \bigcup_{n\geqslant 1} I_n$, so we can apply $\mathbf{HB}_i$ to obtain a positive integer N such that $K \subset \bigcup_{n=1}^{N} I_n$. It follows that for $k > j > N$,

$$\sum_{n=j+1}^{k} |I_n| \leqslant \mu(S - K) < \varepsilon.$$

Hence in either case we have $\sum_{n=j+1}^{k} |I_n| < \varepsilon$ for all sufficiently large j and k. Since $\varepsilon > 0$ is arbitrary, we conclude that the partial sums of $\sum_{n=1}^{\infty} |I_n|$ form a Cauchy sequence, and therefore that the series converges in $\mathbb{R}$.

Acknowledgement. The author thanks Ayan Mahalanobis for reminding him of Mandelkern's work and thereby leading him to the ideas of this paper. Hannes Diener suggested to me the proof that the set S in the recursive example is not colocated.

References

1. Aczel, P., Rathjen, M.: Notes on Constructive Set Theory, Report No. 40, Institut Mittag-Leffler, Royal Swedish Academy of Sciences (2001)
2. Berger, J., Bridges, D.S., Mahalanobis, A.: The anti-Specker property, a Heine–Borel property, and uniform continuity, preprint, University of Canterbury (January 2007)
3. Bishop, E.A.: Foundations of Constructive Analysis, McGraw-Hill, New York (1967)
4. Bishop, E.A., Bridges, D.S.: Constructive Analysis. Springer, Heidelberg (1985)
5. Bridges, D.S., Vîţă, L.S.: Techniques of Constructive Analysis, Universitext. Springer, New York (2006)
6. Bridges, D.S., Richman, F.: Varieties of Constructive Mathematics. London Math. Soc. Lecture Notes, vol. 97. Cambridge Univ. Press, Cambridge (1987)
7. Mandelkern, M.: Located sets on the line. Pacific J. Math 95(2), 401–409 (1981)
8. Specker, E.: Nicht konstruktiv beweisbare Sätze der Analysis. J. Symb. Logic 14, 145–158 (1949)

Computing with Genetic Gates

Nadia Busi[1] and Claudio Zandron[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A. Zamboni 7, I-40127 Bologna, Italy
busi@cs.unibo.it

[2] Dipartimento di Informatica, Sistemistica e Comunicazione,
Università di Milano-Bicocca,
via Bicocca degli Arcimboldi 8, I-20126, Milano, Italy
zandron@disco.unimib.it

Abstract. We introduce Genetic Systems, a formalism inspired by genetic regulatory networks and suitable for modeling the interactions between the genes and the proteins, acting as regulatory products.

The generation of new objects, representing proteins, is driven by genetic gates: a new object is produced when all the activator objects are available in the system, and no inhibitor object is available. Activators are not consumed by the application of such an evolution rule. Objects disappear because of degradation: each object is equipped with a lifetime, and the object decays when such a lifetime expires.

We investigate the computational expressiveness of Genetic Systems: we show that they are Turing equivalent by providing an encoding of Random Access Machines in Genetic Systems.

1 Introduction

Most biological processes are regulated by networks of interactions between regulatory products and genes. To investigate the dynamical properties of these genetic regulatory networks, various formal approaches, ranging from discrete to stochastic and to continuous models, have been proposed (see [3] for a review).

The goal of this paper is to investigate the ability of genetic networks to act as a computational device. To this aim, we introduce *Genetic Systems*, a simple discrete formalism for the modeling of genetic networks.

The basic ingredients of the model are *genetic gates*, that are rules modeling the behaviour of genes, and *objects*, representing proteins. Proteins both regulate the activity of a gene – by activating or inhibiting transcription – and represent the product of the activity of a gene.

A genetic gate is essentially a *contextual rewriting rule* consisting of three components: the set of activators, the set of inhibitors and the transcription product. A genetic gate is activated if the activator objects are present in the system, and all inhibitor objects are absent. The result of the application of a genetic gate rule is the production of a new object. It is worthwhile to note that the application of such a rule does not remove the activator objects from the system.

In biological systems, proteins can disappear in (at least) two ways (see, e.g., [1]): a protein can either decay because its lifetime is elapsed, or be neutralized by a repressor protein.

To model the decaying process, we equip objects with a lifetime, which is decremented at each computational step. When the lifetime of an object becomes equal to zero, the object disappears. In our model we represent both decaying and persistent objects: while the lifetime of a decaying object is a natural number, persistent objects are equipped with an infinite lifetime.

The behaviour of repressor proteins is modeled through *repressor rules*, consisting of two components: the repressor object and the object to be destroyed. The rule is activated when both objects are present in the system. When the rule is applied, the object to be destroyed disappears, while the repressor is not removed.

We investigate the computational power of Genetic Systems by presenting an encoding of Random Access Machines (RAMs) [9], a well-known, deterministic Turing equivalent formalism. The encoding we provide turns out to be deterministic (i.e., in each state of the system, at most one computational step can be performed). As a consequence, the encoding enjoys the following property: a RAM terminates if and only if its encoding terminates (this means that no additional divergent or failed computations are added in the encoding).

The paper is organized as follows. In Section 2 we provide some basic definitions that will be used throughout the paper. The syntax and the semantics of Genetic Systems is presented in Section 3, and in Section 4 we show that Genetic Systems are Turing equivalent, by providing an encoding of RAMs. Section 5 reports some conclusive remarks.

2 Basic Definitions

In this section we provide some basic definitions that will be used throughout the paper. With $I\!N$ we denote the set of natural numbers, whereas $I\!N_\infty$ denotes $I\!N \cup \{\infty\}$. We start with the definition of multisets and multiset operations.

Definition 1. *Given a set S, a* finite multiset *over S is a function $m : S \rightarrow I\!N$ such that the set $dom(m) = \{s \in S \,|\, m(s) \neq 0\}$ is finite. The* multiplicity *of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S is denoted by $\mathcal{M}_{fin}(S)$; variables $m, m', \ldots$ range over $\mathcal{M}_{fin}(S)$. A multiset m such that $dom(m) = \emptyset$ is called* empty*. The empty multiset is denoted by $\emptyset$.*

Given the multiset m and m', we write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$ while $\oplus$ denotes their multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes* multiset difference*: $(m \setminus m')(s) =$ if $m(s) \geq m'(s)$ then $m(s) - m'(s)$ else 0.*

The set of parts of a set S is defined as $\mathcal{P}(S) = \{X \mid X \subseteq S\}$.

Given a set $X \subseteq S$, with abuse of notation we use X to denote also the multiset

$$m_X(s) = \begin{cases} 1 & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

We provide some basic definitions on strings, cartesian products and relations.

Definition 2. *A string over S is a finite (possibly empty) sequence of elements in S. Given a string $u = x_1 \dots x_n$, the length of u is the number of occurrences of elements contained in u and is defined as follows: $|u| = n$. The empty string is denoted by λ.*

With S^* we denote the set of strings over S, and $u, v, w, \dots$ range over S. Given $n \geq 0$, with S^n we denote the set of strings of length n over S. Given a string $u = x_1 \dots x_n$, the multiset corresponding to u is defined as follows: for all $s \in S$, $m_u(s) = |\{i \mid x_i = s \wedge 1 \leq i \leq n\}|$. With abuse of notation, we use u to denote also m_u[1].

Definition 3. *With $S \times T$ we denote the cartesian product of sets S and T, with $\times_n S$, $n \geq 1$, we denote the cartesian product of n copies of set S and with $\times_{i=1}^{n} S_i$ we denote the cartesian product of sets $S_1, \dots, S_n$, i.e., $S_1 \times \dots \times S_n$.*

Given a binary relation R over a set S, with R^n we denote the composition of n instances of R, with R^+ we denote the transitive closure of R, and with R^* we denote the reflexive and transitive closure of R.

3 Genetic Systems

In this section, we present the definition of Genetic Systems (G Systems for short) and the definitions which we need to describe their functioning. To this aim, given a set X, we define $\mathcal{R}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times (X \times I\!N_\infty)$.

Definition 4. *A* Genetic System with timed degradation *is a tuple*

$$G = (V, GR, RR, w_0)$$

where

1. *V is a finite alphabet whose elements are called* objects*;*
2. *GR is a finite multiset*[2] *over $\mathcal{R}_V$ of* genetic gates *over V; these gates are of the forms $u_{act}, \neg u_{inh} :\to (b, t)$ where $u_{act} \cap u_{inh} = \emptyset$. $u_{act} \subseteq V$ is the* positive regulation (activation)[3]*, $u_{inh} \subseteq V$ is the* negative regulation (inhibition)*, $b \in V$ is the transcription of the gate*[4] *and $t \in I\!N_\infty$ is the* duration *of object b;*
3. *RR is a finite set*[5] *of* repressor rules *of the form $(rep : b \to)$ where $rep, b \in V$ and $rep \neq b$;*

1 In some cases we denote a multiset by one of its corresponding strings, because this permits to define functions on multisets in a more insightful way.

2 Here we use multisets of rules, instead of sets, because each rule can be used at most once in each computational step.

3 We consider sets of activators, meaning that a genetic gate is never activated by more than one instance of the same protein.

4 Usually the expression of a genetic gate consists of a single protein.

5 We use sets rules, instead of multisets, because each repressor rule denotes a chemical law; hence, a repressor rule can be applied for an unbounded number of times in each computational step.

4. *w_0 is a string over $V \times I\!N_\infty$, representing the multiset of objects contained in the system at the beginning of the computation. The objects are of the form (a, t), where a is a symbol of the alphabet V and $t > 0$ represents the decay time of that object.*

We say that a gate is *unary* if $|u_{act} \oplus u_{inh}| = 1$. The multiset represented by w_0 constitutes the *initial state* of the system. A transition between states is governed by an application of the transcription rules (specified by the genetic gates) and of the repressor rules.

The gate $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ can be activated if the current state of the system contains enough free activators and no free inhibitors. If the gate is activated, the regulation objects (activators) in the set u_{act} are bound to such a gate, and they cannot be used for activating any other gate in the same maximal parallelism evolution step.

In other words, the gate $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ in a state formed by a multiset of (not yet bound) objects w can be activated if u_{act} is contained in w and no object in u_{inh} appears in w; if the gate performs the transcription, then a new object (b, t) is produced. Note that the objects in u_{act} and u_{inh} are not consumed by the transcription operation, but will be released at the end of the operation and (if they do not disappear because of the decay process) they can be used in the next evolution step. Each object starts with a decay number, which specifies the number of steps after which this object disappears. The decay number is decreased after each parallel step; when it reaches the value zero, the object disappears. If the decay number of an object is equal to ∞, then the object is persistent and it never disappears.

The repressor rule $(rep : b \rightarrow)$ is activated when both the repressor *rep* and the object b are present, and the repressor *rep* destroys the object b. The parentheses surrounding the repressor rule are omitted when the notation is not ambiguous.

We adopt the following notation for gates. The activation and inhibition sets are denoted by one of the corresponding strings, i,e, $a, b, \neg c :\rightarrow (c, 5)$ denotes the gate $\{a, b\}, \neg\{c\} :\rightarrow (c, 5)$. If either the activation or the inhibition is empty then we omit the corresponding set, i.e., $a :\rightarrow (b, 3)$ is a shorthand for the gate $\{a\}, \neg\emptyset :\rightarrow (b, 3)$. The *nullary gate* $\emptyset, \neg\emptyset :\rightarrow (b, 2)$ is written as $:\rightarrow (b, 2)$.

3.1 Partial Configurations, Reaction Relation and Maximal Parallelism Step

Having defined Genetic Systems, we are ready to describe their functioning. A transition between two states of the system is governed by an application of the transcription rules (specified by the genetic gates) and of the repressor rules. Different semantics can be adopted, depending on the number of rules that are applied in each computational step, and on the way in which the set of rules to be applied is chosen. We adopt the so-called *maximal parallelism semantics*: all the rules that *can be* applied simultaneously, *must be* applied in the same computational step.

We give now the definitions for partial configuration, configuration, reaction relation, and heating and decaying function.

Definition 5. *Let $G = (V, GR, RR, w_0)$ be a Genetic System. A* partial configuration *of G is a tuple $(w, R, \bar{w}, \bar{R}) \in \mathcal{M}_{fin}(V \times I\!N_\infty) \times \mathcal{M}_{fin}(\mathcal{R}_V) \times \mathcal{M}_{fin}(V \times I\!N_\infty) \times \mathcal{M}_{fin}(\mathcal{R}_V)$.*

The set of partial configurations of G is denoted by $Conf_G$. We use γ, γ', γ_1, ... to range over $Conf_G$.

The multiset w contains the active objects, whereas $\bar{w}$ is the multiset of the frozen (already used) objects; R represents the active gates, while $\bar{R}$ represents the frozen (already used) gates.

A *configuration* is a partial configuration containing no frozen objects; configurations represent the states reached after the execution of a maximal parallelism computation step.

Definition 6. *Let $G = (V, GR, RR, w_0)$ be a Genetic System.*

A configuration *of G is a partial configuration $(w, R, \bar{w}, \bar{R})$ satisfying the following: $\bar{w} = \emptyset$ and $\bar{R} = \emptyset$.*

The initial configuration *of G is the configuration $(w_0, GR, \emptyset, \emptyset)$.*

The activation of a genetic gate is formalized by the notion of reaction relation. In order to give a formal definition we need the function $obj : (V \times I\!N_\infty)^* \to V^*$, defined as follows. Assume that $(a, t) \in (V \times (I\!N_\infty))$ and $w \subseteq (V \times (I\!N_\infty))^*$. Then, $obj(\lambda) = \lambda$ and $obj((a, t)w) = a\ obj(w)$.

We also need to define a function DecrTime which is used to decrement the decay time of objects, destroying the objects which reached their time limit.

Definition 7. *The function $DecrTime : (V \times I\!N_\infty)^* \to (V \times I\!N_\infty)^*$ is defined as follows:*

$DecrTime(\lambda) = \lambda$, and

$$DecrTime((a, t)w) = \begin{cases} (a, t-1)DecrTime(w) & \text{if } t > 1 \\ DecrTime(w) & \text{if } t = 1 \end{cases}$$

We are now ready to give the notion of *reaction relation*.

Definition 8. *Let $G = (V, GR, RR, w_0)$ be a Genetic System.*

The reaction relation $\mapsto$ *over $Conf_G \times Conf_G$ is defined as follows:*
$(w, R, \bar{w}, \bar{R}) \mapsto (w', R', \bar{w}', \bar{R}')$ iff one of the following holds:

- *there exist $u_{act}, \neg u_{inh} :\to (b, t) \in R$ and $w_{act} \subseteq w$ such that*
  - $u_{inh} \cap dom(obj(w)) = \emptyset$
  - $obj(w_{act}) = m_{u_{act}}$[6]
  - $w' = w \setminus w_{act}$
  - $\bar{w}' = \bar{w} \oplus \{(b, t)\} \oplus DecrTime(w_{act})$
  - $R' = R \setminus (u_{act}, \neg u_{inh} :\to (b, t))$
  - $\bar{R}' = \bar{R} \oplus (u_{act}, \neg u_{inh} :\to (b, t))$

[6] We recall that $m_{u_{act}}$ is the multiset containing exactly one occurrence of each object in the set u_{act}. Hence, the operator $=$ is intended here to be the equality operator on multisets.

– *there exists* $rep : b \rightarrow \in RR$ *such that*
  - *there exist* $t_{rep}, t_b \in \mathbb{N}_\infty$ *such that* $\{(rep, t_{rep}), (b, t_b)\} \subseteq w$
  - $w' = w \setminus \{(rep, t_{rep}), (b, t_b)\}$
  - $\bar{w}' = w \oplus DecrTime((rep, t_{rep}))$
  - $\bar{R} = R$

The function *heat&decay* – used in the definition of a maximal parallelism computational step –permits to make the frozen objects and rules active again, and decrements the decaying time of the objects that have not been used in any rule in of the last maximal parallelism step.

Definition 9. *The function heat&decay* : $Conf_G \rightarrow Conf_G$ *is then defined as follows:*
$heat\&decay(w, R, \bar{w}, \bar{R}) = (DecrTime(w)) \oplus \bar{w}), R \oplus \bar{R}, \emptyset, \emptyset)$

Now we are ready to define the maximal parallelism computational step $\Mapsto$:

Definition 10. *Let* $G = (V, GR, RR, w_0)$ *be a Genetic System.*
The maximal parallelism computational step $\Mapsto$ *over (nonpartial) configurations of* G *is defined as follows:* $\gamma_1 \Mapsto \gamma_2$ *iff one of the following holds:*

– *there exists a partial configuration* γ' *s.t.* $\gamma_1 \mapsto^+ \gamma'$, $\gamma' \not\mapsto$ *and* $\gamma_2 = heat\&decay(\gamma')$[7], *or*
– $\gamma_1 = (w, R, \emptyset, \emptyset)$, $\gamma_1 \not\mapsto$, *there exists* $(a, t) \in w$ *s.t.* $t \neq \infty$ *and* $\gamma_2 = (DecrTime(w), R, \emptyset, \emptyset)$.

We say that a configuration γ *is* terminated *if no maximal parallelism step can be performed, i.e.,* $\gamma \not\Mapsto$.

Note that a computational step can be either a maximal set of rules, or the passing of one time unit, in case the system is deadlocked (i.e., no rule can be applied).

We also need computational steps of the second kind, because it may happen that the computation restarts after some object – acting as inhibitor for some rule – decays. Consider, e.g., the system with a negative gate $\neg b :\rightarrow (a, 3)$ and a positive gate $a :\rightarrow (a, 3)$, reaching a configuration containing only the object $(b, 2)$; the system can perform no move, but, after 2 time units have elapsed, an object $(a, 3)$ is produced and, by the positive gate, the system will never terminate.

4 Expressiveness of Genetic Systems

In this section we show that Genetic Systems are Turing powerful. It's easy to see that Genetic systems can be simulated by a Turing complete formalism. In this section we show how to model Random Access Machines (RAMs) [9], a well known Turing powerful formalism, in Genetic Systems. We start recalling the definition of RAMs.

[7] With $\gamma \not\mapsto$ we denote the fact that there exist no γ' such that $\gamma \mapsto \gamma'$.

4.1 Random Access Machines

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [5] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

A state of a RAM is modelled by $(i, c_1, \ldots, c_n)$, where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively. The notation $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ is used to denote that the state of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$, as a consequence of the execution of the i-th instruction.

A state $(i, c_1, \ldots, c_n)$ is *terminated* if the program counter i is strictly greater than the number of instructions m. We say that a RAM R *terminates* if its computation reaches a terminated state. The *output* of the RAM is the contents of register r_1 in the terminated state of the RAM (if such a state exists).

4.2 Encoding RAMs in Genetic Systems

In this section we show how to model RAMs in Genetic Systems. The basic idea consists in representing the contents of registers by sets of copies of persistent objects. More precisely, the fact that register r_i contains value c_i is represented by the presence of c_i copies of object (r_i, ∞) in the state of the system.

The fact that the program counter contains the value i (i.e., the next instruction to be executed is the ith) is represented by the presence of object p_i. At the beginning of the computation, the program counter is represented by the decaying object $(p_1, 1)$. In general, just before starting the execution of the ith instruction, the system contains the object $(p_i, 1)$.

The instructions could be encoded by genetic gates. Basically, the encoding of a successor instruction $i : Succ(r_j)$ produces a new instance of a persistent object (r_j, ∞), as well as the new program counter $(p_{i+1}, 1)$. The encoding of a decrement or jump instruction $i : DecJump(r_j, s)$ may have two different behaviours. One gate checks (by negative regulation) if no object r_j occurs in the system: if this is the case, then the program counter $(p_s, 1)$ is produced. Another gate checks (by positive regulation) if there exists an object r_j in the system: if this is the case, then a repressor for object r_j is produced to model the decrement of the register; the repressor decays just after destroying r_j, and the program counter $(p_{i+1}, 1)$ is produced.

When providing a RAM encoding, we need to specify how to extract the output of the system. As the output of the RAM is the contents of register r_1 in the terminated state of the RAM, we decided to take as output of the RAM encoding the number of occurrences of object (r_1, ∞) in a terminated configuration of the system.

We provide a deterministic RAM encoding which satisfies the following condition: the RAM terminates with output k if and only if the RAM encoding terminates in a state containing exactly k occurrences of object (r_1^1, ∞).

For the sake of brevity, we consider RAMs that satisfy the following constraint: if the RAM has m instructions, then all the jumps to addresses higher than m are jumps to the address $m+1$. This constraint is not restrictive, as for any RAM not satisfying the constraint above it is possible to contruct an equivalent RAM (i.e., a RAM computing the same function) which satisfies it. Given a RAM with m instructions, the first constraint can be easily satisfied by replacing each jump to an address higher than m by a jump to the address $m + 1$.

If the ith instruction is $(i : Succ(r_j))$, then the following sequence of rules is executed[8]. On the left we report the genetic gate that performs a transcription, while on the right we report the set of nonpersistent objects currently contained in the system.

	$(p_i, 1)$
step 1 : $p_i :\rightarrow (inc_1^i, 2)$	
	$(inc_1^i, 2)$
step 2 : $inc_1^i, \neg inc_2^i :\rightarrow (inc_2^i, 2)$	
	$(inc_1^i, 1), (inc_2^i, 2)$
step 3 : $inc_1^i, inc_2^i :\rightarrow (r_j^1, \infty)$	
	$(inc_2^i, 1)$
step 4 : $\neg inc_1^i, inc_2^i :\rightarrow (p_{i+1}, 1)$	
	$(p_{i+1}, 1)$

The main difficulties in the design of the encoding are concerned with the following facts. As the activators of a genetic gate are not removed from the state by the transcription operator, we must ensure that some operations (such as the production of object (r_j, ∞)) are executed exactly once. We make use of decaying objects to ensure that there exists a single time unit where such an operation can be performed. The other difficulty is due to the fact that the activation of a genetic gate produces a single object. As we need to produce both the object (r_j, ∞) and the new program counter, we make use of auxiliary decaying objects. Note that each of the gates reported above can be activated only if at least one of the objects in the set $\{p_i, inc_1^i, inc_2^i\}$ is contained in the current state; as we will see in the following, one of the objects in the above set is

[8] Here we recall that before starting the execution of the ith instruction, the system contains the object $(p_i, 1)$.

present in the system if and only if we are executing the ith instruction; moreover, no one of these objects will be used as activator or inhibitor in the gates used to encode the other instructions. Note also that, when starting from a system containing only $(p_i, 1)$ as nonpersistent object, the only possible execution is the one reported above.

If the i-th instruction is $(i : DecJump(r_j, s))$ and the contents of r_j is zero, then the following rule is executed:

	$(p_i, 1)$
step 1 : $p_i, \neg r_j :\rightarrow (p_s, 1)$	
	$(p_s, 1)$

On the other hand, if the contents of r_j is not zero, then the following sequence of rules is executed:

	$(p_i, 1)$
step 1 : $p_i, r_j :\rightarrow (dec_1^i, 2)$	
	$(dec_1^i, 2)$
step 2 : $dec_1^i, \neg dec_2^i :\rightarrow (dec_2^i, 2)$	
	$(dec_1^i, 1), (dec_2^i, 2)$
step 3 : $dec_1^i, dec_2^i \rightarrow (repr_j, 1)$	
	$(dec_2^i, 1), (repr_j, 1)$
step 4 : $repr_j : r_j \rightarrow$ $\neg dec_1^i, dec_2^i :\rightarrow (p_{i+1}, 1)$	
	$(p_{i+1}, 1)$

If the decrement of r_j is performed, the following operations need to be done: a repressor for object (r_j, ∞) is created, such a repressor removes an instance of (r_j, ∞) and the program counter corresponding to the next instruction is created. We ensure that exactly a single instance of (r_j, ∞) is removed by producing a nonpersistent repressor with timelife of a single unit. Note that in step 4 two rules (a repressor rule and a genetic gate) are executed simultaneously in the same step.

5 Conclusions

We investigate the computational expressiveness of Genetic Systems, a formalism modeling the interactions occurring between genes and regulatory products. A study of the expressiveness of rewriting rules inspired by genetic networks have been carried out by the authors in [2], in the context of Membrane Systems [6]. The result presented in [2] is incomparable with the result presented in this paper, because the semantics of the rules are different (in this paper, the modeling of repressors is more faithful to the biological reality, and a more abstract semantics for genetic gates is used), and because the result in [2] crucially depends on the use of membranes, permitting to localize to a specific compartment the objects and the rules, and of rules modeling the movement of objects across membranes. As we already said in the Introduction, several models for genetic regulatory networks have been proposed to investigate the behaviour of genes; however, to the best of our knowledge, their computational power has never been investigated. While both approaches are inspired by DNA, Genetic Systems turn out to be different from *DNA computing* (see, e.g., [7] for a survey), where the basic ingredients are strings, representing DNA strands, that evolve through the splicing operation.

In the present paper, several ingredients are used to achieve Turing equivalence: maximal parallelism semantics, the use of both persistent and decaying objects, repressor rules, positive and negative regulation. It is worthwhile to wonder if all these ingredients are really needed to get Turing equivalence. We conjecture the following results. If we move to interleaving semantics (that is, a single rule is applied in each computational step) Genetic Systems are Turing equivalent: this can be proved by providing a variant of the RAM presented in this paper. Genetic Systems with interleaving semantics, genetic gates, repressor rules, and persistent objects only (i.e., the lifetime of each object is ∞) are not Turing equivalent; the idea is to provide a reduction of Genetic Systems to Safe Petri Nets [8], that preserves the terminating behaviour. Genetic Systems with either interleaving or maximal parallelism semantics, repressor rules, both persistent and decaying objects, and genetic gates with positive regulation (i.e., no inhibition) are not Turing equivalent; this can be proved by resorting to the theory of Well-Structured Transition Systems [4].

References

1. Blossey, R., Cardelli, L., Phillips, A.: A Compositional Approach to the Stochastic Dynamics of Gene Networks. In: Priami, C., Cardelli, L., Emmott, S. (eds.) Transactions on Computational Systems Biology IV. LNCS (LNBI), vol. 3939, Springer, Heidelberg (2006)
2. Busi, N., Zandron, C.: Computing with Genetic Gates, Proteins and Membranes. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, Springer, Heidelberg (2006)
3. De Jong, H.: Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. Journal of Computatonal Biology 9, 67–103 (2002)
4. Finkel, A., Schnoebelen, Ph.: Well-Structured Transition Systems Everywhere! In: Theoretical Computer Science, vol. 256, pp. 63–92. Elsevier, North-Holland, Amsterdam (2001)
5. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs (1967)
6. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
7. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing. New computing paradigm. Springer, Heidelberg (1998)
8. Reisig, W.: Petri nets: An Introduction. In: EATCS Monographs in Computer Science. Springer, Heidelberg (1985)
9. Shepherdson, J.C., Sturgis, J.E.: Computability of recursive functions. Journal of the ACM 10, 217–255 (1963)

Resource Restricted Computability Theoretic Learning: Illustrative Topics and Problems

John Case[⋆]

Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716-2586 U.S.A.
case@cis.udel.edu

Abstract. Computability theoretic learning theory (machine inductive inference) typically involves learning programs for languages or functions from a stream of complete data about them and, importantly, allows mind changes as to conjectured programs. This theory takes into account algorithmicity but typically does *not* take into account *feasibility* of computational resources. This paper provides some example results and problems for three ways this theory can be constrained by computational feasibility. Considered are: the learner has memory limitations, the learned programs are desired to be optimal, and there are feasibility constraints on obtaining each output program *and* on the number of mind changes.

1 Introduction and Motivation

Let $\mathbb{N}$ = the set of non-negative integers. Computability theoretic (a.k.a recursion theoretic) learning [28,36] typically involves a scenario as depicted in (1) just below.

$$\text{Data } d_0, d_1, d_2, \ldots \xrightarrow{\text{In}} M \xrightarrow{\text{Out}} \text{ Programs } p_0, p_1, p_2, \ldots\ . \tag{1}$$

In (1), $d_0, d_1, d_2, \ldots$ can be, for example, the elements of a (formal) language $L \subseteq \mathbb{N}$ or the successive values of a function $f : \mathbb{N} \to \mathbb{N}$; M is a machine; and, for its *successful* learning, later p_i's $\approx$ compute the L or f. We will consider different criteria of successful learning of L or f by M. **Ex**-style criteria require that all but finitely many of the p_i's are the same *and* do a good job of computing the L or f. **Bc**-style criteria are more relaxed and powerful [4,12,15] and do not require almost all p_i's be the same.

In the present paper we *survey* some *illustrative*, top down, computational *resource restrictions* on essentially the paradigm of (1). In some sections we work instead with simple variants of (1).

In Section 2 below, $d_0, d_1, d_2, \ldots$ are the elements of a language and the p_i's are usually type-0 grammars [26] (equivalently, r.e. or c.e. indices [39]) for languages.

[⋆] Work supported in part by NSF Grant Number CCR-0208616 at UD.

In Section 2 we consider restrictions on the ability of machines M to remember prior data and conjectures. The motivation, in addition to space-complexity theoretic, is from cognitive science. We will provide some open problems too. Much of the material of Section 2 is from [8,13].

In Section 3 below, the d_i's are successive values of functions f in some complexity theoretically interesting subrecursive classes and the p_i's can be programs in some associated subrecursive programming systems [40]. A not-necessarily-realized *desire* is that successful later programs p_i are not too far from optimally efficient.[1] We provide some examples (from [10]) showing how computational complexity of those later p_i's, which *are* successful at computing the f's, is affected by the size of the subrecursive class to be learned.

The paradigm of (1) is sometimes called *learning in the limit*, and the M in (1) can be thought of as computing an appropriately typed, *limiting* functional. One speaks of M's transitions from outputting p_i to outputting p_{i+1} as *mind changes*.[2] Restrictions on the number of such mind changes have been extensively studied in the literature, beginning with [5,15]. [21] first considered counting down mind changes from notations for possibly transfinite constructive ordinals and proved results to the effect that counting down from bigger constructive ordinals gave more learning power. See also [1,29].[3] In (1), the *time* to calculate each p_i may be infeasible, and the total time to reach the *successful* p_i's may not be algorithmic [17]. In Section 4 below, we present some previously unpublished *very preliminary* work on top down, *feasible* variants of (1), and we indicate problems we hope will be worked out in the future [14]. The rough idea, explained in more detail in Section 4 below, is that: 1. one restricts the M of (1) to iterating a *type-2 feasible functional* in the sense of [27,30,35], *and* 2. one counts down, with another type-2 *feasible* functional, the allowed mind changes from *feasible* notations for constructive ordinals.[4]

2 Memory-Limited and U-Shaped Language Learning

Informally, *U-shaped learning* is as follows. For B a task to learn a desired *behavior*, U-shaped learning occurs when, while learning B, a learner *learns* B, then the learner *un*learns B, and, finally, the the learner *re*learns B. U-shaped learning has been empirically observed by cognitive psychologists in various areas of child development. Examples include understanding of various (Piaget-like)

[1] This will be for cases, unlike in Blum Speed-Up Theorems [36], where there *are* optimally efficient programs.

[2] Learning in the limit is essential, for example, for the iterated forward difference method for fitting polynomials to data [25], where the number of mind changes required depends on the degree of the polynomial generating the data.

[3] Outside computability theoretic learning, [2] characterizes *explicitly* Ershov Hierarchy levels [20,19] by constructive ordinal notation *count down*.

[4] For example, algorithmic counting down mind-changes from any notation w for the smallest infinite ordinal ω is equivalent to declaring if and when a first mind change is made *and then* declaring the *finite* number of *further* mind changes allowed.

conservation principles [42], e.g., the interaction between object tracking and object permanence, and verb regularization [34,37,42,43]. Here is an example of the latter. In English, a child first uses *spoke*, the correct past tense form of the verb *speak*. Then the child *in*correctly uses speaked. Lastly, the child returns to using *spoke*.

One main question of the present section: is U-shaped learning *necessary* for full learning power? I.e., are there classes of tasks learnable *only* by *returning* to *abandoned, correct behavior*?

For example, [3,7] answered formalized versions of the previous question for computability theoretic learning with*out* memory limitations. The answer depends interestingly on the criteria of successful learning: roughly, for criteria of power strictly between **Ex** and **Bc**-styles [9] (and also for **Bc**-style), U-shaped learning *is* necessary for full learning power. Humans have memory limitations, both for previously seen data and, to some extent, for previously made conjectures. In the present section we discuss the necessity of U-shaped learning of grammars for whole formal languages *and* in models with such memory restrictions. *First*, though, we discuss in detail the cases with*out* memory limitations.

A sequence T of natural numbers and #'s is a *text* for a language $L \subseteq \mathbb{N} \Leftrightarrow_{\text{def}}$ $L = \{T(i) \mid T(i) \neq \#\}$. The # represents a pause. The only text for the empty language is an infinite sequence of #'s. The present section employs a variant of (1) *where* d_i is $T(i)$ with T a text for a language, and the p_i's are either r.e. indices *or they are* ?*'s*. ?'s signal that M has no program to conjecture.

In the rest of the present section we restrict our attention to **Ex**-style criteria of success.

Formally: a learner M **TxtEx**-*learns* a *class* of languages $\mathcal{L} \Leftrightarrow_{\text{def}}$, for all $L \in \mathcal{L}$, *on* all texts T for L, M eventually stabilizes to outputting *a single* program successfully generating L.

For a language class learning criterion, **C**-learning, such as **TxtEx**-learning just defined and variants defined below, we write **C** to stand for the collection of all languages classes $\mathcal{L}$ such that some machine M **C**-learns each $L \in \mathcal{L}$.

A learner M is *Non-U-Shaped* (abbreviated: **NU**) on a class $\mathcal{L}$ that M **TxtEx**-learns $\Leftrightarrow_{\text{def}}$, on any text for any language L in $\mathcal{L}$, M never outputs a sequence $\ldots, p, \ldots, q, \ldots, r, \ldots$, where p, r accept/generate L, but q doesn't.

Next we discuss three types of *memory limited* language learning models from the prior literature [11,22,33,44].

Iterative Learning: M **It**-*learns* a *class* of languages $\mathcal{L} \Leftrightarrow_{\text{def}}$ M **TxtEx**-learns $\mathcal{L}$ *but* M has access only to its own just previous conjecture (if any) and to its current text datum.

m-Feedback Learning is like **It**-learning, *but*, while the learner has access to its just previous conjecture and to the current text datum, it can also make m simultaneous *recall* queries, i.e., queries as to whether up to m items of its choice have already been seen in input data thus far.

n-Bounded Example Memory Learning is also like **It**-learning, *but*, while the learner has access to its just previous conjecture and to the current text datum, it remembers up to n previously seen data items that it chooses to remember.

For $m, n > 0$, m-Feedback and n-Bounded Example Learning are incomparable, but separately make strict learning hierarchies increasing in m, n [11].

N.B. For the cases of $m, n > 0$, it is completely open as to whether U-shapes are necessary for full power of m-Feedback and n-Bounded Example Learning!

Results about the necessity of U-shapes for **It**-learning and for *more severely restricted* variants of the other models just above *have* been obtained and some are discussed below. For **It**-learning, U-shapes are *not* needed:

Theorem 1 ([13]). $\mathbf{NUIt} = \mathbf{It}$.[5]

Memoryless Feedback Learners are restricted versions of Feedback Learners above: for $m > 0$, an $\mathbf{MLF}_m$*-learner* has *no* memory of previous conjectures but has access to its current text datum and can make $m > 0$ simultaneous recall queries — queries as to whether up to m items of its choice have already been been seen in input data. The $\mathbf{MLF}_m$ learning criteria form a hierarchy increasing in m: $\mathbf{MLF}_m \subset \mathbf{MLF}_{m+1}$ [8].

Theorem 2 ([8]). U-shaped learning is necessary for each level of this hierarchy: for $m > 0$, $\mathbf{NUMLF}_m \subset \mathbf{MLF}_m$.

Bounded Memory States Learners [32] are restricted variants of Bounded Example Learners above: for $c > 1$, a $\mathbf{BMS}_c$-Learner does *not* remember any previous conjectures, has access to current text datum, *and* can store *any one of* c different values it chooses in its memory. This latter corresponds exactly to remembering $\log_2(c)$ bits. The $\mathbf{BMS}_c$ learning criteria form a hierarchy increasing in $c > 1$: $\mathbf{BMS}_c \subset \mathbf{BMS}_{c+1}$ [32].

Theorem 3 ([8]). U-shaped learning is *not needed* for 2-Bounded Memory States Learners: $\mathbf{NUBMS}_2 = \mathbf{BMS}_2$.

Open Questions: is U-shaped learning necessary for $\mathbf{BMS}_c$*-learning with $c > 2$?* Humans remember some bits, remember some prior data, can recall whether they've seen some data, and, likely, store their just prior conjecture. *Is U-shaped learning necessary for such combinations?*

3 Complexity of Learned Programs

Results in the present section are selected from many in [10], and we employ (1) in the case that d_i is $f(i)$, *where* $f \in \mathcal{F} \subseteq \mathcal{R}_{0,1}$, the class of all (total) computable functions : $\mathbb{N} \rightarrow \{0, 1\}$.

Suppose $a \in \mathbb{N} \cup \{*\}$. a is for anomaly count. When $a = *$, a stands for finitely many.

$\mathcal{F} \in \mathbf{Ex}^a \Leftrightarrow_{\text{def}} (\exists M)(\forall f \in \mathcal{F})$
$[M$ fed $f(0), f(1), \ldots,$ outputs $p_0, p_1, \ldots \wedge (\exists t)[p_t = p_{t+1} = \cdots \wedge p_t$ computes f — except at up to a inputs $]]$.

[5] As per our convention above, **It**, respectively **NUIt**, stands for the collection of all languages classes $\mathcal{L}$ such that some machine M **It**-learns, respectively **NUIt**-learns, each $L \in \mathcal{L}$.

$\mathcal{F} \in \mathbf{Bc}^a \Leftrightarrow_{\text{def}} (\exists M)(\forall f \in \mathcal{F})$
$[M$ fed $f(0), f(1), \ldots,$ outputs $p_0, p_1, \ldots \wedge (\exists t)[p_t, p_{t+1}, \ldots$ each computes f — except at up to a inputs$]]$.

Turing machines herein are multi-tape.

For $k \geq 1$, $\mathcal{P}^k =_{\text{def}}$ the class of all $\{0,1\}$-valued functions computable by Turing Machines in $O(n^k)$ time, where n is the length of the input expressed in *dyadic* notation.[6] $\mathcal{P} =_{\text{def}} \bigcup \mathcal{P}^k$.

Let slow be any fixed slow growing unbounded function $\in \mathcal{P}^1$, e.g., $\leq$ an inverse of Ackermann's function as from [16, Section 21.4]. $\mathcal{Q}^k =_{\text{def}}$ the class of all $\{0,1\}$-valued functions computable in $O(n^k \cdot \log(n) \cdot \text{slow}(n))$ time. $\mathcal{P}^k \subset \mathcal{Q}^k \subset \mathcal{P}^{k+1}$. The first proper inclusion is essentially from [24,26] and appears to be best known.

$\mathcal{P} \in \mathbf{Ex}^0$. $\mathcal{P}^k \in \mathbf{Ex}^0$ too (where each output conjecture runs in k-degree polytime).

$\mathcal{CF}$, the class of all characteristic functions of the context free languages [26], $\in \mathbf{Ex}^0$ [23].

From [15] (with various credits to Bārzdiņš, the Blums, Steel, and Harrington): $\mathbf{Ex}^0 \subset \mathbf{Ex}^1 \subset \mathbf{Ex}^2 \subset \cdots \subset \mathbf{Ex}^* \subset \mathbf{Bc}^0 \subset \mathbf{Bc}^1 \subset \cdots \subset \mathbf{Bc}^*$, and $\mathcal{R}_{0,1} \in \mathbf{Bc}^*$.

We introduce some basic, useful notation.

$(\forall^\infty x)$ means for all but finitely many $x \in \mathbb{N}$.

$\mathcal{U} =_{\text{def}} \{f \in \mathcal{R}_{0,1} \mid (\forall^\infty x)[f(x) = 1]\}$ $(\subset \mathcal{P}^1)$. $\mathcal{U}$ is an example of a class of particularly *easy* functions.

$\varphi_p^{\text{TM}} =_{\text{def}}$ the partial computable function $: \mathbb{N} \to \mathbb{N}$ computed by Turing machine program (number) p.

$\Phi_p^{\text{TM}}(x) =_{\text{def}}$ the runtime of Turing machine program (number) p on input x, if p halts on x, and undefined, otherwise.

$\Phi_p^{\text{WS}}(x) =_{\text{def}}$ the work space used by Turing machine program (number) p on input x, if p halts on x, and undefined, otherwise.

Clearly, $\mathcal{U} \subset \mathcal{REG}$, the class all characteristic functions of regular languages [26].

$f[n] =_{\text{def}}$ the sequence $f(0), \ldots, f(n-1)$.

$M(f[n]) =_{\text{def}}$ M's output based only on $f[n]$.

Of course, since finite automata do not employ a work tape, $\exists M$ witnessing $\mathcal{REG} \in \mathbf{Ex}^0$ such that $(\forall n, x)[\Phi^{\text{TM}}_{M(f[n])}(x) = |x| + 1 \ \wedge \ \Phi^{\text{WS}}_{M(f[n])}(x) = 0]$.

A result of [41] is strengthened by

Theorem 4 ([10]). Suppose $k \geq 1$ and that M witnesses either $\mathcal{Q}^k \in \mathbf{Ex}^*$ or $\mathcal{Q}^k \in \mathbf{Bc}^0$ (special case: M witnesses $\mathcal{Q}^k \in \mathbf{Ex}^0$).
Then: $(\exists f \in \mathcal{U})(\forall k$-degree polynomials $p)$
$(\forall^\infty n)(\forall^\infty x)[\Phi^{\text{TM}}_{M(f[n])}(x) > p(|x|)]$.

If we increase the generality of a machine M to handle $\mathcal{Q}^k$ instead of merely $\mathcal{P}^k$, this *forces* the run-times of M's successful outputs on some *easy* $f \in \mathcal{U}$ worse

[6] The *dyadic* representation of an input natural number $x =_{\text{def}}$ the x-th finite string over $\{0,1\}$ *in lexicographical order*, where the counting of strings starts with zero [40]. Hence, unlike with binary representation, lead zeros matter.

than any k-degree polynomial bound, i.e., to be *sub*optimal. But, for learning only $\mathcal{P}^k$, this need not happen. Hence, we see, in Theorem 4, ones adding slightly to the generality of a learner M produces final, successful programs with a complexity deficiency. Another complexity deficiency in final programs caused by learning too much is provided by the following

Theorem 5 ([10]). Suppose M $\mathbf{Ex}^*$-learns $\mathcal{CF}$ and $k, n \geq 1$(special case: M witnesses $\mathcal{CF} \in \mathbf{Ex}^0$). Then there is an easy f, an $f \in \mathcal{U}$, such that, if p is M's final program on f, for *some distinct* $x_0, \ldots, x_{n-1}$, program p uses more than k workspace squares on each of inputs $x_0, \ldots, x_{n-1}$.

In [10] there are further such complexity deficiencies in final programs caused by learning too much, again where the deficiencies are on *easy* functions. *Assuming $\mathcal{NP}$ separates* from $\mathcal{P}$ (with $\mathcal{NP}$ also treated as a class of $\{0,1\}$-valued characteristic functions of sets $\subseteq \mathbb{N}$), then one gets a complexity deficiency in final programs caused by learning $\mathcal{NP}$ instead of $\mathcal{P}$. There is a similar result in [10] for $\mathcal{BQP}$, a quantum version of polynomial-time [6], in place of $\mathcal{NP}$ — *assuming $\mathcal{BQP}$ separates* from $\mathcal{P}$. In these results the complexity deficient learned programs have *un*necessary non-determinism or quantum parallelism.

4 Feasible Iteration of Feasible Learning Functionals

The material of this section not credited to someone else is from [14].

One-shot **Ex**-style procedures output at most a single (hopefully correct) conjectured program [28]. *Feasible* deterministic one-shot function learning can be modeled by the polytime multi-tape Oracle Turing machines (OTMs) as used in [27] (see also [30,35]). We call the corresponding functionals *basic feasible functionals.* These polytime OTMs have a query tape and a reply tape. To query an oracle f, an OTM writes the dyadic representation of an $x \in \mathbb{N}$ on the query tape and enters its query state. The query tape is then erased, and the dyadic representation of $f(x)$ appears on the reply tape. The cost model is the same as for non-oracle Turing machines, *except* for the additional cost of a query to the oracle. This is handled with the length-cost model, where the cost of a query is $max(|f(x)|, 1)$, where $|f(x)|$ is the length of the string on the reply tape.[7] The next three definitions provide the formal details re the polytime constraint on basic feasible functionals.

Definition 1 ([30]). The *length* of $f : \mathbb{N} \to \mathbb{N}$ is the function $|f| : \mathbb{N} \to \mathbb{N}$ such that $|f| = \lambda n.max(\{|f(x)| \mid |x| \leq n\})$.

Definition 2 ([30]). A *second-order polynomial* over type-1 variables $g_0, ..., g_m$ and type-0 variables $y_0, ..., y_n$ is an expression of one of the following five forms:

[7] N.B. For the present section, then, the general paradigm (1) from Section 1 above is modified to allow the machine M to *query* input functions f for their values — instead of M's merely passively receiving the successive values of such f's.

$$\begin{gathered} a \\ y_i \\ \mathbf{q}_1 + \mathbf{q}_2 \\ \mathbf{q}_1 \cdot \mathbf{q}_2 \\ g_j(\mathbf{q}_1) \end{gathered}$$

where $a \in \mathbb{N}$, $i \le n$, $j \le m$, and $\mathbf{q}_1$ and $\mathbf{q}_2$ are second-order polynomials over $\overrightarrow{g}$ and $\overrightarrow{y}$.

Definition 3 ([30]). Suppose $k \ge 1$ and $l \ge 0$. Then $F : (\mathbb{N} \to \mathbb{N})^k \times \mathbb{N}^l \to \mathbb{N}$ is a *basic feasible functional* if and only if there is an OTM $\mathbf{M}$ and a second-order polynomial $\mathbf{q}$, such that, for each input $(f_1, ..., f_k, x_1, ..., x_l)$,

(1) $\mathbf{M}$ outputs $F(f_1, ..., f_k, x_1, ..., x_l)$, and
(2) $\mathbf{M}$ runs within $\mathbf{q}(|f_1|, ..., |f_k|, |x_1|, ..., |x_l|)$ time steps.

In the context of learning in the limit, we are interested in how to define *feasible* for *limiting*-computable type-2 functionals. This is discussed below, but, first, is presented some background material on notations for constructive ordinals.

Ordinals are representations of well-orderings. The *constructive ordinals* are just those that have a program, called a *notation*, which specifies how to build them (lay them out end to end, so to speak) [39]. Let O be Kleene's system of notations for each constructive ordinal [39], importantly, with the accompanying $<_o$ relation on O. We omit details but refer the reader to the excellent [39].

Everyone knows how to use (notations for) finite ordinals for counting *down*.

As indicated in Section 1 above, we have in mind *iterating* basic feasible learning functionals with *feasible* counting down of iterations from *feasible* notations for constructive ordinals, We want to see worked out the details of this model of feasible for **Ex** learning. We believe we have a correct formalization of the concept of feasible notations and feasible counting down. From space limitations, below we present *very* simple examples only.

Here is a promised very simple example. It is based on a system of notations we call O_ω. O_ω provides notations for all and only the finite ordinals and ω, the first infinite ordinal. This restricted system especially reduces the complexity of computing notations. $O_\omega =_{\text{def}} \mathbb{N}$. For $u \in O_\omega$, if u is even, u is a notation for the (finite) ordinal $u/2$. Otherwise, u is a notation for the (infinite) ordinal ω. For even $x, y \in O_\omega$, $x < y \Rightarrow x <_{O_\omega} y$, and for any even x and odd y, $x <_{O_\omega} y$. No other pairs of numbers satisfy the $<_{O_\omega}$ relation. For $x \in \mathbb{N}$, $\underline{x}$ is defined as the notation for the *finite* ordinal x, and, in this system, $\underline{x} = 2x$.

Next we present one *of many ways* to define feasibly iterated feasible learning. We are interested to investigate more ways and are currently pursuing this. For $t \in \mathbb{N}$, 0^t is (by definition) the string of 0's of length t. It is common in complexity theory to call 0^t a *tally*. We write ε for 0^t when $t = 0$.

Below φ is acceptable [39] *and* has a linear time implementation of S-m-n [40].

Definition 4. Suppose $u \in O_\omega$. A set of functions $\mathcal{S}$ is $\mathbf{Itr}_u$-*feasibly learnable* if and only if there exist basic feasible functionals $H : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ and $\mathbf{F} : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ such that for all $f \in \mathcal{S}$, there exists $k \in \mathbb{N}$ such that,

(1) $\mathbf{F}(f, \varepsilon) \leq_{O_\omega} u$,
(2) $\mathbf{F}(f, 0^{t+1}) <_{O_\omega} \mathbf{F}(f, 0^t)$, for all $t < k$,
(3) $\mathbf{F}(f, 0^k) = \underline{0}(= 0)$, and
(4) $\varphi_{H(f,0^k)} = f$.

Definition 5. $\mathcal{S}$ ranges over classes of computable functions.
$\mathbf{Itr}_u\mathbf{BffEx}$= $\{\mathcal{S} | \mathcal{S}$ is $\mathbf{Itr}_u$-*feasibly learnable*$\}$.

When an $\mathbf{F}$ from above counts down from a notation in O_ω for ω, it is allowed to jump to a notation for any finite ordinal. Let $\mathcal{S}_0$ be the *example* set of functions f such that f has the following properties:
(a) $f(0) > 1$,
(b) $f(1) > 1$,
(c) $f(x) = 1$, for exactly one x, where $1 < x \leq 2^{|f(0)|} + |f(1)|$,
(d) $f(x) = 0$, everywhere else. We have the following

Theorem 6. $\mathcal{S}_0 \in (\mathbf{Itr}_w\mathbf{Bffex}$ - $\mathbf{Itr}_{\underline{n}}\mathbf{Bffex})$, where w is any notation in O_ω for the ordinal ω, and $\underline{n}$ is the notation in O_ω for a *finite* ordinal $n \in \mathbb{N}$.

The particular scheme of feasibly iterating basic feasible learning functionals in Definitions 4 and 5 above requires the count-down function to bottom out at $\underline{0} = 0$, so one can tell when the iterations are done (and can suppress all the programs output but the last). We were initially surprised that, even for a scheme like this, we get a learning hierarchy result like in the just above theorem. We can prove that, for finite ordinals $n \in \mathbb{N}$, the $\mathbf{Itr}_{\underline{n}}\mathbf{Bffex}$ hierarchy collapses. As noted above, we are interested in the investigation of more ways for feasibly iterating basic feasible learning functionals. We'd like variant results where one cannot suppress all the output programs but the last. Here is another feasible notation system, this one for O_{ω^2}, *where*, for the lineartime computable pairing function $< .,. >$ from [40], the notation for ω^2 is 0, and that for ordinals $\omega \times a + b < \omega^2$ is $1+ < a, b >$ — with $<_{O_{\omega^2}}$ defined in the obvious way. We can prove hierarchy results similar to the above for this system.

It is interesting to question the feasible learnability of the example class above, $\mathcal{S}_0 \in \mathbf{Itr}_w\mathbf{Bffex}$. Of course, the counting down and the ordinal notations were all feasible as well as was the basic feasible functional H. Nevertheless, we can prove that, for $\mathbf{Ex}$-learning of $\mathcal{S}_0$, the total learning time of infinitely many $f \in \mathcal{S}_0$ is *inherently exponential* in $|f(0)|$ — while being polynomial in $|f(1)|$. *However*, $\mathcal{S}_0$ is analyzable in terms of *parameterized complexity* [18]. For parameter $k > 1$, let $\mathcal{S}_0^k = (\mathcal{S}_0 \cap \{f \mid f(0) \leq k\})$. Then each $\mathcal{S}_0^k$ is infinite and feasibly learnable. We would like to see this sort of phenomenon more generally analyzed and understood — including in more sophisticated settings.

We would also like to see studied *probabilistic* variants of feasibly iterated feasible learners — this toward producing practical generalizations of Valiant's *PAC* learning [31] and Reischuk and Zeugmann's [38] *stochastically finite learning*. These latter involve, probabilistic, one-shot learners.

References

1. Ambainis, A., Case, J., Jain, S., Suraj, M.: Parsimony hierarchies for inductive inference. Journal of Symbolic Logic 69, 287–328 (2004)
2. Ash, C., Knight, J.: Recursive structures and Eshov's hierarchy. Mathematical Logic Quarterly 42, 461–468 (1996)
3. Baliga, G., Case, J., Merkle, W., Stephan, F., Wiehagen, W.: When unlearning helps, Journal submission (2007)
4. Bārzdiņš, J.: Two theorems on the limiting synthesis of functions. Theory of Algorithms and Programs, Latvian State University, Riga. 210, 82–88 (1974)
5. Bārzdiņš, J., Freivalds, R.: Prediction and limiting synthesis of recursively enumerable classes of functions. Latvijas Valsts Univ. Zinatn. Raksti 210, 101–111 (1974)
6. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM Journal on Computing 26, 1411–1473 (1997)
7. Carlucci, L., Case, J., Jain, S., Stephan, F.: Non U-shaped vacillatory and team learning. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 241–255. Springer, Heidelberg (2005)
8. Carlucci, L., Case, J., Jain, S., Stephan, F.: Memory-limited U-shaped learning. In: Lugosi, G., Simon, H. (eds.) COLT 2006. LNCS (LNAI), vol. 4005, pp. 244–258. Springer, Heidelberg (2006)
9. Case, J.: The power of vacillation in language learning. SIAM Journal on Computing 28(6), 1941–1969 (1999)
10. Case, J., Chen, K., Jain, S., Merkle, W., Royer, J.: Generality's price: Inescapable deficiencies in machine-learned programs. Annals of Pure. and Applied Logic 139, 303–326 (2006)
11. Case, J., Jain, S., Lange, S., Zeugmann, T.: Incremental concept learning for bounded data mining. Information and Computation 152, 74–110 (1999)
12. Case, J., Lynes, C.: Machine inductive inference and language identification. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, pp. 107–115. Springer, Heidelberg (1982)
13. Case, J., Moelius, S.: U-shaped, iterative, and iterative-with-counter learning, Submitted (2007)
14. Case, J., Paddock, T., Kötzing, T.: Feasible iteration of feasible learning functionals, Work in progress (2007)
15. Case, J., Smith, C.: Comparison of identification criteria for machine inductive inference. Theoretical Computer Science 25, 193–220 (1983)
16. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge, MA (2001)
17. Daley, R., Smith, C.: On the complexity of inductive inference. Information and Control 69, 12–40 (1986)
18. Downey, R., Fellows, M.: Parameterized Complexity. In: Monographs in Computer Science, Springer, Heidelberg (1998)
19. Ershov, Y.: A hierarchy of sets, I. Algebra i Logika, 7(1):47–74, 1968. In Russian (English translation in Algebra and Logic, 7:25–43 1968) (1968)
20. Ershov, Y.: A hierarchy of sets II. Algebra and Logic 7, 212–232 (1968)
21. Freivalds, R., Smith, C.: On the role of procrastination in machine learning. Information and Computation 107(2), 237–271 (1993)
22. Fulk, M., Jain, S., Osherson, D.: Open problems in Systems That Learn. Journal of Computer and System Sciences 49(3), 589–604 (1994)

23. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
24. Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
25. Hildebrand, F.: Introduction to Numerical Analysis. McGraw-Hill, New York (1956)
26. Hopcroft, J., Ullman, J.: Introduction to Automata Theory Languages and Computation. Addison-Wesley Publishing Company, London, UK (1979)
27. Irwin, R., Kapron, B., Royer, J.: On characterizations of the basic feasible functional, Part I. Journal of Functional Programming 11, 117–153 (2001)
28. Jain, S., Osherson, D., Royer, J., Sharma, A.: Systems that Learn: An Introduction to Learning Theory, 2nd edn. MIT Press, Cambridge, MA (1999)
29. Jain, S., Sharma, A.: Elementary formal systems, intrinsic complexity, and procrastination. Information and Computation 132, 65–84 (1997)
30. Kapron, B., Cook, S.: A new characterization of type 2 feasibility. SIAM Journal on Computing 25, 117–132 (1996)
31. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press, Cambridge, MA (1994)
32. Kinber, E., Stephan, F.: Language learning from texts: Mind changes, limited memory and monotonicity. Information and Computation 123, 224–241 (1995)
33. Lange, S., Zeugmann, T.: Incremental learning from positive data. Journal of Computer and System Sciences 53, 88–103 (1996)
34. Marcus, G., Pinker, S., Ullman, M., Hollander, M., Rosen, T.J., Xu, F.: Overregularization in Language Acquisition. In: Monographs of the Society for Research in Child Development (Includes commentary by H. Clahsen), vol. 57(4), University of Chicago Press, Chicago (1992)
35. Mehlhorn, K.: Polynomial and abstract subrecursive classes. Journal of Computer and System Sciences 12, 147–178 (1976)
36. Odifreddi, P.: Classical Recursion Theory, volume II. Elsivier, Amsterdam (1999)
37. Plunkett, K., Marchman, V.: U-shaped learning and frequency effects in a multilayered perceptron: implications for child language acquisition. Cognition 86(1), 43–102 (1991)
38. Reischuk, R., Zeugmann, T.: An average-case optimal one-variable pattern language learner. Journal of Computer and System Sciences (Special Issue for COLT'98) 60(2), 302–335 (2000)
39. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York, 1967. Reprinted, MIT Press (1987)
40. Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. Research monograph in: Progress in Theoretical Computer Science. Birkhäuser Boston (1994)
41. Sipser, M.: Private communication (1978)
42. Strauss, S., Stavy, R. (eds.): U-Shaped Behavioral Growth. Academic Press, NY (1982)
43. Taatgen, N., Anderson, J.: Why do children learn to say broke? A model of learning the past tense without feedback. Cognition 86(2), 123–155 (2002)
44. Wiehagen, R.: Limes-erkennung rekursiver funktionen durch spezielle strategien. Electronische Informationverarbeitung und Kybernetik 12, 93–99 (1976)

Characterizing Programming Systems Allowing Program Self-reference[⋆]

John Case and Samuel E. Moelius III

Department of Computer & Information Sciences
University of Delaware
103 Smith Hall
Newark, DE 19716
{case,moelius}@cis.udel.edu

Abstract. The interest is in characterizing insightfully the power of program self-reference in effective programming systems (epses), the computability-theoretic analogs of programming languages. In an eps in which the *constructive* form of Kleene's Recursion Theorem (**KRT**) holds, it is possible to construct, algorithmically, from an arbitrary algorithmic task, a self-referential program that, in a sense, creates a self-copy and then performs that task on the self-copy. In an eps in which the *not-necessarily*-constructive form of Kleene's Recursion Theorem (**krt**) holds, such self-referential programs exist, but cannot, in general, be found algorithmically.

In an earlier effort, Royer proved that there is *no* collection of recursive denotational control structures whose implementability *characterizes* the epses in which **KRT** holds. One main result herein, proven by a finite injury priority argument, is that the epses in which **krt** holds are, similarly, *not* characterized by the implementability of some collection of recursive denotational control structures.

On the positive side, however, a characterization of such epses of a rather different sort *is* shown herein. Though, perhaps not the insightful characterization sought after, this surprising result reveals that a hidden and inherent constructivity is always present in **krt**.

Know thyself.
– Greek proverb

Keywords: Computability Theory, Programming Language Semantics, Self-Reference.

1 Introduction

The first author has, for some time, been interested in the difficult problem of understanding and insightfully characterizing the power of *program or machine self-reference* (synonym: *program self-reflection*).[1] Initial mathematical attempts

[⋆] This paper received support from NSF Grant CCR-0208616.
[1] This paper does not address *linguistic* self-reference, e.g., in arithmetic [16].

on this subject [10,11,14] were based on conceptualizing the *constructive* form of *Kleene's Recursion Theorem* (**KRT**) (Property 2 below) in terms of an associated *non-denotational control structure* [14] (synonym: *connotational control structure* [14]). Beginning with the next subsection, we explain what we mean by program self-reference and self-knowledge, what **KRT** has to do with these, and how we model control structures. Then, we briefly highlight some relevant results from the prior literature. Finally, we summarize and relevantly interpret the main results of the present paper — which results are the recent progress on the still difficult problem mentioned at the beginning of this paragraph.

1.1 Kleene's Recursion Theorems

Let $\mathbb{N}$ be the set of natural numbers, $\{0, 1, 2, \ldots\}$. Let $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be any fixed, 1-1, onto, computable mapping [13]. The function $\langle \cdot, \cdot \rangle$ enables us to restrict our attention to one-argument partial functions and still handle, with coding by $\langle \cdot, \cdot \rangle$, multiple argument cases.

For all one-argument partial functions ψ and all $p \in \mathbb{N}$, $\psi_p \overset{\text{def}}{=} \psi(\langle p, \cdot \rangle)$. A one-argument *partial computable* function ψ is an *effective programming system (*eps*)* $\overset{\text{def}}{\Leftrightarrow}$ for every one-argument partial computable function α, there exists p, such that $\psi_p = \alpha$ [12,13,9,10,11,14].[2] Informally, one can think of ψ as a programming language (e.g., C++, Java, Haskell) and of p as a *program* within that language. In this sense, ψ_p is the partial computable function coded by ψ-program p. Thus, for all $p, x \in \mathbb{N}$, $\psi(\langle p, x \rangle)$ is the (coded) output on input (coded by) x of the program (coded by) p in that language.

For the remainder of the present subsection (1.1), let ψ be any fixed eps. The following property (Property 1) is the not-necessarily-constructive form of Kleene's Recursion Theorem *for the ψ-system*.[3]

*Property 1 (***krt** *for* eps *ψ).* $(\forall p)(\exists e)(\forall x)[\psi_e(x) = \psi_p(\langle e, x \rangle)]$.

One way to interpret Property 1 is as follows. ψ-program p represents an arbitrary preassigned, algorithmic task to perform with a self-copy; e represents a ψ-program that

1. creates a copy of itself, external to itself, and, *then*,
2. runs the preassigned task p on the pair consisting of this self-copy and e's input x.

The 'e' on the right-hand side of the equation in Property 1 *is* the self-copy of the original 'e' on the left-hand side of this equation. Thus, in an important sense, e is a program that *creates* complete (low level) *self-knowledge*. The way in

[2] In much of the literature on epses, e.g., [12,7,8], they are called *numberings* since, in such systems, programs are conveniently named by numbers. In learning theory contexts, e.g., [6,18,5], they are also referred to as *hypothesis spaces*.

[3] Rogers [13] popularized a fixed-point variant of Property 1: for all computable $f : \mathbb{N} \to \mathbb{N}$, there exists e such that $\psi_e = \psi_{f(e)}$. His variant should *not* be confused with Property 1. Riccardi [10] explored their interconnections.

which e uses this self-knowledge is according to how the preassigned task p says to.[4] Infinite regression is *not* needed since e projects its self-copy *externally* to itself [4]. We say above that this self-knowledge is complete since it *is* e's syntactic code-script, wiring/flow diagram, etc. For higher level knowledge about, say, e's behavioral propensities, e.g., ψ-program e runs in polynomial time, p can run a safe theorem prover on e perchance to prove such things about e, but e having access to e itself is more basic and fundamental than e merely having access to facts such as that it runs in polynomial time.

Self-knowledgeable programs have long been known to be an elegant theoretical tool in computability theory. Such programs can, when relevant, provide *succinct* solutions to problems "that would otherwise require extensive, complex treatment" [13] (see also [15]). Self-knowledge can also serve as a useful game-theoretic aid to strategy [4], e.g., in the *game* played between a robot and its environment [1,3].

Of course, Property 1 asserts that, given p, there merely *exists* an e satisfying the equation in Property 1 for p. It is another problem to *find* such an e algorithmically *from* p. Here, then, is Property 2, the *constructive* form of Kleene's Recursion Theorem for the ψ-system, which makes this stronger assertion.

*Property 2 (***KRT** *for* eps ψ*).* There exists computable $r : \mathbb{N} \rightarrow \mathbb{N}$ such that $(\forall p, x)\big[\psi_{r(p)}(x) = \psi_p\big(\langle r(p), x\rangle\big)\big]$.

In Property 2, $r(p)$ plays the role of e in Property 1. Since r is computable, $r(p)$ can be found algorithmically from p.

1.2 Control Structures

From a programming languages standpoint, the r in Property 2 represents an *instance* (or *implementation*) of a *control structure* [10,11,14,8,5]. In the context of epses, an instance of a control structure provides a means of forming a composite program from given constituent programs and/or data. For comparison, an *instance in an* eps ψ *of the control structure* **if-then-else** is (by definition [10,11]) a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that, for all a, b, c, and x,

$$\psi_{f(\langle a,b,c\rangle)}(x) = \begin{cases} \psi_b(x), & \text{if } \psi_a(x) \text{ converges}^{5} \text{ and } \psi_a(x) > 0; \\ \psi_c(x), & \text{if } \psi_a(x) \text{ converges and } \psi_a(x) = 0; \\ \text{divergent, otherwise.} \end{cases} \tag{1}$$

An instance such as f above of **if-then-else** combines three ψ-programs, a, b, and c (and *no* data) to form a fourth (composite) ψ-program $f(\langle a, b, c\rangle)$.

[4] We care, of course, that **krt** provides not only self-knowledgeable programs, but also, self-knowledgeable programs that can *use* that knowledge in any preassigned algorithmic way. *Usable*, as opposed to empty, self-knowledge is what we care about.

[5] For all one-argument partial functions ψ and $x \in \mathbb{N}$, $\psi(x)$ *converges* iff there exists $y \in \mathbb{N}$ such that $\psi(x) = y$; $\psi(x)$ *diverges* iff there is *no* $y \in \mathbb{N}$ such that $\psi(x) = y$. If ψ is partial computable, and x is such that $\psi(x)$ diverges, then one can imagine that a program associated with ψ *goes into an infinite loop* on input x.

if-then-else is an example of a *nonrecursive denotational control structure* (synonym: *nonrecursive extensional control structure*). A *non*recursive denotational control structure is one for which the I/O behavior of a composite program may depend *only* upon the I/O behavior of the constituent programs and upon the data (see (a) of Definition 1 below). So, for example, the I/O behavior of such a composite program can*not* depend upon the number of symbols in, or the run-time complexity of, a constituent program.

A *recursive denotational control structure* (synonym: *recursive extensional control structure*) is like a *non*recursive denotational control structure where the I/O behavior of a composite program may depend, additionally, upon the I/O behavior of the composite program itself (see (b) of Definition 1 below). Consider the following example, chosen for illustrative purposes. Let an effective instance in an eps ψ of **recursive unbounded minimization** be a computable function $f : \mathbb{N} \to \mathbb{N}$ such that, for all a, b, and x,

$$\psi_{f(\langle a,b\rangle)}(x) = \begin{cases} \psi_b(x), & \text{if } \psi_a(x) \text{ converges and } \psi_a(x) > 0; \\ \psi_{f(\langle a,b\rangle)}(x+1), & \text{if } \psi_a(x) \text{ converges and } \psi_a(x) = 0; \\ \text{divergent}, & \text{otherwise.} \end{cases} \tag{2}$$

Note the use of $\psi_{f(\langle a,b\rangle)}$ in the second if-clause in (2). This use of $\psi_{f(\langle a,b\rangle)}$ is what makes **recursive unbounded minimization** a *recursive* denotational control structure.

For many recursive denotational control structures, there is *wiggle room* in how they may be implemented. For **recursive unbounded minimization**, this wiggle room manifests itself in the extreme diversity of the functions f that satisfy (2). For example, suppose that $f_1 : \mathbb{N} \to \mathbb{N}$ is computable and that, for all a, b, and x,

$$\psi_{f_1(\langle a,b\rangle)}(x) = \begin{cases} \psi_b(z), & \text{where } z \text{ is } \textit{least} \text{ such that } z \geq x, \\ & \quad (\forall y \in \{x, ..., z\})[\psi_a(y) \text{ converges}], \\ & \quad (\forall y \in \{x, ..., z-1\})[\psi_a(y) = 0], \\ & \quad \text{and } \psi_a(z) > 0, \text{ if such a } z \text{ exists;} \\ \text{divergent, otherwise.} \end{cases} \tag{3}$$

Then, $f = f_1$ is a solution of (2).

Next, suppose that a_0 is a ψ-program such that, for all x, $\psi_{a_0}(x)$ converges and $\psi_{a_0}(x) = 0$. Note that when $a = a_0$ in (2), (2) merely insists that $\psi_{f(\langle a,b\rangle)}(0) = \psi_{f(\langle a,b\rangle)}(1) = ...$, for any b. Thus, if $f_2 : \mathbb{N} \to \mathbb{N}$ is computable and, for all a, b, and x,

$$\psi_{f_2(\langle a,b\rangle)}(x) = \begin{cases} 5, & \text{if } a = a_0; \\ \psi_{f_1(\langle a,b\rangle)}(x), & \text{otherwise;} \end{cases} \tag{4}$$

then, $f = f_2$ is also a solution of (2). (In (4), the number 5 was chosen arbitrarily.)[6]

[6] Readers familiar with denotational semantics may recognize that f_1 provides a minimal fixed-point solution of (2); whereas, f_2 provides a *non*-minimal fixed-point solution of (2) [14,17].

Of course, a composite program produced by a recursive denotational control structure *may* choose to *ignore* its own behavior. In this sense, recursive denotational control structures are a generalization of *non*recursive denotational control structures.

KRT, when viewed as a control structure, is *not* denotational in any sense.[7] So, a problem that the first author posed to Royer was to find a collection of denotational control structures whose implementability characterizes the epses in which **KRT** holds. The thinking was that *denotational control structures are easier to understand*, and such a collection would be a decomposition of **KRT** into more easily understood components. Royer proved that *no* such characterization exists, even if one allows the collection to contain *recursive* denotational control structures [14].

1.3 Summary of Results

krt is the focus in the present paper as it ostensibly involves *pure* self-reference without the required constructivity of **KRT**. So, a question we had is whether Royer's negative result mentioned above still holds if one replaces **KRT** by **krt**, i.e., whether there exists a collection of (possibly) recursive denotational control structures whose implementability characterizes the epses in which **krt** holds. One of our main results, Corollary 1 in Section 3, says that *no* such characterization exists. The proof is by a finite injury priority argument.[8]

In Section 3, we also consider a *relatively constructive* variant of **krt**. Suppose ξ is an eps and that ψ is partial computable, but not-necessarily an eps. We say that ξ-**KRT** *holds in* $\psi \overset{\text{def}}{\Leftrightarrow}$ there exists computable $r : \mathbb{N} \rightarrow \mathbb{N}$ such that $(\forall p, x)\big[\psi_{r(p)}(x) = \xi_p\big(\langle r(p), x\rangle\big)\big]$. Here, $r(p)$ is a self-knowledgeable ψ-program, where the preassigned task for $r(p)$ to employ on its self-copy is ξ-program p.

Theorem 2, our other main result, says: ψ is an eps in which **krt** holds $\Leftrightarrow$ $(\exists$ eps $\xi)[\xi$-**KRT** holds in $\psi]$. This implies that, if, for some eps ξ, ξ-**KRT** holds in merely partial computable ψ, then both ψ is an eps and **krt** holds in ψ. It also surprisingly implies that, if **krt** holds in an eps ψ, then it holds with *some degree of constructivity* — constructivity with respect to *some* eps ξ.

Section 2 just below provides notation and preliminaries.

Due to space constraints, many of the details of the proofs in Section 3 have been omitted.

2 Notation and Preliminaries

Computability-theoretic concepts not explained below are treated in [13]. $\mathbb{N}$ denotes the set of natural numbers. $2\mathbb{N}$ and $2\mathbb{N}+1$ denote the sets of even and odd natural numbers, respectively. Lowercase Roman letters, with or without decorations, range over elements of $\mathbb{N}$ unless stated otherwise.

[7] Such control structures are called *connotational* [14].

[8] Rogers [13] explains priority arguments.

The pairing function $\langle\cdot,\cdot\rangle$ was introduced in Section 1. For all x, $\langle x\rangle \overset{\text{def}}{=} x$. For all $x_1,...,x_n$, where $n>2$, $\langle x_1,...,x_n\rangle \overset{\text{def}}{=} \langle x_1, \langle x_2,...,x_n\rangle\rangle$.

$\mathcal{P}$ denotes the collection of all one-argument partial functions. α, ξ, Ξ, σ, ψ, and Ψ, with or without decorations, range over elements of $\mathcal{P}$. We use Church's lambda-notation [13] to name partial functions, including total functions and predicates, as is standard in many programming languages. For example, $\lambda x.\,(x+1)$ denotes the one-argument (total) function that maps a natural number to its successor.

For all α and x, $\alpha(x)\downarrow$ denotes that $\alpha(x)$ converges; $\alpha(x)\uparrow$ denotes that $\alpha(x)$ diverges. We use $\uparrow$ in expressions to indicate divergence. For example, $\lambda x.\uparrow$ denotes the everywhere divergent partial computable function. For all α, $\mathrm{dom}(\alpha) \overset{\text{def}}{=} \{x : \alpha(x)\downarrow\}$ and $\mathrm{rng}(\alpha) \overset{\text{def}}{=} \{y : (\exists x)[\alpha(x)=y]\}$. We identify a partial function with its graph, e.g., we identify α with the set $\{(x,y) : \alpha(x)=y\}$. As noted in the introduction, for all ψ and p, $\psi_p \overset{\text{def}}{=} \psi_p(\langle p,\cdot\rangle)$.

$F_0, F_1, ...$ denotes a fixed, canonical enumeration of all one-argument finite functions [13,9].

φ denotes a fixed, acceptable eps.[9] Φ denotes a fixed Blum complexity measure for φ [2].[10] For all p and t, φ_p^t and W_p^t are as follows.

$$\varphi_p^t \overset{\text{def}}{=} \{(x,y) : x \le t \ \wedge\ \Phi_p(x) \le t \ \wedge\ \varphi_p(x) = y\}. \tag{5}$$

$$W_p^t \overset{\text{def}}{=} \mathrm{dom}(\varphi_p^t). \tag{6}$$

Γ and Θ, with or without decorations, range over mappings of type $\mathbb{N}^m \times \mathcal{P}^n \to \mathcal{P}$, where $m+n>0$.

For all $\Gamma : \mathbb{N}^m \times \mathcal{P}^n \to \mathcal{P}$, where $m+n>0$, Γ is a *computable operator*[11] $\overset{\text{def}}{\Leftrightarrow}$ there exists p such that, for all $x_1,...,x_m$, $\alpha_1,...,\alpha_n$, y, and z,

$$\begin{gathered}\Gamma(x_1,...,x_m,\alpha_1,...,\alpha_n)(y) = z \\ \Leftrightarrow \\ (\exists i_1,...,i_n,t)\big[(\forall j\in\{1,...,n\})[F_{i_j} \subseteq \alpha_j] \\ \wedge\ \langle x_1,...,x_m,i_1,...,i_n,y,z\rangle \in W_p^t\big].\end{gathered} \tag{7}$$

Intuitively, Γ is a computable operator iff there exists an algorithm for listing the *graph of* the partial function $\Gamma(x_1,...,x_m,\alpha_1,...,\alpha_n)$ from $x_1,...,x_m$ and the *graphs of* the partial functions $\alpha_1,...,\alpha_n$ — independently of the enumeration order chosen for each of $\alpha_1,...,\alpha_n$. $\mathcal{C}$ ranges over collections of computable operators. For all computable operators $\Gamma : \mathbb{N}^m \times \mathcal{P}^n \to \mathcal{P}$, where $m+n>0$, and for

[9] An eps ψ is *acceptable* $\overset{\text{def}}{\Leftrightarrow}$ ($\forall$ epses ξ)($\exists$ computable $t : \mathbb{N}\to\mathbb{N}$)$(\forall p)[\psi_{t(p)} = \xi_p]$ [12,13,9,10,11,14]. Thus, the acceptable epses are exactly those epses into which every eps can be compiled. Any eps corresponding to a real-world, general purpose programming language (e.g., C++, Java, Haskell) is acceptable.

[10] For any partial computable function, e.g., an eps, many such measures exist.

[11] Rogers [13] calls the computable operators, *recursive operators*. We have chosen to use the former term so that we may reserve the term *recursive* for something that *refers to itself*.

all t, $\Gamma^t : \mathbb{N}^m \times \mathcal{P}^n \rightarrow \mathcal{P}$ is the computable operator such that, for all $x_1, ..., x_m$, $\alpha_1, ..., \alpha_n$, y, and z,

$$\begin{gathered} \Gamma^t(x_1, ..., x_m, \alpha_1, ..., \alpha_n)(y) = z \\ \Leftrightarrow \\ (\exists i_1, ..., i_n)\big[(\forall j \in \{1, ..., n\})[F_{i_j} \subseteq \alpha_j] \\ \wedge \langle x_1, ..., x_m, i_1, ..., i_n, y, z\rangle \in W_p^t\big], \end{gathered} \tag{8}$$

where p is any fixed φ-program as in (7) above for Γ. Clearly, for all computable operators $\Gamma : \mathbb{N}^m \times \mathcal{P}^n \rightarrow \mathcal{P}$, where $m+n > 0$, there exists an algorithm for finding j from t, $x_1, ..., x_m$, and $i_1, ..., i_n$, such that $F_j = \Gamma^t(x_1, ..., x_m, F_{i_1}, ..., F_{i_n})$.

Definition 1. For all epses ψ, and all $f : \mathbb{N} \rightarrow \mathbb{N}$, (a) and (b) below.

(a) For all computable operators $\Gamma : \mathbb{N}^m \times \mathcal{P}^n \rightarrow \mathcal{P}$, where $m + n > 0$, *f is an effective instance in ψ of the* non*recursive denotational control structure determined by Γ* $\Leftrightarrow$ f is computable and, for all $x_1, ..., x_{m+n}$,

$$\psi_{f(\langle x_1,...,x_{m+n}\rangle)} = \Gamma(x_1, ..., x_m, \psi_{x_{m+1}}, ..., \psi_{x_{m+n}}). \tag{9}$$

(b) For all computable operators $\Theta : \mathbb{N}^m \times \mathcal{P}^{n+1} \rightarrow \mathcal{P}$, where $m + n > 0$, *f is an effective instance in ψ of the recursive denotational control structure determined by Θ* $\Leftrightarrow$ f is computable and, for all $x_1, ..., x_{m+n}$,

$$\psi_{f(\langle x_1,...,x_{m+n}\rangle)} = \Theta(x_1, ..., x_m, \psi_{x_{m+1}}, ..., \psi_{x_{m+n}}, \psi_{f(\langle x_1,...,x_{m+n}\rangle)}). \tag{10}$$

3 Results

Royer [14] proved that there is *no* collection of recursive denotational control structures whose implementability characterizes the epses in which **KRT** holds. Corollary 1, below, proves the analogous result for **krt**. Thus, even the *pure* self-reference embodied by **krt** cannot be decomposed into recursive denotational control structures.[12] Our proof is by a finite injury priority argument.

Definition 2. For all computable operators $\Theta : \mathbb{N}^m \times \mathcal{P}^{n+1} \rightarrow \mathcal{P}$, where $m+n > 0$, *Θ is recursively denotationally omnipresent* $\Leftrightarrow$ for all epses ψ, there exists an effective instance in ψ of the recursive denotational control structure determined by Θ.

Theorem 1. Suppose that computable operator $\Theta : \mathbb{N}^m \times \mathcal{P}^{n+1} \rightarrow \mathcal{P}$, where $m + n > 0$, is *not* recursively denotationally omnipresent. Then, there exists an eps ψ such that (a) and (b) below.

(a) **krt** holds in ψ.
(b) There is *no* effective instance in ψ of the recursive denotational control structure determined by Θ.

Proof (Sketch). We give here the construction. We omit the proof of its correctness. Let Θ be as stated. Since Θ is *not* recursively denotationally omnipresent, there exists an eps ξ such that there is *no* effective instance in ξ of the recursive denotational control structure determined by Θ.

[12] N.B. Our result does *not* subsume Royer's.

ψ is constructed via a finite injury priority argument. The requirements, in order of *decreasing* priority, are: $S, R_0, R_1, ...$, where, for all q, R_q and S are as follows.

$$\begin{aligned} &R_q \Leftrightarrow (\exists a)[\psi_a = \varphi_q(\langle a, \cdot \rangle)]. \\ &S \ \ \Leftrightarrow (\forall \text{ computable } g : \mathbb{N} \to \mathbb{N})(\exists x_1, ..., x_{m+n}) \\ &\qquad [\psi_{g(\langle x_1,...,x_{m+n}\rangle)} \neq \Theta(x_1, ..., x_m, \psi_{x_{m+1}}, ..., \psi_{x_{m+n}}, \psi_{g(\langle x_1,...,x_{m+n}\rangle)})]. \end{aligned}$$

The satisfaction of R_q, for all q, ensures that ψ is an eps in which **krt** holds. The satisfaction of S ensures that there is *no* effective instance in ψ of the recursive denotational control structure determined by Θ.

ψ is constructed in stages. For all a and t, ψ_a^t denotes ψ_a at the beginning of stage t. For all a, $\psi_a^0 = \lambda x \centerdot \uparrow$. For all a, t, and x, $\psi_a^{t+1}(x) = \psi_a^t(x)$ unless stated otherwise.

In conjunction with ψ, a partial computable σ and a limit-computable d are constructed. σ and d are used to help satisfy the S requirement. For all t, σ^t and d^t denote σ and d, respectively, at the beginning of stage t. For all a, σ^0 and d^0 are as follows.

$$\sigma^0(a) = \begin{cases} (a-1) \div 2, & \text{if } a \in 2\mathbb{N}+1; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{11}$$

$$d^0(a) = 0. \tag{12}$$

For all t and a, $\sigma^{t+1}(a) = \sigma^t(a)$ unless stated otherwise. Similarly, for all t and a, $d^{t+1}(a) = d^t(a)$ unless stated otherwise. The following will be clear, by construction.

$$(\forall t)[\text{dom}(\sigma^t) \cap 2\mathbb{N} \text{ is finite}]. \tag{13}$$

$$\lambda t, a \centerdot [\sigma^t(a)\downarrow] \text{ is a computable predicate.} \tag{14}$$

Let r be such that, for all t and q,

$$r^t(q) = \begin{cases} 2\langle q, i\rangle, \text{ where } i \text{ is } \textit{least} \text{ such that } \sigma^t(2\langle q, i\rangle)\uparrow \\ \quad \text{and } (\forall p < q)[2\langle q, i\rangle > r^t(p)]. \end{cases} \tag{15}$$

r is used to help satisfy the R requirements. It can be shown, by a straightforward induction, that, if (13) holds as claimed, then, for all t, r^t is total and monotonically increasing. Furthermore, if (14) holds as claimed, then $\lambda t, q \centerdot r^t(q)$ is computable. Clearly, by (15), if t, q, and i are such that $r^t(q) = 2\langle q, i\rangle$, then $\sigma^t(2\langle q, i\rangle)\uparrow$. It follows that, for all t, $\text{dom}(\sigma^t) \cap \text{rng}(r^t) = \emptyset$.

For all q and t, it can be seen that R_q is injured in stage t whenever $r^{t+1}(q) \neq r^t(q)$. There are two ways that this can occur. The first is when $[(\sigma^t \circ r^t)(q)\uparrow \ \wedge \ (\sigma^{t+1} \circ r^t)(q)\downarrow]$, equivalently, $(\sigma \circ r^t)(q)$ becomes defined in stage $t+1$. The second is when, for some $p < q$, $[r^t(p) < r^t(q) \ \wedge \ r^{t+1}(p) \geq r^t(q)]$. In this latter case, R_q is injured as a result of a *cascading effect*. Clearly, either condition causes $r^{t+1}(q) \neq r^t(q)$.

Let Ξ be a Blum complexity measure for ξ. For all p and t, let

$$\xi_p^t = \{(x, y) : x \leq t \ \wedge \ \Xi_p(x) \leq t \ \wedge \ \xi_p(x) = y\}. \tag{16}$$

Construct ψ, σ, and d by executing successive stages $t = 0, 1, ...$ as follows.

Stage $t = \langle a, i\rangle$.

Case $\sigma^t(a)\downarrow$. Let $p = \sigma^t(a)$ and, for all $x \le t+1$ such that $[\psi_a^t(x)\uparrow \wedge \xi_p^t(x)\downarrow]$, set $\psi_a^{t+1}(x) = \xi_p^t(x)$.

Case $a \in \mathrm{rng}(r^t)$. Perform steps (1) and (2) below.

(1) Let $s = d^t(a)$ and determine whether conditions (a) and (b) below are satisfied.

  (a) $\psi_a^t(s)\downarrow$.

  (b) For all $\langle x_1, ..., x_{m+n}\rangle < s$ and b such that $\psi_a^t(\langle x_1, ..., x_{m+n}\rangle) = b$, (i) and (ii) below.

    (i) $\psi_b^s \subseteq \Theta^t(x_1, ..., x_m, \psi_{x_{m+1}}^t, ..., \psi_{x_{m+n}}^t, \psi_b^t)$.

    (ii) $\Theta^s(x_1, ..., x_m, \psi_{x_{m+1}}^s, ..., \psi_{x_{m+n}}^s, \psi_b^s) \subseteq \psi_b^t$.

  If conditions (a) and (b) are satisfied, then perform substeps $(*)$ and $(**)$ below.

  $(*)$ Let $c = \psi_a^t(s)$. If $[c > a \ \wedge \ \sigma^t(c)\uparrow]$, then find any p such that $\psi_c^t \subseteq \xi_p$ and set $\sigma^{t+1}(c) = p$.

  $(**)$ Set $d^{t+1}(a) = s+1$.

(2) Let q be be such that $r^t(q) = a$ and, for all $x \le t+1$ such that $[\psi_a^t(x)\uparrow \ \wedge \ \varphi_q^t(\langle a, x\rangle)\downarrow]$, set $\psi_a^{t+1}(x) = \varphi_q^t(\langle a, x\rangle)$.

End of construction of ψ, σ, and d. $\approx\square$ *(Theorem 1)*

Corollary 1. There is *no* collection of computable operators $\mathcal{C}$ such that (a) *and* (b) below.

(a) For each $\Theta \in \mathcal{C}$, Θ has type $\mathbb{N}^m \times \mathcal{P}^{n+1} \to \mathcal{P}$, for some m and n, where $m + n > 0$.[13]

(b) For all epses ψ, **krt** holds in $\psi \Leftrightarrow (\forall \Theta \in \mathcal{C})$[there exists an effective instance in ψ of the recursive denotational control structure determined by Θ].

Theorem 2, below, is our other main result. It reveals that a hidden and inherent constructivity is always present in **krt**.

Definition 3. For all epses ξ and partial computable ψ, ξ-**KRT** *holds in* $\psi \Leftrightarrow$ $(\exists$ computable $r : \mathbb{N} \to \mathbb{N})(\forall p, x)\big[\psi_{r(p)}(x) = \xi_p\big(\langle r(p), x\rangle\big)\big]$.

Theorem 2. For all partial computable ψ, ψ is an eps in which **krt** holds $\Leftrightarrow$ $(\exists$ eps $\xi)[\xi$-**KRT** holds in $\psi]$.

Proof (Sketch).

$(\Rightarrow)$ Let ψ be as stated. Let ξ be such that, for all a, b, x_1, and x_2,

$$\xi_{\langle a,b\rangle}(\langle x_1, x_2\rangle) = \begin{cases} \psi_a(x_2), & \text{if } x_1 = a; \\ \psi_b(\langle x_1, x_2\rangle), & \text{otherwise.} \end{cases} \tag{17}$$

Clearly, ξ is partial computable. Thus, to show that ξ is an eps, it suffices to show that, for all b, there exists a such that $\xi_{\langle a,b\rangle} = \psi_b$. Let ψ-program b be fixed. By **krt** in ψ, there exists a such that, for all x, $\psi_a(x) = \psi_b(\langle a, x\rangle)$. For all x_1 and x_2, consider the following cases.

[13] (a) ensures that each $\Theta \in \mathcal{C}$ determines a recursive denotational control structure (see (b) of Definition 1).

CASE $x_1 = a$. Then,

$$\begin{aligned}\xi_{\langle a,b\rangle}(\langle a, x_2\rangle) &= \psi_a(x_2) && \{\text{by (17)}\}\\ &= \psi_b(\langle a, x_2\rangle) && \{\text{by the choice of } a\}.\end{aligned}$$

CASE $x_1 \neq a$. Then, by (17), $\xi_{\langle a,b\rangle}(\langle x_1, x_2\rangle) = \psi_b(\langle x_1, x_2\rangle)$.

Furthermore, it can be shown that $\lambda\langle a, b\rangle\centerdot a$ witnesses ξ-**KRT** in ψ (details omitted).

($\Leftarrow$) Omitted. $\approx\square$ *(Theorem 2)*

References

1. Adami, C.: What do robots dream of? Science 314, 1093–1094 (2006)
2. Blum, M.: A machine independent theory of the complexity of recursive functions. Journal of the ACM 14, 322–336 (1967)
3. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science 314, 1118–1121 (2006)
4. Case, J.: Infinitary self-reference in learning theory. Journal of Experimental and Theoretical Artificial Intelligence 6, 3–16 (1994)
5. Case, J., Jain, S., Suraj, M.: Control structures in hypothesis spaces: The influence on learning. Theoretical Computer Science 270(1-2), 287–308 (2002)
6. Freivalds, R., Kinber, E., Wiehagen, R.: Inductive inference and computable one-one numberings. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 28, 463–479 (1982)
7. Goncharov, S., Sorbi, A.: Generalized computable numberings and non-trivial Rogers semilattices. Algebra and Logic 36, 359–369 (1997)
8. Jain, S., Nessel, J.: Some independence results for control structures in complete numberings. Journal of Symbolic Logic 66(1), 357–382 (2001)
9. Machtey, M., Young, P.: An Introduction to the General Theory of Algorithms. North Holland, New York (1978)
10. Riccardi, G.: The Independence of Control Structures in Abstract Programming Systems. PhD thesis, SUNY Buffalo (1980)
11. Riccardi, G.: The independence of control structures in abstract programming systems. Journal of Computer and System Sciences 22, 107–143 (1981)
12. Rogers, H.: Gödel numberings of partial recursive functions. Journal of Symbolic Logic 23, 331–341 (1958)
13. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York, 1967. Reprinted, MIT Press (1987)
14. Royer, J.: A Connotational Theory of Program Structure. LNCS, vol. 273. Springer, Heidelberg (1987)
15. Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. Research monograph in Progress in Theoretical Computer Science. Birkhäuser Boston (1994)
16. Smorynski, C.: Fifty years of self-reference in arithmetic. Notre Dame Journal of Formal Logic 22(4), 357–374 (1981)
17. Winskel, G.: The Formal Semantics of Programming Languages: An Introduction. Foundations of Computing Series. MIT Press, Cambridge (1993)
18. Zeugmann, T., Lange, S.: A guided tour across the boundaries of learning recursive languages. In: Jantke, K.P., Lange, S. (eds.) Algorithmic Learning for Knowledge-Based Systems. LNCS (LNAI), vol. 961, pp. 190–258. Springer, Heidelberg (1995)

K-Trivial Closed Sets and Continuous Functions

George Barmpalias[1], Douglas Cenzer[2,⋆], Jeffrey B. Remmel[3], and Rebecca Weber[4]

[1] School of Mathematics, University of Leeds,
Leeds LS2 9JT, England
georgeb@maths.leeds.ac.uk
[2] Department of Mathematics, University of Florida,
P.O. Box 118105, Gainesville, Florida 32611
cenzer@math.ufl.edu
[3] Department of Mathematics, University of California, San Diego
La Jolla, CA 92093-0112
jremmel@ucsd.edu
[4] Department of Mathematics, Dartmouth College,
Hanover, NH 03755-3551
rweber@math.dartmouth.edu

Abstract. We investigate the notion of K-triviality for closed sets and continuous functions. Every K-trivial closed set contains a K-trivial real. There exists a K-trivial Π^0_1 class with no computable elements. For any K-trivial degree $\mathbf{d}$, there is a K-trivial continuous function of degree $\mathbf{d}$.[1]

Keywords: Computability, Randomness, Π^0_1 Classes.

1 Introduction

The study of algorithmic randomness has been an active area of research in recent years. The basic problem is to quantify the randomness of a single real number. Here we think of a real $r \in [0, 1]$ as an infinite sequence of 0's and 1's, i.e as an element in $2^{\mathbb{N}}$. There are three basic approaches to algorithmic randomness: the measure theoretic, the compressibility and the betting approaches. All three approaches have been shown to yield the same notion of (algorithmic) randomness. Here we will only use notions from the compressibility approach, incorporating a number of non-trivial results in this area. For background and history of algorithmic randomness we refer to [11,10,13].

Prefix-free (Chaitin) complexity for reals is defined as follows. Let M be a prefix-free function with domain $\subset \{0, 1\}^*$. For any finite string τ, let $K_M(\tau) = min\{|\sigma| : M(\sigma) = \tau\}$. There is a *universal* prefix-free function U such that, for any prefix-free M, there is a constant c such that for all τ

$$K_U(\tau) \leq K_M(\tau) + c.$$

⋆ Corresponding author.

[1] This research was partially supported by NSF grants DMS 0532644 and 0554841. Remmel was also partially supported by NSF grant DMS 0400507.

We let $K(\sigma) = K_U(\sigma)$. Then x is said to be *random* if there is a constant c such that $K(x\lceil n) \geq n - c$ for all n. This means a real x is random exactly when its initial segments are not *compressible*.

In a series of recent papers [1,2,3,4], P. Brodhead, S. Dashti and the authors have defined the notion of (algorithmic) randomness for closed sets and continuous functions on $2^{\mathbb{N}}$. Some definitions are needed. For a finite string $\sigma \in \{0,1\}^n$, let $|\sigma| = n$. For two strings σ, τ, say that τ *extends* σ and write $\sigma \prec \tau$ if $|\sigma| \leq |\tau|$ and $\sigma(i) = \tau(i)$ for $i < |\sigma|$. For $x \in 2^{\mathbb{N}}$, $\sigma \prec x$ means that $\sigma(i) = x(i)$ for $i < |\sigma|$. Let $\sigma^\frown\tau$ denote the concatenation of σ and τ and let $\sigma^\frown i$ denote $\sigma^\frown(i)$ for $i = 0, 1$. Let $x\lceil n = (x(0), \ldots, x(n-1))$. Two reals x and y may be coded together into $z = x \oplus y$, where $z(2n) = x(n)$ and $z(2n+1) = y(n)$ for all n. For a finite string σ, let $I(\sigma)$ denote $\{x \in 2^{\mathbb{N}} : \sigma \prec x\}$. We shall call $I(\sigma)$ the *interval* determined by σ. Each such interval is a clopen set and the clopen sets are just finite unions of intervals. Now a nonempty closed set P may be identified with a tree $T_P \subseteq \{0,1\}^*$ where $T_P = \{\sigma : P \cap I(\sigma) \neq \emptyset\}$. Note that T_P has no dead ends. That is, if $\sigma \in T_P$, then either $\sigma^\frown 0 \in T_P$ or $\sigma^\frown 1 \in T_P$. For an arbitrary tree $T \subseteq \{0,1\}^*$, let $[T]$ denote the set of infinite paths through T. For a detailed development of Π^0_1 classes, see [5].

We define a measure μ^* on the space $\mathcal{C}$ of closed subsets of $2^{\mathbb{N}}$ as follows. Given a closed set $Q \subseteq 2^{\mathbb{N}}$, let $T = T_Q$ be the tree without dead ends such that $Q = [T]$. Let $\sigma_0, \sigma_1, \ldots$ enumerate the elements of T in order, first by length and then lexicographically. We then define the code $x = x_Q = x_T$ by recursion such that for each n, $x(n) = 2$ if both $\sigma_n{}^\frown 0$ and $\sigma_n{}^\frown 1$ are in T, $x(n) = 1$ if $\sigma_n{}^\frown 0 \notin T$ and $\sigma_n{}^\frown 1 \in T$, and $x(n) = 0$ if $\sigma_n{}^\frown 0 \in T$ and $\sigma_n{}^\frown 1 \notin T$. We then define μ^* by setting

$$\mu^*(\mathcal{X}) = \mu(\{x_Q : Q \in \mathcal{X}\}) \tag{1}$$

for any $\mathcal{X} \subseteq \mathcal{C}$ and μ is the standard measure on $\{0,1,2\}^{\mathbb{N}}$. Informally this means that given $\sigma \in T_Q$, there is probability $\frac{1}{3}$ that both $\sigma^\frown 0 \in T_Q$ and $\sigma^\frown 1 \in T_Q$ and, for $i = 0, 1$, there is probability $\frac{1}{3}$ that only $\sigma^\frown i \in T_Q$. In particular, this means that $Q \cap I(\sigma) \neq \emptyset$ implies that for $i = 0, 1$, $Q \cap I(\sigma^\frown i) \neq \emptyset$ with probability $\frac{2}{3}$. Brodhead, Cenzer, and Dashti [2] defined a a closed set $Q \subseteq 2^{2^{\mathbb{N}}}$ to be (Martin-Löf) random if x_Q is (Martin-Löf) random. Note that the equal probability of $\frac{1}{3}$ for the three cases of branching allows the application of Schnorr's theorem that Martin-Löf randomness is equivalent to prefix-free Kolmogorov randomness. Then in [2,3], the following results are proved. Every random closed set is perfect and contains no computable elements (in fact, it contains no n-c.e. elements). Every random closed set has measure 0 and has box dimension $\log_2 \frac{4}{3}$.

A continuous function $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ may be represented by a function $f : \{0,1\}^* \to \{0,1\}^*$ such that the following hold, for all $\sigma \in \{0,1\}^*$.

- $|f(\sigma)| \leq |\sigma|$.
- $\sigma_1 \prec \sigma_2$ implies $f(\sigma_1) \prec f(\sigma_2)$.
- For every n, there exists m such that for all $\sigma \in \{0,1\}^m$, $|f(\sigma)| \geq n$.
- For all $x \in 2^{\mathbb{N}}$, $F(x) = \bigcup_n f(x\lceil n)$.

We define the space $\mathcal{F}$ of representing functions $f : \{0,1\}^* \to \{0,1\}^*$ to be those which satisfy clauses (1) and (2) above. There is a one-to-one correspondence between $\mathcal{F}$ and $\{0,1,2\}^{\mathbb{N}}$ defined as follows. Enumerate $\{0,1\}^*$ in order, first by length and then lexicographically, as $\sigma_0, \sigma_1, \ldots$. Thus $\sigma_0 = \emptyset$, $\sigma_1 = (0), \sigma_2 = (1), \sigma_3 = (00), \ldots$. Then $r \in \{0,1,2\}^{\mathbb{N}}$ corresponds to the function $f_r : \{0,1\}^* \to \{0,1\}^*$ defined by declaring that $f_r(\emptyset) = \emptyset$ and that, for any σ_n with $|\sigma_n| \geq 1$,

$$f_r(\sigma_n) = \begin{cases} f_r(\sigma_k), & \text{if } r(n) = 2; \\ f_r(\sigma_k)^\frown i, & \text{if } r(n) = i < 2. \end{cases}$$

where k is such that $\sigma_n = \sigma_k{}^\frown j$ for some j. Every continuous function F has a representative f as described above, and, in fact, it has infinitely many representatives. We define a measure μ^{**} on $\mathcal{F}$ induced by the standard probability measure on $\{0,1,2\}^{\mathbb{N}}$. Brodhead, Cenzer, and Remmel [4] defined an (Martin-Löf) random continuous function on $2^{\mathbb{N}}$ which has a representation in $\mathcal{F}$ which is Martin-Löf random. The following results are proved in [1,4]. Random Δ^0_2 continuous functions exist, but no computable function can be random and no random function can map a computable real to a computable real. The image of a random continuous function is always a perfect set and hence uncountable. For any $y \in 2^{\mathbb{N}}$, there exists a random continuous function F with y in the image of F. Thus the image of a random continuous function need not be a random closed set. The set of zeroes of a random continuous function is a random closed set (if nonempty).

There has been a considerable amount of work on studying reals whose complexity is "low" or trivial from the point of view of randomness. Chaitin defined a real x to be K-trivial if $K(x\lceil n) \leq K(1^n)+O(1)$. We recall that there are noncomputable c.e. sets which are K-trivial and that the K-trivial reals are downward closed under Turing reducibility. The latter is a highly non-trivial result of Nies [15] who also showed that the K-trivial reals form a Σ^0_3-definable ideal in the Turing degrees. In particular, this means that if α and β are K-trivial, then the join $\alpha \oplus \beta$ is also K-trivial.

The main goal of this paper is to study K-triviality for closed subsets of $2^{\mathbb{N}}$ and for continuous functions on $2^{\mathbb{N}}$. We define a closed set Q to be K-trivial if the code x_Q is K-trivial and we define a continuous function $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ to be K-trivial if it has a representing function $f \in \mathcal{F}$ which is K-trivial.

2 K-Trivial Closed Sets

Since every K-trivial real is Δ^0_2, we have that every K-trivial closed set is a strong Π^0_2 class. Note also that the canonical code of a Π^0_1 class has c.e. degree and that there are K-trivial reals with non-c.e. degree. Hence there are K-trivial closed sets which are not Π^0_1 classes.

Analogous to the existence of c.e. K-trivial reals, we will construct several examples of K-trivial Π^0_1 classes. Note that a Π^0_1 class P is said to be *decidable* if the canonical tree T_P is computable, which is if and only if the canonical code for P is computable. Thus we want to construct K-trivial Π^0_1 classes which are not decidable. The degree of a closed set Q is the degree of the tree T_Q and also the degree of the canonical code for T_Q.

We begin with those non-decidable Π^0_1 classes with the simplest structure, that is, countable classes with a unique limit path. Our first construction relies on the following notion. If $A = \{a_0 < a_1 < \cdots\}$ is an infinite set, then A is said to be *retraceable* if there is a partial computable function ϕ such that $\phi(a_{n+1}) = a_n$ for all n. The initial subsets of A are A together with the finite sets $\{a_0, \ldots, a_{n-1}\}$ for each n. Dekker and Myhill [9] showed that every c.e. degree contains a retraceable Π^0_1 set A. Cenzer, Downey, Jockusch and Shore [6] showed that a Π^0_1 set A is retraceable if and only if the family $I(A)$ of initial subsets is a Π^0_1 class. Clearly $I(A)$ has unique limit element A.

Theorem 1. *For any noncomputable K-trivial c.e. degree* $\mathbf{d}$*, there exists a K-trivial Π^0_1 class P of degree* $\mathbf{d}$ *such that P has a unique, noncomputable limit element.*

Proof. Let A be a retraceable Π^0_1 set of degree $\mathbf{d}$. Then A is K-trivial and noncomputable and is the unique limit element of the Π^0_1 class $P = I(A)$ as shown above. It remains to show that the tree T_P has the same degree as A. Certainly $T_P \leq_T A$, since

$$\sigma \in T_P \iff (\forall i < |\sigma|)[\sigma(i) = 1 \rightarrow (i \in A \ \& \ (\forall j < i)(j \in A \rightarrow \sigma(j) = 1))].$$

On the other hand, $A \leq_T T_P$ since

$$a \in A \iff (\exists \sigma \in \{0,1\}^{a+1})(\sigma \in T_P \ \& \ \sigma(a) = 1). \qquad \square$$

We next construct a K-trivial class having only computable members.

Theorem 2. *For any K-trivial c.e. degree* $\mathbf{d}$*, there exists a K-trivial Π^0_1 class of degree* $\mathbf{d}$ *with unique limit path 0^ω and all elements computable.*

Proof. Let B be a co-c.e. set of degree $\mathbf{d}$ and let $Q = \{0^\omega\} \cup \{\{n\} : n \in B\}$. Clearly Q has all elements computable and unique limit element 0^ω. It is easy to check that $T_Q \equiv_T B$. $\square$

Next we wish to obtain a Π^0_1 class with no computable members (a *special* Π^0_1 class) such that the code for the class is K-trivial. To do so we rely heavily on the fact that K-triviality is closed under Turing equivalence. Note first that since the K-trivials form an ideal in the Turing degrees, the separating class for two K-trivial sets A, B will be K-trivial, as the set of its extendible nodes (and hence its code) is Turing-equivalent to $A \oplus B$. It remains to show there are recursively inseparable K-trivial sets. The following proof due to Steve Simpson.

Theorem 3. *There is a K-trivial Π^0_1 class with no computable members.*

Proof. Let B be a noncomputable c.e. K-trivial set. Split B into disjoint noncomputable c.e. A_1, A_2 as in the Friedberg splitting theorem. Ohashi [17] observed that the proof of the Friedberg splitting theorem in fact gives that A_1, A_2 are recursively inseparable. By the downward closure of K-triviality, they are also K-trivial. Let S be their separating class. Then by the discussion above, S is a special K-trivial Π^0_1 class. $\square$

Now a separating class always has measure zero. Next we construct K-trivial classes of arbitrarily large positive measure yet still containing no computable members. The proof makes use of the well-established *cost function* method from the area of algorithmic randomness, first used in Kucera-Terwijn [14] and later made explicit, e.g. in Downey-Hirschfeldt-Nies-Stephan [12].

Theorem 4. *There is a K-trivial Π^0_1 class (of arbitrarily large measure) with no computable paths (thus perfect).*

Proof. There is a well established framework for constructing K-trivial reals in the Cantor space 2^ω in terms of *cost functions*. A good presentation of this can be found in Nies [16]. It is clear that the same method applies to the space 3^ω. Let K be the prefix-free complexity and

$$cost(x,t) = \sum_{x<w\leq t} 2^{-K_t(w)}.$$

It is well known that $\lim_x \sup_t cost(x,t) = 0$. In order to construct a K-trivial Π^0_1 class P it suffices to give a monotone approximation (P_t) to P (in the sense that $P_t \supseteq P_{t+1}$) such that if c_t is the code for P_t and x_s is the least number such that $c_{s-1}(x) \neq c_s(x)$ then

$$\sum_{s>0} cost(x_s, s) \leq 1. \tag{2}$$

Indeed in [16] it is shown that c is K-trivial iff it has a Δ^0_2 approximation (c_t) which satisfies (2). To make sure that there are no computable paths through P it suffices to satisfy the following requirements:

$$R_e : \Phi_e \text{ is total } \Rightarrow \Phi_e \notin P$$

where (Φ_e) is an effective enumeration of all Turing functionals with binary values. The strategy for R_e is to modify the code c at some stage so that the tree represented by c no longer extends some initial segment of Φ_e. This is done by switching a 2 in c to a 0 or 1 according to which has the desired effect. First note that each c_t will consist of all 2's except for a finite initial segment, so we will find a suitable digit to switch. Second note that when we change a position in c from 2 to something else (0 or 1), we can effectively adjust the tail of c (the digits after the modified digit) so that the code describes the tree that we get if we cut that branch from the branching node corresponding to the 2 above. This means that if we let R_e act on c in the way described above, we get an approximation to P which is co-c.e. (so P is a Π^0_1 class).

The last consideration is that R_e cannot change digit n at stage s unless $cost(n,s) < 2^{-(e+1)}$. This will make c K-trivial. Let $\mathbb{N}^{[e]}$ be the e-th column of $\mathbb{N}$, i.e. the set of numbers of the form $\langle e,t\rangle$ for some $t \in \mathbb{N}$ where $\langle .,. \rangle$ is a computable bijection from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. The symbol $\restriction$ denotes restriction of the object that precedes it to the numbers $< x$. For example $\Phi_e \restriction x \downarrow$ means that Φ_e is defined at all arguments $< x$. All parameters in the construction are in formation and only have current values which correspond to the current stage.

Construction. At stage s look for the least $e < s$ such that R_e has not acted and there is a positive $x \in \mathbb{N}^{[e]}$ with the property that

- $\Phi_e \upharpoonright x \downarrow$ and is on P_s
- $cost(k(x,s),s) < 2^{-(e+1)}$, where $k(x,s)$ is the position of node $\Phi_e \upharpoonright (x-1)$ in the code c_s of P_s.

If there is no such e go to the next stage. Otherwise note that since R_e has not acted and $x \in \mathbb{N}^{[e]}$, no strategy has chopped any branch from node $\Phi_e \upharpoonright (x-1)$ and so the latter is branching. Now switch $k(x,s)$ from 2 to $1-\Phi_e(x-1)$ (so that $\Phi_e \upharpoonright x \notin P$) and let larger positions describe the tree that we get by chopping that branch. Go to the next stage.

For the verification, the comments before the description of R_e explain why the approximation (c_t) defined in the construction corresponds to a co-c.e. approximation of P, so that P is a Π^0_1 class. Each R_e is satisfied by the standard cost-function argument: there is some x_0 such that for all $x > x_0$ and all s, $cost(x,s) < 2^{-(e+1)}$ (by the properties of $cost$). Finally c is K-trivial since the approximation (c_t) given in the construction satisfies (2) (that each R_e acts at most once and contributes cost at most $2^{-(e+1)}$). Finally note that by choosing the witnesses x sufficiently large we can make sure that P has measure arbitrarily close to 1. □

Theorem 5. *If P is a K-trivial Π^0_1 class then the leftmost path is a K-trivial real.*

Proof. The leftmost path is computable from the (code of the) Π^0_1 class P and since K-triviality is downward closed under Turing reductions it must be K-trivial. □

By Nies' top low$_2$ theorem (see [11]), there is a low$_2$ c.e. degree above all K-trivial degrees. By Theorem 5, this means that the sets computed by it form a basis for the K-trivial Π^0_1 classes (while no incomplete c.e. degree has this property with respect to all Π^0_1 classes). The following theorem shows that such a c.e. degree cannot be low. Note however that there are low PA degrees, i.e. low degrees such that the sets computed by them form a basis for all Π^0_1 classes. The corresponding problem for K-trivial reals—whether there is a low degree bounding all K-trivials—is a major open problem.

Theorem 6. *If A is c.e. and low then there is a K-trivial Π^0_1 class which contains no A-computable paths. In other words, there is no c.e. low set A such that the sets computed by A form a basis for the K-trivial Π^0_1 classes.*

Proof. This is similar to the proof that for every c.e. low A there is a K-trivial B such that $B \not\leq_T A$ (in the same way that the proof of Theorem 4 is similar to the construction of a non-computable K-trivial set). If the reader is not familiar with that construction, (s)he might like to have a look at it [16]. We wish to follow the construction of Theorem 4 only now we need to satisfy the following more demanding requirements:

$$R_e : \Phi^A_e \text{ is total } \Rightarrow \ \Phi^A_e \notin P.$$

In general it is impossible to satisfy these requirements but if we know that A is low we can use the following trick (due to Robinson) to succeed. During the construction we will ask $\emptyset'$ a $\Sigma^0_1(A)$ question (for the sake of R_e). Note that since A is low, $\emptyset'$ can answer such questions. At each stage we will only have an approximation to $\emptyset'$ and so we will get a correct answer possibly after a finite number of false answers. Requirement R_e will use *witnesses* (in the sense of the proof of Theorem 4) from $\mathbb{N}^{[e]}$. We will ask the following:

Is there a stage s and a witness x such that
- $\Phi^A_e \upharpoonright x[s] \downarrow$ with correct A-use and $\Phi^A_e \upharpoonright x[s] \in P_s$
- $cost(k(x,s),s) < 2^{-(n_e+e+3)}$

where n_e is the number of times that some branch of P has been pruned (i.e. some digit of c has been changed) for the sake of R_e?

First notice that the above question refers to the partial computable sequences (P_s), $(n_e[s])$ which are defined during the very construction. By the recursion theorem we can ask such questions and approximate the right answers: given any partial computable sequence (P'_s) of Π^0_1 classes and uniformly partial computable sequences $(n'_e[s])$, we will effectively define a construction in which the questions refer to the given parameters. All of these constructions will define a sequence (P_s) of Π^0_1 classes which monotonically converges to a K-trivial Π^0_1 class P which however does not necessarily satisfy the other requirements; also each will define a uniformly partial computable sequence $(n_e[s])$. The (double) recursion theorem will give a construction in which the questions asked actually refer to (P_s) and $(n_e[s])$. Such a construction will succeed in satisfying all requirements. Let $g(e,s)$ be a computable function approximating the true answer to the questions above, when these are set to refer to the given parameters (P'_s), $(n'_e[s])$.

Construction. For stage s and each $e < s$ such that there is an *unused* $x \in \mathbb{N}^{[e]}$ satisfying $\Phi^A_e \upharpoonright x[s] \in P_s$ and $cost(k(x,s),s) < 2^{-(n_e[s]+e+3)}$ (where $n_e[s]$ is as above) do the following. Wait for a stage $t \geq s$ such that $g(e,t) = 1$ or the computation $\Phi^A_e \upharpoonright x[s]$ has been spoiled. In the first case switch $k(x,s)$ from 2 to $1 - \Phi_e(x-1)$ (so that $\Phi_e \upharpoonright x \notin P$) and let larger positions describe the tree that we get by chopping that branch (say that x has been *used*); proceed to stage $s+1$. In the latter case do nothing and test the next value of e. If the above has run over all $e < s$ and we are still at stage s, go to stage $s+1$.

For the verification, note that if x is unused at some stage, then currently all nodes of the xth level of P are branching. So each construction defines a (possibly finite) monotone sequence of clopen sets P_s (and so a Π^0_1 class P as a limit). Also, for every values of the input (P'_s), $(n'_e[s])$ the resulting class P is K-trivial as the condition (2) from the proof of Theorem 4 holds (at any stage at most one requirement acts and the cost of that action is small by construction). By the double recursion theorem there is a construction such that

$$n_e[s] = n'_e[s] \ \wedge \ P_s = P'_s$$

for all s, e; i.e. the input and output as (double) partial computable sequences are the same. This construction must be total (in the sense that it passes through all stages) since every search halts (for example if $\Phi_e^A \upharpoonright x[s] \in P_s$, $cost(k(x,s),s) < 2^{-(n_e[s]+e+3)}$ and the computation is true then $g(e)$ has to settle at 1 as it guesses correctly). Finally suppose that R_e is not satisfied. This means that the answer to the e-question is a negative one. So $g(e)$ would settle to 0 (since it approximates the correct answer to the e-question) and R_e would act finitely often. But then the cost requirement (in particular n_e) would remain constant and (by the properties of $cost$) for some large enough x, s the computation $\Phi_e^A[s] \upharpoonright x$ will be correct and $\Phi_e^A \upharpoonright x[s] \in P_s$, $cost(k(x,s),s) < 2^{-(n_e[s]+e+3)}$ which is a contradiction. □

3 *K*-Trivial Continuous Functions

In [4], the notion of randomness was extended to continuous functions on $2^{\mathbb{N}}$. Thus it will be natural to consider K-trivial continuous functions. It was shown in [4] that a random continuous function maps any computable real to a random real. It follows immediately from the closure under join of K-trivial degrees that a K-trivial continuous function maps any computable real to a K-trivial real. It was shown in [4] that the set of zeroes of a random continuous function is either empty or random. It follows by downward closure of the K-trivial degrees that the set of zeroes of a K-trivial continuous function is either empty or K-trivial.

We consider a continuous functions $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ always in terms of one its representing functions $f : 2^{<\mathbb{N}} \to 2^{<\mathbb{N}}$, or, equivalently, in terms of the code of one of its representing functions. Note that by slowing the convergence of the function on finite strings, we may code information into the code of the function. Hence the codes of a given function on Cantor space are always closed upwards in the Turing degrees, so the K-degree of a function should be the K-degree of the canonical code, that which converges as rapidly as possible. However, the canonical code of a function F may be computed from *any* code, so it follows from the downward closure of K-triviality that F is K-trivial if and only if the canonical code is K-trivial.

Theorem 7. *For any K-trivial degree* $\mathbf{d}$*, there is a continuous function $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ with canonical code of degree* $\mathbf{d}$*. Moreover, if* $\mathbf{d}$ *is c.e., F may be chosen to have left-c.e. canonical code.*

Proof. Let $A = \{a_1, a_2, \ldots\}$ be a set of degree $\mathbf{d}$. We define F monotonically increasing such that $F(0^\omega) = 0^\omega$ and $F(1^\omega) = \chi_A$, the characteristic function of A. We work via $f : 2^{<\mathbb{N}} \to 2^{<\mathbb{N}}$. To begin, let $f(0) = 0^{(a_1+1)}$ and $f(1) = 0^{a_1}1$. Now suppose we have defined $f(\sigma) = \tau$ for $|\sigma| = n-1$, and that $a_n - a_{n-1} = m$. Then let $f(\sigma 0) = \tau 0^m$ and $f(\sigma 1) = \tau 0^{m-1}1$. It is clear that $f \equiv_T A$, so f is of degree $\mathbf{d}$. Furthermore, if $\mathbf{d}$ is a c.e. degree and A is chosen c.e., the code given by f will be left-c.e., as shown by an analysis of the construction.

The code for f may be thought of as composed of blocks of length 2^n for $n \geq 1$, in order of increasing size, corresponding to different levels of the tree. At level n, if $n-1 \notin A$, the block will be all zeros. If $n-1 \in A$ and $|A \upharpoonright n| = k$, the block will consist of 2^k subblocks of 2^{n-k} bits each, beginning with a subblock of all zeros and alternating to end with a subblock of all 1s. Thus the structure of the n^{th} block is determined entirely by whether $n-1$ is in A, and if so, how many values $< n-1$ are also in A.

Given an enumeration of A as A_s, $s \in \omega$, we may define an approximation to the function F with corresponding canonical code C_s. We show that as s increases, a bit of C_s holding a one may only change to zero if a preceding bit changes from zero to one; this shows that C_s is an increasing approximation. As the enumeration A_s is computable by assumption, the canonical code of F is then left-c.e. Without loss of generality we consider a single level of the tree, n, and a single stage, s. If the corresponding block of C_{s-1} is all zeros, this level causes no trouble at stage s: either it remains all zeros or half of its zeros change to ones. If the n^{th} block of C_{s-1} is half zeros and half ones, then enumeration into A at stage s may cause the subblocks to multiply and rearrange. However, this only occurs when some $k < n-1$ enters A_s, causing the corresponding earlier level to change from all zeros to half zeros and half ones. □

4 Medvedev Degrees of *K*-Trivial Classes

The degrees of difficulty of K-trivial closed sets should be of interest. Simpson [18], Cenzer and Hinman [7] and others have developed the subject of the Medvedev (or strong) degrees of Π^0_1 classes. Here $P \leq_M Q$ means that there is a computable function mapping Q into P. The Medvedev degrees form a lattice where the meet operation is the disjoint union and the join is the product, $P \otimes Q = \{\alpha \oplus \beta \mid \alpha \in P \text{ and } \beta \in Q\}$. There is a least degree 0_M consisting of the classes with a computable member and a highest degree 1_M which can be viewed as a universal Π^0_1 class. A related structure are the *Muchnik (or weak) degrees*, where P is weakly reducible to Q if for every $\beta \in Q$ there exists $\alpha \in P$ such that $\alpha \leq_T \beta$.

One general problem is where the K-trivial Π^0_1 classes fit into the Medvedev (or Muchnik) degrees of the Π^0_1 classes. We have only a few results so far. Since the K-trivial reals form an ideal in the Turing degrees, it follows that the family of Π^0_1 classes which contain a K-trivial real form an ideal in the lattice of Medvedev degrees (and also in the lattice of Muchnik degrees). The following proposition says that the K-trivial Π^0_1 classes are closed under the meet and the join operation in the Medvedev degrees.

Proposition 1. *The K-trivial Π^0_1 classes are closed under disjoint unions and under products.*

Proof. The degree of the code of the disjoint union of two Π^0_1 classes is the join of the degrees of the codes of these Π^0_1 classes. The same holds for products and since K-triviality is invariant in the Turing degrees and closed under join (in the Turing degrees) the proposition follows. □

Note however that K-triviality (for Π^0_1 classes) is not closed under Medvedev equivalence. For example the least Medvedev degree contains Π^0_1 classes with computable leftmost path but with a canonical code which computes the halting problem. Hence we could call a Medvedev degree K-trivial if it contains a K-trivial class. Since there is no c.e. complete K-trivial real and any Medvedev complete Π^0_1 class is also c.e. complete, it follows that no K-trivial Π^0_1 class is Medvedev complete. A relevant question is whether there a top Medvedev degree among the K-trivials, or even a maximal one.

Theorem 8. *There is no maximal K-trivial Medvedev or Muchnik degree.*

Proof. Given any K-trivial Π^0_1 class Q it suffices to construct a K-trivial Π^0_1 class P which is not weakly reducible to Q. Indeed, in that case $P \otimes Q$ would be K-trivial strongly above Q and not weakly below Q. We argue as follows. By the Low Basis Theorem, Q contains a member α of low Turing degree. Now by Theorem 6, there is a K-trivial Π^0_1 class P with no path computed by α. This means that P is not weakly reducible to Q. □

The above proof also shows that there is no Π^0_1 class P which has low canonical code and is weakly above all K-trivial Π^0_1 classes.

References

1. Barmpalias, G., Brodhead, P., Cenzer, D., Remmel, J.B., Weber, R.: Algorithmic Randomness of Continuous Functions (in preparation)
2. Brodhead, P., Cenzer, D., Dashti, S.: Random closed sets. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 55–64. Springer, Heidelberg (2006)
3. Barmpalias, G., Brodhead, P., Cenzer, D., Dashti, S., Weber, R.: Algorithmic randomness of closed sets, J. Logic and Computation (to appear)
4. Brodhead, P., Cenzer, D., Remmel, J.B.: Random continuous functions (Information Berichte, FernUniversität (2006). In: Cenzer, D., Dillhage, R., Grubb, T., Weihrauch, K. (eds.) CCA 2006. Third International Conference on Computability and Complexity in Analysis. Electronic Notes in Computer Science, pp. 79–89. Springe, Heidelberg (2006)
5. Cenzer, D.: Π^0_1 Classes, ASL Lecture Notes in Logic (to appear)
6. Cenzer, D., Downey, R., Jockusch, C.G., Shore, R.: Countable thin Π^0_1 classes. Ann. Pure Appl. Logic 59, 79–139 (1993)
7. Cenzer, D., Hinman, P.G.: Density of the Medvedev lattice of Π^0_1 classes. Archive for Math. Logic 42, 583–600 (2003)
8. Cenzer, D., Remmel, J. B.: Π^0_1 classes, In: Ersov, Y., Goncharov, S., Marek, V., Nerode, A., Remmel, J. (eds.): Handbook of Recursive Mathematics, Vol. 2: Recursive Algebra, Analysis and Combinatorics, Elsevier Studies in Logic and the Foundations of Mathematics, Vol. 139 pp. 623–821 (1998)
9. Dekker, J., Myhill, J.: Retraceable sets. Canad. J. Math. 10, 357–373 (1985)
10. Downey, R.: Five Lectures on Algorithmic Randomness. In: Chong, C.T. (ed.) Computational Prospects of Infinity Proc. 2005 Singapore meeting (to appear 2005)
11. Downey, R., Hirschfeldt, D.: Algorithmic Randomness and Complexity, in preparation. Current draft available at http://www.mcs.vuw.ac.nz/~downey/.
12. Downey, R., Hirschfeldt, D., Nies, A., Stephan, F.: Trivial reals. In: Proc. 7th and 8th Asian Logic Conference, pp. 101–131. World Scientific Press, Singapore (2003)

13. Downey, R.: Some computability-theoretic aspects of reals and randomness. In: Cholak, P. (ed.) The Notre Dame Lectures ASL Lecture Notes in Logic (2005)
14. Kučera, A., Terwijn, S.: Lowness for the class of random sets. Journal of Symbolic Logic 64, 1396–1402 (1999)
15. Nies, A.: Lowness properties and randomness. Advances in Mathematics 197, 274–305 (2005)
16. Nies, A.: Computability and Randomness, in preparation. Current draft available at http://www.cs.auckland.ac.nz/~nies.
17. Ohashi, K.: A stronger form of a theorem of Friedberg. Notre Dame J. Formal Logic 5, 10–12 (1964)
18. Simpson, S.G.: Mass problems and randomness. Bull. Symbolic Logic 11, 1–27 (2005)

Pseudojump Operators and Π^0_1 Classes

Douglas Cenzer[1,⋆], Geoffrey LaForte[2], and Guohua Wu[3,⋆⋆]

[1] Department of Mathematics, University of Florida
P.O. Box 118105, Gainesville, Florida 32611, USA
cenzer@math.ufl.edu

[2] Department of Computer Science, University of West Florida
Pensacola, Florida 32514, USA
glaforte@coginst.uwf.edu

[3] School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore 639798
guohua@ntu.edu.sg

Abstract. For a pseudojump operator V^X and a Π^0_1 class P, we consider properties of the set $\{V^X : X \in P\}$. We show that there always exists $X \in P$ with $V^X \leq_T \mathbf{0}'$ and that if P is Medvedev complete, then there exists $X \in P$ with $V^X \equiv_T \mathbf{0}'$. We examine the consequences when V^X is Turing incomparable with V^Y for $X \neq Y$ in P and when $W^X_e = W^Y_e$ for all $X, Y \in P$. Finally, we give a characterization of the jump in terms of Π^0_1 classes.

Keywords: Computability, Π^0_1 Classes.

Pseudojump operators have been of great interest in computability theory and were explicitly introduced by Jockusch and Soare in [7]. If ϕ^X_e is the eth partial computable functional with oracle X, then $W^X_e = \{n : \phi^X_e(n)\downarrow\}$ and the eth pseudojump operator J_e maps X to $X \oplus W^X_e$. In particular, the jump operator $J(X) = X' = \{e : \phi^X_e(e)\downarrow\}$ is also a pseudojump operator. We will often denote a pseudojump operator by V and let V^X denote the pseudojump of X. Friedberg [3] constructed a noncomputable c.e. set A such that $A' \equiv_T \mathbf{0}'$. The fundamental theorem for pseudojumps, from [7], states that for any index e, there exists a noncomputable c.e. set A such that $J_e(A) \equiv_T \mathbf{0}'$. This generalizes the result of Friedberg that $A' \equiv_T \mathbf{0}'$ for some noncomputable c.e. set A. On the other hand, if V^X is obtained from the construction of a *low*^{X} set, then $(V^A)' = A'$, so that if $V^A \equiv_T \mathbf{0}'$, then $A' = \mathbf{0}''$. In each of these examples, $X <_T V^X$ for all X. We will say that a pseudojump operator V is *strongly nontrivial* if $X <_T V^X$ for all X. In the recent paper [2], it was shown that for any pseudojump operator V with $A <_T V^A$ for all c.e. sets A, there exist Turing incomparable c.e. sets A and B such that $V^A \equiv_T V^B \equiv_T \mathbf{0}'$.

⋆ Research was partially supported by National Science Foundation grants DMS 0532644 and 0554841. Corresponding author.

⋆⋆ Wu is partially supported by start-up grant No. M48110008 and research grant No. RG58/06 from NTU.

The study of pseudojump operators is a natural extension of the study of c.e. sets and degrees, which are fundamental in computability theory. Another natural extension is the study of effectively closed sets (Π^0_1 classes), which are sets of reals and play an important role in many areas of computable mathematics. The degrees of members of Π^0_1 classes is of great interest here. For example, every Π^0_1 class $Q \subseteq 2^{\mathbb{N}}$ has a member of c.e. degree, but there exist Π^0_1 classes with no computable member. A survey of results on Π^0_1 classes may be found in [1].

In this paper, we consider the interaction between pseudojump operators and Π^0_1 classses, in particular how pseudojump operators act on Π^0_1 classes. Recent work of Simpson [8] on the Medvedev degrees of Π^0_1 classes has characterized the complete degrees in several ways. The main result is that if V is a pseudojump operator and P is a Medvedev complete Π^0_1 class, then there exists $X \in P$ with $V^X \equiv_T \mathbf{0}'$. (It follows that there exist infinitely many such $X \in P$.)

We also give a new characterization of the jump in terms of Π^0_1 classes and consider for a Π^0_1 class Q, properties of the set $\{V^X : X \in Q\}$. That is, we examine the consequences of having $W^X_e = W^Y_e$ for all $X \in Q$ and of having W^X_e Turing incomparable with W^Y_e for all $X \neq Y$ in Q.

It is easy to find a nonempty Π^0_1 class P and a pseudojump operator V such that $V^X \not\equiv_T \mathbf{0}'$ for any $X \in P$. For example, if P contains only computable elements and V^X is low^X, then $X' \equiv \mathbf{0}'$ for all $X \in P$. Our intuition is that if P is complicated enough, then it should have a member with $V^X \equiv_T \mathbf{0}'$.

For Π^0_1 classes with no computable members, we still might not have a c. e. member or even a member of c.e. degree with $V^X \equiv_T \mathbf{0}'$. We can find examples of such *special* Π^0_1 classes with no members X of c. e. degree such that $V^X \equiv_T \mathbf{0}'$. Jockusch [4] constructed a Π^0_1 class P with no c. e. members at all. Jockusch and Soare [5] constructed a Π^0_1 class Q such that for any c. e. degree $\mathbf{b}$ and any $X \in P$, if $X \leq_T \mathbf{b}$, then $\mathbf{b} = 0'$. Thus if X has c. e. degree and $X \in Q$, then $X \equiv_T \mathbf{0}'$, so that if $V^X \leq_T \mathbf{0}'$, then $V^X \equiv_T X$, so that V fails to be strongly non-trivial. Recall that the Low Basis Theorem of Jockusch and Soare [6] shows that any nonempty Π^0_1 class $P \subseteq 2^{\mathbb{N}}$ must contain a member of low degree. The previous result implies that this member need not have c.e. degree.

Since $V^X \leq_T X'$ for any set X and any pseudojump operator V, the following is an immediate corollary of the low basis theorem. We sketch a proof in preparation for the main theorem. Let K denote the Halting Problem $\{e : \phi_e(e)\downarrow\}$.

Proposition 1. *For any pseudojump operator V and any nonempty Π^0_1 class P, there exists $X \in P$ with $V^X \leq_T K$.*

Proof. This is an easy modification of the Low Basis Theorem [6]. Let $P = [T]$ and fix e such that $V^X = W^X_e = \{m : \phi^X_e(m)\downarrow\}$. For each a, define the computable tree

$$U_a = \{\sigma \in \{0,1\}^* : \phi^\sigma_e(a)\uparrow\}.$$

Then $[U_a] = \{X : \phi^X_e(a)\uparrow\}$. Now define a sequence of Π^0_1 trees $\{S_n : n < \omega\}$ as follows. Let $S_0 = T$ and for each n, define

$$S_{n+1} = \begin{cases} S_n \cap U_n, & \text{if } S_n \cap U_n \text{ is infinite,} \\ S_n, & \text{otherwise.} \end{cases}$$

Now let $S = \cap_n S_n$ and $Q = [S] = \cap_n [S_n]$. By assumption, P is nonempty so that T is infinite and it follows from the construction, by induction, that each S_n is infinite. Thus Q is nonempty.

The construction is computable in K and therefore $\{n : S_n \cap U_n$ is infinite$\}$ is computable in K. Now for $X \in [S_{n+1}]$, it is clear that if $S_n \cap U_n$ is infinite, then $n \notin V^X$. On the other hand, if $S_n \cap U_n$ is finite, then $[S_n] \cap [U_n] = \emptyset$, so that for $X \in [S_n]$, $n \in V^X$. This gives a definition of V^X using K. Note that for any $X, Y \in Q$, we have $V^X = V^Y$. □

We now turn to the main result. Let $\mathcal{Q}$ be the computable Boolean algebra of clopen sets in $\{0,1\}^{\mathbb{N}}$. A clopen set is simply a finite union of intervals. A Π^0_1 class P is said to be *productive* if there is a computable *splitting* function $g : \mathbb{N} \to \mathcal{B}$ such that, for any e, if $P_e \cap P$ is nonempty, then both $P_e \cap P \cap g(e)$ and $P_e \cap P - g(e)$ are nonempty. Simpson showed that a Π^0_1 class is productive if and only if it is Medvedev complete. The Medvedev complete classes are the most *difficult* in the sense that if Q is Medvedev complete and P is any Π^0_1 class, then there exists a computable map Φ mapping Q into P.

Theorem 1. *Let V be a pseudojump operator V and let P be a Medvedev complete Π^0_1 class. Then there exists $X \in P$ with $V^X \equiv_T K$.*

Proof. Let $P = P_c = [T]$ be Medvedev complete and let g be a splitting function for P. We now give a modification of the proof of Proposition 1 above. The idea is that the Halting Problem K will be coded into V^X via a function $f : \mathbb{N} \to \mathcal{Q}$, computable in V^X, such that

$$X \in f(n) \iff n \in K.$$

Fix e such that $V^X = W^X_e$ and let U_a be defined as above. Now define the sequences $\{R_n : n < \omega\}$ and $\{Q_n : n < \omega\}$ of Π^0_1 classes as follows. Let $R_0 = P = P_c$ and let

$$R_n = \begin{cases} Q_n \cap [U_n], & \text{if } Q_n \cap [U_n] \text{ is nonempty,} \\ Q_n, & \text{otherwise.} \end{cases}$$

Let $R_n = P_{r(n)}$. By the construction, R_n is a nonempty subset of P, so that $R_n \cap g(r(n))$ and $R_n - g(r(n))$ are both nonempty subsets of P. Then define

$$Q_{n+1} = \begin{cases} R_n \cap g(r(n)), & \text{if } n \in K, \\ R_n - g(r(n)), & \text{otherwise.} \end{cases}$$

As before, let $Q = \cap_n Q_n$. It follows by induction that each tree each Q_n is nonempty and hence Q is nonempty. Once again, the construction is computable in K and it follows as in the proof of Proposition 1 that, for $X \in Q$, $V^X \leq_T K$ and that, for any $X \in Q$,

$$(*) \qquad V^X = \{n : Q_n \cap [U_n] \text{ is nonempty}\}.$$

On the other hand, suppose that $X \in Q$ and we use V^X as an oracle. Note that $X \leq_T V^X$ so that we can also use X in our computation from V^X. Then we can recursively compute the function $r(n)$ as follows. Informally, we can compute R_n using V^X and then we can compute Q_{n+1} using X.

More formally, we may define functions r and q, computable from V^X, so that $R_n = P_{r(n)}$ and $Q_n = P_{q(n)}$. That is, $Q_0 = P$, so $q(0) = c$. Given $q(n)$, we have

$$P_{r(n)} = \begin{cases} P_{q(n)} \cap [U_n], & \text{if } n \in V^X, \\ P_{q(n)}, & \text{otherwise.} \end{cases}$$

Then we have

$$P_{q(n+1)} = \begin{cases} P_{r(n)} \cap g(r(n)), & \text{if } X \in g(r(n)), \\ P_{r(n)} - g(r(n), & \text{otherwise.} \end{cases}$$

It follows that the functions $q(n)$ and $r(n)$ are computable from V^X. Finally $K \leq_T V^X$ since

$$n \in K \iff X \in g(r(n)).$$

Note that in fact $V^X \equiv_T K$ for all $X \in Q$.

To obtain infinitely many X with $V^X \equiv_T K$, note that for any σ such that $P \cap I(\sigma) \neq \emptyset$, $P \cap I(\sigma)$ is also Medvedev complete. This is because the splitting function for P is easily adapted to a splitting function for $P \cap I(\sigma)$. This means that for every σ such that $P \cap I(\sigma) \neq \emptyset$, there exists $X \in I(\sigma)$ with $V^X \equiv_T K$. Thus there are infinitely many such $X \in$P. □

Although the class Q constructed in the theorem is not a Π^0_1 class, it is a strong Π^0_2 class with the property that $\{V^X : X \in Q\}$ is a singleton and this unique V^X is $\leq_T K$. It seems natural to consider the question of a Π^0_1 class P where V^X is unique for $X \in P$. A classical result is that if $P = \{X\}$ itself is a singleton, then X is computable. By our definition, $V^X = V^Y$ implies that $X = Y$, so we consider just W^X_e.

Proposition 2. *Let P be a Π^0_1 class and suppose that $W^X_e = W^Y_e = W_P$ for all $X, Y \in P$.*

(a) The unique W^X_e for $X \in P$ is a c.e. set.
(b) If $X \leq_T W^X_e$ for all X, then $X \leq_T W_P$, so that P is countable and therefore has a computable member.
(c) Suppose that $X \leq_T W^X_e$ for all X and further that $W^R_e <_T K$ for any recursive R. Then $W_P <_T K$.

Proof. Fix a computable tree T such that $P = [T]$.

(a) Claim: $a \in V^X \iff (\exists n)[(\forall \sigma \in \{0,1\}^n \cap T \;\rightarrow\; a \in V^\sigma)$.

Suppose first that $a \in V^X$ for all $X \in P$. Then by compactness, there exists m such that $a \in V^{X \lceil m}$ for all $X \in P$. Let $S = \{\sigma \in \{0,1\}^m : P \cap I(\sigma) \neq \emptyset\} =$

$\{X\lceil m : X \in P\}$. For $\sigma \in \{0,1\}^m - S$, T contains only finitely many extensions of σ. Thus we can find $n > m$ such that $\tau\lceil m \in S$ for all $\tau \in \{0,1\}^n \cap T$. This n satisfies the formula above.

Next suppose that n exists as in the formula. Then for every $X \in P$, $a \in V^{X\lceil n}$ and therefore $a \in V^X$.

(b) There can be only countably many $X \leq_T W_P$, so it follows from (a) that P is countable and hence P has a computable member.

(c) Finally, let R be a computable member of P which exists by (b). Then for any $X \in P$, $V^X = V^R <_T K$. □

For the other extreme, suppose that V^X is Turing incomparable with V^Y for all $X \neq Y$ in P. It was also shown in [6] that there exist Π^0_1 classes containing continuum many elements, with each pair Turing incomparable. This will serve as an example with $V^X = X$.

Of course if $V^X = X'$, then any Π^0_1 class Q must contain X with $V^X = K$ and therefore if nontrivial, Q must contain distinct X, Y with $V^X \equiv_T K \equiv_T V^Y$.

Proposition 3. *Let W^X denote either W^X_e or $X \oplus W^X_e$ and suppose that P is an infinite Π^0_1 class such that W^X and W^Y are Turing incomparable for any $X, Y \in P$. Then there is no $X \in P$ such that $K \leq_T W^X$.*

Proof. Suppose by way of contradiction that $K \leq_T W^X$ for some $X \in P$. Since P is infinite, there is some $Y \in P$ with $Y \neq X$. Let n be the least such that $X(n) \neq Y(n)$ and let $Q = P \cap I(Y\lceil n+1)$. By Proposition 1, there exists $Z \in Q$ with $W^Z \leq_T K \leq_T V^X$. □

Finally, we observe that Π^0_1 classes may be used to define the jump and also pseudojumps.

Proposition 4. *For any set X, $\{e : X \in P_e\} \equiv_T X'$.*

Proof. Let $W^X = \{e : X \in P_e\}$. Then $W^X \leq_T X'$ since

$$e \in W^X \iff (\forall n) X\lceil n \notin W_e.$$

For the completeness, use the s-m-n theorem to define a computable function f such that

$$P_{f(e)} = \{X : \phi^X_e(e) \uparrow\}.$$

Then

$$e \in X' \iff f(e) \notin W^X$$

gives a reduction of X' to W^X. □

One can define a pseudojump using Π^0_1 classes as follows. Let $\pi_i(P)$ be the projection of P onto the ith coordinate, where $\pi_i(X) = Y$ means that $X = \langle X_1, X_2, \ldots\rangle$ and $Y = X_i$.

Then let

$$V_e^X = \{i : X \in \pi_i(P_e)\}.$$

It can be seen that $V_e^X \equiv_T X'$ when P_e is a particular Medvedev complete class, such that $\pi_i(P)$ runs over all Π^0_1 classes. It is an interesting question whether every pseudojump can be expressed in this form.

References

1. Cenzer, D., Remmel, J.B.: Π^0_1 classes in mathematics, In: Ershov, Y., Goncharov, S., Nerode, A., Remmel, J. (eds.) Handbook of Recursive Mathematics, Part Two, Elsevier Studies in Logic. vol. 139 pp. 623-821. (1998)
2. Coles, R., Downey, R., Jockusch, C., LaForte, G.: Completing pseudojump operators. Ann. Pure and Appl. Logic 136, 297–333 (2005)
3. Friedberg, R.M.: A criterion for completeness of degrees of unsolvability. J. Symbolic Logic 22, 159–160 (1957)
4. Jockusch, C.G.: Π^0_1 classes and boolean combinations of recursively enumerable sets. J. Symbolic Logic 39, 95–96 (1974)
5. Jockusch, C.G., Soare, R.: Degrees of members of Π^0_1 classes. Pacific J. Math 40, 605–616 (1972)
6. Jockusch, C., Soare, R.: Π^0_1 classes and degrees of theories. Trans. Amer. Math. Soc 173, 35–56 (1972)
7. Jockusch, C., Shore, R.: Pseudojump operators I: the r.e. case. Trans. Amer. Math. Soc 275, 599–609 (1983)
8. Simpson, S.: Mass problems and randomness. Bull. Symbolic Logic 11, 1–27 (2005)
9. Soare, R.: Recursively Enumerable Sets and Degrees. Springer, Heidelberg (1987)

Sofic Trace Subshift of a Cellular Automaton[⋆]

Julien Cervelle[1], Enrico Formenti[2,⋆⋆], and Pierre Guillon[1]

[1] Institut Gaspard Monge, Université de Marne la Vallée
77454 Marne la Vallée Cedex 2, France
{julien.cervelle,pierre.guillon}@univ-mlv.fr

[2] Laboratoire I3S, Université de Nice-Sophia Antipolis
2000 Route des Lucioles, 06903 Sophia Antipolis, France
enrico.formenti@unice.fr

Abstract. The trace subshift of a cellular automaton is the subshift of all possible columns that may appear in a space-time diagram. In this paper we study conditions for a sofic subshift to be the trace of a cellular automaton.

Keywords: discrete-time dynamical systems, cellular automata, symbolic dynamics, sofic systems, formal languages.

1 Introduction

Cellular automata are well-known formal models for complex systems. They are used in a huge variety of different scientific fields including mathematics, physics and computer science.

A *cellular automaton* (CA) consists in an infinite number of identical *cells* arranged on a regular lattice indexed by $\mathbb{Z}$. Each cell is a finite automaton which *state* takes value in a finite set A. All cells evolve synchronously according to their own state and those of their neighbors.

The study and classification of the evolutions of cellular automata is one of the standing open problems in the field [1,2,3,4,5,6]. Indeed, the simple definition of CA contrasts the wide variety of their evolutions. An interesting idea is to classify these behaviors according to some notion of complexity. Of course, the word "complexity" means different things to different researchers. For this reason, in literature one finds classifications according to topological entropy, measure theory, dimension theory, attractors, algorithmic complexity, etc.

In this paper we follow a formal languages approach. Each CA is associated with a language. The idea is that the more complex is the language, the more complex is the automaton.

The associated language is defined as follows (see Section 2 for more precise definitions). Each CA can be seen as a discrete dynamical system $(A^{\mathbb{Z}}, F)$, where

⋆ This work has been supported by the Interlink/MIUR project "Cellular Automata: Topological Properties, Chaos and Associated Formal Languages", by the ANR Blanc "Projet Sycomore".

⋆⋆ Corresponding author.

F is the global function. Let $\beta = \beta_1, \beta_2, \dots, \beta_k$ a finite partition of $A^{\mathbb{Z}}$. Then an orbit of initial condition x, $\mathcal{O}_F(x) = (x, F(x), \dots, F^n(x), \dots)$ can be associated with the infinite word w such that $\forall n \in \mathbb{N}, w_n = i$ if $F^n(x) \in \beta_i$. Then, w is the *β-trace* of F with initial condition x. The *β-trace set* Σ of F is the set of β-traces with all possible initial conditions. Remark that when $A^{\mathbb{Z}}$ is endowed with the Cantor topology (see Section 2), Σ is a closed shift-stable set i.e. a subshift. The language $\mathcal{L}(\Sigma)$ of factors occurring in configurations of Σ is the language associated with $(A^{\mathbb{Z}}, F)$.

In [7], Kůrka classified factor subshifts of CA according to their language complexity. He devised three classes: bounded periodic; regular but not bounded periodic; not regular. In this paper we address a somewhat complementary question, namely, given a subshift Σ of a certain language complexity we wonder if it can be the trace of a CA.

Motivations come both from classical symbolic dynamics but also from physics. Indeed, when observing natural phenomena due to physical constraints, one can keep trace only of a finite number of measurements. This set of measurements, usually, takes into account only a minor part of the parameters ruling the phenomenon under investigation. Hence, to some extent, what is observed is the “trace” of the phenomenon left on the “instruments” rather than the overall phenomenon.

It is a very important issue (Galilean principle) to find a formal model which can reproduce the observed trace. Restating this reasoning in our context: given a subshift Σ, one wonders which discrete dynamical system can produce it. In particular, one can ask if there exists a CA having Σ as a trace.

Giving a complete answer to this question seems very hard. In this paper we give some sufficient conditions for a regular language to be traceable. The proof is constructive. We believe that the construction of the CA is of some interest in its own.

Because of the lack of space some of the proofs are omitted. They can be found in [8].

2 Definitions

Let $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$. For $i, j \in \mathbb{N}$ with $i \leq j$, $[i, j]$ denotes the set of integers between i and j. For any function F from $A^{\mathbb{Z}}$ into itself, F^n denotes the n-fold composition of F with itself. A set $S \subset A^{\mathbb{Z}}$ is *F-stable* if $F(S) \subset S$.

Languages. Let A be a finite *alphabet* with at least two *letters*. A *word* is a finite sequence of letters $w = w_0 \dots w_{|w|-1} \in A^*$. Its *reverse* $\bar{w}$ is $w_{|w|-1} \dots w_0$ and its *rotation* $\gamma(w)$ is $w_1 \dots w_{|w|-1} w_0$. A *factor* of a word $w = w_0 \dots w_{|w|-1} \in A^*$ is a word $w_{[i,j]} = w_i \dots w_j$, for $0 \leq i \leq j < |w|$. We note $w_{[i,j]} \sqsubset w$. The empty word is denoted by ε. Given two languages $C, D \subset A^*$, CD denotes their concatenation, $C + D$ their union, $C^* = \bigcup_{n \in \mathbb{N}} C^n$, and $C^\omega = \left\{ z \in A^{\mathbb{N}} \middle| \forall j \in \mathbb{N}, \exists k \geq j, z_{[0,k-1]} \in C^* \right\}$. When no confusion is possible, given a word w, we also denote w the language $\{w\}$.

Configurations. A *configuration* is a biinfinite sequence of letters $x \in A^{\mathbb{Z}}$. The set $A^{\mathbb{Z}}$ of configurations is the *phase space*. The definition of *factor* can be naturally extended to configurations: for $x \in A^{\mathbb{Z}}$ and $i \leq j$, $x_{[i,j]} = x_i \dots x_j \sqsubset x$. If $u \in A^*$, then u^ω is the infinite word consisting in periodic repetitions of u, and ${}^\omega u^\omega$ is the configuration consisting in periodic repetitions of u.

Topology. We endow the phase space with the *Cantor topology*. A base for open sets is given by cylinders. For $j, k \in \mathbb{N}$ and a finite set W of words of length j, we will note $[W]_k$ the *cylinder* $\left\{ w \in A^{\mathbb{Z}} \middle| w_{[k,k+j-1]} \in W \right\}$. $[W]_k^C$ is the complement of the cylinder $[W]_k$.

Cellular automata. A (one-dimensional) *cellular automaton* is a parallel synchronous computation model consisting in cells distributed over a regular lattice $\mathbb{Z}$. Each cell has a state in the finite alphabet A, which evolves depending on the state of their neighbors according to a *local rule* $f : A^d \to A$, where $m \in \mathbb{Z}$ and $d \in \mathbb{N}^*$ are the *anchor* and the *diameter* of the CA, respectively. The *global function* of the CA is $F : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ such that $F(x)_i = f(x_{[i-m,i-m+d]})$ for every $x \in A^{\mathbb{Z}}$ and $i \in \mathbb{Z}$. The *space-time diagram* of initial configuration $x \in A^{\mathbb{Z}}$ is the sequence of the configurations of the orbit $(F^j(x))_{j \in \mathbb{N}}$. Usually they are graphically represented by a two-dimensional diagram like in Figure 1.

The *shift map* $\sigma : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is a particular CA global function defined by $\sigma(x)_i = x_{i+1}$ for every $x \in A^{\mathbb{Z}}$ and $i \in \mathbb{Z}$. According to the Hedlund theorem [9], the global functions of CA are exactly the continuous self-maps of $A^{\mathbb{Z}}$ commuting with the shift map.

Any local rule f of a CA can be extended naturally to an application on words $f(w) = (f(w_{[i,i+d-1]}))_{0 \leq i \leq |w|-d}$, for all $w \in A^* A^d$.

Dynamical systems. A *dynamical system* is a couple (X, F) where X is a set called the *phase space*, and $F : X \to X$ is a continuous self-map. (Y, F) is a *subsystem* of (X, F) if Y is a closed F-stable subset of X. A set Y is *F-stable* if $F(Y) \subset Y$.

Morphisms. A *morphism* of the dynamical system (X, F) into the dynamical system (Y, G) is a continuous map $\phi : X \to Y$ such that $\phi \circ F = G \circ \phi$. A *conjugacy* (resp. a *factorization*) is a bijective (resp. surjective) morphism; in that case we say that (Y, G) is conjugate to (resp. a factor of) (X, F) (we expect the reader not to confuse between the use of the word "factor" in the dynamical system context and the language theory one).

Subshifts. The *onesided shift*, also noted σ is the self-map of $A^{\mathbb{N}}$ such that $\sigma(z)_i = z_{i+1}$, for every $z \in A^{\mathbb{N}}$ and $i \in \mathbb{N}$. A *onesided subshift* (or simply a *subshift*) $\Sigma \subset A^{\mathbb{N}}$ is a σ-stable closed set of infinite words. The language of Σ is $\mathcal{L}(\Sigma) = \left\{ w \in A^* \middle| \exists z \in \Sigma, w \sqsubset z \right\}$, and characterizes Σ, since $\Sigma = \left\{ z \in A^{\mathbb{N}} \middle| \forall w \sqsubset z, w \in \mathcal{L}(\Sigma) \right\}$. The *alphabet* of the subshift Σ is the set $\left\{ a \in A \middle| \exists z \in \Sigma, az \in \Sigma \right\}$ i.e. the set of letters that appear in infinite words belonging to Σ. A subshift Σ is *transitive* if for every words $u, v \in \mathcal{L}(\Sigma)$, there exists $w \in A^*$ such that $uwv \in \mathcal{L}(\Sigma)$. A subshift can also be characterized by a language $\mathcal{F} \subset A^*$ of *forbidden words*, i.e. such that $\Sigma = \left\{ z \in A^{\mathbb{N}} \middle| \forall u \in \mathcal{F}, x \not\sqsubset z \right\}$. A subshift is of *finite type* (SFT for short) if it has a finite language of forbidden words. It is a k-SFT (for $k \in \mathbb{N}$) if it has a finite set of forbidden words of length k. A subshift Σ is *sofic* if $\mathcal{L}(\Sigma)$ is a regular language. The following characterization of sofic subshifts will be very useful in the sequel. For more about subshifts, see for instance [10].

Theorem 1 (Weiss [11]). *A subshift is sofic if and only if it is a factor of a SFT.*

Hedlund's theorem can be extended as follows.

Theorem 2 (Hedlund [9]). *A function ϕ is a morphism of a subshift (X, σ) on alphabet A into a subshift (Y, σ) on alphabet B if and only if there is a radius $r \in \mathbb{N}$ and a local rule $f : A^{r+1} \to B$ such that $\forall j \in \mathbb{N}, \forall x \in X, \phi(x)_j = f(x_j \dots x_{j+r})$ (we say ϕ is an r-block map).*

3 Traces

In this section we define the main notion introduced in the paper, namely, the trace of a CA and the traceability of a subshift. Moreover, we give a simple necessary condition for a subshift being traceable.

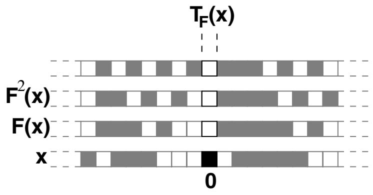


Fig. 1. The trace seen on the space-time diagram

Definition 1 (Trace). *Given a CA F, the* trace applications *are defined for $k \in \mathbb{Z}$ by $T_F^k(x) = (F^j(x))_{j \in \mathbb{N}}$. In other words, $T_F^k(x)$ is the k^{th} column of the space-time diagram of initial configuration x (see Figure 1). We note $T_F = T_F^0$. We say that $T_F(x)$ is the trace of F with initial condition x.*

The study of trace applications can be reduced to the study of T_F, because of shift-invariance of CA.

It can be noticed that this notion of trace corresponds to the β-trace as defined in the introduction, with $\beta = \{ [a] | a \in A \}$ being the partition of $A^{\mathbb{Z}}$ into cylinders of width 1.

Definition 2 (Traceability). *The* trace subshift *of a CA F is $\tau(F) = T_F(A^{\mathbb{Z}})$. It is a factor subshift of $(A^{\mathbb{Z}}, F)$, since T_F is continuous and commutes with σ. A subshift Σ is* traceable *if there exists a CA F for which $\Sigma = \tau(F)$.*

We begin with a condition for the traceability of a subshift. Proposition 2 proves that it is necessary.

Definition 3 (T0 subshift). *A subshift is* T0 *if it includes a* 2*-SFT with the same alphabet.*

Proposition 1. *A subshift is T0 if and only if there exists a map $\phi : A \to A$ such that for every letter $a \in A$, $(\phi^j(a))_{j\in\mathbb{N}} \in \Sigma$.*

Proof. If $\forall a \in A, (\phi^j(a))_{j\in\mathbb{N}} \in \Gamma$, then $\left\{ (\phi^j(a))_{j\in\mathbb{N}} \middle| a \in A \right\}$ is a 2-SFT of alphabet A and set of forbidden words $\bigcup_{a\in A} a(A \setminus \{\phi(a)\})$. Conversely, consider a 2-SFT Γ of alphabet A. Define $\phi(a) = b$ such that $ab \in \mathcal{L}(\Gamma)$. Since Γ is a 2-SFT, we have that $\forall a \in A, (\phi^j(a))_{j\in\mathbb{N}} \in \Gamma$. □

Example 1. Consider the following subshifts :

- $\Sigma = (1 + \varepsilon)(01)^\omega$; it is finite T0 (with $\phi(0) = 1$ and $\phi(1) = 0$);
- $\Sigma = 0^\omega + 0^*1^\omega$; it is infinite T0 (with $\phi(0) = \phi(1) = 1$);
- $\Sigma = (01 + 1 + \varepsilon)(001)^\omega$; it is finite but not T0.

Proposition 2. *The trace subshift of a CA is T0.*

Proof. Consider a CA F. For any $a \in A$, define $\phi(a)$ as $F({}^\omega a^\omega)_0$. Then $(\phi^j(a))_{j\in\mathbb{N}} = T_F({}^\omega a^\omega) \in \tau(F)$. By Proposition 1, $\tau(F)$ is T0. □

Theorem 3. *Any* 2*-SFT is traceable.*

Proof. Consider a 2-SFT Σ and let A be its alphabet. For each $a \in A$, define $\phi(a) = b$ such that $ab \in \mathcal{L}(\Sigma)$. Consider the CA of anchor 0, diameter 2 and local rule f defined by $f(x_0, x_1) = x_1$ if $x_0x_1 \in \mathcal{L}(\Sigma)$, and $\phi(x_0)$ otherwise. If $x \in A^{\mathbb{Z}}$, then the definition of the rule gives that every factor of length 2 of its trace is in $\mathcal{L}(\Sigma)$. Conversely, if $z \in \Sigma$, then we can see by induction that Σ is the trace of F with initial condition x as soon as $x_{[0,+\infty)} = z$. □

4 k-Traceability

In order to establish finer results we first need a weaker condition for traceability, namely k-traceability. A subshift of alphabet A is k-traceable if it is the set of columns of a CA on the alphabet A^k. The difference between a CA tracing a subshift of alphabet A and one tracing a subshift of alphabet A^k is that the rule of the latter can use the knowledge of the position of a letter of A in a word over A^k. This results in a much simpler construction.

Notation. If Σ is a subshift on an alphabet $B \subset A^k$, and $q \in [0, k-1]$, then the q^{th} projection is defined as

$$\pi_q : \begin{array}{c} B^{\mathbb{N}} \to A^{\mathbb{N}} \\ (z_j)_{j\in\mathbb{N}} \mapsto ((z_j)_q)_{j\in\mathbb{N}} \end{array} .$$

We also note $\pi(\Sigma) = \bigcup_{0\le q<k} \pi_q(\Sigma)$, which is a subshift on A.

In this section we will limit our study to onesided CA, i.e. with anchor 0.

Definition 4 (k-traceability). *Given a CA F on the alphabet $B \subset A^k$, the* k-trace subshift *is defined by* $\overset{\circ}{\tau}(F) = \bigcup_{0 \le q < k} \left\{ ((F^j(x)_0)_q)_{j\in\mathbb{N}} \middle| x \in B^{\mathbb{Z}} \right\} = \pi(\tau(F))$.

A subshift is k-traceable *if it is the k-trace of a onesided CA on the alphabet $B \subset A^k$.*

Similarly to what done in the previous section we give a necessary condition for being k-traceable.

Definition 5 (T1 subshift). *A subshift Σ on the alphabet A is* T1 *if there exists $k \in \mathbb{N}^*$ and a 2-SFT $\Gamma \subset (A^k)^{\mathbb{N}}$, such that $\pi_0(\Gamma) = \pi(\Gamma) = \Sigma$ (in particular, Σ is a factor of Γ).*

Theorem 4. *A T1 subshift is k-traceable for some $k \in \mathbb{N}^*$.*

Proof. By Theorem 3, the corresponding $\Gamma \subset (A^k)^{\mathbb{N}}$ is the trace of a CA F on some alphabet $B \subset A^k$. Hence, $\overset{\circ}{\tau}(F) = \pi(\tau(F)) = \pi(\Gamma) = \Sigma$. □

Example 2. Consider the subshift $\Sigma = (01 + 1 + \varepsilon)(001)^\omega$. It is T1 (define the 2-SFT $\Gamma = (uvw)^\omega$ on the alphabet $B = \{u, v, w\} \subset A^3$, where $u = 001$, $v = 010$ and $w = 100$). It is thus 3-traceable, but not traceable since it is not T0.

Theorem 5. *Any SFT is T1.*

Proof. Let Σ be a k-SFT for some $k \in \mathbb{N}^*$. $\Gamma = \left\{ (z_{[j,j+k-1]})_{j\in\mathbb{N}} \middle| z \in \Sigma \right\}$ is a 2-SFT, and $\pi(\Gamma) = \bigcup_{0\le q<k} \left\{ (z_{j+q})_{j\in\mathbb{N}} \middle| z \in \Sigma \right\} = \bigcup_{0\le q<k} \sigma^q(\Sigma) = \Sigma = \pi_0(\Gamma)$. □

This result allows us to prove the next proposition, which is a less restrictive condition for being T1.

Proposition 3. *A subshift Σ is T1 if and only if it is a factor of a SFT Γ on alphabet A^k for some $k \in \mathbb{N}^*$ such that $\pi(\Gamma) \subset \Sigma$.*

Now we extend the results on k-traceability to sofic subshifts (with some additional properties).

Definition 6 (T2 subshift). *A subshift is* T2 *if it is sofic and includes an infinite transitive subshift.*

Theorem 6. *Any T2 subshift is T1.*

The proof of Theorem 6 is given using the following lemmas.

Lemma 1. *A sofic transitive subshift on alphabet A is infinite if and only if for all $n \ge 2$, it includes a subshift B^ω with $B \subset A^k$, $|B| \ge n$ and some $k \in \mathbb{N}^*$.*

Lemma 2. *If Σ is a factor subshift of a SFT Γ on alphabet B such that $B^\omega \subset \Sigma$, then Σ is T1.*

Proof (of Theorem 6). Let Σ a T2 subshift. From Theorem 1, it is a factor of a SFT Γ. Thanks to Lemma 1, there is an arbitrarily large set of words B on A such that $B^\omega \subset \Sigma$, so we can assume without loss of generality that Γ is a subshift on such an alphabet $B \subset A^k$ for some $k \in \mathbb{N}^*$. From Lemma 2, we conclude that Σ is T1. □

5 From k-Trace to Trace

In the previous section, we gave a sufficient condition for a particular subshift Σ to be k-traced by a CA G on an alphabet $B \subset A^k$. In this section, we show how to simulate G with another CA, on alphabet A, in such way that its trace is Σ. This can be done if we add a further condition to our subshift.

Definition 7 (T3 subshift). *A subshift Σ is* T3 *if there is a map $\phi : A \to A$ such that for every letter $a \in A, (\phi^j(a))_{j\in\mathbb{N}} \in \Sigma$ (it is T0) and there is a word $w \in A^* \setminus \phi(A)^*$ such that $w^\omega \in \Sigma$.*

Example 3. Consider the following subshifts.

- $\Sigma = (1+\varepsilon)(01)^\omega$ is not T3.
- $\Sigma' = 0^\omega + 0^*1^\omega$ is T3 (with $\phi(0) = \phi(1) = 1$ and $w = 0$).

Theorem 7. *Any T3 k-traceable subshift (for some $k \in \mathbb{N}^*$) is traceable.*

This section presents a sketch of the proof of Theorem 7. Remark that it is well known that a CA on any alphabet can be simulated by a CA on any other alphabet (with at least two letters), provided that its diameter is wide enough. In particular, any CA on $B \subset A^k$ can be simulated by a CA on A. Each cell can see its neighborhood as words of A^k and evolve accordingly. The problem is that all cells must have the same local rule, so they have to find from the neighborhood which *column* of the A^k simulation they are representing. This is usually done using a special *border* word to delimit the words of A^k.

In this section, Σ denotes a T3 subshift on alphabet which is k-traceable by a onesided CA G. Let ϕ and w be as in Definition 7. Assume G has diameter 2 (the construction can easily be generalized) and local rule $g : B^2 \to B$.

In order to achieve the simulation, we first define border words to delimit A^k cells. We have two *execution modes*: a *simulation mode* will simulate properly the execution of the CA on alphabet B, and a *default mode* will be applied if the neighborhood contains invalid information. This adds some issues: default evolution must be in Σ; border evolution must also evolve according to Σ; and we have to ensure that when a mode is applied to a cell, the same mode keeps being applied there in the following generations, since a change of mode would produce an invalid trace. These problems will be solved in the three following subsections.

5.1 Borders

In order to make our simulations, we need to delimit computation zones. This is obtained by using some special words called borders and defined as follows:

$$\Upsilon = \left\{ a^{|w|} v \overline{v} a^{k+3|w|} \middle| a \in \phi(A), v \in \mathcal{O}_\gamma(w) \right\} \subset A^l,$$

where $\mathcal{O}_\gamma(w) = \{\gamma^q(w) | \, 0 \le q < |w| \}$, and $l = k + 6\,|w|$.

Borders have the property that they cannot have a too wide overlap.

Border evolution. The border words of Υ must have an evolution in Σ. The following rule (of diameter 1) respects that condition:

$$\Delta_\Upsilon : \begin{array}{c} \Upsilon \to \Upsilon \\ a^{|w|}v\overline{v}a^{k+3|w|} \mapsto \phi(a)^{|w|}\gamma(v)\overline{\gamma(v)}\phi(a)^{k+3|w|} \end{array} .$$

Macrocells. We will decompose our configurations into *macrocells.* A macrocell is the concatenation of a border word and a valid word of B. We can simulate the local rule g by a *macroevolution rule* (local rule on macrocells of $B\Upsilon \subset A^h$, where $h = k + l = 2k + 6\,|w|$, and of diameter 2):

$$\Delta : \begin{array}{c} (B\Upsilon)^2 \to B\Upsilon \\ (u,v) \mapsto g(u_{[0,k-1]}, v_{[0,k-1]})\Delta_\Upsilon(u_{[k,h-1]}) \end{array} .$$

5.2 Default Mode

Our CA will work as follows. Valid zones (with macrocells), which evolve according to the macroevolution rule Δ so that they remain valid zones. Invalid zones run a microdefault mode so that they remain invalid zones. Nevertheless, frontiers between the zones must not move. In the frontiers, a macrodefault mode is applied in order for a macrocell to have the opportunity to evolve without taking into account its neighbors; that way, each cell will keep the same execution mode.

Macrodefault mode. To do so, we extend the macroevolution to a function on ΘA^h, where $\Theta = B\Upsilon A^h \setminus \bigcup\limits_{0<i<h} A^i B\Upsilon A^{h-i}$ because it does not take into account overlapping macrocells. This is crucial in order to define a local rule. If the central macrocell has a neighbor macrocell in Θ, we apply a simulation step of the CA. Otherwise, we evolve as a macrodefault mode (simulation from a monochromatic configuration):

$$\Delta : \begin{array}{rl} \Theta A^h \to & B\Upsilon \\ u \mapsto & \left| \begin{array}{l} g(u_{[0,k-1]}, u_{[h,h+k-1]})\Delta_\Upsilon(u_{[k,h-1]}) \text{ if } u \in B\Upsilon\Theta \\ g(u_{[0,k-1]}, u_{[0,k-1]})\Delta_\Upsilon(u_{[k,h-1]}) \text{ otherwise} \end{array} \right. \end{array} .$$

Microdefault mode. The function ϕ (corresponding to the fact Σ is T0) allows to define a microdefault mode for a neighborhood that does not contain any macrocell. We are now able to transform the function Δ into a local rule on A. Indeed, we can define, for anchor $m = h - 1$ and diameter $d = 3h - 1$:

$$f : \begin{array}{rl} A^d \to & A \\ w \mapsto & \left| \begin{array}{ll} \Delta(u)_i & \text{if } w \in A^{m-i}uA^i, \text{ where } u \in \Theta A^h, i \in [0, h-1] \\ \phi(w_0) & \text{otherwise} \end{array} \right. \end{array}$$

since such an integer i, and such a word u would be unique (from the construction of Θ). This local rule is such that $f(A^m u A^m) = \Delta(u)$ for every $u \in \Theta A^h$, which is what we wanted: it can simulate in one step the behavior of our CA on B. Let F be the corresponding global rule.

The following lemma guarantees that no column changes its evolution mode.

Lemma 3. *The preimage of cylinder* $[B\Upsilon]$ *is cylinder* $[\Theta]$*. Moreover, cylinder* $[\Theta]$ *and its complementary* $[\Theta]^C$ *are* F*-stable (in particular, we cannot create a border).*

A configuration which is a valid encoding of some $y \in B^{\mathbb{Z}}$ (simulation mode), then its trace is some projection of the trace of y. Otherwise, microdefault and microdefault mode also produce a trace which is in Σ. This concludes the proof of Theorem 7.

6 Examples

Example 4 (Finite untraceable T1 subshift). No CA traces subshift $\Sigma = \{0^\omega, (01)^\omega, (10)^\omega\}$, even though it is T1.

Example 5 (T1, T3, non-SFT, non-T2 subshift). The subshift $\Sigma = (0^*1 + 1^*)0^\omega$ is neither a SFT nor T2, but it is T1 and T3. Hence, by Theorems 4 and 7 it is traceable.

Example 6 (Traceable non-T3 subshift). Let f be the local rule of anchor 3 and diameter 7 such that $f(u_{-3}000111) = 1$, $f(000111u_3) = 0$, $f(u_{-3}001011) = 0$, $f(001011u_3) = 1$, and $f(u) = u_0$ otherwise. The trace subshift of the corresponding CA is $\tau(F) = \{0^\omega, (10)^\omega, (01)^\omega, 1^\omega\}$. In this case, $\tau(F)$ is finite but not T3.

Example 7 (Traceable non-sofic subshift). Let F be the CA on alphabet $A = \{b, r, l, w\}$ (the white, the right, the left and the wall particles, respectively) defined by the following local rule f of anchor 1 and diameter 3:

$x_{-1}x_0x_1$	$rl?$	$?rl$	$r?l$	$?w?$	$?rw$	$wl?$	$r??$	$??l$	$???$
$f(x_{-1}x_0x_1)$	w	w	w	w	l	r	r	l	b

where ? stands for any letter in A and the first applicable rule is used (left to right). Then, $\tau(F)$ is not sofic.

7 Putting Things Together

In this paper, we have given sufficient conditions for a subshift to be the trace of a CA. The following summarizes all these results:

Theorem 8. *Any T3+T1, T3 SFT, or T3+T2 subshift is traceable.*

The present result follows other works on the structure that the trace of a CA can have. Here, we take the problem the other way around: we construct a CA that traces a particular kind of subshifts. Though we do not have a necessary and sufficient condition for traceability, Conditions T0, T1, T2 and T3 are a first step toward a better understanding of what makes a sofic subshift traceable or not.

Moreover, we expect our construction to be generalizable to weaker conditions. Nevertheless, the non-sofic case is still obscure. The general feeling is that it needs a completely different approach.

In [7], it is proved that every factor subshift of a CA F is a factor of some β-trace, where β is a partition of $A^{\mathbb{Z}}$, and every β-trace is a factor of some *column factor* (i.e. a subshift $\left\{ (F^j(x)_{[0,q-1]})_{j\in\mathbb{N}} \middle| x \in A^{\mathbb{Z}} \right\}$, for some $q \in \mathbb{N}$). Hence we can wonder now whether this kind of result can be generalized, in particular to the *canonical factor* $\left\{ (F^j(x)_{[0,d-1]})_{j\in\mathbb{N}} \middle| x \in A^{\mathbb{Z}} \right\}$ of the CA.

References

1. Gilman, R.H.: Classes of linear automata. Erg. Th. & Dyn. Sys. 7, 105–118 (1988)
2. Hurley, M.: Attractors in cellular automata. Erg. Th. & Dyn. Sys. 10, 131–140 (1990)
3. Dubacq, J.C., Durand, B., Formenti, E.: Kolmogorov complexity and cellular automata classification. Th. Comp. Sci. 259(1–2), 271–285 (2001)
4. Durand, B., Formenti, E., Varouchas, G.: On undecidability of equicontinuity classification for cellular automata. In: Morvan, M., Rémila, E., (eds.) DMCS'03. Volume AB of DMTCS Proc. Disc. Math. and Th. Comp. Sci. pp. 117–128 (2003)
5. Culik, K., Yu, S.: Undecidability of cellular automata classification schemes. Comp. Sys. 2, 177–190 (1988)
6. Braga, G., Cattaneo, G., Flocchini, P., Vogliotti, C.Q.: Pattern growth in elementary cellular automata. Th. Comp. Sci. 145(1–2), 1–26 (1995)
7. Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Erg. Th. & Dyn. Sys. 17, 417–433 (1997)
8. Formenti, E., Cervelle, J., Guillon, P.: Sofic trace of a cellular automaton. Technical report, Institut Gaspard Monge (march 2007), http://hal.archives-ouvertes.fr/hal-00135811
9. Hedlund, G.A.: Endomorphism and automorphism of the shift dynamical system. Math. Sys. Theory 3, 320–375 (1969)
10. Marcus, B., Lind, D.: An introduction to symbolic dynamics and coding. Cambridge University Press, Cambridge (1995)
11. Weiss, B.: Subshifts of finite type and sofic systems. Monatshefte für Mathematik 77(5), 462–474 (1973)

Thin Maximal Antichains in the Turing Degrees[⋆]

Chi Tat Chong[1] and Liang Yu[2]

[1] Department of Mathematics, Faculty of Science, National University of Singapore, Lower Kent Ridge Road, Singapore 117543
chongct@math.nus.eud.sg

[2] Institute of Mathematical Sciences, Nanjing University, Nanjing, Jiangsu Province 210093, P.R. of China
yuliang.nju@gmail.com

Abstract. We study existence problems of maximal antichains in the Turing degrees. In particular, we give a characterization of the existence of thin Π^1_1 maximal antichains in the Turing degrees in terms of (relatively) constructible reals. A corollary of our main result gives a negative solution to a question of Jockusch under the assumption that every real is constructible.

1 Introduction

Let $\langle \mathfrak{D}, \leq \rangle$ denote the structure of the Turing degrees. If $\mathbf{A} \subset \mathfrak{D}$, then it is an *antichain* if $\mathbf{x} \not\leq \mathbf{y}$ and $\mathbf{y} \not\leq \mathbf{x}$ for any distinct $\mathbf{x}$, $\mathbf{y} \in \mathbf{A}$. $\mathbf{A}$ is *maximal* if it is not properly contained in an antichain. By contrast, $\mathbf{A}$ is a *chain* if all of its elements are pairwise Turing comparable. $\mathbf{A}$ is a *maximal chain* if it is not properly contained in any chain. In [3] we studied the existence problem of maximal chains in $\mathfrak{D}$ under various set-theoretic assumptions. In this paper we turn our attention to existence problems of maximal antichains in $\mathfrak{D}$. Since there are $2^{\aleph_0}$ many minimal degrees, we have immediately the following proposition.

Proposition 1 (Folklore). *(ZFC) Every maximal antichain has size* $2^{\aleph_0}$.

In parallel with Turing degrees, we say that $A \subset 2^\omega$ is an antichain if its elements are pairwise Turing incomparable. We define the related notions similarly. Our interest here are twofold: (i) In view of Proposition 1.1, is there an analytically definable (say $\mathbf{\Pi^1_1}$) maximal antichain? (ii) Does every maximal antichain $A \subset 2^\omega$ contain a perfect subset? Theorem 2.5 (ii) says that under ZFC, the existence of a thin $\mathbf{\Pi^1_1}$ maximal antichain of Turing degrees is equivalent to the assertion that $2^\omega = (2^\omega)^{L[x]}$ for some real x. Comparing the consistency strength of the existence of a thin $\mathbf{\Pi^1_1}$ maximal antichain in the Turing degrees with that of a $\mathbf{\Pi^1_1}$ maximal chain, where a large cardinal axiom is needed for it to be refuted (see [3])), one sees that the former is a much weaker statement.

[⋆] The research of the authors was respectively supported in part by NUS grant WBS 146-000-054-123, and NSF of China No. 10471060 and No. 10420130638.

In §3, we apply the results of §2 to study a measure-theoretic problem on the Turing degrees, and provide a negative answer to a question raised by Jockusch.

The following notations are adopted: x, y, z etc. denote elements of 2^ω, while the collection of paths of a perfect tree T is denoted by $[T]$.

2 Thin Maximal Antichains

Firstly, it is a consequence of ZFC that there does exist a thin maximal antichain in the Turing degrees:

Proposition 2. *(ZFC)There exists a thin maximal antichain.*

Proof. Fix an enumeration $\{[T_\alpha]\}_{\alpha<2^{\aleph_0}}$ of perfect sets whose Turing degrees form an antichain, and fix an enumeration of all reals $\{x_\alpha\}_{\alpha<2^{\aleph_0}}$. We construct a thin set $A = \{z^0_\alpha | \alpha < 2^{\aleph_0}\} \cup \{z^1_\alpha | \alpha < 2^{\aleph_0}\}$ whose Turing degrees form a maximal antichain, by induction on $\alpha < 2^{\aleph_0}$, so that both $A - [T_\alpha]$ and $[T_\alpha] - A$ are nonempty.:

At step α, check whether $\{x_\alpha\} \cup \{z^i_\beta | \beta < \alpha \wedge i \leq 1\}$ is an antichain. If the answer is yes, then check whether the Turing degrees of $\{x_\alpha\} \cup \{z^i_\beta | \beta < \alpha \wedge i \leq 1\} \cup [T_\alpha]$ form an antichain. There are two cases to consider:

(i) If they form an antichain, select a real $y \equiv_T x_\alpha$ but $y \neq x_\alpha$. Obviously $y \notin T_\alpha$. Define $z^0_\alpha = y$. Then select another real $y_0 \notin [T_\alpha]$ so that there is a real $y_1 \in [T_\alpha]$ with $y_0 \equiv_T y_1$. Define $z^1_\alpha = y_0$.
(ii) Otherwise, define $z^0_\alpha = z^1_\alpha = x_\alpha$.

If the Turing degrees of $\{x_\alpha\} \cup \{z^i_\beta | \beta < \alpha \wedge i \leq 1\}$ do not form an antichain, check whether $\{z^i_\beta | \beta < \alpha \wedge i \leq 1\} \cup [T_\alpha]$ is an antichain.

(iii) If the answer is yes, select a real $x \in [T_\alpha] - \{z^i_\beta | \beta < \alpha \wedge i \leq 1\}$. Then select a real $y \equiv_T x_\alpha$ but $y \neq x_\alpha$. Define $z^0_\alpha = z^1_\alpha = y$.
(iv) Otherwise, define $z^0_\alpha = z^1_\alpha$ to be any real forming an antichain with $\{z^i_\beta | \beta < \alpha \wedge i \leq 1\}$.

The set $A = \{z^i_\alpha | \alpha < 2^{\aleph_0} \wedge i \leq 1\}$ is an antichain by construction. We claim that it is maximal. Otherwise, there is a real x_α whose Turing degree is incomparable with those of all the reals in A. Let α_0 be the least ordinal α for which x_α has this property. Then according to (i) and (ii) at step α_0, either x_{α_0} or some real y of the same degree is chosen to be $z^i_{\alpha_0}$ for some (or all) $i \leq 1$, which is a contradiction. Furthermore, for each α, both $A - [T_\alpha]$ and $[T_\alpha] - A$ are nonempty since $A \cup [T_\alpha]$ is not an antichain. Thus A is a maximal antichain that is thin.

How complicated must a thin maximal antichain be? Since every maximal antichain of reals has size $2^{\aleph_0}$, it cannot be $\mathbf{\Sigma}^1_1$ (else it would contain a perfect subset). We show it is consistent with ZF that there exists a Π^1_1 thin maximal antichain. The idea of the proof is similar to that used in constructing a Π^1_1 maximal chain presented in [3]. But the technique required to derive the result is quite different.

Lemma 1. *(ZF) Let $X \cup \{x_0\}$ be a countable antichain in the Turing degrees. Let x_1 be a real. Then there is a z such that*

1. *$z'' \geq_{\mathrm{T}} x_1$;*
2. *$\{z\} \cup X$ is an antichain;*
3. *$z \geq_{\mathrm{T}} x_0$.*

Proof. Let $X = \{y_i\}_{i \in \omega}$. We construct a real z so that the following requirements are satisfied:

$$N_{e,i} : \Phi_e^{z \oplus x_0} \text{ is total } \implies \Phi_e^{z \oplus x_0} \neq y_i.$$

Then $\{z \oplus x_0\} \cup X$ is an antichain. We also need to make $(z \oplus x_0)'' \geq_{\mathrm{T}} x_1$. We construct a sequence of finite strings $\sigma_0 \prec \sigma_1 \prec ...$ so that $z = \bigcup_n \sigma_n$.

Construction:
At step 0, define $\sigma_0 = \emptyset$.
At step $n+1 = \langle e, i \rangle$.

Substep 1: (Satisfying $N_{e,i}$). Consider the following statement:

$$(\exists \tau \succeq \sigma_n)(\forall \tau_0 \succeq \tau)(\forall \tau_1 \succeq \tau)(\forall m)(\Phi^{\tau_0 \oplus x_0}(m) \downarrow \wedge \Phi^{\tau_1 \oplus x_0}(m) \downarrow \implies \Phi^{\tau_0 \oplus x_0}(m) = \Phi^{\tau_1 \oplus x_0}(m)).$$

If the statement is true, then find the least τ (in a recursive well ordering of strings) and define $\sigma_{n+1}^0 = \tau$. Then for every real $z \succ \sigma_{n+1}^0$, $\Phi_e^{z \oplus x_0}$ is total implies $\Phi_e^{z \oplus x_0} \leq_T x_0$. Thus $\Phi_e^{z \oplus x_0} \neq y_i$ since $X \cup \{x_0\}$ is an antichain. If the statement is not true, find the least $\tau_0 \succeq \sigma_n$ for which there exists (a least) $\tau_1 \succeq \sigma_n$ such that $\Phi^{\tau_0 \oplus x_0}(m) \downarrow \neq \Phi^{\tau_1 \oplus x_0}(m) \downarrow$ for some m. Define $\sigma_{n+1}^0 = \tau_k$ for the $k < 2$ where $\Phi^{\tau_k \oplus x_0}(m) \neq y_i(m)$.

Substep 2: (Coding x_1). Define $\sigma_{n+1} = (\sigma_{n+1}^0)^\frown(x_1(n))$.

Finally, define $z = \bigcup_n \sigma_n$. This finishes the construction.

Since $x_0 \not\leq_{\mathrm{T}} y_i$ for all i, $z \oplus x_0 \not\leq_{\mathrm{T}} y_i$ for all i. By the construction above, $z \oplus x_0 \not\geq_{\mathrm{T}} y_i$ for all i, so $X \cup \{z \oplus x_0\}$ is an antichain.

To see that $(z \oplus x_0)'' \geq_{\mathrm{T}} x_1$, we look at the statement considered in Substep 1. The statement is decidable by x_0''. If the statement is true, then we can x_0''-recursively find the τ. Then $\tau = \sigma_{n+1}^0$. Otherwise, we can x_0''-recursively find both τ_0 and τ_1. Then we use z to decide which one is the σ_{n+1}^0. Thus $x_1(n) = 0$ if and only if $z(|\sigma_{n+1}^0| + 1) = 0$. Moreover, $\sigma_{n+1} = z \upharpoonright (|\sigma_{n+1}^0| + 1)$. So the sequence $\{\sigma_n\}_n$ can be computed from $z \oplus x_0''$. Hence $(z \oplus x_0)'' \geq_{\mathrm{T}} z \oplus x_0'' \geq_{\mathrm{T}} x_1$.

Corollary 1. *(ZF+DC) Let $X \cup \{x_0\}$ be a countable antichain in the Turing degrees. Then there is a real x_1 so that for all real $y \geq_{\mathrm{T}} x_1$ there is a real z such that*

1. *$z'' \equiv_{\mathrm{T}} y$;*
2. *$\{z\} \cup X$ is an antichain;*
3. *$z \geq_{\mathrm{T}} x_0$.*

Proof. Fix an enumeration $\{y_i\}_{i\in\omega}$ of X. Then the set

$$B = \{y | (\exists z)(z'' = y \wedge X \cup \{z\} \text{ is an antichain} \wedge z \geq_{\mathrm{T}} x_0)\}$$

is a Borel set. Moreover, by Lemma 1, for each real x, there is a real $y \in B$ so that $y \geq_{\mathrm{T}} x$. By Borel determinacy [8], there exists a real x_1 so that for all $x \geq_{\mathrm{T}} x_1$, there is a real $y \in B$ so that $y \equiv_{\mathrm{T}} x$.

The proof of the following theorem depends heavily on the results of Boolos and Putnam [2]. Call a set $E \subseteq \omega \times \omega$ an arithmetical copy of a structure $(S, \in)$ if there is a 1-1 function $f : S \to \omega$ so that for all $x, y \in S$, $x \in y$ if and only if $(f(x), f(y)) \in E$. In ([2]) it is proved that if $(L_{\alpha+1} \setminus L_\alpha) \cap 2^\omega \neq \emptyset$ then there is an arithmetical copy $E_\alpha \in L_{\alpha+1}$ of $(L_\alpha, \in)$ so that any $x \in (L_{\alpha+1} \setminus L_\alpha) \cap 2^\omega$ is arithmetical in E_α (i.e. E_α is a master code for α in the sense of Jensen [6]). Moreover, each $z \in L_\alpha \cap 2^\omega$ is one-one reducible to E_α. Hence E_α may be viewed as a real. Note that for each constructibly countable β, there is an $\alpha > \beta$ such that $(L_{\alpha+1} \setminus L_\alpha) \cap 2^\omega \neq \emptyset$. For a given ordinal α and $X \subseteq \alpha \times \omega$, we denote by $X[\beta]$ the real $\{n \in \omega | (\beta, n) \in X\}$. We may regard X as a sequence of reals of length α.

Lemma 2. *Assume $V = L$. There exists a Π^1_1 thin maximal antichain in the Turing degrees.*

Proof. A set A of reals is Π^1_1 if and only if there is a Σ_0-formula φ such that

$$y \in A \Leftrightarrow (\exists x \in L_{\omega_1^y}[y])(L_{\omega_1^y}[y] \models \varphi(x, y)),$$

where ω_1^y is the least ordinal $\alpha > \omega$ such that $L_\alpha[y]$ is admissible (see [1] and [10]).

Our proof combines Corollary 1 and the argument in [3] which is based on [5].

Assuming $V = L$, we define a function F on $\omega_1 \times \bigcup_{\alpha<\omega_1} \mathcal{P}(\alpha \times \omega)$ as follows:

For each $\alpha < \omega_1$ and antichain $X \subseteq \alpha \times \omega$ with $\alpha < \omega_1$, we define $F(\alpha, X)$ to be the real z such that there exists a lexicographically least triple (β, E, e_0) (where the ordering on the second coordinate is $<_L$) satisfying the following properties:

1. There is a 1-1 function $h \in L_\beta$ which maps ω onto α, a real $x_0 \in L_\beta$ so that $\{x_0\} \cup \{X[h(n)] | n \in \omega\}$ is an antichain and $(L_{\beta+1} \setminus L_\beta) \cap 2^\omega \neq \emptyset$;
2. $E \in L_{\beta+1}$ is an arithmetical copy of $(L_\beta, \in)$ as described above,
3. $z \geq_{\mathrm{T}} x_0$ and $\{z\} \cup \{X[h(n)] | n \in \omega\}$ is an antichain. Furthermore,
4. $z'' \equiv_{\mathrm{T}} E$ and
5. $z = \Phi^E_{e_0}$.

We show that $F(\alpha, X)$ is defined if X is an antichain.

Fix (α, X) where X is an antichain. Since $V = L$, there is a $\gamma > \alpha$ such that there is a real $x_0 \in L_\gamma$ with $\{x_0\} \cup X$ forming an antichain. Choose a real x_1 for $X \cup \{x_0\}$ as guaranteed by Corollary 1. Since $V = L$, there is a $\beta > \gamma$ so that $x_1 \in L_\beta$, $(L_{\beta+1} \setminus L_\beta) \cap 2^\omega \neq \emptyset$ and there is a function h_α mapping ω onto

α. By the discursion above, there is an arithmetical copy $E \subseteq \omega \times \omega$ in $L_{\beta+1}$ so that $E \geq_T x_1$. By Corollary 1, there is a real $z \geq_{\mathrm{T}} x_0$ so that $z'' \equiv_{\mathrm{T}} E$ and $\{z\} \cup \{X[h_\alpha(n)] | n \in \omega\}$ is an antichain. Obviously, $L_{\beta+1} \in L_{\omega_1^z}[z]$. By the absoluteness of $<_L$, it is easy to see that F is a well-defined function.

Moreover, one can verify using the absoluteness of $<_L$ that there is a Σ_0 formula $\varphi(\alpha, X, z, y)$ such that $F(\alpha, X) = z$ if and only if $L_{\omega_1^{(X,z)}}[X, z] \models (\exists y)\varphi(\alpha, X, z, y)$, with a function $h \in L_{\omega_1^{(X,z)}}[X, z]$ mapping ω onto α.

Thus we can perform transfinite induction on α to construct a maximal antichain of Turing degrees. But care has to be exercised here since in general sets constructed this way are Σ_1 over L_{ω_1}, i.e. Σ^1_2 and not necessarily Π^1_1.

Define $G(\alpha) = z$ if and only if $\alpha < \omega_1^z$ and there is a function $f : \alpha + 1 \to 2^\omega$ with $f \in L_{\omega_1^z}[z]$ so that for all $\beta \leq \alpha$, $f(\beta) = F(\beta, \{(\gamma, n) | n \in f(\gamma) \wedge \gamma < \beta\})$ and $f(\alpha) = z$. Since $L_{\omega_1^z}[z]$ is admissible, $\{f(\gamma) | \gamma \leq \alpha\} \in L_{\omega_1^z}[z]$. So $G(\alpha) = z$ if and only if there is a function $f : \alpha + 1 \to 2^\omega$ with $f \in L_{\omega_1^z}[z]$ such that

$$L_{\omega_1^z}[z] \models ((\forall \beta \leq \alpha)(\exists y)\varphi(\beta, \{(\gamma, n) | n \in f(\gamma) \wedge \gamma < \beta\}, y, f(\beta))) \wedge f(\alpha) = z.$$

Since $L_{\omega_1^z}[z]$ is admissible, G is Σ_1-definable. In other words, $G(\alpha) = z$ if and only if there is a function $f : \alpha + 1 \to 2^\omega$ with $f \in L_{\omega_1^z}[z]$ such that

$$L_{\omega_1^z}[z] \models$$
$$((\exists s)(\forall \beta \leq \alpha)(\exists y \in s)\varphi(\beta, \{(\gamma, n) | n \in f(\gamma) \wedge \gamma < \beta\}, y, f(\beta))) \wedge f(\alpha) = z.$$

Define the range of G to be T. Then $z \in T$ if and only if there exists an ordinal $\alpha < \omega_1^y$ and a function $f : \alpha + 1 \to 2^\omega$ with $f \in L_{\omega_1^z}[z]$ such that

$$L_{\omega_1^z}[z] \models$$
$$((\exists s)(\forall \beta \leq \alpha)(\exists y \in s)\varphi(\beta, \{(\gamma, n) | n \in f(\gamma) \wedge \gamma < \beta\}, y, f(\beta))) \wedge f(\alpha) = z.$$

So T is Π^1_1.

All that remains is to show that G is a well-defined total function on ω_1. This can be done using the same argument as that for showing the recursion theorem over admissible structures (see Barwise [1]). The only difficult part is to argue, as was done earlier, that the function f defined above exists. We leave this to the reader.

We show that G is a maximal antichain. Suppose not, then there is a $<_L$-least real $x_0 \notin G$ so that $\{x_0\} \cup G$ is an antichain. Since $V = L$, $x_0 \in L_\gamma$ for some $\gamma < \omega_1$. Then, by the construction above, there must be some real $y \in G$ so that $y \geq_{\mathrm{T}} x_0$, a contradiction.

To see that G is thin, it suffices to show that $z \in L_{\omega_1^z}$ if z is in the range of F. By (2), $E \in L_{\beta+1}$ and $\beta + 1 < \omega_1^E$. So $E \in L_{\omega_1^E}$. By (4), $\omega_1^E = \omega_1^z$ and $z \in L_{\beta+2} \subseteq L_{\omega_1^E}$. So $z \in L_{\omega_1^z}$. By a result of Mansfield-Solovay [7], G is a thin set.

Theorem 1. *(ZFC)*

(i) There is a thin Π^1_1 maximal antichain of Turing degrees if and only if $(2^\omega)^L = 2^\omega$.

(ii) There is a thin $\mathbf{\Pi}^1_1$ maximal antichain of Turing degrees if and only if $(2^\omega)^{L[x]} = 2^\omega$ for some real x.

Proof.

(i) Suppose A is a thin Π^1_1 maximal antichain. Then, by Solovay's result [11], $A \subset L$. Now let x be a real. By a theorem of Cooper [4], there is a real y of minimal degree such that $x \leq_T y'$. Since A is a maximal antichain, there is a real $z \in A$ with $z \geq_{\mathrm{T}} y$. So $x \leq_{\mathrm{T}} z'$. Hence $x \in L$.
Conversely, suppose $(2^\omega)^L = 2^\omega$. Fix a Π^1_1 set G as in Lemma 2. Since the statement "G is an antichain in the Turing degrees" is Π^1_2 and

$$L \models \text{“}G \text{ is an antichain in the Turing degrees”},$$

G is an antichain in the Turing degrees by absoluteness. Fix a real x. Since $(2^\omega)^L = 2^\omega$, $x \in L$. The statement $T(x)$:"there exists $y \in G$ so that y is Turing comparable with x" is $\Sigma^1_2(x)$ and $L \models T(x)$. It follows that $T(x)$ is true. Thus G is a maximal antichain.

(ii) Relativize the proof of (i).

It follows that to construct a model in which there is no thin $\mathbf{\Pi}^1_1$ maximal antichain of Turing degrees, one just needs to refute CH in the model. It is natural to ask whether there is a model of $ZFC + CH$ with no thin $\mathbf{\Pi}^1_1$ maximal antichain of Turing degrees. The answer is yes: Apply iterated Cohen forcing with finite support of length $(\omega_1)^L$, i.e. conditions of the form $((<\omega, 2)_\alpha, <: \alpha < \omega_1)$ over L to obtain a generic set G. Notice that this notion of forcing satisfies the (set-theoretic) countable chain condition (c.c.c), and so preserves all cardinals. Now $L[G] \models ZFC + CH$. If there is a real $x \in L[G]$ so that $(2^\omega)^{L[x]} = 2^\omega$, then $x \in L[G_\alpha]$ for some $\alpha < \omega_1$ where G_α is the generic set obtained from iterated forcing up to α. Then for any real $y \in L[G] - L[G_\alpha]$, y is not constructible in x. It follows from Theorem 1 that there is no thin $\mathbf{\Pi}^1_1$ maximal antichain of Turing degrees in $L[G]$.

3 Applications to the Measure Theory of Turing Degrees

In [12], Yu investigated measure theoretic aspects of the Turing degrees. In this section, we continue the investigation by applying the results in the previous section to study some problems in this area.

Given a set A of reals, we define $\mathcal{U}(A) = \{y | \exists x(x \in A \wedge x \leq_{\mathrm{T}} y)\}$. We have the following proposition.

Proposition 3. *If A is a Π^1_1 thin set, then $\mu(\mathcal{U}(A)) = 0$.*

Proof. Fix a ZFC model $\mathfrak{M}$. If A is Π^1_1, then $\mathcal{U}(A)$ is Π^1_1 and so measurable. By a result of Sacks [9], the set $C = \{n \in \omega | \mu(\mathcal{U}(A)) > 2^{-n}\}$ is Π^1_1. Since A is a thin Π^1_1 set, $A \subset L$. Extend $\mathfrak{M}$ to a generic $\mathfrak{N}$ by any notion of forcing that collapses $(\omega_1)^L$ to ω. Then A is still thin by absoluteness since the statement "A is a thin set" is Π^1_2. In the generic extension $\mathfrak{N}$, A is countable since A is a subset of constructible reals. So $\mathcal{U}(A)$ is a null set in $\mathfrak{N}$. I.e. "$\forall n(n \notin C)$" is true in $\mathfrak{N}$. Since the statement "$\forall n(n \notin C)$" is Σ^1_1, it is true in $\mathfrak{M}$. Thus $\mu(\mathcal{U}(A)) = 0$ in $\mathfrak{M}$.

Together with Theorem 1, we have the following corollary.

Corollary 2. *Assume* $(2^\omega)^L = 2^\omega$. *There is a maximal antichain* A *in the Turing degrees such that* $\mu(A) = \mu(\mathcal{U}(A)) = 0$.

We say that a set $X \subset 2^\omega$ is a *quasi-antichain* in the Turing degrees if there is an antichain $\mathbf{X} \subset \mathfrak{D}$ so that $X = \{x | x \text{ is of } \mathbf{x} \wedge \mathbf{x} \in \mathbf{X}\}$.

Yu [12] showed that there is a nonmeasurable quasi-antichain in the Turing degrees and no quasi-antichain has positive measure. In response to Yu's results, Jockusch [12] asked the following question:

Question 1 (Jockusch). Is every maximal quasi-antichain in the Turing degrees nonmeasurable?

We answer this question in the negative under the assumption that every real is constructible:

Corollary 3. *Assume* $(2^\omega)^L = 2^\omega$. *There is a null maximal quasi-antichain in the Turing degrees.*

Proof. By Corollary 2, there is a maximal antichain A so that $\mu(\mathcal{U}(A)) = 0$. Then $B = \{y | \exists x (x \in A \wedge x \equiv_{\mathrm{T}} y)\} \subseteq \mathcal{U}(A)$ is a null maximal quasi-antichain.

References

1. Barwise, J.: Admissible sets and structures. Springer, Heidelberg (1975)
2. Boolos, G., Putnam, H.: Degrees of unsolvability of constructible sets of integers. J. Symbolic Logic 33, 497–513 (1968)
3. Chong, C.T., Yu, L.: Maximal chains in the turing degrees. J. Symbolic Logic (To appear)
4. Cooper, S.B.: Minimal degrees and the jump operator. J. Symbolic Logic 38, 249–271 (1973)
5. van Engelen, F., Miller, A.W., Steel, J.: Rigid Borel sets and better quasi-order theory. In: Logic and combinatorics (Arcata, Calif, 1985), volume 65 of Contemp. Math, pp. 199–222. Amer. Math. Soc, Providence, RI (1987)
6. Jensen, R.B.: The fine structure of the constructible hierarchy. Ann. Math. Logic, 4:229–308; erratum, ibid. 4 (1972), 443 (1972)
7. Mansfield, R.: Perfect subsets of definable sets of real numbers. Pacific J. Math. 35, 451–457 (1970)
8. Martin, D.A.: Borel determinacy. Ann. of Math (2) 102(2), 363–371 (1975)
9. Sacks, G.E.: Measure-theoretic uniformity in recursion theory and set theory. Trans. Amer. Math. Soc. 142, 381–420 (1969)
10. Sacks, G.E.: Higher recursion theory. In: Perspectives in Mathematical Logic, Springer, Heidelberg (1990)
11. Solovay, R.M.: On the cardinality of Σ_2^1 sets of reals. In: Foundations of Mathematics (Symposium Commemorating Kurt Gödel, Columbus, Ohio, 1966), pp. 58–73. Springer, New York (1969)
12. Yu, L.: Measure theory aspects of locally countable orderings. J. Symbolic Logic 71(3), 958–968 (2006)

Effective Computation for Nonlinear Systems

Pieter Collins[⋆]

Centrum voor Wiskunde en Informatica,
Postbus 94079,
1090 GB Amsterdam,
The Netherlands
Pieter.Collins@cwi.nl

Abstract. Nonlinear dynamical and control systems are an important source of applications for theories of computation over the the real numbers, since these systems are usually to complicated to study analytically, but may be extremely sensitive to numerical error. Further, computer-assisted proofs and verification problems require a rigorous treatment of numerical errors. In this paper we will describe how to provide a semantics for effective computations on sets and maps and show how these operations have been implemented in the tool Ariadne for the analysis, design and verification of nonlinear and hybrid systems.

Keywords: computable analysis, nonlinear systems, Ariadne.

1 Introduction

A simple but important problem in nonlinear systems theory is that of *safety verification*. Given an autonomous system, $x_{n+1} = f(x_n)$ or $\dot{x} = f(x)$, we wish to determine whether the evolution of the system starting in an initial set X_0 remains in some safe set S. To solve such a problem we need a method which can rigorously compute the image of a set or integrate a differential equation over a set of points. Further, we want our algorithm to be efficient enough to be practically useful, and optimal, in the sense that if it is theoretically possible to decide safety, then the algorithm returns the correct answer. For hybrid systems (see [1]), which combine continuous-time and discrete-time evolution, the problem is even more challenging.

There has been much work recently on *computable analysis* [2,3], which provides a theory of computation on objects from geometry and analysis such as sets and maps, and is essential equivalent to approaches based on Scott domains [4]. This theory can be applied to discuss the computability of the *reachable set* [5], which plays a crucial role in safety verification.

There are many algorithms for rigorously integrating differential equations, dating back to work of Moore [6] from the early days of interval analysis, and.

Many tools have been developed which are capable of computing or approximating reachable sets. The package GAIO [7] may be used to compute the

[⋆] This work was partially supported by the Nederlandse Wetenschappelijk Organisatie (NWO) through VIDI project number 639.032.408.

image of a set, but does not provide a rigorous method for integrating differential equations. The Lohner method for rigorous integration is available in the CAPD Library [8], and higher-order Taylor methods are implemented in VNODE [9]. Other tools are available for reachability analysis of hybrid systems, but only Checkmate [10] can handle nonlinear dynamics. However, there is still a need for a general-purpose open-source tool which can solve a wide variety of problems in nonlinear dynamic systems.

The goal of the computational framework Ariadne [11] is to provide a syntax, semantics and implementation of fundamental operations from geometry and analysis, guided by formal computability theory. In this paper, we describe how the numerical kernel of Ariadne, written in C++, implements computation with sets and maps, in a way which can be efficiently implemented and is powerful enough to solve the safety verification problem.

2 Reachability and Safety Computation

Consider a discrete-time system with state-space X described by the continuous map $f : X \to X$, a set of initial states X_0 and a set of safe states S. The *safety verification* problem is decide whether every orbit of f starting in X_0 remains in S. We can express the safety verification problem in terms of the *reachable set* as:

$$\mathrm{reach}(f, X_0) \subset S, \text{ where } \mathrm{reach}(f, X_0) := \textstyle\bigcup_{n=0}^{\infty} f^n(X_0).$$

To even consider how to solve this problem, we first need to have a description of the sets S and X_0 and the map f. Since the set of continuous functions on $\mathbb{R}^n$ has continuum cardinality, we cannot represent all functions exactly. Instead, following Weihrauch [2], we describe elements of an infinite set Y by a *representation* $\delta :\subset \Sigma^\omega \to Y$, which encodes $y \in Y$ by a sequences over some alphabet Σ. To be useful, we must be able to obtain approximations to y from a initial part of some $p \in \delta^{-1}(y)$. Given representations $\delta_0, \ldots, \delta_k$ of sets $Y_0, \ldots, Y_k$, we say a function $f : Y_1 \times \cdots \times Y_k \to Y_0$ is *computable* if there is a Turing-computable function $\mathcal{M} :\subset \Sigma^\omega \times \cdots \times \Sigma^\omega \to \Sigma^\omega$ such that $\delta_0(\mathcal{M}(p_1, \ldots, p_k)) = f(\delta_1(p_1), \ldots, \delta_k(p_k))$.

If (Y, τ) is a topological space with countable sub-base σ labelled by a partial surjective function $\nu :\subset \Sigma^* \to \sigma$, then the *standard representation* δ of (Y, τ, σ, ν) is defined by

$$\delta(\langle w_1, w_2, \ldots\rangle) = y \iff \{\nu(w_i) \mid i \in \mathbb{N}\} = \{J \in \sigma \mid y \in J\}.$$

Informally, we say that δ encodes a list of all $J \in \sigma$ such that $y \in J$.

In this paper, we fix a state space X (such as $\mathbb{R}^n$) with countable base β. As well as the set of points of X, the open sets $\mathcal{O}$, closed sets $\mathcal{A}$, compact sets $\mathcal{K}$, and continuous self-maps $\mathcal{C}$ of X all have natural topologies giving rise to the following standard representations:

– The representation ρ of X encodes $\{J \in \beta \mid x \in J\}$.
– The lower representation $\theta_<$ of $\mathcal{O}$ encodes $\{I \in \beta \mid \overline{I} \subset U\}$.

- The lower representation $\psi_<$ of $\mathcal{A}$ encodes $\{J \in \beta \mid J \cap A \neq \emptyset\}$.
- The upper representation $\psi_>$ of $\mathcal{A}$ encodes $\{I \in \beta \mid \overline{I} \cap A = \emptyset\}$.
- The upper representation $\kappa_>$ of $\mathcal{K}$ encodes $\{(J_1, \ldots, J_k) \in \beta^* \mid C \subset \bigcup_{i=1}^{k} J_i\}$.
- The compact-open representation δ of $\mathcal{C}$ encodes $\{(I, J) \in \beta \times \beta \mid \overline{I} \subset f^{-1}(J)\}$.

In [5], we show that the optimal $\kappa_>$-semicomputable over-approximation to $\mathrm{reach}(f, X_0)$ is the *chain-reachable set*

$$\mathrm{chain\,reach}(f, X_0) := \bigcap \big\{ C \in \mathcal{K} \mid X_0 \cup f(C) \subset C^\circ \big\},$$

which may be much larger than the reachable set. This means that it is impossible to prove safety (using only the approximate information about f, X_0 and S given by the standard representations) if $\mathrm{chain\,reach}(f, X_0) \not\subset S$, even if $\mathrm{reach}(f, X_0) \subset S$. Similarly, it is only possible to disprove safety if $\mathrm{reach}(f, X_0) \not\subset \overline{S}$. Hence the best possible solution to the safety verification problem (in an approximative setting) is an algorithm which computes:

$$\mathrm{verify}(f, X_0, S) := \begin{cases} \top & \text{if } \mathrm{chain\,reach}(f, X_0) \subset S; \\ \bot & \text{if } \mathrm{reach}(f, X_0) \not\subset \overline{S}; \\ \uparrow & \text{otherwise.} \end{cases}$$

3 Representations as Interfaces

The elements of a countable (i.e. discrete) set such as the integers or rational numbers may be described exactly by finite data, and correspond to *concrete types*. It is straightforward to implement types `Integer` and `Rational` which implement integer and rational numbers, such that arithmetic and comparison operators are computable.

The elements of an uncountable set such as the real numbers cannot be described by a finite amount of data. However, we shall see that they can be adequately described by *abstract interfaces* with properties reflecting the standard representation.

Throughout the paper, we use typeface `x` to denote data types and x to denote the corresponding mathematical object.

3.1 The Standard Representation of Points

Recall that we can describe an element of a Hausdorff space with countable base β using the standard representation, which encodes a point x by listing its basic open neighbourhoods. We could describe the standard representation by a method `neighbourhood` taking an integer argument and returning an element of β implemented by a `BasicSet` class:

```
virtual BasicSet Point::neighbourhood(Integer i);
```

In practice, it is more useful to be able to determine whether x lies in some element I of β. Unfortunately, the standard representation ρ merely yields a semi-decision algorithm for the predicate $x \in I$. If $x \notin \overline{I}$, we can indeed deduce that $x \notin \overline{I}$, since there exists $J \in \beta$ such that $x \in J$ and $I \cap J = \emptyset$. But if $x \in \partial I$, then no such J exists, so we are unable to show $x \notin I$. Hence, any implementation of a method `bool Point::in(BasicSet)` using only approximate information about x will fail to terminate on `x.in(I)` if $x \in \partial I$.

In order to obtain a function which always terminates, we give a *precision* argument p, and return an *"indeterminate"* value if we cannot decide $x \in I$ or $x \notin I$ to precision p. We therefore have a method

```
virtual tribool Point::in(BasicSet I, Integer p);
```

where `tribool` is the enumerated type {`true,false,indeterminate`}.

We now give conditions under which the `in` method specifies a point in X uniquely. For consistency, we clearly require,

(C) If `x.in(I,p)==true` and `x.in(J,r)==false`, then $I \not\subset J$.

Define $\eta = \{J \in \beta \mid \exists p \in \mathbb{N}$ s.t. `x.in(J,p)==true`$\}$. Then η must satisfy the following intersection, refinement and approximation properties:

(I) $I_1, I_2 \in \eta \implies I_1 \cap I_2 \neq \emptyset$.
(R) $I \in \eta$ and $I \subset J_1 \cup \cdots \cup J_k \implies \exists i \in \{1, \ldots, k\}$ such that $J_i \in \eta$.
(A) $J \in \eta \implies \exists I \in \eta$ such that $\overline{I} \subset J$.

Further, if η is any set satisfying these properties, then there exists x such that $\eta = \{J \in \beta \mid x \in J\}$.

To simplify code using the `in` method, we can assume that the method is implemented such that the following precision and monotonicity properties are satisfied:

(P) If `r>p` and `x.in(I,p)!=indeterminate`, then `x.in(I,r)==x.in(I,p)`.
(M) If $I \subset J$, then `x.in(I,p)` $\Longrightarrow$ `x.in(J,p)`, and `!x.in(J,p)` $\Longrightarrow$ `!x.in(I,p)`.

3.2 Representations of Sets

A closed set is uniquely specified by the basic open sets it intersects, or by the basic closed sets it is disjoint from. These specifications give rise to the *lower* and *upper* representations $\psi_<$ and $\psi_>$, respectively. Since it cannot be true that both $A \cap I \neq \emptyset$ and $A \cap \overline{I} = \emptyset$, we can specify an interface for closed sets by the single method

```
virtual tribool ClosedSet::disjoint(BasicSet I, Integer p);
```

where p is the precision argument. Using the method `disjoint`, we can compute

$$\alpha_< = \{I \in \beta \mid \exists p \in \mathbb{N} \text{ s.t. } \texttt{A.disjoint(I,p)==false}\},$$
$$\alpha_> = \{I \in \beta \mid \exists p \in \mathbb{N} \text{ s.t. } \texttt{A.disjoint(I,p)==true}\}.$$

To ensure that `disjoint` gives consistent results, we require the following condition on $\alpha_<$ and $\alpha_>$.

(DC) If $I \in \alpha_<$ and $J_i \in \alpha_>$ for $i = 1, \ldots, k$, then $I \not\subset \bigcup_{i=1}^k \overline{J}_i$.

Given $\alpha_<$ and $\alpha_>$, we can recover sets $A_<$ and $A_>$ by:

$A_< = \{x \mid \exists (J_i)_{i\in\mathbb{N}}$ with $J_i \in \alpha_<$ s.t. $J_{i+1} \subset J_i$ and $\bigcap_{i\in\mathbb{N}} J_i = \{x\}\}$.
$A_> = X \setminus U_>$, where $U_> = \{x \mid \exists I \in \alpha_>$ s.t. $x \in \overline{I}\}$.

For $\alpha_<$, we impose the following monotonicity, refinement and approximation properties:

(DM$_<$) If $I \subset J$ and $I \in \alpha_<$, then $J \in \alpha_<$.
(DR$_<$) If $I \subset \bigcup_{i=1}^k J_i$ and $I \in \alpha_<$, then there exists i such that $J_i \in \alpha_<$.
(DA$_<$) If $J \in \alpha_<$, then there exists $\overline{I} \subset J$ such that $I \in \alpha_<$.

The condition $\mathbf{DM}_<$ ensures that the set $A_<$ is closed, and the stronger condition $\mathbf{DR}_<$ ensures that every basic set $I \in \alpha_<$ contains a point in $A_<$.

In order that $A_>$ is a closed set, we require:

(DA$_>$) If $I \in \alpha_>$, then there exists $J \supset \overline{I}$ such that $J \in \alpha_>$.

To simplify algorithms using the `disjoint` method, we usually require either:

(DM$_>$) If $\overline{I} \subset \overline{J}$ and $J \in \alpha_>$, then $I \in \alpha_>$.
(DR$_>$) If $\overline{I} \subset \bigcup_{i=1}^k \overline{J}_i$ and $J_i \in \alpha_>$ for all i, then $I \in \alpha_>$.

In order that $\alpha_<$ and $\alpha_>$ yield equal closed sets, we require:

(DE) For all $J \in \beta$, there exists $I \subset J$ such that $I \in \alpha_< \cup \alpha_>$.

The following result gives conditions under which the `disjoint` method contains equivalent information to the standard representations of closed sets.

Theorem 1

1. *If* ***(DR$_<$,DA$_<$)****, then $A_<$ is closed and $\alpha_< = \{J \in \beta \mid A_< \cap J \neq \emptyset\}$.*
2. *If* ***(DA$_>$)*** *then $A_>$ is closed; if also* ***(DR$_>$)*** *then $\alpha_> = \{I \in \beta \mid A_> \cap \overline{I} = \emptyset\}$.*
3. *If* ***(DC,DR$_<$,DA$_<$,DA$_>$,DE)****, then $A_< = A_>$.*

The proof is omitted.

Since an open set is the complement of a closed set, we immediately obtain an interface for open sets:

```
virtual tribool OpenSet::superset(BasicSet,Integer);
```

The `superset` method defines a set $U_< = \bigcup\{\overline{I} \mid \exists p$ s.t. `U.superset(I)`$\}$. Given properties analogous to those for the `disjoint` method, we can prove that the `superset` interface is equivalent to the standard representation of open sets $\mathcal{O}$.

Since a compact set is just a bounded closed set, we can specify a compact set using the `disjoint` method, together with the method

```
virtual BasicSet CompactSet::BoundingBox();
```

which yields a basic set $I \supset C$.

3.3 Representations of Continuous Functions

Continuous functions can be described by the interface

```
virtual BasicSet Map::apply(BasicSet);
```

An object `f` of class `Map` represents a continuous function f if the following consistency and refinement conditions hold:

(FC) $f(I) \subset$ `f.apply(I)`,
(FR) if $J \ni f(x)$, then there exists I such that $x \in I$ and `f.apply(I)` $\subset J$.

It is also useful in practise to impose the monotonicity property:

(FM) if $I \subset J$, then `f.apply(I)` $\subset$ `f.apply(J)`, and

A differentiable function can be specified by giving the Jacobian derivative explicitly as a matrix of interval values

```
virtual Matrix<Interval> C1Map::jacobian(BasicSet);
```

4 Implementation in Ariadne

4.1 Numerical Types

Ariadne supports various types which can be used to represent real numbers, namely `Float64`, `MPFloat`, `Rational` and `ComputableReal`. These types are classified in terms of the way they handle arithmetic and approximation.

The `ComputableReal` type supports arbitrary arithmetical, algebraic and transcendental functions, which return exact results. The `Rational` type only supports arithmetic, but this is also exact. The *floating-point* types `Float64` and `MPFloat` do not support exact arithmetic, since in general, arithmetical operations cannot be computed exactly for these types. Instead, floating-point types support *interval* arithmetic and functions, which return an object of type `Interval<Float>` which must contain all possible results which could be obtained.

The *precision* of a floating point type is the number of bits/bytes used to store the number. The type `Float64` is a *fixed-precision* type with 64 bits, and the type `MPFloat` is a *multiple-precision* type. The precision of an object of type `MPFloat` is set when the object is constructed, and is only changed on explicit `set_precision` function call. The precision may be given explicitly, but it is usually more convenient to allow the precision to be determined implicitly by the precision of the arguments (and of the result, if the object is a temporary intermediate in a computation), and by a default precision. To avoid accidental truncation, it is an error to assign a number to an `MPFloat` object of lower precision, though assignment to `Interval<MPFloat>` objects is allowed.

The memory for an `MPFloat` is allocated on the heap, which may be slow, but once the object is created, no memory management is required, so computation is reasonably fast.

The fixed-precision types are useful for problems where speed of execution is paramount. The multiple-precision type can be used if the fixed-precision types do not give sufficient accuracy, which may be the case if the problem depends sensitively on initial data or is not robust to perturbations. Rational numbers are useful when exact arithmetical results are required or efficiency is not an issue, and computable real numbers are useful for problem specification.

Currently, the type `Float64` is implemented using the IEEE `double` floating-point numbers, the `MPFloat` type using `mpfr_t` from the MPFR library [12], the `Rational` using `mpq_t` from the GMP library [13], and the `ComputableReal` type using the iRRAM package [14].

4.2 Linear Algebra

Linear algebra is important when working with derivatives of maps, and with polyhedral sets. In Ariadne, we provide `Vector`, `Matrix` and `Tensor` classes, which can be used with rational numbers, floating-point numbers and intervals, and also robust linear programming solvers for testing geometric predicates.

4.3 Geometric Calculus

The geometric calculus used by Ariadne is based around elementary *basic set* types. A *denotable set* is a finite union of basic sets (of a given type), and can be stored using a finite amount of data. Arbitrary sets are either described by the interfaces given in Section 3.2, or by approximations in terms of denotable sets. Operations on sets can be built on the fundamental geometric predicates of *disjointness* and *subset*, and the operations of *union*, *intersection*, *subdivision* and *approximation*.

To simplify the interface, we do not use an explicit precision argument in Ariadne. Instead, the precision of an operation acting on sets represented using floating-point types is determined by the precision used for the arguments and the default precision. Basic sets with interval coefficients are returned by operations which require arithmetic on sets represented using floating-point types. Operations using sets based on rational numbers are computed exactly.

The simplest type of basic set for Euclidean space are the rational or dyadic hypercubes of the form $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n]$. In Ariadne, hypercubes are represented by the template `Cuboid<R>`, where `R` is the numerical type used to represent the a_i and b_i. This may be a `Float` type, an `Interval<Float>` or a `Rational`. The core interface is given below:

```
class Cuboid<R> {
  Integer dimension();
  R lower_bound(Integer);
  R upper_bound(Integer);
  Integer precision();
};
```

Besides cuboids, ARIADNE provides basic set classes

`Zonotope`: $\{x \in \mathbb{R}^n \mid x = c + Ge \text{ where } c \in \mathbb{R}^n,\ G \in \mathbb{R}^{n\times k} \text{ and } e \in [-1,1]^k\}$.
`Polytope`: $\{x \in \mathbb{R}^n \mid x = Vs \text{ where } V \in \mathbb{R}^{n\times k},\ s \in \mathbb{R}^k_+ \text{ and } \sum_{i=1}^k s_i = 1\}$.
`Polyhedron`: $\{x \in \mathbb{R}^n \mid Ax \le b \text{ where } A \in \mathbb{R}^{k\times n} \text{ and } b \in \mathbb{R}^k\}$.

The `Zonotope` class is useful to support higher-order integration of vector fields and iteration of maps. The `Polytope` and `Polyhedron` classes are general classes which are useful for computing geometric predicates; additionally, `Polyhedron` class may be useful for specifying input sets. Any polyhedral set can be converted to a `Polytope` or `Polyhedron`, although in most cases the conversion cannot be done exactly using floating-point arithmetic. `Parallelotope`, `Simplex`, `Sphere` and `Ellipsoid` classes are also provided. The basic set classes support the binary predicates given below.

```
tribool contains(Cuboid<R1>,Point<R2>);
tribool disjoint(Cuboid<R1>,Cuboid<R2>);
tribool subset(Cuboid<R1>,Cuboid<R2>);
```

These predicates return `indeterminate` if the result is not robust with respect to changes in the parameters. Basic set types also support the `Cuboid bounding_box()`, `subdivide` and `over_approximation` functions. Other binary operations are provided where natural, such as `minkowski_sum`, `convex_hull`, `open_intersection` and `closed_intersection`.

A *denotable set* is a set which is described exactly as a finite union of basic sets, and typically represents an approximation to some other set. Denotable sets support the fundamental geometric operations, iteration through their elements and the `union` function.

ARIADNE currently supports classes `ListSet<BasicSet>`, `GridCellListSet`, `GridMaskSet` and `PartitionTreeSet`. A `ListSet` is an arbitrary finite union of basic sets. The other types are *partition sets*, since they are based on a topological partition of the state space. The `GridMaskSet` class is easy to work with and is an efficient way of storing unstructured sets. The `GridCellListSet` class is useful to represent approximations to basic sets. The `PartitionTreeSet` class is a highly efficient way of storing structured sets with dynamically-varying resolution.

We say a denotable set is an *under-approximation* to a set S if $\bigcup_{i=1}^k \overline{I}_i \subset S$, and an *over-approximation* if $\bigcup_{i=1}^k \overline{I}_i \supset S$. A list set $\bigcup_{i=1}^k J_i$ is a *lower-approximation* to S if $J_i \cap S \neq \emptyset$ for all i. Approximations on partition sets can be specified by giving a `Grid` or other `Partition`, or by directly adjoining elements of an existing partition set:

```
void PartitionSet::adjoin_over_approximation(Set);
```

General sets can be specified by how they interact with `Cuboid` basic sets, as given by the methods `disjoint`, `superset`, `subset` and `bounding_box`. Using `superset` we can compute under approximations, using `disjoint` we can compute lower-approximations, and using `disjoint` and `subset` or `bounding_box`, we can compute over-approximations.

4.4 Computing the Image of Sets

One of the most important tasks in Ariadne is to compute the image of a set under a continuous function. Using just the apply method of the Map interface described in Section 3.3, we can compute images and preimages of general sets:

```
CompactSet image(Map, CompactSet);
ClosedSet lower_image(Map, ClosedSet);
OpenSet lower_preimage(Map, OpenSet);
```

The image of a closed, but not compact set, and the preimage of an open set are only lower-semicomputable, in the sense that we can only effectively compute convergent lower-approximations to result. The image of a compact set can be computed to arbitrary accuracy.

Although the apply method is in principle sufficient to compute set images and preimages, convergence to a good approximation tends to be slow due to the "wrapping effect" of interval arithmetic. It is therefore preferable to use higher-order algorithms if derivatives of the function are available. Since the class of zonotopes is closed under affine maps, we can use zonotopes to compute first-order approximations to the image of a set.

```
Zonotope< Interval<R>> apply(C1Map<R>, Zonotope<R>);
```

The return type is an interval set to avoid expensive approximation operations. The image of a rational zonotope under an affine map can be computed exactly. If higher-order derivatives are available, even more accurate Taylor methods can be used.

4.5 The Verification Algorithm

We can now sketch an algorithm to solve the safety problem.

To attempt to verify $\operatorname{chain reach}(f, X_0) \subset S$ for some bounded open set S, we compute an over-approximation $\widehat{X}_0$ to X_0 and an under-approximation $\widetilde{S}$ of S on a grid G using the over_approximate and under_approximate functions. We then discretise f by computing an over-approximation $\widehat{f}(\overline{I})$ to $f(\overline{I})$ for every grid cell $\overline{I}$ using the apply(Map,Zonotope) and over_approximate(Zonotope,Grid) functions. Finally, we can compute the reachable set $\widehat{X}_0$ under $\widehat{f}$ combinatorially. It is straightforward to show that if $\operatorname{reach}(\widehat{f}, \widehat{X}_0) \subset \widetilde{S}$ then $\operatorname{chain reach}(f, X_0) \subset S$, and that the converse holds if the grid G is sufficiently fine.

To attempt to verify $\operatorname{reach}(f, X_0) \not\subset S$, we find a basic set I and a natural number n such that $X_0 \cap I \neq \emptyset$ and $f^n(\overline{I}) \cap \overline{S} = \emptyset$. We can use the apply function to compute an over-approximation $\widehat{I}_n$ to $f^n(\overline{I})$, and the disjoint method to show $X_0 \cap I \neq \emptyset$ and $\overline{S} \cap \widehat{I}_n = \emptyset$.

In practice, over-approximations to $f^n(X_0)$ are adaptively computed on-the-fly, and counterexamples from the discretised safety verification algorithm are used to guide safety falsification.

5 Concluding Remarks

In this paper, we have outlined a scheme for rigorous computation on sets and maps based on the theory of computable analysis and standard representation of topological space. This scheme is being used as the interface for the implementation in C++ of the numerical kernel of the tool Ariadne for reachability analysis and verification of nonlinear and hybrid dynamical and control systems. The operations required by the interface have a natural syntax and have efficient implementations, some which are based on well-studied general algorithms such as the simplex algorithm or integration algorithms.

Ongoing work includes improving the efficiency of the existing algorithms in Ariadne, providing more advanced capabilities for applying maps and integration vector fields based on higher-order Taylor methods, providing interfaces to external packages, and extending the interface to cover more advanced problems.

References

1. van der Schaft, A., Schumacher, H.: An introduction to hybrid dynamical systems. Lecture notes in control and information sciences, vol. 251. Springer, London (2000)
2. Weihrauch, K.: Computable analysis - An introduction. In: Texts in Theoretical Computer Science, Springer, Heidelberg (2000)
3. Brattka, V., Presser, G.: Computability on subsets of metric spaces. Theoretical Comp. Sci. 305, 43–76 (2003)
4. Stoltenberg-Hansen, V., Lindström, I., Griffor, E.R.: Mathematical Theory of Domains. Cambridge University Press, Cambridge (1994)
5. Collins, P.: Continuity and computability of reachable sets. Theor. Comput. Sci. 341, 162–195 (2005)
6. Moore, R.E.: Interval Analysis. Prentice-Hall, Englewood Cliffs, N.J (1966)
7. Dellnitz, M., Froyland, G., Junge, O.: The algorithms behind GAIO-set oriented numerical methods for dynamical systems. In: Ergodic theory, analysis, and efficient simulation of dynamical systems, pp. 145–174. Springer, Heidelberg (2001)
8. Mrozek, M., et al.: CAPD Library (2007), http://capd.wsb-nlu.edu.pl/
9. Nedialkov, N.S.: VNODE-LP: A validated solver for initial value problems in ordinary differential equations. Technical report, McMaster University (2006) CAS-06-06-NN.
10. Izaias Silva, B., Keith Richeson, B.K., Chutinan, A.: Modeling and verification of hybrid dynamical system using CheckMate. In: Proceedings of the International Conference on Automation of Mixed Processes. pp. 189–194 (2000)
11. Balluchi, A., Casagrande, A., Collins, P., Ferrari, A., Villa, T., Sangiovanni-Vincentelli, A.L.: Ariadne: a framework for reachability analysis of hybrid automata. In: Proceedings of the International Syposium on Mathematical Theory of Networks and Systems (2006)
12. Hanrot, G., et al.: The MPFR library (2000), http://www.mpfr.org/
13. Granlund, T., et al.: The GMP library (2005), http://swox.com/gmp/
14. Müller, N., et al.: iRRAM (2006), http://www.informatik.uni-trier.de/iRRAM/

On Rules and Parameter Free Systems in Bounded Arithmetic

Andres Cordòn-Franco, Alejandro Fernández-Margarit, and Francisco Felix Lara-Martín[⋆]

Facultad de Matemáticas. Universidad de Sevilla
C/ Tarfia, s/n, 41012 Sevilla, Spain
{acordon,afmargarit,fflara}@us.es

Abstract. We present model–theoretic techniques to obtain conservation results for first order bounded arithmetic theories, based on a hierarchical version of the well known notion of an existentially closed model.

Keywords: Bounded Arithmetic, conservation results, parameter free schemes.

1 Introduction

Bounded arithmetic theories are formal systems tailored to capture computational complexity classes. The foundational work in this area is [3], where S. Buss introduced the families of theories S_2^i and T_2^i ($i \geq 0$) and showed that they can be considered as formal counterparts of the Polynomial Time Hierarchy PH. Since then a variety of related systems have been introduced in order to deal with other complexity classes. Among the fundamental results on these systems two groups can be isolated: (a) characterizations of their computational strength, mainly, by determining their Σ_i^b*–definable functions*; and (b) relationship among different axiomatizations, especially, *conservation results*.

Here we present model–theoretic methods to obtain both kinds of results for restricted versions of Buss's theories S_2^i, T_2^i as well as for the Σ_i^b–replacement scheme $BB\Sigma_i^b$. Systems S_2^i and T_2^i are axiomatized over a certain base theory by *axiom schemes* expressing (respectively) the polynomial and the usual induction principles restricted to Σ_i^b–formulas. We shall weaken these theories in two ways: (1) by formalizing the corresponding induction or replacement principle as an *inference rule* instead of an axiom scheme, or (2) by restricting the induction schemes to *parameter free* formulas. In the first case we drop the axiom scheme and consider the closure of the base theory under first order logic and nested applications of the corresponding inference rule. In the second case we still deal with an axiom scheme but now it is restricted to formulas with no other free variables than the induction variable. While the effects of these restrictions have been extensively investigated for fragments of Peano Arithmetic, it is not the

[⋆] Partially supported by grants MTM2005-08658 of MEC and TIC-137 of Junta de Andalucía, Spain. Part of this work was done while the first author was visiting the Mathematical Institute of the Academy of Sciences of the Czech Republic in 2004.

case of Bounded Arithmetic. To our best knowledge, parameter free systems have only been systematically studied by S. Bloch in the second part of his thesis [2]. On the other hand, systems described by inference rules in the sense above seldom appear in an explicit manner in the literature. A recent exception is J. Johannsen and C. Pollett's work [7], where the authors study the Δ_1^b–bit–comprehension rule in connection with the complexity class TC^0 of functions computable by uniform threshold circuit families of polynomial size and constant depth. Moreover, both in [2] and in [7] the analysis of those systems has been carried out by means of proof–theoretic methods.

In this paper we shall develop a model–theoretic approach to the investigation of these restricted systems. To this end, the key ingredient is the notion of an $\exists\hat{\Pi}_i^b$*–maximal model*, a hierarchical version of the well known notion of an existentially closed model. These models allow us to clarify the relationships between the considered theories and their restricted versions in a particularly simple way. Namely, if T denotes S_2^i, T_2^i or $B B\Sigma_i^b$ and T^R (resp. T^-) denotes its inference rule (resp. parameter free) version, then (see Thm. 1 and Prop. 3)

- every $\exists\hat{\Pi}_i^b$–maximal model of T^R is a model of T, and
- every theory extending T^- is closed under the corresponding inference rule and, so, every $\exists\hat{\Pi}_i^b$–maximal model of $T^-$is a model of T.

From these facts we shall derive our main results (see Theorems 3 and 4): (1) S_2^i, T_2^i and $B B\Sigma_i^b$ are $\forall\Sigma_i^b$–conservative over their inference rule versions; and (2) S_2^i, T_2^i are $\exists\forall\Sigma_i^b$–conservative over their parameter free versions. As far as we know, these results are new, and the $\exists\forall\Sigma_i^b$–conservation results for parameter free systems improve previous $\forall\Sigma_i^b$–conservation obtained in [2].

Finally, in Sect. 4 we apply the results obtained for Σ_1^b–replacement to the analysis of the Δ_1^b–bit–comprehension rule Δ_1^b–BCR. This rule was introduced in [7] to capture the complexity class TC^0 and is the final refinement of a series of theories introduced in [5,6,7] in the quest for natural theories for TC^0. In [7] it is proved that TC^0 coincides with the Σ_1^b-definable functions of the system Δ_1^b–CR given by the closure under Δ_1^b–BCR of a certain base theory; and that the Σ_0^b–replacement scheme $B B\Sigma_0^b$ is $\forall\Sigma_1^b$–conservative over Δ_1^b–CR. Here, we prove that TC^0 also coincides with the $\hat{\Sigma}_1^b$–definable functions of the (apparently) weaker system $\hat{\Delta}_1^b$–CR and reformulate this system in terms of Σ_i^b–replacement rule, obtaining as a corollary a new proof of the conservation result in [7]. Our analysis is of independent interest in view of the open problems on Δ_1^b–BCR posed in [7]; however, it also supports Johannsen–Pollett's claim on Δ_1^b–CR as a minimal natural theory for TC^0 and makes more transparent the close relationship between Δ_1^b–bit–comprehension and Σ_1^b–replacement.

2 Fragments of Bounded Arithmetic

In what follows we state some definitions and results on Bounded Arithmetic that will be used through this paper (see [3,8] for more information). The first order language of Bounded Arithmetic $\mathcal{L}_2$ comprises the usual language of Peano

Arithmetic $\{0, S, +, \cdot, \leq\}$ together with five new function symbols: $\lfloor \frac{x}{2} \rfloor$, $|x|$, $\#$, MSP and $\dot{-}$; where $\lfloor \frac{x}{2} \rfloor$ is x divided by 2 rounded down, $|x|$ is the length of x in binary notation, $x\#y$ is $2^{|x|\cdot|y|}$, $MSP(x,i)$ is $\lfloor \frac{x}{2^i} \rfloor$, and $x \dot{-} y$ is the subtraction function. As usual, we also write $x+1$ and $2^{|x|}$ for Sx and $1\#x$, respectively. Bounded formulas of $\mathcal{L}_2$ are classified in a hierarchy of sets Σ_i^b and Π_i^b by counting the alternations of bounded quantifiers ($\exists x \leq t$, $\forall x \leq t$), ignoring sharply bounded quantifiers ($\exists x \leq |t|$, $\forall x \leq |t|$).

The *induction* axiom for $\varphi(x)$, I_φ, is the formula

$$\varphi(0) \wedge \forall x\, (\varphi(x) \rightarrow \varphi(Sx)) \;\rightarrow\; \forall x\, \varphi(x)$$

The *length induction* axiom for $\varphi(x)$, $LIND_\varphi$, and the *double length induction* axiom for $\varphi(x)$, $LLIND_\varphi$, are obtained replacing the consequent of I_φ by $\forall x\, \varphi(|x|)$ and $\forall x\, \varphi(||x||)$, respectively. The *polynomial induction* axiom for $\varphi(x)$, $PIND_\varphi$, is the formula

$$\varphi(0) \wedge \forall x\, (\varphi(\lfloor \frac{x}{2} \rfloor) \rightarrow \varphi(x)) \;\rightarrow\; \forall x\, \varphi(x)$$

In all cases, $\varphi(x)$ may contain other free variables, which are called parameters. On a par with these induction axioms, we consider induction *inference rules.* The *induction rule* for $\varphi(x)$, IR, is

$$\frac{\varphi(0)\,,\; \forall x\, (\varphi(x) \rightarrow \varphi(Sx))}{\forall x\, \varphi(x)}$$

Similarly, PINDR, LINDR and LLINDR are defined.

$BASIC$ denotes a finite set of open (quantifier–free) axioms specifying the interpretations of the nonlogical symbols of $\mathcal{L}_2$. Following [7,10], our base theory will be LIOpen $= BASIC + \{LIND_\varphi \,:\, \varphi \text{ is open}\}$. As shown there, LIOpen allows for simple definitions of tuple–encoding and sequence–encoding functions. First, observe that there are $\mathcal{L}_2$–terms $Bit(x,i)$ and $LSP(x,i)$ returning the value of the bit in the 2^i position of the binary representation of x, and the number consisting of the low i bits of x, respectively. The code of a sequence $\{b_0, b_1, \ldots, b_{|s|}\}$ with all its elements less than or equal to some a is the number $w < 4(a\#2s)$ whose binary representation consists of a 1 followed by the binary representations of the elements b_i concatenated, each padded with zeroes to length $|a|$ (we shall write $bd(a,s)$ for the bounding term $4(a\#2s)$). Thus, the $\mathcal{L}_2$–term $\beta_a(w,i) := MSP(LSP(w, Si\cdot|a|), i\cdot|a|)$ returns the i–th element of such a sequence. As for tuple–encoding, pairs are coded as $\langle x,y\rangle := (B+y)\cdot 2B + (B+x)$, where $B = 2^{|max(x,y)|}$. Then there is an open formula $ispair(u)$ defining the range of the function $\langle x,y\rangle$; and there are terms $(u)_0$, $(u)_1$ returning the left and right coordinates from a coded pair (see [10] for details). Interestingly, the encoding and decoding functions are all $\mathcal{L}_2$–terms so can be used in an $\mathcal{L}_2$–formula without altering its quantifier complexity.

The theories we shall deal with are defined as follows. Let Γ be a set of formulas and let $\mathbf{E}$ denote one of the schemes: I, $PIND$, $LIND$, $LLIND$. First, the theory $\mathbf{E}\Gamma$ is LIOpen $+ \{\mathbf{E}_\varphi : \varphi \in \Gamma\}$. Second, the fragment $T + \Gamma$–$\mathbf{E}$R is the closure of T under first order logic and nested applications of the $\mathbf{E}$–rule restricted to formulas in Γ, where T is an arbitrary $\mathcal{L}_2$–theory extending LIOpen. Finally, $\mathbf{E}\Gamma^-$ is LIOpen $+ \{\mathbf{E}_\varphi : \varphi(x) \in \Gamma^-\}$, where $\varphi(x) \in \Gamma^-$ means that x is the only free variable occurring in φ.

With this terminology, the three classic families of Bounded Arithmetic theories T^i_2, S^i_2 and R^i_2 correspond to $I\Sigma^b_i$, $PIND\Sigma^b_i$ and $LLIND\Sigma^b_i$, respectively. Let us remark, however, that $\mathcal{L}_2$ differs from the language of Buss's original theories S^i_2 and T^i_2, which does not include the MSP and $\dot{-}$ symbols. In addition, Buss's theories are axiomatized over $BASIC$ instead of over LIOpen. But these facts are inessential for sufficiently strong theories since both additional functions are Σ^b_1–definable in Buss's S^1_2, and this last theory implies LIOpen.

Bounded formulas of $\mathcal{L}_2$ are also classified in a hierarchy of sets *strict* Σ^b_i ($= \hat{\Sigma}^b_i$) and *strict* Π^b_i ($= \hat{\Pi}^b_i$), where no sharply bounded quantifier is allowed to precede a quantifier that is not sharply bounded. Each Σ^b_i (resp. Π^b_i) formula is equivalent to a $\hat{\Sigma}^b_i$ (resp. $\hat{\Pi}^b_i$) formula and the $\hat{\Pi}^b_{i-1}$–replacement scheme $BB\hat{\Pi}^b_{i-1}$ is a natural theory which proves that equivalence. The *replacement* or *bounded collection* axiom for a formula $\varphi(x, y)$ and a term $t(x)$, BB_φ, is

$$\begin{aligned} &\forall x \le |s|\, \exists y \le t(x)\, \varphi(x, y) \rightarrow \\ &\qquad \exists w < bd(t^*(|s|), s)\, \forall x \le |s|\, (\beta_{t^*(|s|)}(w, x) \le t(x) \wedge \varphi(x, \beta_{t^*(|s|)})), \end{aligned}$$

where t^* denotes an $\mathcal{L}_2$–term canonically associated to t so that, provably in LIOpen, t^* is monotonic and $t \le t^*$ (see [7,10] for details).

$BB\Gamma$ is LIOpen $+ \{BB_\varphi : \varphi \in \Gamma\}$. Similarly, the inference rule versions $T + \Gamma$–BBR are defined. In [10] it is shown that every Σ^b_i formula is provably equivalent in $BB\hat{\Pi}^b_{i-1}$ to a $\hat{\Sigma}^b_i$–formula, and that $PIND\hat{\Sigma}^b_i$ implies $BB\hat{\Pi}^b_{i-1}$ ($i \ge 1$). As a consequence, the author obtains the equivalences $T^i_2 \equiv I\hat{\Sigma}^b_i$ and $S^i_2 \equiv PIND\hat{\Sigma}^b_i \equiv LIND\hat{\Sigma}^b_i$. Finally, reasoning as in the proof of result 3.2 in [5], it is easy to show that $BB\hat{\Sigma}^b_{i+1} \equiv BB\hat{\Pi}^b_i$, and $T + \hat{\Sigma}^b_{i+1}$–BBR $\equiv T + \hat{\Pi}^b_i$–BBR.

3 On $\exists\hat{\Pi}^b_i$–Maximal Models and Conservation Results

In this section we present our methods for proving conservation results. To illustrate these methods, we prove that S^i_2, T^i_2 and $BB\Sigma^b_i$ are $\forall\Sigma^b_i$–conservative over their inference rule versions; and we use these results to show that S^i_2 and T^i_2 are $\exists\forall\Sigma^b_i$–conservative over their parameter free versions. The main idea involves a basic model–theoretic argument: we show that each (countable) model of the weak theory has a $\hat{\Sigma}^b_i$*–elementary extension* to a model of the strong theory ($\mathfrak{B}$ is a Γ–elementary extension of $\mathfrak{A}$, $\mathfrak{A} \prec_\Gamma \mathfrak{B}$, if $\mathfrak{A} \subseteq \mathfrak{B}$ and, for all $\varphi(\vec{x}) \in \Gamma$ and $\vec{a} \in \mathfrak{A}$, it holds that $\mathfrak{A} \models \varphi(\vec{a}) \Longleftrightarrow \mathfrak{B} \models \varphi(\vec{a})$). The key ingredient for this construction is the notion of an $\exists\hat{\Pi}^b_i$*–maximal model* for a theory T.

Definition 1. *Let $\mathfrak{A}$ be a model of T. We say that $\mathfrak{A}$ is $\exists\hat{\Pi}^b_i$–maximal for T if, for each $\mathfrak{B} \models T$, it holds that* $\mathfrak{A} \prec_{\hat{\Sigma}^b_i} \mathfrak{B} \implies \mathfrak{A} \prec_{\exists\hat{\Pi}^b_i} \mathfrak{B}$.

This notion is a suitably modified version of the general concept of an existentially closed model. The use of similar notions to prove conservation results for arithmetic systems was presented in a general setting in J. Avigad's [1] (our work is inspired by the methods in that paper). First of all, observe that $\exists\hat{\Pi}^b_i$–maximal models do exist. The proof is an easy modification of the standard iterative argument to construct existentially closed models.

Proposition 1. *Suppose T is $\forall\exists\hat{\Pi}^b_i$–axiomatizable and $\mathfrak{A}$ is a countable model of T. Then there is $\mathfrak{B} \models T$ such that $\mathfrak{A} \prec_{\hat{\Sigma}^b_i} \mathfrak{B}$ and $\mathfrak{B}$ is $\exists\hat{\Pi}^b_i$–maximal for T.*

Next, we prove the main property of these models of interest to us: each $\exists\hat{\Pi}^b_i$–maximal model for $T + \hat{\Sigma}^b_i$–**E**R also satisfies the corresponding scheme $\mathbf{E}\hat{\Sigma}^b_i$. We first need the following result (the proof is a standard compactness argument).

Proposition 2. *Let $\mathfrak{A}$ be $\exists\hat{\Pi}^b_i$–maximal for T, $\vec{a} \in \mathfrak{A}$ and $\varphi(\vec{x}, \vec{v}) \in \hat{\Sigma}^b_i$ and let $\hat{\Pi}^b_i$–Diag($\mathfrak{A}$) denote the set of all the $\hat{\Pi}^b_i$-formulas (with parameters in $\mathfrak{A}$) valid in $\mathfrak{A}$. The following conditions are equivalent.*

1. *$\mathfrak{A} \models \forall\vec{x}\,\varphi(\vec{x}, \vec{a})$.*
2. *There is $\theta(\vec{a}, \vec{b})$ in $\hat{\Pi}^b_i$–Diag($\mathfrak{A}$) satisfying $T + \theta(\vec{a}, \vec{b}) \vdash \forall\vec{x}\,\varphi(\vec{x}, \vec{a})$.*

Theorem 1. *Let **E** denote one of the following schemes: BB, I, $PIND$, $LIND$, $LLIND$. If $\mathfrak{A}$ is $\exists\hat{\Pi}^b_i$–maximal for $T + \hat{\Sigma}^b_i$–**E**R, then $\mathfrak{A} \models \mathbf{E}\hat{\Sigma}^b_i$.*

Proof. (Collection scheme): Assume $\mathfrak{A}$ is $\exists\hat{\Pi}^b_i$–maximal for $T + \hat{\Sigma}^b_i$–BBR and $\mathfrak{A} \models \forall x \leq |s|\,\exists y \leq t\,\varphi(x, y, a)$, where $\varphi(x, y, v) \in \hat{\Sigma}^b_i$, $a \in \mathfrak{A}$ and s, t are $\mathcal{L}_2$–terms (for notational simplicity we omit the possible parameters in t, s). By Proposition 2 there are $b \in \mathfrak{A}$ and $\theta(v, u)$ in $\hat{\Pi}^b_i$ such that $\mathfrak{A} \models \theta(a, b)$, and $(T + \hat{\Sigma}^b_i\text{–BBR}) + \theta(a, b) \vdash \forall x \leq |s|\,\exists y \leq t\,\varphi(x, y, a)$. So, $T + \hat{\Sigma}^b_i\text{–BBR} \vdash \theta(v, u) \rightarrow \forall x \leq |s|\,\exists y \leq t\,\varphi(x, y, v)$. Define $\delta(x, y, v, u)$ to be $\neg\theta(v, u) \vee \varphi(x, y, v)$. Clearly, δ is $\hat{\Sigma}^b_i$ and $T + \hat{\Sigma}^b_i$–BBR proves the antecedent of the bounded collection axiom for $\delta(x, y)$. Applying $\hat{\Sigma}^b_i$–BBR in $\mathfrak{A}$ and taking $v = a$ and $u = b$, we get

$$\mathfrak{A} \models \exists w < bd(t^*(|s|), s)\,\forall x \leq |s|\,(\beta_{t^*(|s|)}(w, x) \leq t \wedge \delta(x, \beta_{t^*(|s|)}(w, x), a, b))$$

Since $\mathfrak{A} \models \theta(a, b)$, $\mathfrak{A} \models \delta(x, y, a, b) \rightarrow \varphi(x, y, a)$ and hence the consequent of the bounded collection axiom for $\delta(x, y, a)$ is true in $\mathfrak{A}$.

(Induction schemes): We only write the proof for the usual induction scheme I, the remaining cases being analogous. Assume $\mathfrak{A}$ is $\exists\hat{\Pi}^b_i$–maximal for $T + \hat{\Sigma}^b_i$–IR. To prove that $\mathfrak{A} \models I\hat{\Sigma}^b_i$, assume $\mathfrak{A} \models \varphi(0, a) \wedge \forall x\,(\varphi(x, a) \rightarrow \varphi(x + 1, a))$, where $\varphi(x, v) \in \hat{\Sigma}^b_i$ and $a \in \mathfrak{A}$. We must show $\mathfrak{A} \models \forall x\,\varphi(x, a)$. Put $\varphi(x, v)$ as $\exists y \leq t(x, v)\,\varphi_0(x, y, v)$, where $\varphi_0(x, y, v) \in \hat{\Pi}^b_{i-1}$ and $t(x, v)$ is a term. By prenex operations, the antecedent of the induction axiom for φ can be reexpressed as

$$\forall x\,\forall y\,[\varphi(0, a)\ \wedge\ (\neg(y \leq t(x, a)) \vee \neg\varphi_0(x, y, a) \vee \varphi(x + 1, a))]$$

Let us denote by $\psi(x, y, a)$ the $\hat{\Sigma}^b_i$–formula in brackets [ ] above. Since $\mathfrak{A}$ is $\exists\hat{\Pi}^b_i$–maximal for $T + \hat{\Sigma}^b_i$–IR and $\mathfrak{A} \models \forall x, y\,\psi(x, y, a)$, by Proposition 2 it follows that there are $b \in \mathfrak{A}$ and $\theta(v,u) \in \hat{\Pi}_i^b$ satisfying $\mathfrak{A} \models \theta(a,b)$, and $(T + \hat{\Sigma}_i^b\text{–IR}) + \theta(a,b) \vdash \forall x, y\, \psi(x,y,a)$. Hence,

$$T + \hat{\Sigma}_i^b\text{–IR} \vdash \theta(v,u) \;\rightarrow\; (\varphi(0,v) \wedge \forall x\,(\varphi(x,v) \rightarrow \varphi(x+1,v)))$$

Now define $\delta(x,v,u)$ to be the $\hat{\Sigma}_i^b$–formula $\neg\theta(v,u) \vee \varphi(x,v)$. Clearly, $T + \hat{\Sigma}_i^b$–IR proves the antecedent of the induction axiom for $\delta(x,v,u)$. By applying $\hat{\Sigma}_i^b$–IR, we get $\mathfrak{A} \models \forall x, v, u\, \delta(x,v,u)$. In particular, $\mathfrak{A} \models \forall x\,(\neg\theta(a,b) \vee \varphi(x,a))$, and hence $\mathfrak{A} \models \forall x\, \varphi(x,a)$ since $\theta(a,b)$ is true in $\mathfrak{A}$. □

Combining Proposition 1 and Theorem 1, we can derive our $\forall\Sigma_i^b$–conservation results. The proof is in two steps. First, we prove this conservation result only for $\forall\hat{\Sigma}_i^b$–formulas. Second, we show how to extend it to general $\forall\Sigma_i^b$–formulas.

Theorem 2. *Let* $\mathbf{E}$ *denote one of the following schemes:* BB, I, $PIND$, $LIND$, $LLIND$ *and let* T *be a* $\forall\exists\hat{\Pi}_i^b$*–axiomatizable theory. Then* $T + \mathbf{E}\hat{\Sigma}_i^b$ *is* $\forall\hat{\Sigma}_i^b$*–conservative over* $T + \hat{\Sigma}_i^b$–$\mathbf{E}$R.

Proof. By contradiction, assume $T + \mathbf{E}\hat{\Sigma}_i^b \vdash \varphi$ but $T + \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R} \nvdash \varphi$, where $\varphi \in \forall\hat{\Sigma}_i^b$. Let $\mathfrak{A}$ be a countable model of $(T + \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R}) + \neg\varphi$. Since T is $\forall\exists\hat{\Pi}_i^b$–axiomatizable, so is $T + \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R}$ (for $\mathbf{E} = BB$, recall that $\hat{\Sigma}_i^b$–BBR and $\hat{\Pi}_{i-1}^b$–BBR are equivalent rules). By Proposition 1 there is $\mathfrak{B} \models T + \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R}$ such that $\mathfrak{A} \prec_{\hat{\Sigma}_i^b} \mathfrak{B}$ and $\mathfrak{B}$ is $\exists\hat{\Pi}_i^b$–maximal for $T + \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R}$. From Theorem 1 it follows that $\mathfrak{B} \models \mathbf{E}\hat{\Sigma}_i^b$. Hence, $\mathfrak{B} \models T + \mathbf{E}\hat{\Sigma}_i^b + \neg\varphi$, which is a contradiction. □

Since T_2^i and S_2^i are $\forall\hat{\Sigma}_{i+1}^b$–axiomatizable, a first application of Theorem 2 is the following strengthening of the well known facts that S_2^{i+1} implies T_2^i and R_2^{i+1} implies S_2^i, and of theorem 68 in [10] stating that $BB\hat{\Sigma}_{i+1}^b$ implies S_2^i.

Corollary 1

1. *LIOpen* $+ \hat{\Sigma}_{i+1}^b$–LINDR *implies* T_2^i.
2. *Both LIOpen* $+ \hat{\Sigma}_{i+1}^b$–LLINDR *and LIOpen* $+ \hat{\Sigma}_{i+1}^b$–BBR *imply* S_2^i.

To extend previous conservation result to $\forall\Sigma_i^b$–formulas, we need the following lemma (the proof is by induction on the complexity of Σ_i^b–formulas).

Lemma 1. $(i \geq 1)$ *Let* $\varphi(\vec{v}) \in \Sigma_i^b$*. There exists* $\hat{\varphi}(\vec{v}) \in \hat{\Sigma}_i^b$ *such that:*
(1) $BB\hat{\Pi}_{i-1}^b \vdash \varphi(\vec{v}) \leftrightarrow \hat{\varphi}(\vec{v})$*, and* *(2)* $BB\hat{\Pi}_{i-2}^b \vdash \hat{\varphi}(\vec{v}) \rightarrow \varphi(\vec{v})$.
(For $i = 1$, $BB\hat{\Pi}_{-1}^b$ *denotes LIOpen.)*

Theorem 3. *Let* $\mathbf{E}$ *denote one of the following schemes:* BB, I, $PIND$, $LIND$*. Then, LIOpen* $+ \hat{\Sigma}_i^b$–$\mathbf{E}$R *axiomatizes the* $\forall\Sigma_i^b$*–consequences of* $\mathbf{E}\Sigma_i^b$.

Proof. Assume $\mathbf{E}\Sigma_i^b \vdash \forall\vec{v}\, \varphi(\vec{v})$, where $\varphi \in \Sigma_i^b$. Let $\hat{\varphi}(\vec{v}) \in \hat{\Sigma}_i^b$ as in Lemma 1. Since S_2^i implies $BB\Pi_{i-1}^b$ (see [3]), $\mathbf{E}\Sigma_i^b$ implies $BB\Pi_{i-1}^b$ and LIOpen $+ \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R}$ implies $BB\Pi_{i-2}^b$ by Corollary 1. Hence, $\mathbf{E}\Sigma_i^b \vdash \forall\vec{v}\, \hat{\varphi}(\vec{v})$ and LIOpen $+ \hat{\Sigma}_i^b\text{–}\mathbf{E}\text{R} \vdash \hat{\varphi}(\vec{v}) \rightarrow \varphi(\vec{v})$. So, this last theory proves $\forall\vec{v}\, \varphi(\vec{v})$ by Theorem 2. □

In what follows we deal with parameter free versions of T_2^i and S_2^i. Notice that there are two natural candidates for their parameter free counterparts: restricting the axiom scheme to parameter free Σ_i^b–formulas, or to *strict* parameter free Σ_i^b formulas. Since we are interested in conservation results over these theories, we choose the weakest ones to make the results stronger. That is, we fix $T_2^{i,-} \equiv I\hat{\Sigma}_i^{b,-}$ and $S_2^{i,-} \equiv PIND\hat{\Sigma}_i^{b,-}$. We derive the conservation theorems from our previous work on inference rules. The key observation is the following:

Proposition 3

1. *If* T *implies* $T_2^{i,-}$ *then* T *is closed under* $\hat{\Sigma}_i^b$–IR.
2. $(i \geq 1)$ *If* T *implies* $PIND\Sigma_1^{b,-} + S_2^{i,-}$ *then* T *is closed under* $\hat{\Sigma}_i^b$–PINDR.

Proof. (1): Assume T proves $\varphi(0,v) \wedge \forall x\,(\varphi(x,v) \rightarrow \varphi(x+1,v))$, where $\varphi(x,v)$ is $\hat{\Sigma}_i^b$. We must show $T \vdash \forall v\,\forall x\,\varphi(x,v)$. The idea is to codify the parameter v and the induction variable x in a single variable u using the pairing function and to apply $I\hat{\Sigma}_i^{b,-}$. To this end, define $\theta(u)$ to be the following $\hat{\Sigma}_i^b$–formula:

$$(ispair(u) \wedge (u)_0 < (u)_1 \wedge ispair((u)_1)) \rightarrow \varphi((u)_0, (u)_{1,1})$$

Trivially, $T \vdash \theta(0)$ since $\neg\, ispair(0)$. Let us see that $T \vdash \forall u\,(\theta(u) \rightarrow \theta(u+1))$. Reasoning in T, we assume $\theta(u)$ and $(ispair(u') \wedge (u')_0 < (u')_1 \wedge ispair((u')_1)$, where $u' = u+1$. We must show $\varphi((u')_0, (u')_{1,1})$.

Case 1: $(u')_0 = 0$. Then $\varphi(0, (u')_{1,1})$ since $T \vdash \forall v\,\varphi(0,v)$.

Case 2: $(u')_0 > 0$. Since $(u')_0 < (u')_1$, $max((u')_0 - 1, (u')_1) = (u')_1$ and hence by the definition of the pairing function u codifies the pair $\langle (u')_0 - 1, (u')_1 \rangle$ (that is, $(u)_0 = (u')_0 - 1$ and $(u)_1 = (u')_1$). Consequently, from $\theta(u)$ it follows $\varphi((u')_0 - 1, (u')_{1,1})$ and hence $\varphi((u')_0, (u')_{1,1})$ since $T \vdash \varphi(x,v) \rightarrow \varphi(x+1,v)$.

From the induction axiom for $\theta(u)$ (available in T since it contains $I\hat{\Sigma}_i^{b,-}$) it follows that $T \vdash \forall u\,\theta(u)$. To show $T \vdash \forall v \forall x\,\varphi(x,v)$, observe that $\varphi(x,v)$ can be inferred from $\theta(\langle x, \langle x, v\rangle\rangle)$.

(2): The proof is similar to that of *1* but now we need to define a new tuple–encoding function *compatible* with the PIND axioms: roughly speaking, if u codifies the pair $(x, \vec{v})$ and $x > 0$, then $\lfloor \frac{u}{2} \rfloor$ must codify the pair $(\lfloor \frac{x}{2} \rfloor, \vec{v})$. In [2] Bloch proposed the following encoding function satisfying that property:

$$[x,v,z] = u \;\equiv\; \begin{cases} |v| < z^2 \leq |u| < (z+1)^2 \wedge \\ u = Concat(v + 2^{min(z^2,|u|)}, x + 2^{|x|}) \end{cases}$$

where $Concat(x,y) = x \cdot 2^{|y| \dot{-} 1} + y \dot{-} 2^{|y| \dot{-} 1}$. In words, we pad v to length z^2 and concatenate the result with x (notice that the $Concat$ function operates on bit–strings rather than on binary numbers, that is, $Concat(1x, 1y) = 1xy$). Observe that the encoding function $[x,v,z]$ itself is not total, but it is total for all z sufficiently large. Namely, as shown in [2], $PIND\Sigma_1^{b,-}$ proves:

(a) $|x| \leq 2z \wedge |v| < z^2 \;\rightarrow\; \exists! u\,([x,v,z] = u)$,
(b) $u > 0 \;\rightarrow\; \exists! x, v, z \leq u\,([x,v,z] = u)$, and
(c) $u = [x,v,z] \wedge x > 0 \;\rightarrow\; \lfloor \frac{u}{2} \rfloor = [\lfloor \frac{x}{2} \rfloor, v, z]$

Equipped with this encoding function, we can infer the PIND axiom for the $\hat{\Sigma}^b_i$–formula $\varphi(x,v)$ from the PIND axiom for the (parameter free) $\hat{\Sigma}^b_i$–formula $\theta(u) \equiv u > 0 \rightarrow \exists x, v, z \leq u\, ([x,v,z] = u \wedge \varphi(x,v))$. □

Observe that from the previous result and Theorem 2 it immediately follows that $T_2^{i+1,-}$ implies T^i_2 and that $PIND\Sigma^{b,-}_1 + S^{i+1,-}_2$ implies S^i_2.

Theorem 4

1. *T^i_2 is $\exists\forall\Sigma^b_i$–conservative over $T^{i,-}_2$.*
2. *$(i \geq 1)$ S^i_2 is $\exists\forall\Sigma^b_i$–conservative over $PIND\Sigma^{b,-}_1 + S^{i,-}_2$.*

Proof. Using Lemma 1 as in Theorem 3, it suffices to show $\exists\forall\hat{\Sigma}^b_i$–conservation. We only write the proof of 1. Assume φ is an $\exists\forall\hat{\Sigma}^b_i$–sentence such that $T^i_2 \vdash \varphi$ but $T^{i,-}_2 \nvdash \varphi$. Then $T = T^{i,-}_2 + \neg\varphi$ is consistent and $\forall\exists\hat{\Pi}^b_i$–axiomatizable. Let $\mathfrak{A}$ be an $\exists\hat{\Pi}^b_i$–maximal model for T. By Proposition 3, T is closed under $\hat{\Sigma}^b_i$–IR. Hence, $\mathfrak{A} \models T + T^i_2$ by Theorem 1. So, $\mathfrak{A} \models T^i_2 + \neg\varphi$, which is a contradiction. □

As for parameter free $BB\Sigma^b_i$, we can prove that $BB\Sigma^b_i$ is $\exists\forall\Sigma^b_i$–conservative over $UBB\hat{\Sigma}^b_i$ as in Theorem 4 (UBB_φ is obtained quantifying universally the parameters of $\varphi(x,y)$ in both the antecedent and the consequent of BB_φ).

4 On Replacement and Bit–Comprehension Rules

In this section we shall study an inference rule closely tied to Σ^b_1–replacement: Δ^b_1–bit–comprehension rule. This rule was defined in [7] as follows:

$$\Delta^b_1\text{–BCR}: \quad \frac{\varphi(x) \leftrightarrow \psi(x)}{\exists y < 2^{|u|}\, \forall x < |u|\, (Bit(y,x) = 1 \leftrightarrow \varphi(x))}$$

where $\varphi(x) \in \Sigma^b_1$ and $\psi(x) \in \Pi^b_1$. In [7], it is proved that $BB\Sigma^b_0$ (denoted there by C^0_2) is a $\forall\Sigma^b_1$–conservative extension of Δ^b_1–CR (the theory LIOpen $+ \Delta^b_1$–BCR). So, in view of Theorem 3, it is natural to investigate the relationship between Δ^b_1–BCR and $\hat{\Sigma}^b_1$–BBR. In this section, we consider the apparently weaker rule for *strict* formulas $\hat{\Delta}^b_1$–BCR and show that LIOpen + $\hat{\Delta}^b_1$–BCR (denoted in what follows by $\hat{\Delta}^b_1$–CR) is equivalent to LIOpen $+ \hat{\Sigma}^b_1$–BBR, see Theorem 5. In fact, over LIOpen, the four rules Σ^b_1–BBR, $\hat{\Sigma}^b_1$–BBR, Δ^b_1–BCR and $\hat{\Delta}^b_1$–BCR are equivalent and, by Theorem 3, provide axiomatizations of the $\forall\Sigma^b_1$–consequences of $BB\Sigma^b_1$. Moreover, in [9], answering a question posed in [7], it is shown that Δ^b_1–CR is finitely axiomatizable; so, a finite number of *nested* applications of any of the rules above axiomatizes the $\forall\Sigma^b_1$–consequences of $BB\Sigma^b_1$. However,

Problem 1. *Is LIOpen$+\hat{\Sigma}^b_1$–BBR equivalent to [LIOpen; $\hat{\Sigma}^b_1$–BBR], the closure of LIOpen under first order logic and* unnested *applications of $\hat{\Sigma}^b_1$–BBR?*

Our work suggests a positive answer to Problem 1 since this is the case for the analogous problem for collection rule in the usual language of Peano Arithmetic.

Now we prove that LIOpen $+ \hat{\Sigma}^b_1$–BBR and $\hat{\Delta}^b_1$–CR are equivalent. Our work also provides a new proof of Johannsen–Pollett's conservation theorem.

Firstly, observe that it can be easily shown that LIOpen $+ \hat{\Sigma}^b_1$–BBR is closed under $\hat{\Delta}^b_1$-BCR. On the other hand, since C^0_2 coincides with $BB\Sigma^b_0$, by Theorem 2, C^0_2 is $\forall\hat{\Sigma}^b_1$ conservative over LIOpen$+\hat{\Sigma}^b_1$–BBR. So, in order to simultaneously get the equivalence of LIOpen $+ \hat{\Sigma}^b_1$–BBR and $\hat{\Delta}^b_1$–CR, and Johannsen–Pollett's theorem, it suffices to prove that $\hat{\Delta}^b_1$–CR is closed under $\hat{\Sigma}^b_1$–BBR. Next two lemmas are the key ingredients of the proof. The first one provides a weak form of replacement available in $\hat{\Delta}^b_1$–CR (the proof is straightforward and we omit it). The second one is a selection (or witnessing) principle for $\hat{\Delta}^b_1$–CR.

Lemma 2. *Let $\varphi(x, y) \in \hat{\Sigma}^b_1$ such that $\hat{\Delta}^b_1$–CR $\vdash \forall x \leq |s| \, \exists! y \leq t \, \varphi(x, y)$. Then*

$$\hat{\Delta}^b_1\text{–}CR \vdash \exists w < bd(t^*(|s|), s) \, \forall x \leq |s| \, (\beta_{t^*(|s|)}(w, x) \leq t \wedge \varphi(x, \beta_{t^*(|s|)}(w, x)))$$

Lemma 3. *Let $\varphi(x, y) \in \hat{\Sigma}^b_1$ such that $\hat{\Delta}^b_1$–CR $\vdash \forall x \, \exists y \leq t \, \varphi(x, y)$, then there exists $\psi(x, y) \in \hat{\Sigma}^b_1$ such that $\hat{\Delta}^b_1$–CR proves*

$$(1) \quad \forall x \, \exists! y \leq t \, \psi(x, y), \qquad \textit{and} \qquad (2) \quad \forall x \, \forall y \, (\psi(x, y) \rightarrow \varphi(x, y))$$

Proof. (Sketch) The proof we present here leans upon an analysis of the class of $\hat{\Sigma}^b_1$–definable functions $\hat{\Delta}^b_1$–CR. We refine corollary 1 in [7] and prove that TC^0 is the class of $\hat{\Sigma}^b_1$–definable functions of $\hat{\Delta}^b_1$–CR. The basic result is a machine–independent characterization of TC^0 given by Clote and Takeuti in [4]:

Let BF be the set of basic functions $\{o,\, s_0,\, s_1,\, \#,\, \times,\, |\cdot|\} \cup \{\Pi^n_i :\ 1 \leq i \leq n\}$, where $o(x) = 0$, $s_0(s) = 2x$, $s_1(x) = 2x + 1$, $|x| = \lceil log_2(x + 1)\rceil$, $x\#y = 2^{|x|\cdot|y|}$, $\times$ denotes the usual product and $\Pi^n_i(x_1, \ldots, x_n) = x_i$.

Given $g : \omega^n \rightarrow \omega$ and $h_0, h_1 : \omega^{n+1} \rightarrow \{0, 1\}$, a function f is defined by *concatenation recursion on notation* (CRN) from g, h_0 and h_1 if

$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}) \\ f(2n, \vec{x}) &= 2 \cdot f(n, \vec{x}) + h_0(n, \vec{x}), \quad \text{provided } n \neq 0 \\ f(2n + 1, \vec{x}) &= 2 \cdot f(n, \vec{x}) + h_1(n, \vec{x}) \end{aligned}$$

Clote and Takeuti proved that TC^0 is the smallest class of functions containing BF and closed under composition and CRN. In order to show that every function in TC^0 is $\hat{\Sigma}^b_1$–definable in $\hat{\Delta}^b_1$–CR, we show a stronger technical result:

For each function $f \in TC^0$ there exist a formula $\psi(\vec{x}, y, z_1, \ldots, z_n) \in \Sigma^b_0$ and terms $t(x), t_1(x, y), t_2(x, y, z_1), \ldots, t_n(x, y, z_1, \ldots, z_{n-1})$ such that the $\hat{\Sigma}^b_1$–formula $\exists z_1 \leq t_1 \ldots \exists z_n \leq t_n \, \psi(x, y, \vec{z})$ defines f in the standard model and

$$\text{LIOpen} + \hat{\Delta}^b_1\text{–BCR} \vdash \forall \vec{x} \, \exists! y \leq t \, \exists! z_1 \leq t_1 \ldots \exists! z_n \leq t_n \, \psi(x, y, \vec{z})$$

The proof proceeds by induction, using Clote–Takeuti's characterization of TC^0. The claim obviously holds for the basic functions and for f defined by composition from functions verifying the claim. So it suffices to prove the result for functions defined by CRN and this can be done as in theorem 4 in [7].

It is not difficult to verify that, if $f \in TC^0$ is defined by CRN from g, h_0 and h_1 then the proof of the previous technical result provide $\hat{\Sigma}^b_1$–formulas defining the functions involved and such that the relations stated by the recursion equations of CRN can be proved in $\hat{\Delta}^b_1$–CR. Bearing this fact in mind, we can introduce a universally axiomatized and conservative extension of $\hat{\Delta}^b_1$–CR, denoted by

CRNA. This universal theory can be defined in such a way that the functions in TC^0 are defined by terms of CRNA. In this way we can prove that every function $\hat{\Sigma}_1^b$–definable in $\hat{\Delta}_1^b$–CR is in TC^0 by a typical application of Herbrand's theorem. The whole argument is very similar to the Herbrand analyses of S_2^i developed by W. Sieg in [11].

Finally, we derive Lemma 3 from Herbrand's theorem applied to CRNA. $\square$

Theorem 5. *The theories $\hat{\Delta}_1^b$–CR and LIOpen + Σ_1^b–BBR are equivalent and axiomatize the class of the $\forall\Sigma_1^b$–consequences of C_2^0.*

Proof. Observe that C_2^0 extends LIOpen + Σ_1^b–BBR, which in turn extends $\hat{\Delta}_1^b$–CR; so, since LIOpen + Σ_1^b–BBR is $\forall\Sigma_1^b$–axiomatized, it suffices to prove that C_2^0 is $\forall\Sigma_1^b$–conservative over $\hat{\Delta}_1^b$–CR. Finally, by Theorem 3 it is enough to show that $\hat{\Delta}_1^b$–CR is closed under $\hat{\Sigma}_1^b$–BBR. Let us work in $\hat{\Delta}_1^b$–CR.

Let $\varphi(x,y) \in \hat{\Sigma}_1^b$ and t, s be terms such that $\forall x \leq |s|\, \exists y \leq t\, \varphi(x,y)$. Define $\theta(x,y) \in \hat{\Sigma}_1^b$ to be $(x > |s| \wedge y = 0) \vee (x \leq |s| \wedge \varphi(x,y))$. Then $\forall x\, \exists y \leq t\, \theta(x,y)$ and, by Lemma 3, there is $\psi(x,y) \in \hat{\Sigma}_1^b$ such that (1) $\forall x\, \exists! y \leq t\, \psi(x,y)$, and (2) $\forall x\, \forall y\, (\psi(x,y) \rightarrow \theta(x,y))$. By (1) and Lemma 2, it holds that

$$\exists w < bd(t^*(|s|), s)\, \forall x \leq |s|\, (\beta_{t^*(|s|)}(w,x) \leq t \wedge \psi(x, \beta_{t^*(|s|)}(w,x)))$$

Hence, $\exists w < bd(t^*(|s|), s)\, \forall x \leq |s|\, (\beta_{t^*(|s|)}(w,x) \leq t \wedge \varphi(x, \beta_{t^*(|s|)}(w,x)))$, since, by (2) and the definition of θ, we have $x \leq |s| \wedge \psi(x,y) \rightarrow \varphi(x,y)$. $\square$

Corollary 2. *(Johannsen–Pollett) C_2^0 is $\forall\Sigma_1^b$–conservative over Δ_1^b–CR.*

References

1. Avigad, J.: Saturated models of universal theories. Annals of Pure and Applied Logic 118, 219–234 (2002)
2. Bloch, S.: Divide and Conquer in Parallel Complexity and Proof Theory, Ph. D. Thesis. University of California, San Diego (1992)
3. Buss, S.: Bounded Arithmetic. Bibliopolis, Napoli (1986)
4. Clote, P., Takeuti, G.: First order bounded artihmetic and small boolean circuit complexity classes. In: Clote, P., Remmel, J (eds.) Feasible Mathematics II, pp. 154–218. Birkhäuser, Boston (1995)
5. Johannsen, J.: A Bounded Arithmetic Theory for Constant Depth Threshold Circuits. In: Gödel'96. Lecture Notes in Logic, vol. 6, pp. 224–234. Springer, Heidelberg (1996)
6. Johannsen, J., Pollett, C.: On Proofs About Threshold Circuits and Counting Hierarchies. In: Proc. 13th IEEE Symposium on Logic in Computer Science (1998)
7. Johannsen, J., Pollett, C.: On the Δ_1^b–Bit–Comprehension Rule. Logic Colloquium'98. Lecture Notes in Logic, ASL 13, 262–279 (2000)
8. Krajíček, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. Cambridge University Press, Cambridge (1995)
9. Nguyen, P., Cook, S.: Theories for TC^0 and other small complexity classes. Logical Methods in Computer Science 2, 1–40 (2006)
10. Pollett, C.: Structure and Definability in General Bounded Arithmetic Theories. Annals of Pure and Applied Logic 100, 189–245 (1999)
11. Sieg, W.: Herbrand Analyses. Archive for Mathematical Logic 30, 409–441 (1991)

The New Promise of Analog Computation

José Félix Costa[1,2,⋆], Bruno Loff[2], and Jerzy Mycka[3]

[1] Department of Mathematics, Instituto Superior Técnico
Universidade Técnica de Lisboa
Lisboa, Portugal
fgc@math.ist.utl.pt

[2] Centro de Matemática e Aplicações Fundamentais do Complexo Interdisciplinar
Universidade de Lisboa
Lisbon, Portugal
bruno.loff@gmail.com

[3] Institute of Mathematics,
University of Maria Curie-Skłodowska
Lublin, Poland
Jerzy.Mycka@umcs.lublin.pl

Abstract. We show that, using our more or less established framework of *inductive definition of real-valued functions* (work started by Cristopher Moore in [9]) together with ideas and concepts of standard computability we can prove theorems of Analysis. Then we will consider our ideas as a bridging tool between the standard *Theory of Computability (and Complexity)* on one side and *Mathematical Analysis* on the other, making real recursive functions a possible branch of *Descriptive Set Theory*. What follows is an Extended Abstract directed to a large audience of CiE 2007, Special Session on Logic and New Paradigms of Computability. (Proofs of statements can be found in a detailed long paper at the address http://fgc.math.ist.utl.pt/papers/hierarchy.pdf.)

1 Statement of the Conjecture and Its Solution

Consider a class of real-valued functions closed under the operations of composition, of finding the solution to a first order differential equation and the taking of an infinite limit. Thinking briefly about the last two operations, one may observe that they seem to be related. For instance,

$$\exp(x) = \lim_{y\to\infty} (1 + \frac{x}{y})^y,$$

and also

$$\exp(0) = 1, \ \partial_y \exp(y) = \exp(y).$$

The number π can be expressed by a differential equation that gives arctan, since $\pi = 4\arctan(1)$, and we also know, e.g., that

$$\pi = \lim_{y\to\infty} \frac{2^{4y+1} y!^4}{(2y+1)(2y)!^2}.$$

⋆ Corresponding author.

Many other examples may lead us to wonder if this property is universal, i.e., if we can replace the taking of an infinite limit of a function f by the solution of a first order differential equation involving functions no more complex than f. We may also wonder if there is a limit of definability in Analysis, e.g., to know if via limits we can always define new functions or else if all functions can be defined using an upper bound in the number of limit taking.

We will use the toolbox of computability theory to show that while we can always express the solution of a first order differential equation through infinite limits, we cannot always do the opposite.

2 The Model of Recursive Real-Valued Functions

In a sequence of papers, starting with Cristopher Moore's seminal paper [9], we have established a robust framework to think about a theory of definability of real-valued functions. This theory covers a large spectrum of functions from classes of recursive functions extended to the real numbers to the characteristic functions of predicates of the Analytic Hierarchy.

An in-depth overview of the achievements of this theory can be studied in our reference papers [4,11,12,14,8,15] together with a most recent one by Bruno Loff (see [7]) submitted to this Conference, and [2,3] for other, no less relevant contributions (and the new trend represented by several recent papers by Olivier Bournez, Manuel Campagnolo, Daniel Graça and Emmanuel Hainry).

In the original paper by Cristopher Moore, the key idea we acknowledge nowadays (among all motivations that such a paper provided) is the replacement of the standard recurrence scheme for recursive functions by the so-called *differential recursion scheme*. In its simplest form (removing the vector formulation) this scheme reads as follows: the $(n+1)$-ary function h is defined from a n-ary function f and a $(n+2)$-ary function g

$$h(x_1,\ldots,x_n,0) = f(x_1,\ldots,x_n),$$

$$\partial_y h(x_1,\ldots,x_n,y) = g(x_1,\ldots,x_n,y,h(x_1,\ldots,x_n,y)),$$

as the solution of this first-order differential equation, if some conditions hold, where x_1, ..., x_n are the *parameters*, y is the *recurrence variable*, and the last variable of g stands for the *transport variable*.

In 2004, we introduced in [12] the *limit scheme* as a replacement for classical *minimalization*: the n-ary function h is defined from a $(n+1)$-ary function f via infinite limit taking

$$h(x_1,\ldots,x_n) = \lim_{y\to\infty} f(x_1,\ldots,x_n,y).$$

The definition of *Real Recursive Functions* runs now semi-formally as follows:

Definition 2.1. *The class of* Real Recursive Functions, *$REC(\mathbb{R})$ for short, is the smallest class of real-valued functions which contains some constants (-1, 0 and 1 suffice) and the standard projections and which is closed under composition, differential recursion and the taking of infinite limits.*

By now, many readers know the nice starting examples which enchant our eyes due to their simplicity, such as

$$h(x,0)=x,\ \partial_y h(x,y)=1,$$

having the function of addition $\lambda xy.\ x+y$ as solution, or

$$h(x,0)=0,\ \partial_y h(x,y)=x,$$

which gives $\lambda xy.\ x\times y$, and

$$h(0)=1,\ \partial_y h(y)=h(y),$$

resulting in the *exponential*.

Another class of interesting examples uses infinite limits:

$$\delta(x)=\lim_{y\to\infty}(\frac{1}{x^2+1})^y$$

which is Kronecker's δ function over the reals, or

$$sgn(x)=\lim_{y\to\infty}\ \frac{arctan(xy)}{\frac{\pi}{2}}$$

which is the *signal* function, or

$$\Theta(x)=\frac{\delta(x)+sgn(x)+1}{2},$$

the Heaviside function defined by composition.

A real recursive number in our framework is the value of real recursive function on a basic constant like 0. Notice that the class of real recursive functions is countable infinite, thus the set of real recursive numbers is also countable. It turns out that a number $y=h(x)$, where x is a previously defined real recursive number and h some real recursive function, is also real recursive. E.g., Neper's e is given by $\exp(1)$ and π is given by $4\arctan(1)$. Numbers can be thought of as entire computable structures, indivisible entities [9], or computable by digits (as in the classical way), using continued fractions.

Let us add at this point that theory of real recursive functions is intended to be more analytic in its form than the well-known approach of computable analysis. However, on some levels these theories coincide (see [1]).

We tried to show that our framework is versatile: from a careful and not so complex definition of the (countable) set of recursive functions over the reals we show by means of the toolbox of Analysis that: (a) Laplace transform can be used to quickly obtain useful real recursive functions and to measure their rate of growth, (b) the embedding of Turing machines into continuous time recursive functions is trivial (take a look at the newest definition in [15]), (c) a (limit) hierarchy of real recursive functions exist to classify hardness of functions.

The fact that the set of real recursive functions is countable gives us a possibility to consider decidability questions for these functions. For example it has been proved in [14] that for a real recursive function the problem of its domain is undecidable and the identity of two real recursive functions cannot be determined by any real recursive function.

Let us stop here to study a bit further the mentioned hierarchy of real recursive functions. If $\eta(f)$ counts the smallest rank of the limit operator — the number of nested limits — in every description of a function f, then we can define the following hierarchy of sets:

$$H_i = \{f : \eta(f) \leq i\}.$$

In [12] we established the results that follow.

Proposition 2.1. *The functions* $+$, $\times$, $-$, exp, sin, cos, log *(*inter alia*) are in* H_0*, Kronecker's* δ *function, the function sgn, and Heaviside's* θ *function (*inter alia*) are in* H_1*. Euler's* Γ *function and Riemann's* ζ *function are in* H_1*.*

We can add separation results such as:

Proposition 2.2. $H_0 \neq H_1$ *(since Euler's* Γ *function and Riemann's* ζ *function are in* H_1 *and not in* H_0*.*

About this η-hierarchy (of limits), we may add further topics. We showed in [12] that we can embed the entire arithmetical hierarchy within the limit hierarchy up to some finite level (up to a finite number of limit operations), where the analytic hierarchy starts. The use of limits gives rise to uncomputable functions, e.g., at some level we get the halting problem solved.

This means, *inter alia*, that strong uncompressible numbers like Chaitin's halting probability are found in very precise levels of the limit hierarchy.

Proposition 2.3. *The classical* halting problem *is decidable in some level (*H_3*) of the* η*-hierarchy. Chaitin's* Ω *is a real recursive constant. The Arithmetical Hierarchy is confined to a finite level of the* η*-hierarchy (*H_6*, where the Analytical Hierarchy starts).*

We can prove that the $(H_i)_{i \in \mathbb{N}}$ does not collapse (the full proof can be found in the submitted paper [8]) and contains the whole Arithmetical Hierarchy and the whole Analytical Hierarchy. In fact, Bruno Loff proved in [7] the following most interesting characterization (interesting both for real recursive functions, and for the analytical hierarchy — the later becoming defined without quantifiers and in a single inductive step):

Proposition 2.4. *Real recursive functions are those functions* f *such that the predicate expression* $y = f(x)$ *is in* Δ^1_ω*.*

To these previous aspects, we should add the impact of a further one: (d) in the basis of the limit hierarchy we can still find a set of functions over the real numbers indeed computable by physical means, theoretically by Claude Shannon's General Purpose Analog Computer and practically by the Differential

Analyzer of Vannevar Bush (see [5]). Hence, in H_0 we have truly computable functions in the physical sense (and also in the sense of computable analysis). Is the GPAC the ultimate limit of analog computability? Nobody really knows, but we can add that Rubel improved the GPAC in the 90's building up the conceptual Extended Analog Computer, in a such a way that some limits become physically realizable.

3 Proof Methods

In the full version of this paper we prove that if we have a first order differential equation that gives us some function, we can always find an infinite limit that describes the same function, using a numerical approximation which asymptotically behaves in the intended manner. Nonetheless, given a function expressed by an infinite limit, we cannot always find a first order differential equation that results in the same function, because if we could, the η-hierarchy would collapse.

We finish our extended abstract by describing how such a statement can be proved. First, we show that

Proposition 3.1. *There is no universal real recursive function, i.e., there is no real recursive binary scalar function Ψ such that, for all $n \in \mathbb{N}$, $x \in \mathbb{R}$, $\Psi(n, x) = \phi_n(x)$, where ϕ_0, ϕ_1, ϕ_2, ... denotes an enumeration of all real recursive functions: ϕ_n is the function given by a description coded by n.*

Furthermore, there is no universal real recursive function ψ which verifies $\psi(n, x) = \phi_n(x)$ if n codes for a description with the smallest possible rank of the limit operators for the described function.

Finally, we prove that

Proposition 3.2. *There is a universal real recursive function for each level of the η-hierarchy, i.e., for every level H_n of the η-hierarchy, there is a real recursive binary function Ψ_n such that whenever the number of nested limits in a description e is less than n, we have $\Psi_n(e, x) = \phi_e(x)$.*

These statements taken together prove that the η-hierarchy does not collapse. The function Ψ_n is most probably not in H_n, but it suffices to show that it exists in a higher level of the η-hierarchy.

We conclude that there is no real recursive universal Ψ function, nor even a restriction of Ψ to low-rank codes, but that there are real recursive universal Ψ_n functions for every level of the η-hierarchy. This assures that while we cannot have real recursive characteristics for the problems of domain and identity for every function, we can still have them for every function up to any level of the η-hierarchy. Based on these two statements we prove the main theorem:

Theorem 3.1. *There is no limit to inductive definability of real-valued functions by composition, solving first-order differential equations and infinite limit taking.*

This result makes us feel that our framework can be considered a branch of *Descriptive Set Theory*. For the purpose we recall some words of Yiannis N. Moschovakis (see [10]): *Lebesgue defined the collection of analytically representable functions as the smallest set which contains all constants and projections and which is closed under sums, products and the taking of limits. [...] Today we recognize Lebesgue [1905] [see [6]] as a classical work in the theory of definability. It introduced and studied systematically several natural notions of definable functions and sets and it established the first important hierarchy theorems and structure results for collections of definable objects.* So do we! How close is real recursive function theory to *Descriptive Set Theory*? We do not know, and the answer to this question is an open problem in our research program.

What about connections between *Mathematical Analysis* and *Theory of Computability (and Complexity))* in the other direction? We believe that our most general framework, with infinite limits ([12,15,8]), has enough ingredients to allow a good translation of classical computability and classical computational complexity problems into Analysis. We do believe that such translations might be a solution to open problems described in analytic terms: we are much involved in the definition of analog classes P and NP, and to find one good analytic representation of the $P \neq NP$ conjecture (see [13]).

Acknowledgements. We would like to thank Fernando Ferreira for having compelled us a bit further towards *Descriptive Set Theory*. Although the title mentions analog computation, our theory is much more about recursive functions over the reals. This change of view is due to Martin Davis who has been systematically disturbed by our proposals: to him we have much to thank.

References

1. Bournez, O., Campagnolo, M., Graça, D., Hainry, E.: The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 631–643. Springer, Heidelberg (2006)
2. Bournez, O., Hainry, E.: Real recursive functions and real extensions of recursive functions. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 116–127. Springer, Heidelberg (2004)
3. Bournez, O., Hainry, E.: Elementarily computable functions over the real numbers and $\mathbb{R}$-sub-recursive functions. Theoretical Computer Science 348(2–3), 130–147 (2005)
4. Campagnolo, M., Moore, C., Costa, J.F.: An analog characterization of the Grzegorczyk hierarchy. Journal of Complexity 18(4), 977–1000 (2002)
5. Graça, D., Costa, J.F.: Analog computers and recursive functions over the reals. Journal of Complexity 19(5), 644–664 (2003)
6. Lebesgue, H.: Sur les fonctions représentables analytiquement. J. de Math. 1, 139–216 (1905)
7. Loff, B.: A functional characterisation of the analytical hierarchy, (submitted 2007)
8. Loff, B., Costa, J.F., Mycka, J.: Computability on reals, infinite limits and differential equations, (accepted for publication 2006)

9. Moore, C.: Recursion theory on the reals and continuous-time computation. Theoretical Computer Science 162(1), 23–44 (1996)
10. Moschovakis, Y.N.: Descriptive set theory. North–Holland, Amsterdam (1980)
11. Mycka, J.: μ-recursion and infinite limits. Theoretical Computer Science 302, 123–133 (2003)
12. Mycka, J., Costa, J.F.: Real recursive functions and their hierarchy. Journal of Complexity 20(6), 835–857 (2004)
13. Mycka, J., Costa, J.F.: The $P \neq NP$ conjecture in the context of real and complex analysis. Journal of Complexity 22(2), 287–303 (2006)
14. Mycka, J., Costa, J.F.: Undecidability over continuous-time. In: Logic Journal of the IGPL, vol. 14(5), pp. 649–658. Oxford University Press, Oxford (2006)
15. Mycka, J., Costa, J.F.: A new conceptual framework for analog computation. Theoretical Computer Science, Accepted for publication (2007)

Comparing C.E. Sets Based on Their Settling Times

Barbara F. Csima[⋆]

Department of Pure Mathematics
University of Waterloo
Waterloo, ON, Canada N2L 3G1
csima@math.uwaterloo.ca
www.math.uwaterloo.ca/~csima

Abstract. To each computable enumerable (c.e.) set A with a particular enumeration $\{A_s\}_{s\in\omega}$, there is associated a settling function $m_A(x)$, where $m_A(x)$ is the last stage when a number less than or equal to x was enumerated into A. In [7], R.W. Robinson classified the complexity of c.e. sets into two groups of complexity based on whether or not the settling function was dominant. An extension of this idea to a more refined ordering of c.e. sets was first introduced by Nabutovsky and Weinberger in [6] and Soare [9], for application to differential geometry. There they defined one c.e. set A to settling time dominate another c.e. set B ($B >_{\mathrm{st}} A$) if for every computable function f, for all but finitely many x, $m_B(x) > f(m_A(x))$. In [4] Csima and Soare introduced a stronger ordering, where $B >_{\mathrm{sst}} A$ if for all computable f and g, for almost all x, $m_B(x) > f(m_A(g(x)))$. We give a survey of the known results about these orderings, make some observations, and outline the open questions.

1 Introduction

An integral part of a computably enumerable (c.e.) set is it's enumeration. Obviously, a c.e. set has more than one enumeration, so when attempting to compare c.e. sets based on their settling times, it might be more correct to compare particular enumerations of the sets. It turns out that the notions introduced by Robinson, Nabutovsky, Weinberger and Soare, are all enumeration independent, so it is possible to compare c.e. sets, not just particular enumerations.

For Computability Theory, we follow the notation of Soare's *Recursively Enumerable Sets and Degrees* [8] and new notation from Soare's *Computability Theory and Applications* [10], which we also define. See also Cooper's *Computability Theory* [1] for a modern treatment of the subject.

We let $\{W_{e,s}\}_{e,s\in\omega}$ be any standard enumeration of the c.e. sets. We write "$\forall^\infty x$" for "for all but finitely many x".

We first recall the definition of a dominant function.

[⋆] Partially supported by Canadian NSERC Discovery Grant 312501.

Definition 1. *A function g is said to be* dominant *if for every computable function f, $(\forall^\infty x)[g(x) > f(x)]$.*

We also recall Martin's characterization of high sets in terms of dominant functions.

Theorem 1 (Martin [5]). *A set A is high ($A' \equiv_T \emptyset''$) if and only if there exists an A-computable dominant function.*

We now give the definition of the settling function associated to an enumeration of a c.e. set.

Definition 2. *For every computably enumerable (c.e.) set W_e and associated enumeration $\{W_{e,s}\}_{s\in\omega}$, we define the* settling (or modulus) function*: $m_e(x) = (\mu s)[W_{e,s} \upharpoonright\upharpoonright x = W_e \upharpoonright\upharpoonright x]$ where $A \upharpoonright\upharpoonright x = \{y \le x \mid y \in A\}$.*

In [7], Robinson sorted c.e. sets into two groups depending on whether or not their settling functions were dominant.

Definition 3. *For every c.e. set W_e, we say that W_e is* dominant *if its settling function is; that is if*

$$(\forall \textit{ computable } f)(\forall^\infty x)[m_e(x) > f(x)]$$

Robinson referred to dominant sets as "high" and non-dominant sets as "low"; we will avoid this terminology due to the obvious confusion it would cause with the currently prevalent meaning of these words. Robinson showed that if a set has one enumeration that is dominant, then all are dominant, so the definition is independent of the enumeration chosen.

For use in an application to differential geometry, Nabutovsky and Weinberger [6] and Soare [9] introduced the following ordering on c.e. sets based on their settling times.

Definition 4. *For c.e. sets A and B, we say A* settling time dominates *B and write $A >_{st} B$ iff $(\exists W_i = A)$ $(\exists W_j = B)$*

$$(\forall \textit{ computable } f)(\forall^\infty)[\ m_i(x) \ > \ f(m_j(x))\ ]. \tag{1}$$

Andre Nies showed that this is equivalent to $(\forall W_i = A)(\forall W_j = B)$ [(1) holds]. We denote the structure of the c.e. sets with the relation $<_{\text{st}}$ as $\mathcal{E}_{\text{st}}$.

Nabutovsky and Weinberger wanted a descending sequence in $\mathcal{E}_{\text{st}}$ to use in the construction of various manifolds that gave information on the geometry of Riemannian metrics modulo diffeomorphisms. Soare constructed such a sequence as described in [9]. See Soare [9], Csima and Soare [4], and Weinberger [11], for general background information on the ordering and its applications to differential geometry.

Though not designed for that purpose, the ordering $<_{\text{st}}$ gives a natural extension of the idea of domination to an ordering on c.e. sets. Indeed, we make the following observation.

Observation 2. *For a c.e. set A, the following are equivalent:*

(i) A is dominant.
(ii) There exists an infinite computable set C such that $A >_{\rm st} C$.
(iii) $A >_{\rm st} C$ for every computable set C.

Proof. Clearly *(iii)* $\Rightarrow$ *(ii)*.

For *(i)* $\Rightarrow$ *(iii)*, suppose A is a dominant c.e. set, and that C is any computable set. Then m_C is computable, so for any computable function f, $f \circ m_C$ is computable. Thus since A is dominant, $(\forall^\infty x)[m_A(x) > f(m_C(x))]$, and so $A >_{\rm st} C$.

For *(ii)* $\Rightarrow$ *(i)*, suppose that A settling time dominates an infinite computable set C. Since C is infinite and computable, then it has an enumeration such that m_C is computable and such that $m_C(x) \geq x$ for almost every x. Let f be any computable function. Let f^* be a non-decreasing computable function such that $f^*(x) \geq f(x)$ for all x. Then since $A >_{\rm st} C$, and since f^* is computable, $(\forall^\infty x)[m_A(x) \geq f^*(m_C(x))]$. Since f^* is non-decreasing and since $(\forall^\infty x)[m_C(x) \geq x]$, we have $(\forall^\infty x)[m_A(x) \geq f^*(m_C(x)) \geq f^*(x) \geq f(x)]$. Since f was arbitrary, this shows that A is dominant.

Along similar lines, Csima and Soare observed the following.

Observation 3 (Csima, Soare [4]). *If $A >_{\rm st} B$ and B is infinite then A is dominant.*

Just as Turing reducibility refines the dichotomy of computable/non-computable to an order on the non-computable sets, $<_{\rm st}$ gives an order on the dominant sets.

However, $<_{\rm st}$ is not the only obvious choice for extending the idea of domination. In [4] and [3], the following strong settling time domination was introduced.

Definition 5. *For c.e. sets A and B we say A strongly settling time dominates B, $A >_{\rm sst} B$, if for all computable functions f and g, for almost every x,*

$$m_A(x) > f(m_B(g(x))).$$

The associated strict partial ordering on c.e. sets is denoted by $\mathcal{E}_{\rm sst}$.

The original motivation for this came as follows. To help simplify proofs in differential geometry, Nabutovsky and Weinberger asked whether there exits a sequence $\{A_i\}_{i\in\omega}$ of c.e. sets such that

$$(\forall \text{ computable } f)(\forall n)(\forall^\infty x)[m_{A_i}(x) > f(m_{A_{i+1}}(nx))]$$

This question was answered in my thesis [2] as follows.

Definition 6. *Let g be a computable function. For c.e. sets A and B we say $A >_{g-\rm st} B$ if for all computable functions f, for almost every x,*

$$m_A(x) > f(m_B(g(x))).$$

Theorem 4. *For any computable g, there exists a sequence $\{A_i\}$ of c.e. sets such that*

$$A_i >_{g-\mathrm{st}} A_{i+1}$$

This answered the question of Nabutovsky and Weinberger using $g(x) = x^2$.

However, this also raised the question of whether there are sets A and B such that $A >_{g-\mathrm{st}} B$ for all computable g. That is, whether there exist A and B such that $A >_{\mathrm{sst}} B$.

Observation 5. *For a c.e. set A, the following are equivalent:*

(i) A is dominant.
(ii) There exists an infinite computable set C such that $A >_{\mathrm{sst}} C$.
(iii) $A >_{\mathrm{sst}} C$ for every computable set C.

Proof. Clearly *(iii)⇒(ii)*.

For *(ii)⇒(i)*, if $A >_{\mathrm{sst}} C$ for some infinite computable set C, then $A >_{\mathrm{st}} C$, so by Observation 2, A is dominant.

For *(i)⇒(iii)*, suppose A is a dominant c.e. set, and that C is any computable set. Then m_C is computable, so for any computable functions f and g, $f \circ m_C \circ g$ is computable. Thus since A is dominant, $(\forall^\infty x)[m_A(x) > f(m_C(g(x)))]$, and so $A >_{\mathrm{sst}} C$.

Thus, relative to computable sets, the orderings $<_{\mathrm{st}}$ and $<_{\mathrm{sst}}$ behave the same way. However, the two orderings $<_{\mathrm{st}}$ and $<_{\mathrm{sst}}$ are distinct on the dominant sets; that is, $<_{\mathrm{st}}$ is a proper refinement of $<_{\mathrm{sst}}$.

2 Dominant Sets

We now summarize Robinson's results on dominant c.e. sets.

Observation 6 (Robinson [7]). *A dominant set must be high.*

Indeed, a c.e. set A is dominant if it's settling function m_A dominates every computable function. By Martin's characterization [5], a set is high if and only if it can compute a dominant function. Since A can certainly compute m_A, and since m_A is dominant, A is high.

Robinson further showed that every high c.e. degree contains a dominant set.

Theorem 7 (Robinson [7]). *Every high c.e. Turing degree contains a dominant c.e. set.*

Robinson showed this by again using Martin's characterization of high sets in terms of dominant functions.

On the other hand, a high c.e. set need not be dominant. Indeed, Robinson showed the following.

Theorem 8 (Robinson [7]). *Every c.e. degree contains a non-dominant set.*

Corollary 1 (Robinson [7]). *The dichotomy of the c.e. sets into dominant and non-dominant does not respect Turing reducibility.*

On the other hand, Robinson showed that wtt-reducibility is respected.

Theorem 9 (Robinson [7]). *If A is dominant and $A <_{\mathrm{wtt}} B$ then B is dominant.*

3 Settling Time Reducibility

As we saw earlier, if A is a c.e. set that settling time dominates an infinite c.e. set, then A must be dominant. So when comparing sets using the $<_{\mathrm{st}}$ reducibility, all sets except those on the bottom will be dominant (and hence high).

The $<_{\mathrm{st}}$ ordering works by comparing initial segments of the sets to one another, as we can see by the following theorems. We first introduce/recall some notation.

Definition 7

(i) A set A is bounded Turing reducible *to a set B ($A \leq_{\mathrm{bT}} B$) if $A \leq_T B$ and there is a computable function $h(x)$ and a Turing reduction $A = \Phi_e^B$ with use function $u(x) \leq h(x)$. This is commonly written $A <_{\mathrm{wtt}} B$, but we introduce this notation with an eye towards part (ii) of the definition.*

(ii) A set A is identity bounded Turing reducible *to B ($A \leq_{\mathrm{ibT}} B$) if $A \leq_{\mathrm{bT}} B$ with $h(x) = x$, namely $A = \Phi_e^B$ with use function $u(x) \leq x$ for all x.*

Theorem 10 (Csima, Soare [4]). *The $<_{\mathrm{st}}$ ordering is well defined on ibT-degrees. Indeed, let A, B, and C be c.e. sets with enumerations $\{A_s\}_{s\in\omega}$, $\{B_s\}_{s\in\omega}$, and $\{C_s\}_{s\in\omega}$, respectively. If $A <_{\mathrm{st}} B$ and $B \leq_{\mathrm{ibT}} C$ then $A <_{\mathrm{st}} C$. If $A \leq_{\mathrm{ibT}} B$ and $B <_{\mathrm{st}} C$ then $A <_{\mathrm{st}} C$.*

On the other hand, the $<_{\mathrm{st}}$ ordering is not well-defined on bT-degrees (wtt-degrees). Indeed, it does not even preserve computable isomorphism.

Theorem 11 (Csima, Shore [3]). *The order $<_{\mathrm{st}}$ is not well defined on* 1*-degrees. Indeed, there exist c.e. sets A and B such that $A \equiv_1 B$ but $A >_{\mathrm{st}} B$.*

Certainly if $A >_{\mathrm{st}} B$ then $A \geq_{\mathrm{T}} B$. From the above, we see it is possible for A and B to have the same Turing degree. Csima and Soare showed that it is possible to have a sequence of c.e. sets descending strictly in both the Turing degrees and in $\mathcal{E}_{\mathrm{st}}$.

Theorem 12 (Csima, Soare [4]). *There exists a sequence $\{A_n\}_{n\in\omega}$ of c.e. sets such that $A_n >_{\mathrm{T}} A_{n+1}$ and $A_n >_{\mathrm{st}} A_{n+1}$ for all n.*

When $<_{\mathrm{st}}$ was first introduced, it was for the application to differential geometry, which only required a descending sequence as above. However, the natural question arose as to what kind of partial orders can be embedded into $\mathcal{E}_{\mathrm{st}}$.

Theorem 13 (Csima, Shore [3]). *Any countable partial ordering can be embedded into $\mathcal{E}_{\mathrm{st}}$.*

Notice that $\mathcal{E}_{\mathrm{st}}$ is a strict partial ordering. Indeed, for any c.e. set A, $A \not>_{\mathrm{st}} A$. The largest equivalence relation on $\mathcal{E}_{\mathrm{st}}$ (or any strict partial order) that respects the given ordering and gives a reflexive partial ordering as a quotient is given by $A \equiv_{\mathrm{st}} B \Leftrightarrow \{C \mid C >_{\mathrm{st}} A\} = \{C \mid C >_{\mathrm{st}} B\}$ & $\{C \mid C <_{\mathrm{st}} A\} = \{C \mid C <_{\mathrm{st}} B\}$. One could instead of $\mathcal{E}_{\mathrm{st}}$ then reasonably study its quotient $\mathcal{E}^*_{\mathrm{st}}$ by this equivalence relation with the natural partial ordering $\leq$. Note that the above theorem shows that every countable partial ordering can be embedded in $\mathcal{E}^*_{\mathrm{st}}$ as well, since a given partial order P can be modified by adding on extra elements to produce a partial order P' such that any embedding of P' into $\mathcal{E}_{\mathrm{st}}$ will restrict to one of P into $\mathcal{E}^*_{\mathrm{st}}$.

Theorem 14 (Csima, Shore [3]). *There exists a maximal set in $\mathcal{E}_{\mathrm{st}}$. That is, there exists an A such that for all e, $W_e \not>_{\mathrm{st}} A$.*

As for minimal sets, certainly any set that is not high would be minimal. If we consider the computable sets to be the simplest, then we can ask for the existence of a non-trivial minimal set in the sense that $A >_{\mathrm{st}} C$ for any computable set C, but there exist no B with $A >_{\mathrm{st}} B >_{\mathrm{st}} C$.

Theorem 15 (Csima, Shore [3]). *There exists a non-trivial minimal set in $\mathcal{E}_{\mathrm{st}}$. Indeed, there exists a c.e. set A such that $A >_{\mathrm{st}} C$ for every computable C, and if W_e is noncomputable then $A \not>_{\mathrm{st}} W_e$.*

Also, in [3], it is shown that infs and sups need not exist in $\mathcal{E}_{\mathrm{st}}$.

Theorem 16 (Csima, Shore [3]). *There are c.e. sets A and B such that A and B have no infimum in the $<_{\mathrm{st}}$ ordering, indeed, such A and B can be found with $A \not\equiv_{\mathrm{st}} B$. There are c.e. sets C and D such that C and D have no supremum in the $<_{\mathrm{st}}$ ordering, indeed, such C and D can be found with $C \not\equiv_{\mathrm{st}} D$.*

The above theorem is proved by embedding a particular finite partial ordering into $\mathcal{E}_{\mathrm{st}}$ with an added property that two of the sets have a *gap* between them. That is, the two sets A and C are such that $A >_{\mathrm{st}} C$, but there is no c.e. set H with $A >_{\mathrm{st}} H >_{\mathrm{st}} C$. The strategy for constructing two sets with a gap between them is similar to that of constructing a minimal set.

4 Strong Settling Time Reducibility

Clearly if $A >_{\mathrm{sst}} B$, then $A >_{\mathrm{st}} B$, so that $<_{\mathrm{sst}}$ is a collapsing of $<_{\mathrm{st}}$. Hence sets in this ordering are still all high (except for the ones on the bottom).

Unlike $<_{\mathrm{st}}$, the ordering $<_{\mathrm{sst}}$ is well defined on bT-degrees (wtt-degrees). This difference also shows that the two orderings are distinct.

Theorem 17 (Csima, Shore, [3]). *The $<_{\mathrm{sst}}$ ordering is well defined on bT-degrees. In fact, it respects bT-reducibility in the sense that if $A \leq_{\mathrm{bT}} B <_{\mathrm{sst}} C$ or $A <_{\mathrm{sst}} B \leq_{\mathrm{bT}} C$ then $A <_{\mathrm{sst}} C$.*

The proof of this is essentially the same as the proof that $<_{\mathrm{st}}$ is well defined on ibT-degrees, but the extra strength of $<_{\mathrm{sst}}$ allows arbitrary computable functions to be absorbed, rather than just the identity function.

On the other hand, $<_{\mathrm{sst}}$ is not well defined on Turing degrees.

Theorem 18. *The $<_{\mathrm{sst}}$ ordering is not well defined on Turing degrees. Indeed, if $A >_{\mathrm{sst}} B$ and B is infinite, then there exists a c.e. set $C \equiv_{\mathrm{T}} A$ such that $C \not>_{\mathrm{sst}} B$.*

Proof. In [4], this same theorem is shown with $<_{\mathrm{st}}$ in place of $<_{\mathrm{sst}}$. But, if $A \not>_{\mathrm{st}} B$, then certainly $A \not>_{\mathrm{sst}} B$. It can also be seen as an immediate consequence of Corollary 1 and Observation 2.

In as far as embedding of partial orderings, Csima and Shore have the following partial result, which amounts to showing that certain linear orders can be embedded into $\mathcal{E}_{\mathrm{sst}}$.

Theorem 19 (Csima, Shore [3]). *Let P be a computable partial ordering on $\mathbb{N}$ with no infinite ascending sequence. There exists a computable sequence $\{A_n\}_{n\in\mathbb{N}}$ of c.e. sets such that if $m <_P n$ then $A_m <_{\mathrm{sst}} A_n$.*

Since we have shown the existence of maximal and minimal sets for $\mathcal{E}_{\mathrm{st}}$, we get the following for free in $\mathcal{E}_{\mathrm{sst}}$.

Theorem 20. *There exists a maximal set in $\mathcal{E}_{\mathrm{sst}}$.*

Proof. By Theorem 14, there exists a c.e. set A such that for all e, $W_e \not>_{\mathrm{st}} A$. But then certainly for all e, $W_e \not>_{\mathrm{sst}} A$.

Theorem 21. *There exists a non-trivial minimal set in $\mathcal{E}_{\mathrm{sst}}$. That is, there exists a c.e. set A such that $A >_{\mathrm{sst}} C$ for any computable C, but for all W_e, if $A >_{\mathrm{sst}} W_e$ then $W_e \not>_{\mathrm{sst}} C$.*

Proof. By Theorem 15, there exists a c.e. set A such that for all computable C, $A >_{\mathrm{st}} C$, and for all e, if W_e is non-computable, then $A \not>_{\mathrm{st}} W_e$. Now if $A >_{\mathrm{st}} C$ for all computable C, then A is dominant, and so $A >_{\mathrm{sst}} C$ for all computable C, by Observations 2 and 5. And if $A \not>_{\mathrm{st}} W_e$ for all e, then certainly $A \not>_{\mathrm{sst}} W_e$ for all e. So the non-trivial minimal set in $\mathcal{E}_{\mathrm{st}}$ is still non-trivial and minimal in $\mathcal{E}_{\mathrm{sst}}$.

5 Questions

It was shown that any countable partial order can be embedded into $\mathcal{E}_{\mathrm{st}}$.

Question 1. Can this be done while simultaneously embedding into the Turing degrees?

In Theorem 19, Csima and Shore embed certain linear orders into $\mathcal{E}_{\mathrm{sst}}$.

Question 2. Can arbitrary countable partial orders be embedded into $\mathcal{E}_{\mathrm{sst}}$?

The techniques of Theorem 19 can probably more quickly be adapted to answer the following easier question in the affirmative.

Question 3. Can every finite partial order be embedded into $\mathcal{E}_{\mathrm{sst}}$?

We have seen that infs and sups need not exist in $\mathcal{E}_{\mathrm{st}}$.

Question 4. Must infs and sups exist in $\mathcal{E}_{\mathrm{sst}}$?

To prove that infs and sups do not exist in $\mathcal{E}_{\mathrm{sst}}$, the argument for showing there is a minimal set was lifted to a more complicated setting. The way the existence of a non-trivial minimal set in $\mathcal{E}_{\mathrm{sst}}$ was shown was to observe that the non-trivial minimal set in $\mathcal{E}_{\mathrm{st}}$ was still non-trivial in $\mathcal{E}_{sst}$. This was because $<_{\mathrm{st}}$ and $<_{\mathrm{sst}}$ behave the same relative to the computable sets; though of course this is not true in general. So it is unclear whether the gap argument can go through in the $\mathcal{E}_{\mathrm{sst}}$ setting. The following is probably true, and would be a good first step towards showing infs and sups do not exist in $\mathcal{E}_{\mathrm{sst}}$.

Question 5. Do there exist non-computable A and B such that $A >_{\mathrm{sst}} B$ and such that there exist no C with $A >_{\mathrm{sst}} C >_{\mathrm{sst}} B$?

Certainly it cannot be true that given any high sets such that $A <_{\mathrm{T}} B$ then $A <_{\mathrm{st}} B$, since Robinson already showed this must fail. However, Robinson did show that every high c.e. degree contains a dominant set, so the following may be possible:

Question 6. If $\mathbf{a} < \mathbf{b}$ are high c.e. degrees, must there exist c.e. sets $A \in \mathbf{a}$ and $B \in \mathbf{b}$ such that $A <_{\mathrm{st}} B$? $A <_{\mathrm{sst}} B$?

Question 7. More generally, given any partial order of high degrees, can there be realized a partial order of c.e. sets with the same relationship under $<_{\mathrm{st}}$ ($<_{\mathrm{sst}}$) in exactly those high degrees?

References

1. Cooper, S.B.: Computability Theory. Chapman & Hall/CRC, Boca Raton, FL (2004)
2. Csima, B.F.: Applications of Computability Theory to Prime Models and Differential Geometry, Ph.D. thesis, The University of Chicago (2003)
3. Csima, B.F., Shore, R.A.: The Settling-Time Reducibility Ordering. Journal of Symbolic Logic 71(4), 1394–1410 (2006)
4. Csima, B.F., Soare, R.I.: Computability Results Used in Differential Geometry, Journal of Symbolic Logic (To appear)
5. Martin, D.A.: Classes of recursively enumerable sets and degrees of unsolvability. Z. Math. Logik Grundlagen Math. 12, 295–310 (1966)
6. Nabutovsky, A., Weinberger, S.: The Fractal Nature of Riem/Diff I. Geometrica Dedicata 101, 1–54 (2003)

7. Robinson, R.W.: A dichotomy of the recursively enumerable sets. Z. Math. Logik Grundlag. Math. 14, 339–356 (1968)
8. Soare, R.I.: Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets, Springer, Heidelberg (1987)
9. Soare, R.I.: Computability theory and differential geometry. Bull. Symb. Logic 10, 457–486 (2004)
10. Soare, R.I.: Computability Theory and Applications, Springer-Verlag, Heidelberg (To appear)
11. Weinberger, S.: Computers, rigidity and moduli. The large scale fractal geometry of Reimannian moduli space (M.B. Porter Lectures). Princeton UNiversity Press, Princeton NJ (2005)

Time-Complexity Semantics for Feasible Affine Recursions

Norman Danner[1] and James S. Royer[2]

[1] Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT 06459, USA
ndanner@wesleyan.edu

[2] Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13210, USA
royer@ecs.syr.edu

Abstract. The authors' ATR programming formalism is a version of call-by-value PCF under a complexity-theoretically motivated type system. ATR programs run in type-2 polynomial-time and all standard type-2 basic feasible functionals are ATR-definable (ATR types are confined to levels 0, 1, and 2). A limitation of the original version of ATR is that the only directly expressible recursions are tail-recursions. Here we extend ATR so that a broad range of affine recursions are directly expressible. In particular, the revised ATR can fairly naturally express the classic insertion- and selection-sort algorithms, thus overcoming a sticking point of most prior implicit-complexity-based formalisms. The paper's main work is in extending and simplifying the original time-complexity semantics for ATR to develop a set of tools for extracting and solving the higher-type recurrences arising from feasible affine recursions.

1 Two Algorithms in Search of a Type-System

As Hofmann [9] has noted, a problem with implicit characterizations of complexity classes is that they often fail to capture many natural *algorithms*—usually because the complexity-theoretic types used to control primitive recursion impose draconian restrictions on programming. Here is an example. In Bellantoni and Cook's [3] and Leivant's [11] well-known characterizations of the polynomial-time computable functions, a recursively-computed value is prohibited from driving another recursion. But, for instance, the recursion clause of insertion-sort has the form $\mathsf{ins\_sort}(\mathsf{cons}(a, l)) = \mathsf{insert}(a, \mathsf{ins\_sort}(l))$, where insert is defined by recursion on its second argument; selection-sort presents analogous problems.

Hofmann [9, 8] addresses this problem by noting that the output of a non-size-increasing program (such as $\mathsf{ins\_sort}$) should be permitted to drive another recursion, as it cannot cause the sort of complexity blow-up the B-C-L restrictions guard against. To incorporate such recursions, Hofmann defines a higher-order language with typical first-order types and a special type $\Diamond$ through which functions defined recursively must "pay" for any use of size-increasing constructors, in effect guaranteeing that there is no size increase. Through this scheme Hofmann is able to implement many natural algorithms while still ensuring that any typable program is non-size-increasing polynomial-time computable (Aehlig and Schwichtenberg [1] sketch an extension that captures all of polynomial-time).

Our earlier paper [5, 6], hereafter referred to as *ATS*, takes a different approach to constructing a usable programming language with guaranteed resource usage. We introduce a type-2 programming formalism called ATR (for Affine Tail Recursion, which we rechristen in this paper as Affine *Tiered* Recursion) based on PCF. ATR's type system is motivated by the tiering and safe/normal notions of [11] and [3] and serves to control the size of objects. Instead of restricting to primitive recursion, ATR has an operator for recursive definitions; affine types and explicit clocking on the operator serve to control time. We give a denotational semantics to ATR types and terms in which the size restrictions play a key part. This allows us, for example, to give an ATR *definition* of a primitive-recursion-on-notation combinator (with appropriate types and without explicit bounding terms) that preserves feasibility. We also give a *time-complexity semantics* and use it to prove that each type-2 ATR program has a (second-order) polynomial run-time.[1] Finally, we show that the standard type-2 basic feasible functionals (an extension of polynomial-time computability to type-2) of Mehlhorn [12] and Cook and Urquhart [4] are ATR definable. Moreover, our underlying model of computation (and complexity) is just a standard abstract machine that implements call-by-value PCF. However, ATR is still somewhat limited as its only base type is binary words and the only recursions allowed are tail-recursions.

What is new in this paper. In this paper we extend ATR to encompass a broad class of feasible affine recursions. We demonstrate these extensions by giving fairly direct and natural versions of insertion- and selection-sorts on lists. As additional evidence of ATR's support for programming we do not add lists as a base type, but instead show how to implement them over ATR's base type of binary words.

The technical core of this paper is a simplification and generalization of the time-complexity semantics of *ATS*. We construct a straightforward framework in which recursion schemes in ATR lead to time-complexity recurrences that must be solved to show that these schemes preserve feasibility. This gives a route to follow when adding new forms of recursion to ATR. We follow this route to show that the recursions used to implement lists and insertion-sort are (second-order) polynomial-time bounded. We also discuss how to extend these results to handle the recursions present in selection-sort. Thus along with significantly extending our existing system to the point where many standard algorithms can be naturally expressed, we also provide a set of basic tools for further extensions.

2 Programming in ATR

The ATR *formalism.* An ATR base type has the form N_L, where *labels* L are elements of the set $(\Box\Diamond)^* \bigcup \Diamond(\Box\Diamond)^*$ (our use of $\Diamond$ is not directly related to

[1] These kinds of results may also have applications in the type of static analysis for time-complexity that Frederiksen and Jones [7] investigate.

$$\begin{aligned} s,t ::= &\ V \mid K \mid O \mid (\lambda V.s) \mid (st) \\ &\mid (\mathsf{c}_a\, s) \mid (\mathsf{d}\, s) \mid (\mathsf{t}_a\, s) \mid (\mathsf{if}\ s\ \mathsf{then}\ t_0\ \mathsf{else}\ t_1) \mid (\mathsf{down}\, s\, t) \mid (\mathsf{crec}\, K (\lambda_r f.t)) \end{aligned}$$

Fig. 1. ATR expressions. V is a set of variable symbols and O a set of oracle symbols.

Hofmann's). The labels are ordered by $\varepsilon \leq \Diamond \leq \Box\Diamond \leq \Diamond\Box\Diamond \leq \cdots$ We define a subtype relation on the base types by $\mathsf{N}_L \leq: \mathsf{N}_{L'}$ if $L \leq L'$ and extend it to function types in the standard way. Roughly, we can think of type-N_ε values as basic string inputs, type-$\mathsf{N}_\Diamond$ values as the result of polynomial-time computations over N_ε-values, type-$\mathsf{N}_{\Box\Diamond}$-values as the result applying an oracle (a type-1 input) to $\mathsf{N}_\Diamond$-values, type-$\mathsf{N}_{\Diamond\Box\Diamond}$ values as the result of polynomial-time computations over $\mathsf{N}_{\Box\Diamond}$-values, etc. N_L is called an *oracular* (respectively, *computational*) type when $L \in (\Box\Diamond)^*$ (respectively, $\Diamond(\Box\Diamond)^*$). We let b (possibly decorated) range over base types. Function types are formed as usual from the base types.

The base datatype is $K = \{\mathbf{0}, \mathbf{1}\}^*$, and the ATR terms are defined in Figure 1. The term forming operations correspond to adding and deleting a left-most bit (c_0, c_1, and d), testing whether a word begins with a $\mathbf{0}$ or a $\mathbf{1}$ (t_0 and t_1), and a conditional. The intended interpretation of $\mathsf{down}\, s\, t$ is s if $|s| \leq |t|$ and ε otherwise. The recursion operator is crec, standing for clocked recursion.

The typing rules are given in Figure 2. Type contexts are split (after Barber and Plotkin's DILL [2]) into intuitionistic and affine zones. Variables in the former correspond to the usual $\rightarrow$ introduction and elimination rules and variables in the latter are intended to be recursively defined; variables that occur in the affine zone are said to *occur affinely* in the term. The crec-**I** rule serves as both introduction and elimination rule for the implicit $\multimap$ types (in the rule $\mathbf{b} = \mathsf{b}_1, \ldots, \mathsf{b}_k$ and $\boldsymbol{v}{:}\mathbf{b}$ stands for $v_1{:}\mathsf{b}_1, \ldots, v_k{:}\mathsf{b}_k$). We use λ_r as the abstraction operator for variables introduced from the affine zone of the type context to further distinguish them from "ordinary" variables. The side-conditions on crec-**I** are that f occurs in cons-tail position[2] in t and if $\mathsf{b}_i \leq: \mathsf{b}_1$ then b_i is oracular (including $i = 0$). The constraint on the types allows us to prove a polynomial size-bound on the growth of the arguments to f, which in turn allows us to prove such bounds on all terms. The typing rules enforce a "one-use" restriction on affine variables by disallowing their occurrence as a free variable in both arguments of down, the argument of an application, the test of a conditional, or anywhere in a crec-term.

The intuition behind the *shifts-to* relation $\propto$ between types is as follows. Suppose $f{:}\mathsf{N}_\varepsilon \rightarrow \mathsf{N}_\Diamond$. We think of f as being a function that does some polynomial-time computation to its input. If we have an input x of type $\mathsf{N}_{\Box\Diamond}$ then recalling the intuition behind the base types, we should be able to assign the type $\mathsf{N}_{\Diamond\Box\Diamond}$ to $f(x)$. The shifts-to relation allows us to shift input types in this way, with

[2] Informally, f occurs in *cons-tail position in* t if in the parse-tree of t a path from the root to a complete application of f passes through only conditional branches (not tests), c_0, c_1, and the left-argument of down; *tail_len*(f, t) is defined to be the maximum number of c_a operations not below any down node in any such path.

$$\textbf{Zero-I}\ \frac{}{\Gamma;\Delta\vdash\varepsilon:\mathsf{N}_\varepsilon}\qquad \textbf{Const-I}\ \frac{}{\Gamma;\Delta\vdash K:\mathsf{N}_\Diamond}$$

$$\textbf{Int-Id-I}\ \frac{}{\Gamma,v:\sigma;\Delta\vdash v:\sigma}\qquad \textbf{Aff-Id-I}\ \frac{}{\Gamma;\Delta,v:\sigma\vdash v:\sigma}$$

$$\textbf{Shift}\ \frac{\Gamma;\Delta\vdash s:\sigma}{\Gamma;\Delta\vdash s:\tau}\ (\sigma\propto\tau)\qquad \textbf{Subsumption}\ \frac{\Gamma;\Delta\vdash s:\sigma}{\Gamma;\Delta\vdash s:\tau}\ (\sigma\le:\tau)$$

$$\mathsf{c}_a\textbf{-I}\ \frac{\Gamma;\Delta\vdash s:\mathsf{N}_{\Diamond_d}}{\Gamma;\Delta\vdash(\mathsf{c}_a\ s):\mathsf{N}_{\Diamond_d}}\qquad \mathsf{d}\textbf{-I}\ \frac{\Gamma;\Delta\vdash s:\mathsf{N}_L}{\Gamma;\Delta\vdash \mathsf{d}\,s:\mathsf{N}_L}\qquad \mathsf{t}_a\textbf{-I}\ \frac{\Gamma;\Delta\vdash s:\mathsf{N}_L}{\Gamma;\Delta\vdash \mathsf{t}_a\,s:\mathsf{N}_L}$$

$$\mathsf{down}\textbf{-I}\ \frac{\Gamma;\Delta_0\vdash s:\mathsf{N}_{L_0}\qquad \Gamma;\Delta_1\vdash t:\mathsf{N}_{L_1}}{\Gamma;\Delta_0,\Delta_1\vdash(\mathsf{down}\,st):\mathsf{N}_{L_1}}$$

$$\mathsf{if}\textbf{-I}\ \frac{\Gamma;\_\vdash s:\mathsf{N}_L\qquad \Gamma;\Delta_0\vdash t_0:\mathsf{N}_{L'}\qquad \Gamma;\Delta_1\vdash t_1:\mathsf{N}_{L'}}{\Gamma;\Delta_0\cup\Delta_1\vdash(\mathsf{if}\ s\ \mathsf{then}\ t_0\ \mathsf{else}\ t_1):\mathsf{N}_{L'}}$$

$$\mathsf{crec}\textbf{-I}\ \frac{\_;\_\vdash K:\mathsf{N}_\Diamond\qquad \Gamma,\boldsymbol{v}:\mathbf{b};f:\mathbf{b}\to\mathsf{b}_0\vdash t:\mathsf{b}_0}{\Gamma;\_\vdash\mathsf{crec}\ a\,(\lambda_r f.\lambda\boldsymbol{v}.t):\mathbf{b}\to\mathsf{b}_0}$$

$$\to\textbf{-I}\ \frac{\Gamma,v:\sigma;\Delta\vdash t:\tau}{\Gamma;\Delta\vdash(\lambda v.t):\sigma\to\tau}\qquad \to\textbf{-E}\ \frac{\Gamma;\Delta\vdash s:\sigma\to\tau\qquad \Gamma;\_\vdash t:\sigma}{\Gamma;\Delta\vdash(st):\tau}$$

Fig. 2. ATR typing. The changes from *ATS* are as follows: (1) *ATS* imposed no constraint on b_0 in (crec-**I**);(2) *ATS* restricted (crec-**I**) to tail-recursion; and (3) *ATS* restricted (d-**I**) and (t_a-**I**) to computational types.

a corresponding shift in output type. As a concrete example, the judgment $f:\mathsf{N}_\varepsilon\to\mathsf{N}_\Diamond, x:\mathsf{N}_\varepsilon;\vdash f(fx):\mathsf{N}_{\Diamond\Box\Diamond}$ is derivable using **Subsumption** to coerce the type of $f(x)$ to $\mathsf{N}_{\Box\Diamond}$ and **Shift** to shift the type of the outer application of f. The definition of $\propto$ must take into account multiple arguments and level-2 types and hence is somewhat involved. Since we do not need it for the typings in this paper, we direct the reader to *ATS* for the full definition.

Motivated by the approach of Jones [10], we define the cost of evaluation to be the size of a call-by-value evaluation derivation. This is essentially equivalent to the abstract machine-based cost model of *ATS*, but the derivation-based model helps avoid considerable bookkeeping clutter. Values are string constants, oracles, or abstractions. Environments map term variables to values or to closures over crec terms. A closure $t\rho$ consists of a term t and an environment ρ. The evaluation relation has the form $t\rho\downarrow z\theta$ where $t\rho$ and $z\theta$ are closures and z is a value. The derivation rules for the evaluation are mostly straightforward and mimic the action of the abstract machine of *ATS*; for example, we have

$$\frac{\rho(x)\downarrow z\theta}{x\rho\downarrow z\theta}\qquad \frac{t\rho\downarrow(\mathbf{0}z)\theta}{(\mathsf{d}\,t)\rho\downarrow z\theta}\qquad \frac{s\rho\downarrow w\zeta\qquad t\rho\downarrow z\theta\qquad |w|\le|z|}{(\mathsf{down}\,st)\rho\downarrow w\zeta}.$$

The evaluation rule for crec terms is

$$\frac{}{(\mathsf{crec}\,a(\lambda_r f.\lambda\boldsymbol{v}.t))\rho\downarrow(\lambda\boldsymbol{v}.\mathsf{if}\ |a|<|v_1|\ \mathsf{then}\ t\ \mathsf{else}\ \varepsilon)\rho[f\mapsto\mathsf{crec}(\mathbf{0}a)(\lambda_r f.\lambda\boldsymbol{v}.t)]}$$

which shows how unwinding the recursion increments the clock by one step. The cost of most inference rules is 1, except the $\mathsf{down}\,s\,t$ inference rules have cost

```
val  nil = ε : N_ε

val  cons : N_ε → N_◇ → N_◇ =
     fn w l ⇒ letrec enc : N_ε → N_◇ → N_◇ =
          fn b x ⇒ if x then if t0(x) then c1(c0(enc b (d x)))
                                  else c1(c1(enc b (d x)))
                      else c0(l)
     in enc w w end

val  head : N_ε → N_ε =
     fn l ⇒ letrec dec : N_ε → N_◇ → N_◇ =
          fn b x ⇒ if t1(x) then
                           if t0(d x) then c0(dec b (d(d(x)))) else c1(dec b (d(d(x))))
                      else ε
     in down (dec l l)(l) end

val  tail : N_ε → N_ε =
     fn l ⇒ letrec tail' : N_ε → N_ε → N_ε =
          fn b x ⇒ if t1(x) then tail' b d(d(x)) else d(x)
     in tail' l l end
```

Fig. 3. The basic list operations in ATR

$2|z|+1$ where $t\rho \downarrow z\theta$ and environment and oracle evaluation have length-cost (so, e.g., the cost of the environment rule shown above is $\max(|z|, 1)$ when z is of base type, 1 otherwise).

Implementing lists and sorting. We implement lists of binary words via concatenated self-delimiting strings. Specifically, we code the word $w = b_0 \dots b_{k-1}$ as $s(w) = 1b_0 1b_1 \dots 1b_{k-1}0$ and the list $\langle w_0, \dots, w_{k-1}\rangle$ as $s(w_0) \oplus \dots \oplus s(w_{k-1})$, where $\oplus$ is the concatenation operation. Code for the basic list operations is given in Figure 3.[3] Note that the cons, head, and tail programs all use cons-tail recursion. Insertion-sort is expressed in essentially its standard form, as in Figure 4. This implementation requires another form of recursion, in which the complete application of the recursively-defined function appears in an argument to some operator. In the later part of Section 3 we show how this *recursion in an argument* can be incorporated into ATR. Selection-sort requires yet another form of recursion (a generalization of cons-tail recursion); we discuss how to incorporate it into ATR in Section 4.

Our head and ins_sort programs use the down operator to coerce the type $\mathsf{N}_\diamondsuit$ to N_ε. Roughly, down is used in places where our type-system is not clever enough to prove that the result of a recursion is of size no larger than one of the recursion's initial arguments; the burden of supplying these proofs is shifted off to the correctness argument for the recursion. A cleverer type system (say,

[3] In these code samples, letrec f=s in t end abbreviates $\mathsf{t}[f \mapsto \mathsf{crec}\,\varepsilon(\lambda_r f.s)]$ and we use the ML notation fn $x \Rightarrow \dots$ for λ-abstraction.

```
val  insert  :  N_ε → N_ε → N_◊  =
    fn  w l  ⇒ letrec  ins :  N_ε → N_ε → N_◊  =
        fn  b  l'  ⇒ if  l'  then
                                if   leq  w head(l')  then  cons w l'
                                else  cons (head l')  (ins  b  ( tail  l'))
                            else  cons w nil
    in  ins  l  l  end

val  ins_sort  :  N_ε → N_◊  =
    fn  l  ⇒ letrec  isort  :  N_ε → N_ε → N_◊  =
        fn  b  l'  =  if  l'  then  insert  (head l')  (down (isort  b  ( tail  l'))  l')  else  ε
    in  isort  l  l  end
```

Fig. 4. Insertion-sort in ATR

along the lines of Hofmann's [8]) could obviate many of these down's, but at the price of more complex syntax (i.e., typing), semantics (of values and of time-complexities), and, perhaps, pragmatics (i.e., programming). Our use of down gives us a more primitive (and intensional) system than found in pure implicit complexity,[4] but it also gives us a less cluttered setting to work out the basics of complexity-theoretic compositional semantics—the focus of the rest of the paper. Also, in practice the proofs that the uses of down forces into the correctness argument are for the most part obvious, and thus not a large burden on the programmer.

3 Soundness Theorems

In this section we rework the Soundness Theorem of *ATS* to set up the framework for such theorems, and then use the framework to handle the recursions used to implement insertion-sort (we discuss selection-sort in Section 4). Because of space considerations, we just sketch the main points here and leave detailed proofs to the full paper. The key technical notion is that of *bounding* a closure $t\rho$ by a *time-complexity*, which provides upper bounds on the cost of evaluating $t\rho$ to a value $z\theta$ as well as the *potential* cost of using $z\theta$. The potential of a base-type closure is just its (denotation's) length, whereas the potential of a function f is a function that maps potentials p to the time complexity of evaluating f on arguments of potential p. The bounding relation gives a *time-complexity semantics* for ATR-terms; a *soundness theorem* asserts the existence of a bounding time-complexity for every ATR term. In this paper, our soundness theorems also assert that the bounding time-complexities are *safe*, which in particular implies type-2 polynomial size and cost bounds for the closure. We thereby encapsulate the Soundness, polynomial-size-boundedness, and polynomial-time-boundedness theorems of *ATS* (the *value semantics* for the meaning of ATR terms and corresponding soundness theorem are unchanged).

[4] Leivant's *recursion under a high-tier bound* [11, §3.1] implements a similar idea.

Soundness for tail-recursion. We start by defining *cost*, *potential*, and *time-complexity* types, all of which are elements of the simple product type structure over the *time-complexity base types* $\{\mathsf{T}\} \cup \{\mathsf{T}_L \mid L \text{ is a label}\}$ (we sometimes conflate the syntactic types with their intended meaning, which is the standard set-theoretic semantics when all base types are interpreted as unary numerals). The subtype relation on base types is defined by $\mathsf{T}_L \leq: \mathsf{T}_{L'}$ if $L \leq L'$ and $\mathsf{T}_L \leq: \mathsf{T}$ for all L, and extended to product and function types in the standard way. The only cost type is T, and for each ATR-type σ we define the potential type $\langle\langle\sigma\rangle\rangle$ and time-complexity type $\|\sigma\|$ by $\langle\langle \mathsf{N}_L \rangle\rangle = \mathsf{T}_L$, $\langle\langle \sigma \to \tau \rangle\rangle = \langle\langle\sigma\rangle\rangle \to \|\tau\|$, and $\|\tau\| = \mathsf{T} \times \langle\langle\tau\rangle\rangle$. Write $\mathit{cost}(\cdot)$ and $\mathit{pot}(\cdot)$ for the left- and right-projections on $\|\tau\|$. We introduce *time-complexity variables*, a new syntactic category, and define a time-complexity context to be a finite map from t.c. variables to cost and potential types. For a t.c. context Σ, Σ-Env is the set of Σ environments, defined in the usual way. We extend $\|\cdot\|$ to ATR-type contexts by introducing t.c. variables x_c and x_p for each ATR-variable x and setting $\|\Gamma\| = \cup_{(x:\sigma)\in\Gamma}\{x_c : \mathsf{T}, x_p : \langle\langle\sigma\rangle\rangle\}$. A *time-complexity denotation* of t.c. type γ w.r.t. a t.c. environment Σ is a function $X : \Sigma\text{-Env} \to \gamma$. The projections *cost* and *pot* extend to t.c. denotations in the obvious way.

Definition

1. Suppose $t\rho$ is a closure and $z\theta$ a value, both of type τ; χ a time-complexity of type $\|\tau\|$; and q a potential of type $\langle\langle\tau\rangle\rangle$. Define the *bounding relations* $t\rho \sqsubseteq^\tau \chi$ and $z\theta \sqsubseteq^\tau_{\text{pot}} q$ as follows:[5]
   (a) $t\rho \sqsubseteq^\tau \chi$ if $\mathit{cost}(t\rho) \leq \mathit{cost}(\chi)$ and if $t\rho \downarrow z\theta$, then $z\theta \sqsubseteq^\tau_{\text{pot}} \mathit{pot}(\chi)$.
   (b) $z\theta \sqsubseteq^{\mathsf{b}}_{\text{pot}} q$ if $|z| \leq q$.
   (c) $(\lambda v.t)\theta \sqsubseteq^{\sigma\to\tau}_{\text{pot}} q$ if for all values $z\eta$, if $z\eta \sqsubseteq^\sigma_{\text{pot}} p$, then $t\theta[v \mapsto z\eta] \sqsubseteq^\tau q(p)$.
   (d) $O\theta \sqsubseteq^{\sigma\to\tau}_{\text{pot}} q$ if for all values $z\eta$, if $z\eta \sqsubseteq^\sigma_{\text{pot}} p$, then $(O(z\eta))[] \sqsubseteq^\tau q(p)$.
2. For $\rho \in \Gamma$-Env and $\varrho \in \|\Gamma\|$-Env, we write $\rho \sqsubseteq \varrho$ if for all $v \in \operatorname{Dom} \rho$ we have that $v\rho \sqsubseteq (\varrho(v_c), \varrho(v_p))$.
3. For an ATR-term $\Gamma; \Delta \vdash t{:}\tau$ and a time-complexity denotation X of type $\|\tau\|$ w.r.t. $\|\Gamma; \Delta\|$, we say $t \sqsubseteq X$ if for all $\rho \in (\Gamma;\Delta)$-Env and $\varrho \in \|\Gamma;\Delta\|$-Env such that $\rho \sqsubseteq \varrho$ we have that $t\rho \sqsubseteq X\varrho$.

We define second-order polynomial expressions of tally, potential, and time-complexity types using the operations $+$, $*$, and $\vee$ (binary maximum); the typing rules are given in Figure 5. Of course, a polynomial $\Sigma \vdash p : \gamma$ corresponds to a t.c. denotation of type γ w.r.t. Σ in the obvious way. We shall frequently write p_p for $\mathit{pot}(p)$.

Definition. Let γ be a potential type, b a time-complexity base type, p a potential polynomial, and suppose $\Sigma \vdash p : \gamma$.

1. p is b-strict w.r.t. Σ when $\mathit{tail}(\gamma) \leq: \mathsf{b}$ and every unshadowed[6] free-variable occurrence in p has a type with tail $<: \mathsf{b}$.

[5] We will drop the superscript when it is clear from context.

[6] Roughly, a free-variable occurrence is *shadowed* if it is in a subterm that does not contribute to the size of the term; see *ATS* for details.

$$\frac{}{\Sigma \vdash \varepsilon : \mathsf{T}_\varepsilon} \qquad \frac{}{\Sigma \vdash \mathbf{0}^n : \mathsf{T}_\Diamond} \qquad \frac{}{\Sigma, x:\gamma \vdash x : \gamma}$$

$$\frac{\Sigma \vdash p : \gamma}{\Sigma \vdash p : \gamma'}\,(\gamma \propto \gamma') \qquad \frac{\Sigma \vdash p : \gamma}{\Sigma \vdash p : \gamma'}\,(\gamma \leq: \gamma')$$

$$\frac{\Sigma \vdash p : \mathsf{T}_{\Diamond_k} \qquad \Sigma \vdash q : \mathsf{T}_{\Diamond_k}}{\Sigma \vdash p \bullet q : \mathsf{T}_{\Diamond_k}} \qquad \frac{\Sigma \vdash p : \gamma \qquad \Sigma \vdash q : \gamma}{\Sigma \vdash p \vee q : \gamma}$$

$$\frac{\Sigma, x:\sigma \vdash p : \tau}{\Sigma \vdash \lambda x.p : \sigma \to \tau} \qquad \frac{\Sigma \vdash p : \sigma \to \tau \qquad \Sigma \vdash q : \sigma}{\Sigma \vdash pq : \tau}$$

Fig. 5. Typing rules for time-complexity polynomials. $\bullet$ is $+$ or $*$, γ is a t.c. base type.

2. p is b-chary w.r.t. Σ when $\gamma = \mathsf{b}$ and $p = p_1 \vee \cdots \vee p_m$ with $m \geq 0$ where $p_i = (vq_1 \ldots q_k)$ with each q_i b-strict.
3. p is b-*safe* w.r.t. Σ if:
   (a) γ is a base type and $p = q \odot_{\mathsf{b}} r$ where q is b-strict and r is b-chary, $\odot_{\mathsf{b}} = \vee$ if b is oracular, and $\odot_{\mathsf{b}} = +$ if b is computational.
   (b) $\gamma = \sigma \to (\mathsf{T} \times \tau)$ and $pot(pv)$ is b-safe w.r.t. $\Sigma, v : \sigma$.
4. A t.c. polynomial $\Sigma \vdash q : \mathsf{T} \times \gamma$ is b-*safe* if $pot(q)$ is.
5. A t.c. denotation X of type γ w.r.t. Σ is b-*safe* if X is bounded by a b-safe t.c. polynomial $\Sigma \vdash p : \gamma$.

The Soundness Theorem of *ATS* asserts that every tail-recursive term is bounded by a t.c. denotation for which the cost component is bounded by a type-2 polynomial in the lengths of t's free variables. In the next subsection, we extend this to cons-tail recursion and prove that the bounding t.c. denotation is in fact safe. In particular, we also have that the potential of t's denotation is bounded by a safe polynomial. At base type, this latter statement corresponds to the "poly-max" bounds that can be computed for Bellantoni-Cook and Leivant-style tiered functions (e.g., [3, Lemma 4.1]).

Soundness for cons-tail-recursion. For the remainder of this subsection t is a term such that f is in cons-tail position in t and for which we have a typing $\Gamma, \boldsymbol{v} : \mathbf{b}; f : \mathbf{b} \to \mathsf{b} \vdash t : \mathsf{b}$. We write $\Gamma_{\boldsymbol{v}}$ for for the type context $\Gamma, \boldsymbol{v} : \mathbf{b}$. Define the terms $C_\ell = \mathsf{crec}(\mathbf{0}^\ell a)(\lambda_r f.\lambda \boldsymbol{v}.t)$ and $T_\ell = \mathsf{if}\ |\mathbf{0}^\ell a| < |v_1|\ \mathsf{then}\ t\ \mathsf{else}\ \varepsilon$ (we write $\mathbf{0}^\ell a$ for $\mathbf{0} \ldots \mathbf{0} a$ with ℓ $\mathbf{0}$'s, remembering that this is a string constant), and for any environment ρ, set $\rho_\ell = \rho[f \mapsto C_\ell]$. The main difficulty in proving soundness is constructing a bounding t.c. denotation for crec terms. A key component in the construction is the Affine Decomposition Theorem in Section 14 of *ATS*, which describes how to compute the time-complexity of a term in which f occurs affinely and in tail position. To state it, we need some definitions.

Definition. Let X and Y be t.c. denotations of type $\|\sigma \to \tau\|$ and $\|\sigma\|$, respectively.

1. For a potential $p : \mathsf{T}_L$, $val\, p = (1 \vee p, p)$; if p is of higher type, then $val\, p = (1, p)$. For a t.c. environment ϱ and ATR variable v we write $\varrho[v \mapsto \chi]$ for $\varrho[v_c, v_p \mapsto cost(\chi), pot(\chi)]$.

2. If Y is w.r.t. $\|\Gamma, v : \sigma'\|$, then $\lambda\!\!\lambda_\star v.Y =_{\text{df}} \lambda\!\!\lambda\varrho(1, \lambda\!\!\lambda v_p.Y(\varrho[v \mapsto \mathit{val}\, v_p]))$ is a t.c. denotation of type $\|\sigma' \to \sigma\|$ w.r.t. $\|\Gamma\|$ (we use $\lambda\!\!\lambda x.\cdots$ to denote the map $x \mapsto \cdots$).
3. $X \star Y =_{\text{df}} \lambda\!\!\lambda\varrho(\mathit{cost}(X\varrho) + \mathit{cost}(Y\varrho) + \mathit{cost}(\chi) + 1, \mathit{pot}(\chi))$ is a t.c. denotation of type $\|\tau\|$, where $\chi = \mathit{pot}(X\varrho)(\mathit{pot}(Y\varrho))$ (we write $\lambda\!\!\lambda\varrho.\ldots$ for $\varrho \mapsto \ldots$).
4. $\mathit{dally}(\ell, X) = \lambda\!\!\lambda\varrho(\ell + \mathit{cost}(X\varrho), \mathit{pot}(X\varrho))$ and for $\|\sigma\| = \mathsf{T} \times \mathsf{T}_L$, $\mathit{pad}(\ell, Y) = \lambda\!\!\lambda\varrho(\mathit{cost}(Y\varrho), \ell + \mathit{pot}(Y\varrho))$.
5. For $\|\sigma\| = \mathsf{T} \times \mathsf{T}_L$ and Z also a t.c. denotation of type $\|\sigma\|$, $(Z \uplus Y)\varrho = (\mathit{cost}(Z\varrho) + \mathit{cost}(Y\varrho), \mathit{pot}(Z\varrho) \vee \mathit{pot}(Y\varrho))$.

Theorem 1 (Decomposition Theorem). *Suppose $t \sqsubseteq X$ and Y_i is such that if $ft_1 \ldots t_k$ is a complete application of f in t, then $t_i \sqsubseteq Y_i$. Then*

$$t \sqsubseteq \lambda\!\!\lambda\varrho\left(X\varrho_\varepsilon \uplus \mathit{pad}\big(\mathit{tail\_len}(f,t), \varrho f \star Y_1\varrho_\varepsilon \star \cdots \star Y_k\varrho_\varepsilon\big)\right)$$

where $\varrho_\varepsilon = \varrho[f \mapsto \lambda\!\!\lambda_\star \boldsymbol{v}.(1,0)]$ and $\mathit{tail\_len}(f,t)$ is defined in Footnote 2.

Intuitively, the cost of "getting to" the recursive call is covered by $X\varrho_\varepsilon$, and the cost of the call itself by $\varrho f \star Y_1\varrho_\varepsilon \star \cdots \star Y_k\varrho_\varepsilon$, taking into account any c_a operations after the call (this is an over-estimate if no recursive call is made). The potential (size in this case, since t is of base type) is either independent of any complete application of f or is equal to the size of such an application, again taking into account later c_a operations.

Definition. A *decomposition function* for t is a function $d(\varrho^{\|\Gamma_{\boldsymbol{v}}\|\text{-Env}}, \chi^{\|\gamma\|}) : \|\mathsf{b}\|$ such that $t \sqsubseteq \lambda\!\!\lambda\varrho.d(\varrho_\varepsilon, \varrho f)$ (recall that f is the affinely-restricted variable in t).

Recalling the evaluation rule for crec and the definition of $\sqsubseteq$, we see that we must understand how the closure $T_0\rho_1$ is evaluated for appropriate ρ. It is easy to see that in such an evaluation, the only sub-evaluations of closures over terms of the form T_m are evaluations of closures of the form $T_m\rho_{m+1}[\boldsymbol{v} \mapsto \boldsymbol{z\theta}]$ for some closures $z_i\theta_i$. For the closure $T_0\rho_1$ we say that *the clock is bounded by* K if in every such sub-evaluation we have that $|z_1| < K$.

For a decomposition function d define $\Phi_{d,K}(n) : \|\Gamma_{\boldsymbol{v}}\|\text{-Env} \to \|\mathsf{b}\|$ by

$$\begin{aligned}\Phi_{d,K}(0) &= \lambda\!\!\lambda\varrho.(2K+1, 0)\\ \Phi_{d,K}(n+1) &= \lambda\!\!\lambda\varrho.\, \mathit{dally}\big(2K+1,\ d\big(\varrho_\varepsilon, \mathit{dally}\big(2, (\lambda\!\!\lambda_\star \boldsymbol{v}.\Phi_{d,K}(n))\varrho\big)\big) \vee \big(1,0\big)\big)\end{aligned}$$

We will use $\Phi_{d,K}$ to bound T_ℓ.

Theorem 2 (Recomposition Lemma). *Suppose d is a decomposition function for t, $\rho \in \Gamma_{\boldsymbol{v}}$-Env, $\varrho \in \|\Gamma_{\boldsymbol{v}}\|$-Env, $\rho \sqsubseteq \varrho$, and. that in the evaluation of $T_0\rho_1$ the clock is bounded by K. Then $T_0\rho_1 \sqsubseteq \Phi_{d,K}(K - |a|)(\varrho[v_i \mapsto \mathit{val}(\varrho v_{ip})])$.*

The Recomposition Lemma tells us that $\Phi_{d,K}(n)$ gives us a bound on the time-complexity of our recursion scheme. What we must do now is to "solve" the recurrence used to define Φ and show that it is polynomially-bounded.

Theorem 3 (Bounding Lemma). *Suppose that in Theorem 1 we can assume that X and each Y_i are bounded by t.c. polynomials p and p_i, respectively. Assume further that p is $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe and p_i is $\langle\!\langle \mathsf{b}_i \rangle\!\rangle$-safe w.r.t. $\|\Gamma_{\boldsymbol{v}}\|$. Then there is a $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe polynomial $\|\Gamma_{\boldsymbol{v}}\|, K : \langle\!\langle \mathsf{b}_1 \rangle\!\rangle, n : \langle\!\langle \mathsf{b}_1 \rangle\!\rangle \vdash \varphi(K,n) : \|\mathsf{b}\|$ such that for all K and n, $\Phi_{d,K}(n) \le \varphi(K,n)$.*

Proof. Let d be the decomposition function for t given in Theorem 1. Using the definition of d we can find a $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe polynomial $\|\Gamma_{\boldsymbol{v}}\|, K : \langle\!\langle \mathsf{b}_1 \rangle\!\rangle \vdash (P_0(K), P_1) : \|\mathsf{b}\|$ and a recursive upper bound on $\Phi_{d,K}(n)\varrho$:

$$\begin{aligned} \Phi_{d,K}(0)\varrho &\le (2K+1, 0) \\ \Phi_{d,K}(n+1)\varrho &\le (P_0(K), P_1)\varrho \uplus \mathit{pad}(\ell, \Phi_{d,K}(n)\varrho[v_i \mapsto \mathit{val}(p_{ip}\varrho)]) \end{aligned}$$

where $\ell = \mathit{tail\_len}(f,t)$. An easy proof by induction shows that $\Phi_{d,K}(n) \le (nP_0(K)\xi^{n-1} + 2K + 1, n\ell + P_1\xi^{n-1})$ for $n \ge 1$, where $\xi^0 = \mathrm{id}$ and $(v_{ic}, v_{ip})\xi^{n+1} = \mathit{val}(p_{ip}\xi^n)$. Since $\ell \ne 0$ implies $\mathsf{b}_1 <: \mathsf{b}$, $n\ell + P_1\xi^{n-1}$ is bounded by a $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe polynomial provided that $P_1\xi^{n-1}$ is $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe. Since P_1 is $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe and type-correct substitution of safe polynomials into a safe polynomial yields a safe polynomial (shown in Section 8 of *ATS*), to prove the theorem it suffices to show that $p_{ip}\xi^n$ is a $\langle\!\langle \mathsf{b}_i \rangle\!\rangle$-safe polynomial for each i. The proof of this is essentially the proofs of the One-step and n-step lemmas of Section 10 in *ATS* (it is here that we use the remaining constraints on the types in the crec typing rule).

Proposition 4 (Termination Lemma). *Assume the hypotheses of Theorem 3 hold and that $\rho \sqsubseteq \varrho$. Then in the evaluation of $T_0\rho_1$ the clock is bounded by $p_{1p}\xi^1\varrho$, where ξ^1 is defined as in the proof of Theorem 3.*

Proof. This follows from the details of the proof of Theorem 3.

Theorem 5 (Soundness Theorem). *For every* ATR *term $\Gamma; \Delta \vdash t : \tau$ there is a $\mathit{tail}(\|\tau\|)$-safe t.c. denotation X of type $\|\tau\|$ w.r.t. $\|\Gamma; \Delta\|$ such that $t \sqsubseteq X$.*

Proof. The proof is by induction on terms; for non-crec terms it is essentially as in *ATS*. For $\Gamma; \_ \vdash \mathsf{crec}\, a(\lambda_r f.\lambda \boldsymbol{v}.t) : \mathbf{b} \to \mathsf{b}$, suppose $\tilde{\rho} \in \Gamma$-Env, $\tilde{\varrho} \in \|\Gamma\|$-Env, $\rho \sqsubseteq \varrho$. Use the Bounding, Termination and Recomposition Lemmas to show that $(\lambda \boldsymbol{v}.T_0)\tilde{\rho}_1 \sqsubseteq (\lambda\!\!\!\lambda_\star \boldsymbol{v}.\varphi(p_{1p}\xi^1, p_{1p}\xi^1 - |a|))\tilde{\varrho}$, where p_1, φ, and ξ^n are as in the proof of the Bounding Lemma. We conclude that $\mathsf{crec}\, a\, (\lambda_r f.\lambda \boldsymbol{v}.t) \sqsubseteq \mathit{dally}(1, \lambda\!\!\!\lambda_\star \boldsymbol{v}.\varphi(p_{1p}\xi^1, p_{1p}\xi^1 - |a|))$. Since this last time-complexity is a $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe polynomial, the claim is proved.

Corollary 6. *If $\_;\_ \vdash t : \tau$, then t is computable in type-2 polynomial time.*

Soundness for recursion in an argument. We now address the recursions used in insertion-sort, in which the recursive use of the function occurs inside an argument to a previously-defined function. What we are really after here is structural (primitive) recursion for *defined* datatypes (such as our defined lists). First we adapt our $\to$**-E** rule to allow affine variables to appear in arguments to applications. We still require some restrictions in order to ensure a one-use property; the following is more than sufficient for our needs:

$$\frac{\Gamma; \Delta_0 \vdash s : \sigma \to \tau \qquad \Gamma; \Delta_1 \vdash t : \sigma}{\Gamma; \Delta_0 \cup \Delta_1 \vdash st : \tau}$$

where at most one of Δ_0 and Δ_1 are non-empty, and if *level* $\sigma > 0$, then $\Delta_1 = \emptyset$. Thus an affine variable f may only occur in t if t is of base type, and may not occur simultaneously in s and t. In particular, it is safe for β-reduction to copy a completed f-computation, but not an incomplete one. To simplify notation for the recursion present in insertion-sort we consider the special case in which we allow typings of the form $(*)$ provided $t = \mathsf{if}\ s'\ \mathtt{then}\ s(f\boldsymbol{t})\ \mathtt{else}\ s''$ where f is not free in s' or s'' (we treat the general case in the full paper).

First we must find a decomposition function. Assuming that $s \sqsubseteq X_s$, $t \sqsubseteq X_t$, and $t_i \sqsubseteq Y_i$, we can take as our decomposition function

$$d(\varrho, \chi) = X_t\varrho \uplus \big(cost\big(X_s\varrho\big) + cost\big(\chi \star \boldsymbol{X\varrho}\big) + cost\big(pot(X_s\varrho)(pot(\chi \star \boldsymbol{X\varrho}))\big),$$
$$pot\big(pot(X_s\varrho)(pot(\chi \star \boldsymbol{X\varrho}))\big)\big)$$

where we have written $\chi\star\boldsymbol{X\varrho}$ for $\chi\star X_1\varrho\star\cdots\star X_k\varrho$. Assume the inductively-given bounding t.c. denotations are bounded by safe polynomials p_s, p_t, and $p_1, \ldots, p_k$. The Soundness Theorem follows from the Recomposition Lemma provided we have a polynomial bound on $\Phi_{d,K}(n)$, so now we establish such a bound.

When b is oracular, then since p_{sp} $(= pot(p_s))$ is $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe, we have that $p_{sp} = \lambda z^{\langle\!\langle \mathsf{b} \rangle\!\rangle}.(p, q_s \vee (r_s \vee z))$ where q_s is $\langle\!\langle \mathsf{b} \rangle\!\rangle$-strict and r_s is $\langle\!\langle \mathsf{b} \rangle\!\rangle$-chary and does not contain z. We can therefore find a $\langle\!\langle \mathsf{b} \rangle\!\rangle$-safe t.c. polynomial $(P_0(K, z^{\langle\!\langle \mathsf{b} \rangle\!\rangle}), P_1)$ and derive the following recursive bound on $\Phi_{d,K}$ using the same conventions as in our analysis of cons-tail recursion:

$$\Phi_{d,K}(0)\varrho \leq (2K+1, 0)$$
$$\Phi_{d,K}(n+1)\varrho = (P_0(K, pot(\Phi_{d,K}(n)\varrho')), P_1) \uplus \Phi_{d,K}(n)\varrho'$$

where $\varrho' = \varrho[v_i \mapsto val(p_{ip}\varrho)]$. It is an easy induction to show that for $n \geq 1$ $\Phi_{d,K}(n) \leq ((n \cdot P_0(K, P_1) + 2K + 1)\xi^{n-1}, P_1\xi^{n-1})$ and thus the Bounding and Termination Lemmas that must be proved are exactly those of before.

When b is computational a similar calculation yields the bounding polynomial $((n \cdot P_0((n-2)q_s + P_1) + 2p_{1p})\xi^{n-1}, (n-1)q_s\xi^{n-2} + P_1\xi^{n-1})$ for a $\langle\!\langle \mathsf{b} \rangle\!\rangle$-strict polynomial q_s.

4 Concluding Remarks

In *ATS* we introduced the formalism ATR which captures the basic feasible functionals at type-level ≤ 2. We have extended the formalism with recursion schemes that allow for more natural programming and demonstrated the new formalism by implementing lists of binary strings and insertion-sort and showing that the new recursion schemes do not take us out of the realm of feasibility. We

```
val g :N_ε → N_ε → N_◇ =
    fn y z ⇒ if leq y (head z) then cons y z
               else cons (head z) (cons y ( tail z))

val select :N_ε → N_ε =
    fn l ⇒ letrec sel :N_ε → N_ε → N_ε =
        fn b l' ⇒ if tail(l') then down (g (head l') (sel b ( tail l'))) l' else l'
    in sel l l end

val sel_sort :N_ε → N_◇ =
    fn l ⇒ letrec ssort :N_ε → N_ε → N_◇ =
        fn b l' ⇒ let val m = select l' in cons (head m) (ssort b ( tail m)) end
    in ssort l l end
```

Fig. 6. Selection-sort in ATR. Note: let val x=s in t end abbreviates (fn $x \Rightarrow t)s$ where we restrict x to be of base type.

have also given a strategy for proving that particular forms of recursion can be "safely" added to the base system. Here we indicate some future directions:

More general affine recursions. In the full paper we give a definition of *plain affine recursion* that generalizes cons-tail recursion, allows recursive calls in arguments, and permits recursive calls in the body of let-expressions. In particular, it covers all forms of recursion used in the list operations and insertion- and selection-sort (code for the latter is in Figure 6). At the time of writing, we do not have all the details of the soundness argument in the general case, but we expect it to follow the framework we have developed here.

Lazy ATR*.* A version of ATR with lazy constructors (streams) and evaluation would be very interesting. There are many technical challenges in analyzing such a system but again we expect that the general outline will be the approach we have used in this paper. Of course one can implement streams in the current call-by-value setting in standard ways (raising the type-level), but a direct lazy implementation of streams is likely to be more informative. We expect the analysis of such a lazy-ATR to require an extensive reworking of the various semantic models we have discussed here and in *ATS*.

Real-number algorithms. ATR is a type-2 language, but here we have focused on type-1 algorithms. We are working on implementing real-number algorithms, viewing a real number as a type-1 (stream) oracle. This can be done in either a call-by-value setting (e.g., algorithms that take a string of length n as input and return something like an n-bit approximation of the result) or a lazy setting (in which the algorithm returns bits of the result on demand).

References

[1] Aehlig, K., Schwichtenberg, H.: A syntactical analysis of non-size-increasing polynomial time computation. ACM Transactions on Computation Logic 3(3), 383–401 (2002), http://doi.acm.org/10.1145/507382.507386

[2] Barber, A.: Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Laboratory for Foundations of Computer Science, (1996) http://www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347/index.html

[3] Bellantoni, S., Cook, S.: A new recursion-theoretic characterization of the polytime functions. Computational Complexity 2(2), 97–110 (1992) http://dx.doi.org/10.1007/BF01201998

[4] Cook, S., Urquhart, A.: Functional interpretations of feasibly constructive arithmetic. Annals of Pure and Applied Logic 63(2), 103–200 (1993) http://dx.doi.org/10.1016/0168-0072(93)90044-E

[5] Danner, N., Royer, J.S.: Adventures in time and space. In: Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Charleston, SC 2006), pp. 168–179, Association for Computing Machinery. New York (2006) http://doi.acm.org/10.1145/1111037.1111053

[6] Danner, N., Royer, J.S.: Adventures in time and space. To appear in Logical Methods in Computer Science; full version, at http://arxiv.org/abs/cs/0612116

[7] Frederiksen, C.C., Jones, N.D.: Recognition of polynomial-time programs. Technical Report TOPPS/D-501, DIKU, University of Copenhagen, (2004) http://www.diku.dk/topps/bibliography/2004.html

[8] Hofmann, M.: Linear types and non-size-increasing polynomial time computation. Information and Computation 183(1), 57–85 (2003) http://dx.doi.org/10.1016/S0890-5401(03)00009-9

[9] Hofmann, M.: The strength of non-size increasing computation. In: Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Portland, OR, 2002), pp. 260–269. ACM Press, New York (2002) http://doi.acm.org/10.1145/503272.503297

[10] Jones, N.D.: The expressive power of higher-order types or, life without cons. Journal of Functional Programming 11(1), 55–94 (2001) http://dx.doi.org/10.1017/S0956796800003889

[11] Leivant, D.: Ramified recurrence and computational complexity I: Word recurrence and poly-time. In: Feasible Mathematics II (Ithaca, NY, 1992), pp. 320–343. Birkhäuser Boston, Boston, MA (1995)

[12] Mehlhorn, K.: Polynomial and abstract subrecursive classes. In: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (Seattle, WA, 1974), pp. 96–109. ACM Press, New York, NY, USA (1974) http://doi.acm.org/10.1145/800119.803890

Algebraic Model of an Arithmetic Unit for TTE-Computable Normalized Rational Numbers

Gregorio de Miguel Casado, Juan Manuel García Chamizo, and María Teresa Signes Pont

SPA-Lab,University of Alicante,
03690, San Vicente del Raspeig, Alicante, Spain
{demiguel,juanma,teresa}@dtic.ua.es
http://www.dtic.ua.es/spa-lab/index.html

Abstract. A formal specification of an arithmetic unit for computable normalized rational numbers is proposed. This specification, developed under the scope of the paradigm known as algebraic models of processors, exploits the connection between the signed digit representation for rational numbers in Type-2 Theory of Effectivity and online arithmetic in Computer Arithmetic. The proposal aims for specification formalization and calculation reliability together with implementation feasibility.

Keywords: formal methods for VLSI design, Type-2 Theory of Effectivity, online arithmetic.

1 Introduction

Since the inception of computers, the progress of many scientific fields has become more and more dependent on computer technology advances. As a matter of fact, scientific and engineering computing raise an increasing reliability demand according to the complexity growth of mathematical and physical models. A remarkable example is the field of Physical Oceanography, in which the continuous growth of computing power provides the means to better understand the ocean by combining theory and observations with computer models. Therefore, in this field new computer science paradigms are demanded [15]. In this context, advances in Computable Analysis allow for dealing with the computability and complexity issues and then guarantee the reliability in software development [17].

Some researchers have focused their efforts on the development of specialized software libraries for reliable scientific computing [9]. The IRRAM C++, developed by N. Müller, has shown to be one of the most successful approaches [3]. The interest of these software libraries is motivated by the lack of reliability of the IEEE754/854 floating point standard hardware support for scientific computing applications [14] [16]. An attempt to introduce a better standard based on Interval Arithmetic did not success due to both commercial and theoretical reasons, which led to its rejection [12]. Despite these drawbacks, a recent review by J. Blanck for exact real arithmetic with centred intervals [4] has being done.

Within all the approaches to exact real arithmetic, we claim that signed digit arithmetic resembles an interesting approach [2] as a conceptual convergence between two paradigms which belong to two different Computer Science fields can be realized: Type-2 Theory of Effectivity (TTE) [17], in Computable Analysis, and online arithmetic [7], in Computer Arithmetic. The former, developed by Klaus Weihrauch, bases some computable representations of real numbers on signed digit representations of rational and real numbers which, at the same time, establish the basis for higher abstraction level representations such as common computable spaces of functions. The latter, developed by Trivedi and Ercegovac, deals with the hardware implementation of digit-serial left-to-right (online or Most Significant Bit First) arithmetic operators for signed digit numbers, whose operation dynamics resembles that of the Turing Machine model.

Finally, we summarize some previous work in VLSI design. Within the theoretical research in formal methods, algebraic models for specifying and verifying processors is being consolidated [10] [8] [11]. Complementarily, advances in VLSI and ULSI integration methods for memory technologies provide novel applications [13]. Also, the well known memory growth trend in the whole memory hierarchy of computer systems can still be realized as time goes by [5]. Finally, we remark an interesting processor technology trend based on the hybrid chip approach [1] which deals with the development of conventional processors which also embed Field Programmable Gate Array (FPGA) capabilities. This particular approach explores the possibilities of specialized hardware support embedded in conventional processors for end user applications, keeping the low cost advantages stemming from high scale manufacturing processes.

From the point of view of Computer Architecture, correctness and computability criteria are conventionally considered hardly applicable issues due to the limited utility of mathematical processor models and the inherent limited nature of physical hardware resources, respectively. Nevertheless, we consider that the technology advances analyzed motivate the introduction of paradigms of correctness and computability for designing an arithmetic unit devoted to basic arithmetic operations with TTE-computable signed digit rational numbers. This research extends [6] by exploiting the conceptual connection between TTE and online arithmetic in order to design an arithmetic unit for the operations of addition, multiplication and division, under the scope of the Fox-Harman-Tucker algebraic models of processors.

The paper is organized in the following sections: after the introduction, a normalized signed digit representation for non periodic normalized rational numbers is introduced; the connection between TTE and online arithmetic is analyzed in Section 3 and the arithmetic unit algebraic specification is developed in Section 4. Finally, Section 5 summarizes the conclusions drawn from this research.

2 Computable Normalized Rational Number Signed Digit Notation

Let $\Sigma = \{\overline{1}, 0, 1\}$ be a finite set. Let Σ^* and Σ^w denote the sets of finite and infinite sequences over Σ, respectively. We always assume that there exists an element $\# \notin \Sigma$ and denote $\Sigma_\# := \Sigma \cup \{\#\}$. Let $\iota_w : \Sigma^* \to \Sigma_\#^*$ be a wrapping function that assigns to a word $u = u_0 \ldots u_k \in \Sigma^*$ with $u_0 \ldots u_k \in \Sigma$ and $k \in \mathbb{N}$ the word $\iota_w(u) := \#\#u_0\#u_1\#...\#u_k\#\#$ and let $\iota_u : \Sigma^*_{\#} \to \Sigma^*$ be the corresponding unwrapping function that obtains from a word $v \in \Sigma^*_{\#}$ the word $\iota_u(v) := w$ with $w = w_0 \ldots w_k \in \Sigma^*$, and $k \in \mathbb{N}$. Define the standard representations in TTE of the natural and rational numbers as $\nu_{\mathbb{N}}$ and $\nu_{\mathbb{Z}}$, respectively as in [17, Def. 3.1.2]. Define the unpacking function $s : \Sigma^*_{\#} \to \Sigma^* \times \Sigma^*$ such as for a word $w = uv$ and $w, u, v \in \Sigma^*_{\#}$:

$$s\left(w\right) = s(uv) := \left\{ \left(\iota_u\left(u\right), \iota_u\left(v\right)\right) \ \mid u \sqsubseteq w \text{ and } v \text{ is a suffix of } w \right\}. \qquad (1)$$

Define the count function $c : \operatorname{dom}\left(\nu_{sd}\right) \to \operatorname{dom}\left(\nu_{\mathbb{N}}\right)$, which outputs a natural number corresponding to the position of the dot "$\centerdot$" in the input word. If the dot "$\centerdot$" is not found, then the function outputs the length of the word.

Define the function $a : \Sigma^* \times \operatorname{dom}\left(\nu_{\mathbb{N}}\right) \to \Sigma^* \times \operatorname{dom}\left(\nu_{\mathbb{Z}}\right)$ which removes the 0s at the beginning of an input string $u \in \Sigma^*$ and successively decrements the input $n \in \operatorname{dom}\left(\nu_{\mathbb{N}}\right)$. This function outputs the remaining string $m \in \Sigma^*$ and the decremented input n, which can be negative ($z \in \operatorname{dom}\left(\nu_{\mathbb{Z}}\right)$).

Finally, define the function $i : \Sigma^* \times \operatorname{dom}\left(\nu_{\mathbb{Z}}\right) \to \operatorname{dom}\left(\nu_{sd}\right)$ as

$$i\left(u, z\right) := \begin{cases} 0\centerdot\{0\}^z u & \text{if } z - 1 \leq 0, \\ u_0 \ldots u_{z-1} \centerdot u_z \ldots u_k & \text{if } 0 < z - 1 < \nu_{\mathbb{Z}}\left(k\right), \\ u_0 \ldots u \centerdot \{0\}^{z - \nu_{\mathbb{Z}}(k) - 1} & \text{if } z - 1 \geq \nu_{\mathbb{Z}}\left(k\right), \end{cases} \qquad (2)$$

where $u = u_0 \ldots u_k \in \Sigma^*$, $z \in \operatorname{dom}\left(\nu_{\mathbb{Z}}\right)$ and $k \in \mathbb{N}$.

Now, a normalized notation ν_{nsd} is introduced in order to codify names of non periodic rational numbers with exponent $e \in \mathbb{Z}$ and mantissa $m \in \mathbb{Q}$ by using the notation ν_{sd} [17, Def. 7.2.4]. This notation ν_{nsd} aims for simplifying both the hardware design of the arithmetic operators and final data memory packaging and storage. In addition, in order to match both real circuit design and data memory storage, the proposed representation has to be finally split into positive and negative parts by translating both into the standard $\nu_{b,2}$ notation, as it will be shown in Section 4.

$$\begin{aligned} &\nu_{sd}^{\exp} :\subset \Sigma^* \longrightarrow \mathbb{Z}, \\ &\operatorname{dom}\left(\nu_{sd}^{\exp}\right) := \begin{cases} \text{all } a_n \ldots a_0 \in \Sigma^* \quad \text{for } n \geq 0, \\ a_i \in \Sigma \text{ for } i \leq n, \\ a_n \neq 0, \text{ if } n \geq 0 \text{ and } a_n a_{n-1} \notin \left\{1\overline{1}, \overline{1}1\right\}, \text{ if } n \geq 1, \end{cases} \\ &\nu_{sd}^{\exp}\left(a_n \ldots a_0\right) := \rho_{sd}\left(a_n \ldots a_0 0^w\right). \\ &\nu_{sd}^{\text{man}} :\subset \Sigma^* \longrightarrow \mathbb{Q}, \\ &\operatorname{dom}\left(\nu_{sd}^{\text{man}}\right) := \left\{u\centerdot v \mid u = 0, v \in \Sigma^*, \ u\centerdot v0^w \in \operatorname{dom}\left(\rho_{sd}\right)\right\}, \\ &\nu_{sd}^{\text{man}}\left(u\centerdot v\right) := \rho_{sd}\left(u\centerdot v0^w\right). \end{aligned} \qquad (3)$$

$$\begin{aligned} &\nu_{nsd} :\subset \Sigma^* \longrightarrow \mathbb{Q}, \\ &\operatorname{dom}\left(\nu_{nsd}\right) := \left\{\iota_w(e)\iota_w(m) \middle| \ e \in \operatorname{dom}\left(\nu_{sd}^{\exp}\right), \ m \in \operatorname{dom}\left(\nu_{sd}^{\text{man}}\right)\right\}, \\ &\nu_{nsd}\left(uv\right) := \nu_{sd}\left(e \cdot m\right) \ \text{ with } e = \iota_u\left(u\right) \text{ and } m = 0\centerdot\iota_u\left(v\right). \end{aligned} \qquad (4)$$

Theorem 1. *The notation ν_{nsd} induces the same concept of computability than the standard notation ν_{sd} ($\nu_{nsd} \equiv \nu_{sd}$)*

Proof. ($\nu_{sd} \leq \nu_{nsd}$). The translation from ν_{sd} notation into ν_{nsd} notation can be achieved using TTE-computable string manipulation functions:

1. Count the number of positions from the beginning of the string until the dot character "**.**" is found with the function $c : \mathrm{dom}\,(\nu_{sd}) \rightarrow \mathrm{dom}\,(\nu_{\mathbb{N}})$. This will provide the initial exponent.
2. Remove the dot character "**.**" from the string and then remove the 0s at the beginning of it. At the same time, while removing the 0s, successively decrement the exponent by applying the function $a : \Sigma^* \times \mathrm{dom}\,(\nu_{\mathbb{N}}) \rightarrow \Sigma^* \times \mathrm{dom}\,(\nu_{\mathbb{Z}})$. The mantissa and the exponent at the output are the remaining chunk of the string and the decremented exponent, respectively.
3. Apply the wrapping function in order to wrap the exponent and the mantissa: $\iota_w(e) := u = \#\#e_0\#e_1\#...\#e_k\#\#$, $\iota_w(m) := v = \#\#m_0\#m_1\#...\#m_k\#\#$, and then concatenate u and v such as $w = uv \in \mathrm{dom}\,(\nu_{nsd})$.

Proof. ($\nu_{nsd} \leq \nu_{sd}$). The translation of the ν_{nsd} notation into ν_{sd} notation can be obtained by using TTE-computable string manipulation functions:

1. Apply the search function $s : \Sigma^*_\# \rightarrow \Sigma^* \times \Sigma^*$ to obtain the unwrapped exponent u and the mantissa $v : s\,(w) = (\iota_u\,(x)\,, \iota_u\,(y))\,,$ with $x, y, w \in \Sigma^*_\#$ and then $u = \iota_u\,(x)$ and $v = \iota_u\,(y)\,.$
2. Adjust the exponent by applying the function $v = i\,(u, z)\,, v \in \mathrm{dom}\,(\nu_{sd})\,, u \in \Sigma^*$ and $z \in \mathrm{dom}\,(\nu_{\mathbb{Z}})\,.$

Then, as ν_{sd} is reducible to ν_{nsd} ($\nu_{sd} \leq \nu_{nsd}$) and also ν_{nsd} is reducible to ν_{sd} ($\nu_{nsd} \leq \nu_{sd}$) it can be concluded that ν_{nsd} is equivalent to ν_{sd} ($\nu_{nsd} \equiv \nu_{sd}$).

3 TTE and Online Arithmetic

Digit-serial arithmetic deals with numerical value operations with digit vectors applied at the input and delivered at the output one digit at a time (serially) so that all the digits of the same numerical operand/results share the same digit lines. The main benefit of this approach is the reduction of the number of signal lines connecting modules and the simplification of their interfaces, since these connections and interfaces influence both area an energy dissipation. The drawback is the time (number of cycles) required to receive the inputs and to deliver the results. Nevertheless, this delay can be compensated by overlapping the execution of successive operations (even if dependent), since the successor operation can begin when a few digits of the operands have been received [7, Chap. 9].

In this section the general approach to online algorithm design is first introduced and then the key ideas of the algorithms for the addition/subtraction, multiplication and division are commented.

3.1 The Method for Online Algorithm Design

In the Most Significant Digit First (MSDF) serial mode operation or online arithmetic the total execution time is the sum of two components: the online delay δ, which corresponds to the additional number of operand digits required to determine the first result digit and the time to deliver the n output digits (n cycles for an output of n digits). Therefore, the execution time is $T_n = \delta + 1 + n$.

The general method for designing online algorithms consists in two parts:

1. Obtain the recurrence equation on the residual (internal state) $w_{j+1} \in \text{dom}\,(\nu_{nsd})$.
2. Obtain the result at a given iteration $z_{j+1} \in \text{dom}\,(\nu_{nsd})$.

Let $g : \text{dom}\,(\nu_{nsd}) \times \text{dom}\,(\nu_{nsd}) \times \Sigma \times \text{dom}\,(\nu_{nsd}) \times \Sigma \times \text{dom}\,(\nu_{nsd}) \times \Sigma \to \text{dom}\,(\nu_{nsd})$ and let $f : \text{dom}\,(\nu_{nsd}) \times \text{dom}\,(\nu_{nsd}) \times \Sigma \times \text{dom}\,(\nu_{nsd}) \times \Sigma \times \text{dom}\,(\nu_{nsd}) \to \text{dom}\,(\nu_{nsd})$ be computable functions such as

$$w_{j+1} := g\left(w_j, x_j, x_{j,j+1+\delta}, y_j, y_{j,j+1+\delta}, z_j, z_{j,j+1}\right), \tag{5}$$

$$z_{j+1} := f\left(w_j, x_j, x_{j,j+1+\delta}, y_j, y_{j,j+1+\delta}, z_j\right), \tag{6}$$

for $-\delta \leq j \leq n-1$, δ, $j \in \text{dom}\,(\nu_{\mathbb{Z}})$ and $n - 1 \in \text{dom}\,(\nu_{\mathbb{N}})$.

In the former equations $x_j, y_j \in \text{dom}\,(\nu_{nsd})$are the input operands, $x_{j,j+1+\delta}$, $y_{j,j+1+\delta} \in \Sigma$ are the signed digit digits $j + 1 + \delta$ of the input operands x_j, y_j in the position $j + 1 + \delta$, $z_j \in \text{dom}\,(\nu_{nsd})$ is the result at the iteration step j, $z_j \in \text{dom}\,(\nu_{nsd})$ is the result and $z_{j,j+1+\delta} \in \Sigma$ is the signed digit of the result z_j in the position $j + 1 + \delta$ at the iteration step j.

The computability of the algorithm is absolutely concerned with the possibility to reduce the recurrence equation on the residual to additions and shifts. The proof is omitted due to space constraints.

3.2 Computable Algorithms for Addition, Subtraction, Multiplication and Division

The algorithms for addition/subtraction, multiplication and division have as keystone modules the signed digit adder and the bit shift register. These have being programmed with Visual Basic .NET for supporting calculation procedures in physical oceanography models, such as numerical approximation of integrals [6]. As the software library has implemented as a test bench for a later hardware implementation, it has not being optimized for a particular commercial processor and then the abstraction of the functions has being done by matching final arithmetic hardware modules, as far as possible. This library uses the .NET link technology for connecting to the Mathematica kernel so that to handle numerical values from complex functions, managing graphics and perform error comparisons with IEEE 754/854 format.

4 Specification of the Arithmetic Unit

This section first introduces a functional specification of the arithmetic unit and then develops a part of the algebraic specification at programmer's level. Finally, the data memory packaging scheme of the normalized rational numbers and the set of memory mapping functions for the representation ν_{nsd} is provided.

4.1 Functional Specification

The arithmetic unit (AU) for arithmetic operations with the representation ν_{nsd} (Figure 1) consists of the following modules: the AU controller, the μInstruction Encoder, the cache memory and the arithmetic modules for signed digit addition, signed digit complement (not), shift, comparison, normalization and canonical signed digit recoding. The AU controller manages the virtual digit serial circuits (datapath) according to specific programs of μInstructions for the addition, subtraction, multiplication and division operations. The μInstruction Encoder translates the high abstraction level arithmetic operations into simpler machine instructions, which are associated to the datapath management of the virtual digit serial circuits related to the arithmetic operations. The AU controller also manages a bank or arithmetic registers and a cache memory for speeding up internal operations by reducing the number of accesses to the main memory. It also interfaces the CPU and the main memory system for loading and storing operands as well as partial and final results of the operations.

The instruction set of the processor consists of two management instruction and five arithmetic operations:

– *Status_request(): Idle*$\vee$*Busy*$\vee$*Error(1).* Provides the status of the processor: Idle or Busy.
– *Halt(): Ack*$\vee$*OPs*$\vee$*Error(2)* This is an instruction with priority for stopping the AU. It returns the number of operations done.
– *Add(addRA, lPrec, hPrec,addA, addB, addR):Ack*$\vee$*OPs*$\vee$*Error(3).*
– *Sub(addRA, lPrec, hPrec,addA, addB, addR):Ack*$\vee$*OPs*$\vee$*Error(4).*
– *Mul(addRA, lPrec, hPrec,addA, addB, addR):Ack*$\vee$*OPs*$\vee$*Error(5).*
– *Div(addRA, lPrec, hPrec,addA, addB, addR):Ack*$\vee$*OPs*$\vee$*Error(6).*
– *Not(addRA, lPrec, hPrec,addA, addB, addR):Ack*$\vee$*OPs*$\vee$*Error(7).*
The arithmetic operations return *Ack* (acknowledgement) when completed or the number of operations performed when the operation is halted.

According to estimations done with a register transfer algorithmic approach, a μinstruction counter $AUIC$ and a the following banks of are needed:

– Configuration Registers (CR): LP and HP (precision bounds).
– Base-Address Registers (BA): $AA \leftarrow addA$, $AB \leftarrow addB$, $AC \leftarrow addC$.
– Status Registers (SR): CS for the current status of the AU, μCI for the current μinstruction, OR for result storage, RA result storage address, OC for operation counter.
– Arithmetic Registers (AR). Provide arithmetic support (16 registers, according to a register-to-register algorithmic approach for the operations).

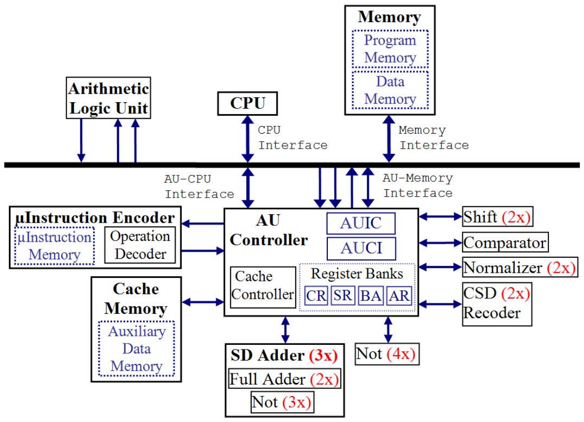


Fig. 1. Arithmetic Unit schematic and connection with the system bus

4.2 Algebraic Specification: Programmer's Level

By introducing an algebraic model for the arithmetic unit formal behavior methods over time and data representation as well as operations can be isolated. In this section, an algebraic specification of the arithmetic unit at programmer's level is developed following [10]. The instruction delay is chosen as abstract system clock T.

The next subsection develops the state and the next-state algebras. The machine algebra and the next-state and output functions are omitted due to space constraints.

The State and Next-State Algebras. The arithmetic unit state consists of the μinstruction counter $AUIC$, the register banks CR, BA, SR, AR, the cache and the main memory system. The register $AUCI$ holds the instruction addressed by the CPU. The μinstruction memory μIM, cache CM and data memory DM are modeled as a mapping of a subset of the natural numbers into the binary numbers: $\mu IM, CM, DM :\subseteq N \times B2$, $N \subseteq \mathrm{dom}\,(\nu_{\mathbb{N}})$ and $B2 \subseteq \mathrm{dom}\,(\nu_{b,2})$. Notice that the address space DM holds the addresses of the operands and the result of the operation.

The state of the machine is defined by:

$$Cc = AUIC \times CR \times BA \times SR \times AR \times Mem, \tag{7}$$

where $Mem = AUIC \longrightarrow CM \cup DM$.

There are 5 inputs with the corresponding outputs:

$$IIn = \{Status\_request(), Halt(), Add(), Sub(), Mul(), Div(), Not()\}, \tag{8}$$

$$\begin{aligned} IOut = \{&Idle \vee Busy \vee Error(1), Ack \vee OPs \vee Error(2),\\ &Ack \vee OPs \vee Error(3), Ack \vee OPs \vee Error(4), Ack \vee OPs \vee Error(5),\\ &Ack \vee OPs \vee Error(6),\ Ack \vee OPs \vee Error(7)\}. \end{aligned} \tag{9}$$

Therefore, the input and output of the arithmetic unit at time $t \in T$ are

$$In = IIn \times AUIC, \tag{10}$$

$$Out = IOut \times AUCI \times Mem, \tag{11}$$

where an instruction is input in $AUCI$ as a request from the CPU. The current μinstruction register $SR.\mu CI$ is updated during the arithmetic operation until the output of the machine is stored in the main memory.

The state algebra can be expressed as:

Algebra Arithmetic Unit State
Sets $T, Cc, In, Out, [T \longrightarrow In]$
Operations $CC : T \times Cc \times [T \longrightarrow In] \longrightarrow Cc \times Out$
End Algebra

CC is defined as:

$$\begin{aligned} &CC_1 (0, g, i) = g, \\ &CC_1 (t+1, g, i) = cc\left(CC_1 (t, g, i), i(t)\right), \\ &CC_2 (0, g, i) = out\left(CC_1 (t, g, i)\right), \end{aligned} \tag{12}$$

where $cc : Cc \times In \longrightarrow Cc$ is the next-state function, and $out : Cc \longrightarrow Out$ is the output function.

Hence, the next-state algebra is defined as:

Algebra Arithmetic Unit Next-State
Sets $T, Cc, In, Out, [T \longrightarrow In]$
Constants $0 : T$
Operations
$t + 1 : T \longrightarrow T$
$cc : Cc \times In \longrightarrow Cc$
$out : Cc \longrightarrow Out$
$eval : T \times [T \longrightarrow In] \longrightarrow In$
End Algebra

The stream evaluation function eval : $T \times [T \longrightarrow In] \longrightarrow In$, eval (t, i) to it (t), is defined by primitive recursive equations over the next-state algebra.

4.3 Data Memory Organization

A realistic memory organization can be achieved by introducing indirect addressing for a flexible mapping process. Figure 2 shows a memory mapping example of a signed digit normalized rational number (exponent and mantissa) and its memory packaging (right-to-left and left-to-right data, respectively).

The storage in memory of source and result values can be done by mapping every value $u \in \text{dom}(\nu_{nsd})$ into a memory designed as an address space $M :\subseteq N \times B2$, $N \subseteq \text{dom}(\nu_{\mathbb{N}})$ and $B2 \subseteq \text{dom}(\nu_{b,2})$. This implies splitting each signed digit value into positive and negative and parts.

Two functions p_{Addr_nrm}, p_{Addr_val} are proposed for the memory mapping process. Their detailed definition is omitted due to space constraints.

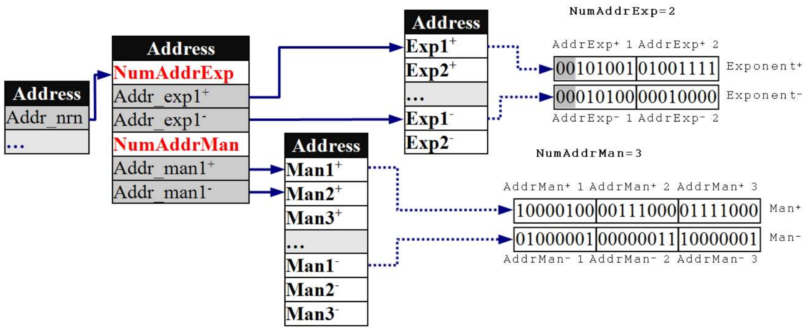


Fig. 2. Memory mapping and data packaging for the ν_{nsd} representation

p_{Addr_nrm} maps the number of addresses of the exponent and mantissa as well as the initial addresses of their values.

p_{Addr_val} maps the positive or negative part of a signed digit number in binary notation into a memory position.

5 Conclusions

An arithmetic unit devoted to the operation with a signed digit TTE representation for normalized rational numbers has being presented. The specification of the arithmetic unit has being done under the scope of the algebraic processor specification model of Fox, Harman and Tucker. By establishing a relationship between a generalization of the method for developing online algorithms in Computer Architecture and TTE, the computable arithmetic operators for addition/subtraction, multiplication and division have being analyzed. This result establishes bounds for the requirements of Type-2 Turing Machines in terms of the working tapes, with fixed time delays of operation for the arithmetic operators analyzed. Finally, the algebraic model of the arithmetic unit provides a formal framework for its description and verification and also helps in filling the gap between the TTE model and a feasible VLSI design.

As future work, the whole specification process, including the abstract circuit level and the verification sketch, and a VHDL prototype for the arithmetic unit presented are going to be developed.

References

1. Andrews, D., Sass, R., Anderson, E., Agron, J., Peck, W., Stevens, J., Baijot, F., Komp, E.: The Case for High Level Programming Models for Reconfigurable Computers. Proc. of the 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms, pp. 21–32 (2006)
2. Avizienis, A.: Signed-digit number representations for fast parallel arithmetic. IRE Trans. Electronic Computers 10, 389–400 (1961)

3. Blanck, J.: Exact real arithmetic systems: Results of competition, Computability and Complexity in Analysis. In: Blank, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 390–394. Springer, Heidelberg (2001)
4. Blanck, J.: Exact real arithmetic using centred intervals and bounded error terms. Journal of Logic and Algebraic Programming 66, 207–240 (2006)
5. Borkar, S.: Getting Gigascale Chips: Challenges and Opportunities in Continuing Moore's Law. ACM Queue 1(7), 26–33 (2003)
6. de Miguel Casado, G., García Chamizo, J.M.: The Role of Algebraic Models and Type-2 Theory of Effectivity in Special Purpose Processor Design. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 137–146. Springer, Heidelberg (2006)
7. Ercegovac, M.D., Lang, T.: Digital Arithmetic. M. Kaufmann, Seattle (2004)
8. Fox, A.C.J., Harman, N.A.: Algebraic Models of Correctness for Abstract Pipelines. The. Journal of Algebraic and Logic Programming 57(1-2), 71–107 (2003)
9. Gowland, P., Lester, D.: A Survey of Exact Arithmetic Implementations. In: Blank, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 30–47. Springer, Heidelberg (2001)
10. Harman, N.A., Tucker, J.V.: Algebraic models of microprocessors: the verification of a simple computer. In: Stravridou, V. (ed.) Mathematics of Dependable Systems II. Oxford: Clarendon Press, pp. 135–170. Oxford University Press, New York (1997)
11. Harman, N.A.: Models of Timing Abstraction in Simultaneous Multithreaded and Multi-Core Processors, Logical Approaches to Computational Barriers, Report Series, vol. CSR 7-2006, pp. 129–139 (2006)
12. Hayes, B.: A Lucid Interval. American Scientist 91, 484–488 (2003)
13. ISSCC Roundtable: Embedded Memories for the Future, IEEE Design and Test of Computers, vol. 20, pp. 66–81 (2003)
14. Lynch, T., Schulte, M.: A High Radix On-line Arithmetic for Credible and Accurate Computing. Journal of UCS 1, 439–453 (1995)
15. Post, D.E., Votta, L.G.: Computational science demands a new paradigm. Physics today 58(1), 35–41 (2005)
16. Schröder, M.: Admissible Representations in Computable Analysis. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 471–480. Springer, Heidelberg (2006)
17. Weihrauch, K.: Computable Analysis. Springer, Heidelberg (2000)

Feasible Depth

David Doty[1,⋆] and Philippe Moser[2,⋆⋆]

[1] Department of Computer Science, Iowa State University, Ames, IA 50011, USA
ddoty@iastate.edu

[2] Dept de Informática e Ingeniería de Sistemas, Centro Politécnico Superior, Zaragoza, Spain
mosersan@gmail.com

Abstract. This paper introduces two complexity-theoretic formulations of Bennett's logical depth: *finite-state depth* and *polynomial-time depth*. It is shown that for both formulations, trivial and random infinite sequences are shallow, and a *slow growth law* holds, implying that deep sequences cannot be created easily from shallow sequences. Furthermore, the E analogue of the halting language is shown to be polynomial-time deep, by proving a more general result: every language to which a non-negligible subset of E can be reduced in uniform exponential time is polynomial-time deep.

Keywords: dimension, depth, randomness, polynomial-time, finite-state.

1 Introduction

Whereas many structures found in nature are highly complex (a DNA sequence, a cell), some seem much simpler, either because of their complete regularity (ice), or their complete randomness (gas). Bennett introduced logical depth [3] to formalize computationally the difference between complex and non-complex (trivial or random) structures. Briefly, a logically deep object is one with a shorter description than itself, but which requires a long time to compute from this short description.

Depth is not a measure of information contained in an object, which correlates with *randomness*, but rather its value, or its *useful* information content. According to classical [18] or algorithmic information theory [14], the information content of a sequence is not representative of its value. Consider an infinite binary sequence produced by random coin tosses. Although the sequence contains a large amount of information in the sense that, with probability 1, it cannot be significantly compressed, its information is not of much value, except

⋆ Corresponding author. This author was partially supported by grant number 9972653 from the National Science Foundation as part of their Integrative Graduate Education and Research Traineeship (IGERT) program.

⋆⋆ This author was partially supported by subvenciones para grupos de investigación Gobierno de Aragón UZ-T27 and subvenciones de fomento de movilidad Gobierno de Aragón MI31/2005.

as a source of input to randomized algorithms. Contrast this with the characteristic sequence of the halting language, access to which enables any computably enumerable language to be decided in linear time. From this perspective, the halting sequence is much more useful than a randomly generated sequence.

Bennett's logical depth separates the sequences that are deep (i.e., that show high internal organization) from those that are shallow (i.e., not deep). Informally, deep sequences are those which contain redundancy, but in such a way that an algorithm requires extensive resources to exploit the redundancy (for instance, to compress or to predict the sequence). In other words, deep sequences are organized, but in a nontrivial way. Highly redundant sequences like 00000... are shallow, because they are trivially organized. Random sequences are shallow, because they are completely unorganized. One of the key features of Bennett's logical depth is that it obeys a *slow growth law* [3,11]: no fast process can transform a shallow sequence into a deep one. Therefore a deep object can be created only through a complex, time-consuming process.

Bennett [3] showed that the halting language is deep, arguing that its depth was evidence of its usefulness. Juedes, Lathrop, and Lutz [11] generalized this result and solidified the connection between usefulness and depth by proving that every *weakly useful* language [8] is deep, where a weakly useful language is one to which a nonnegligible subset of the decidable languages (in the sense of resource-bounded measure theory [15]) reduce in a fixed computable time bound.

Unfortunately, because it is based on Kolmogorov complexity, Bennett's logical depth is not computable. Lathrop and Lutz [13] investigated *recursive computational depth*, which is computable, but not within any feasible time scale. Antunes, Fortnow, van Melkebeek, and Vinodchandran [1] investigated several polynomial-time formulations of depth as instances of the more general concept of computational depth obtained by considering the difference between variants of Kolmogorov complexity. Deep and intriguing connections were demonstrated between depth and average-case complexity, nonuniform circuit complexity, and efficient search for satisfying assignments to Boolean formulas. Nevertheless, some of the depth notions in [1] require complexity assumptions to prove the existence of deep sequences, and not all the depth notions obey slow growth laws. Furthermore, [1] lacks a polynomial-time analogue of the Juedes-Lathrop-Lutz theorem demonstrating that useful objects are necessarily deep.

The aim of this paper is to propose a feasible depth notion that satisfies a slow growth law and in which deep sequences can be proven to exist. We propose two such notions: finite-state depth, and polynomial-time depth. Furthermore, we connect polynomial-time depth to usefulness in deciding languages in the complexity class E. In both cases, the definition of depth intuitively reflects that of Bennett's logical depth: a sequence is deep if it is redundant, but an algorithm requires extensive resources in order to exploit the redundancy.

Our formulation of finite-state depth is based on the classical model of finite-state compressors and decompressors introduced by Shannon [18] and investigated by Huffman [10] and Ziv and Lempel [20]. Informally, a sequence is finite-state deep if given more states, a finite-state machine can decompress the sequence from an input significantly shorter than is possible with fewer states. We show that both finite-state trivial sequences (sequences with finite-state strong dimension [2] equal to 0) and finite-state random sequences (those with finite-state dimension [6] equal to 1, or equivalently normal sequences [4]) are shallow. Our main result in this section shows that finite-state depth obeys a slow growth law: no information lossless finite-state transducer can transform a finite-state shallow sequence into a finite-state deep sequence. We conclude the section by proving the existence of finite-state deep sequences.

Our formulation of polynomial-time depth – contrary to finite-state depth – is not based on compression algorithms but on polynomial-time oblivious predictors. Given a language L, a polynomial-time oblivious predictor is a polynomial-time computable function that, given an input string x, predicts the probability that $x \in L$. Informally, L is polynomial-time deep if, given more time, a predictor is better able to predict membership of strings in L. We show that both E-trivial languages (languages in the complexity class E) and E-random languages are polynomial-time shallow. Our main results in this section are a slow growth law similar to that for finite-state depth and logical depth, and a theorem stating that any language which is "useful" for quickly deciding languages in E must be polynomial-time deep. It follows that H_{E}, the E version of the halting language, is polynomial-time deep.

2 Preliminaries

$\mathbb{N}$ is the set of all nonnegative integers. A *(finite) string* is an element of $\{0,1\}^*$. An *(infinite) sequence* is an element of the Cantor space $\mathbf{C} = \{0,1\}^\infty$. For a string or sequence S and $i, j \in \mathbb{N}$, $S[i\,..\,j]$ denotes the substring consisting of the i^{th} through the j^{th} bits of S, inclusive, and $S \upharpoonright n$ denotes $S[0\,..\,n-1]$. For a string x and a string or sequence S, we write $x \sqsubseteq S$ to denote that $x = S \upharpoonright n$ for some $n \in \mathbb{N}$. For a string x, its length is denoted by $|x|$. $s_0, s_1, s_2 \dots$ denotes the standard enumeration of the strings in $\{0,1\}^*$ in lexicographical order, where $s_0 = \lambda$ denotes the empty string. If x, y are strings, we write $x < y$ if $|x| < |y|$ or $|x| = |y|$ and x precedes y in alphabetical order, and $x \leq y$ if $x < y$ or $x = y$.

A *language* is a subset of $\{0,1\}^*$. A *class* is a set of languages. The *characteristic sequence* of a language L is the sequence $\chi_L \in \{0,1\}^\infty$, whose n^{th} bit is 1 if and only if $s_n \in L$. Because $L \mapsto \chi_L$ is a bijection, we will often speak of languages and sequences interchangeably, with it understood that the "sequence" L refers to χ_L, and the "language" χ_L refers to L. Let $\mathsf{E} = \bigcup_{c \in \mathbb{N}} \mathsf{DTIME}(2^{cn})$ and $\mathsf{EXP} = \bigcup_{c \in \mathbb{N}} \mathsf{DTIME}(2^{n^c})$.

Let $i \leq j \in \mathbb{N}$. The i^{th} projection function $\text{proj}_i : (\{0,1\}^*)^j \to \{0,1\}^*$, is given by $\text{proj}_i(x_1, \dots, x_j) = x_i$.

3 Finite-State Depth

3.1 Finite-State Compression

We use a model of finite-state compressors and decompressors based on finite-state transducers, which was introduced in a similar form by Shannon [18] and investigated by Huffman [10] and Ziv and Lempel [20]. Kohavi [12] gives an extensive treatment of the subject.

A *finite-state transducer (FST)* is a 4-tuple $T = (Q, \delta, \nu, q_0)$, where

– Q is a nonempty, finite set of *states*,
– $\delta : Q \times \{0,1\} \to Q$ is the *transition function*,
– $\nu : Q \times \{0,1\} \to \{0,1\}^*$ is the *output function*,
– $q_0 \in Q$ is the *initial state.*

Furthermore, we assume that every state in Q is reachable from q_0.

For all $x \in \{0,1\}^*$ and $a \in \{0,1\}$, define the *extended transition function* $\widehat{\delta} : \{0,1\}^* \to Q$ by the recursion $\widehat{\delta}(\lambda) = q_0$, and $\widehat{\delta}(xa) = \delta(\widehat{\delta}(x), a)$. For $x \in \{0,1\}^*$, we define the *output* of T on x to be the string $T(x)$ defined by the recursion $T(\lambda) = \lambda$, and $T(xa) = T(x)\nu(\widehat{\delta}(x), a)$ for all $x \in \{0,1\}^*$ and $a \in \{0,1\}$.

A FST can trivially act as an "optimal compressor" by outputting λ on every transition arrow, but this is, of course, a useless compressor, because the input cannot be recovered. A FST $T = (Q, \delta, \nu, q_0)$ is *information lossless (IL)* if the function $x \mapsto (T(x), \widehat{\delta}(x))$ is one-to-one; i.e., if the output and final state of T on input $x \in \{0,1\}^*$ uniquely identify x. An *information lossless finite-state transducer (ILFST)* is a FST that is IL. We write FST to denote the set of all finite-state transducers, and we write ILFST to denote the set of all information lossless finite-state transducers. We say $f : \{0,1\}^\infty \to \{0,1\}^\infty$ is *FS computable* (resp. *ILFS computable*) if there is a FST (resp. ILFST) T such that, for all $S \in \{0,1\}^\infty$, $\lim\limits_{n\to\infty} |T(S \upharpoonright n)| = \infty$ and, for all $n \in \mathbb{N}$, $T(S \upharpoonright n) \sqsubseteq f(S)$. In this case, define $T(S) = f(S)$.

The following well-known theorem [10,12] states that the function from $\{0,1\}^*$ to $\{0,1\}^*$ computed by an ILFST can be inverted – in an approximate sense – by another ILFST.

Theorem 3.1. *For any ILFST T, there exists an ILFST T^{-1} and a constant $c \in \mathbb{N}$ such that, for all $x \in \{0,1\}^*$, $x \upharpoonright (|x| - c) \sqsubseteq T^{-1}(T(x)) \sqsubseteq x$.*

Corollary 3.2. *For any ILFST T, there exists an ILFST T^{-1} such that, for all sequences S, $T^{-1}(T(S)) = S$.*

Fix some standard binary representation $\sigma_T \in \{0,1\}^*$ of each FST T, and define $|T| = |\sigma_T|$. For all $k \in \mathbb{N}$, define

$$\begin{aligned} \mathrm{FST}^{\leq k} &= \{T \in \mathrm{FST} : |T| \leq k\}, \\ \mathrm{ILFST}^{\leq k} &= \{T \in \mathrm{ILFST} : |T| \leq k\}. \end{aligned}$$

Let $k \in \mathbb{N}$ and $x \in \{0,1\}^*$. The *k-FS decompression complexity* (or when k is clear from context, *FS complexity*) of x is

$$\mathrm{D}^k_{\mathrm{FS}}(x) = \min_{p \in \{0,1\}^*} \left\{ \, |p| \; \middle| \; (\exists T \in \mathrm{FST}^{\leq k}) \; T(p) = x \, \right\},$$

i.e., the size of the smallest program $p \in \{0,1\}^*$ such that some k-bit FST outputs x on input p.

For a fixed k, $\mathrm{D}_{\mathrm{FS}}^{k}$ is a finite state analogue of Kolmogorov complexity. For any sequence S, define the *finite-state dimension* of S by

$$\dim_{\mathrm{FS}}(S) = \lim_{k\to\infty} \liminf_{n\to\infty} \frac{\mathrm{D}_{\mathrm{FS}}^{k}(S \upharpoonright n)}{n}, \tag{3.1}$$

and the *finite-state strong dimension* of S by

$$\mathrm{Dim}_{\mathrm{FS}}(S) = \lim_{k\to\infty} \limsup_{n\to\infty} \frac{\mathrm{D}_{\mathrm{FS}}^{k}(S \upharpoonright n)}{n}. \tag{3.2}$$

Finite-state dimension and strong dimension measure the degree of finite-state randomness of a sequence. The above definitions are equivalent [19,7] to several other definitions of finite-state dimension and strong dimension in terms of finite-state gamblers [6,2], entropy rates [20,5], information lossless finite-state compressors [20,6,2], and finite-state log-loss predictors [9].

Schnorr and Stimm [17] (and more explicitly, Bourke, Hitchcock, and Vinodchandran [5]) showed that a sequence has finite-state dimension 1 if and only if it is *normal* in the sense of Borel [4], meaning that for all $k \in \mathbb{N}$, every substring of length k occurs in S with limiting frequency 2^{-k}.

3.2 Finite-State Depth

Intuitively, a sequence is finite-state deep if a finite state transducer, given additional states (or more accurately, additional bits with which to represent the transducer), can decompress the sequence from a significantly shorter input.

Definition 3.3. *A sequence S is* finite-state deep *if*

$$(\exists \alpha > 0)(\forall k \in \mathbb{N})(\exists k' \in \mathbb{N})(\exists^{\infty} n \in \mathbb{N})\ \mathrm{D}_{\mathrm{FS}}^{k}(S \upharpoonright n) - \mathrm{D}_{\mathrm{FS}}^{k'}(S \upharpoonright n) \geq \alpha n.$$

A sequence S is finite-state shallow *if it is not finite-state deep.*

Remark. *All results in this section remain true if the quantification in the definition of finite-state depth is changed to*

$$(\forall k \in \mathbb{N})(\exists \alpha > 0)(\exists k' \in \mathbb{N})(\exists^{\infty} n \in \mathbb{N})\ \mathrm{D}_{\mathrm{FS}}^{k}(S \upharpoonright n) - \mathrm{D}_{\mathrm{FS}}^{k'}(S \upharpoonright n) \geq \alpha n.$$

Note that any sequence deep by the former definition must be deep by the latter definition.

Finite-state trivial and finite-state random sequences are finite-state shallow.

Proposition 3.4. *Let $S \in \mathbf{C}$.*

1. *If $\mathrm{Dim}_{\mathrm{FS}}(S) = 0$, then S is finite-state shallow.*
2. *If S is normal (i.e., if $\dim_{\mathrm{FS}}(S) = 1$), then S is finite-state shallow.*

Finite-state deep sequences cannot be created easily, as the following theorem shows. More precisely, no ILFST can transform a finite-state shallow sequence into a finite-state deep sequence.

Theorem 3.5 (Finite-state slow growth law). *Let S be any sequence, let $f : \{0,1\}^\infty \to \{0,1\}^\infty$ be ILFS computable, and let $S' = f(S)$. If S' is finite-state deep, then S is finite-state deep.*

Theorem 3.6. *There exists a finite-state deep sequence.*

4 Polynomial-Time Depth

Because the time bound defining polynomial-time depth is in terms of the characteristic sequence of a language, we focus on the class E of languages decidable in time $2^{c|s_n|}$ for a fixed $c \in \mathbb{N}$, or equivalently, n^c, where n is the length of the characteristic sequence of a language up to the string s_n.

4.1 Measure in E

We use Lutz's measure theory for the complexity class E, which we now briefly describe. See [16] for more details.

Measure on E is obtained by imposing appropriate resource bounds on a game theoretical characterization of the classical Lebesgue measure of subsets of $\mathbf{C}$. A *martingale* is a function $d : \{0,1\}^* \to [0,\infty)$ such that, for every $w \in \{0,1\}^*$,

$$d(w) = \frac{d(w0) + d(w1)}{2}.$$

We say that a martingale d *succeeds* on a language L if $\limsup_{n\to\infty} d(L \upharpoonright n) = \infty$. Intuitively, d is a gambler that bets money on each successive bit of χ_L, doubling the money bet on the bit that occurs, and losing the rest. It succeeds by making unbounded money.

A class of languages $\mathcal{C}$ has p-*measure zero*, and we write $\mu_{\mathrm{p}}(\mathcal{C}) = 0$, if there is a polynomial-time computable martingale that succeeds on every language in $\mathcal{C}$. $\mathcal{C}$ has *measure zero in* E, denoted $\mu\left(\mathcal{C} | \, \mathsf{E}\right) = 0$, if $\mathcal{C} \cap \mathsf{E}$ has p-measure zero. A class $\mathcal{C}$ has p-*measure one*, denoted $\mu_{\mathrm{p}}(\mathcal{C}) = 1$, if $\overline{\mathcal{C}}$ has p-measure zero, where $\overline{\mathcal{C}}$ denotes the complement of $\mathcal{C}$, and $\mathcal{C}$ has *measure one in* E, denoted $\mu\left(\mathcal{C} | \, \mathsf{E}\right) = 1$, if $\mathsf{E} - \mathcal{C}$ has p-measure zero. We say that a language L is E-*random* if the singleton $\{L\}$ does not have p-measure zero.

Measure in E yields a size notion on the class E similar to Lebesgue measure on the Cantor space. Subsets of E that have p-measure zero are then "small subsets of E"; for example, the singleton set $\{L\}$ for any $L \in \mathsf{E}$. E, being the largest subset of itself, has p-measure one.

4.2 Polynomial-Time Depth

This section proposes a variation of depth based on polynomial-time oblivious predictors, which, given a language L, try to predict $L[n]$ (i.e., the membership of s_n in L), *without* having access to $L[0 \, . \, . \, n-1]$. This is in contrast to a martingale, where the bet on $L[n]$ is by definition a function of $L[0 \, . \, . \, n-1]$. Intuitively, L is polynomial-time deep if giving a polynomial-time predictor more time allows it to predict bits of L with significantly greater accuracy.

An *oblivious predictor* is a function $P : \{0,1\}^* \times \{0,1\} \to [0,1]$ such that, for all $x \in \{0,1\}^*$, $P(x,0) + P(x,1) = 1$. Intuitively, when trying to predict a language L, $P(x,1)$ is the probability with which the predictor predicts that $x \in L$. To measure how well a predictor P predicts L, we consider its associated martingale $p : \{0,1\}^* \to [0,\infty)$ given by

$$p(L \upharpoonright n) = 2^n \prod_{y \leq s_n} P(y, L(y)).$$

We shall consider predictors P such that $P(s_n, b)$ is computable in time polynomial in n (hence computable in time $2^{c|s_n|}$ for some constant c), and call such a P a *polynomial-time oblivious predictor*, and we call the martingale p its *polynomial-time oblivious martingale (pom)*, with the convention that predictors are given in uppercase and pom in lowercase.

Definition 4.1. *A language L is* polynomial-time deep *if there exists $a > 0$ such that, for all pom p, there exists a pom p' such that, for infinitely many $n \in \mathbb{N}$,*

$$\frac{p'(L \upharpoonright n)}{p(L \upharpoonright n)} \geq a \log n,$$

with the convention that $\frac{1}{0} = \infty$. L is polynomial-time shallow *if it is not polynomial-time deep.*

Languages that are trivial or random for E are polynomial-time shallow.

Proposition 4.2. *Let L be a language.*

1. *If $L \in \mathsf{E}$, then L is polynomial-time shallow.*
2. *If L is E-random, then L is polynomial-time shallow.*

4.3 Slow Growth Law

Let $f : \{0,1\}^* \to \{0,1\}^*$. We say f is *monotone* if, for all $x, y \in \{0,1\}^*$, $x < y \implies f(x) < f(y)$. Given $l : \mathbb{N} \to \mathbb{N}$, we say f is *l-bounded* if, for all $x \in \{0,1\}^*$, $|f(x)| \leq l(|x|)$. Given two languages L_1, L_2 and a time bound $t : \mathbb{N} \to \mathbb{N}$ and length bound $l : \mathbb{N} \to \mathbb{N}$, we say that L_1 is *t-time l-bounded monotone many-one reducible* to L_2 (abbreviated *t-l-M reducible*), and we write $L_1 \leq_{\mathrm{M}}^{t,l} L_2$, if there is a Turing machine M computing a monotone, l-bounded reduction $f : \{0,1\}^* \to \{0,1\}^*$ such that, on input s_n, M halts in at most $t(|s_n|) = t(\log n)$ steps and outputs $f(s_n) \in \{0,1\}^*$ such that $s_n \in L_1$ if and only if $f(s_n) \in L_2$. We say L_1 is *E-time linearly bounded monotone many-one reducible* to L_2 (abbreviated *E-Lb-M reducible*), and we write $L_1 \leq_{\mathrm{M}}^{\mathsf{E},\mathrm{Lb}} L_2$, if there exists $c \in \mathbb{N}$ such that $L_1 \leq_{\mathrm{M}}^{2^{c|s_n|}, c|s_n|} L_2$. We follow the convention of letting n refer to the length of a characteristic sequence, rather than the length of the input string s_n. Therefore, equivalently, $L_1 \leq_{\mathrm{M}}^{n^c, n^c} L_2$; i.e., $f(s_n)$ is computable in time n^c, and, if $m \in \mathbb{N}$ is such that $s_m = f(s_n)$, then $m \leq n^c$.

The following result shows that shallow sequences cannot be transformed into deep ones by simple processes.

Theorem 4.3 (Polynomial-time slow growth law). *Let L_1, L_2 be languages such that $L_1 \leq_{\mathrm{M}}^{\mathsf{E},\mathrm{Lb}} L_2$. If L_1 is polynomial-time deep, then L_2 is polynomial-time deep.*

4.4 Languages That are Useful for E

In [3] Bennett showed that the halting language is deep, and Juedes, Lathrop, and Lutz [11] generalized this result by showing every weakly useful [11,8] language is deep. We prove a polynomial-time version of the result of Juedes, Lathrop, and Lutz, namely, that every E-Lb-M weakly useful language is polynomial-time deep.

Following the definition of weakly useful languages from [11] and [8], we define a language L to be E-Lb-M weakly useful if the set of languages in E that are reducible to L – within a fixed time and length bound – is not small (does not have measure zero in E). Intuitively, an E-useful language is somewhere in between an E-hard language and a trivial language, in the sense that the language does not necessarily enable one to decide *all* languages in E, but rather a nonnegligible subset of them. Note, however, that an E-hard (for instance, under polynomial-time many-one reductions) language may not necessarily be E-Lb-M weakly useful because of the requirements that an E-Lb-M reduction be monotone and linearly bounded.

Definition 4.4. *A language L is* E-Lb-M weakly useful *if there is a $c \in \mathbb{N}$ such that the set of languages $2^{c|s_n|}$-$c|s_n|$-M reducible to L does not have measure zero in E, i.e., if*

$$\mu\left(L^{\geq_{\mathrm{M}}^{2^{c|s_n|},c|s_n|}} \middle| \mathsf{E}\right) \neq 0$$

where

$$L^{\geq_{\mathrm{M}}^{2^{c|s_n|},c|s_n|}} = \left\{ A \;\middle|\; A \leq_{\mathrm{M}}^{2^{c|s_n|},c|s_n|} L \right\}.$$

In other words, a language L is weakly useful if a nonneglible subset of E monotonically many-one reduces to L within a fixed exponential time bound and fixed linear length bound. An example of an E-Lb-M weakly useful language is the halting language for E, defined as follows. Fix a standard linear-time computable invertible encoding of pairs of strings $(x, y) \mapsto \langle x, y\rangle$. Let $M_1, M_2, \ldots$ be an enumeration of machines deciding languages in E, where machine M_i runs in time $2^{i|s_n|}$. The E-halting language is given by $H_{\mathsf{E}} = \left\{ \langle 0^i, x\rangle \;\middle|\; M_i \text{ accepts } x \right\}$. It is easy to verify that access to the E-halting language allows one to decide every language $L_i \in \mathsf{E}$, decided by machine M_i, using the $1.01|s_n|$-time-bounded, $1.01|s_n|$-length-bounded, monotone reduction $f(x) = \langle 0^i, x\rangle$; i.e., $\mathsf{E} \subseteq H_{\mathsf{E}}^{\geq_{\mathrm{M}}^{1.01|s_n|,1.01|s_n|}}$, whence H_{E} is E-Lb-M weakly useful.

For every $g : \mathbb{N} \to \mathbb{N}$ and pom p define

$$D_p^g = \left\{ L \in \mathbf{C} \;\middle|\; (\exists \text{ pom } p')(\exists^\infty n \in \mathbb{N})\ \frac{p'(L \upharpoonright n)}{p(L \upharpoonright n)} \geq g(n) \right\}.$$

Note that L is polynomial-time deep if and only if there exists $a > 0$ such that, for all pom p, $L \in D_p^{a \log n}$.

Lemma 4.5. *For any $g : \mathbb{N} \to \mathbb{N}$ such that $g(n) = o(2^n)$ and any pom p, $\mu\left(D_p^g \middle| \mathsf{E}\right) = 1$.*

Theorem 4.6. *Every* E*-Lb-M weakly useful language is polynomial-time deep.*

Corollary 4.7. H_{E} *is polynomial-time deep.*

Corollary 4.8. *No language in* E *is* E*-Lb-M weakly useful.*

Corollary 4.9. *No* E*-random language is* E*-Lb-M weakly useful.*

No decidable language is deep in the sense of Bennett [3] (see also [11, Corollary 5.7]). However, the halting language H is deep and, while not decidable, is computably enumerable. Compare this with the fact that Corollary 4.8 (or a simple diagonalization) implies that $H_{\mathsf{E}} \notin \mathsf{E}$. It is easy to verify, however, $H_{\mathsf{E}} \in \mathsf{DTIME}(2^{|s_n|^2}) \subseteq \mathsf{EXP}$. Thus, polynomial-time depth mirrors Bennett's depth in that E-decidable languages are not polynomial-time deep, but polynomial-time deep languages can be found "close" to E. Similarly, Lemma 4.5 tells us, in an analogous fashion to Corollary 5.10 of [11], that "partially deep" sequences can be found in abundance in E.

References

1. Antunes, L., Fortnow, L., van Melkebeek, D., Vinodchandran, N.: Computational depth: Concept and applications. Theoretical Computer Science (Special issue for selected papers from the 14th International Symposium on Fundamentals of Computation Theory) 354(3), 391–404 (2006)
2. Athreya, K.B., Hitchcock, J.M., Lutz, J.H., Mayordomo, E.: Effective strong dimension, algorithmic information, and computational complexity. SIAM Journal on Computing (to appear). Preliminary version appeared in: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 632–643. Springer, Heidelberg (2004)
3. Bennett, C.H.: Logical depth and physical complexity. In: Herken, R. (ed.) The Universal Turing Machine: A Half-Century Survey, pp. 227–257. Oxford University Press, London (1988)
4. Borel, E.: Sur les probabilités dénombrables et leurs applications arithmétiques. Rendiconti del Circolo Matematico di Palermo 27, 247–271 (1909)
5. Bourke, C., Hitchcock, J.M., Vinodchandran, N.V.: Entropy rates and finite-state dimension Theoretical Computer Science 349, 392–406 (to appear, 2005)
6. Dai, J.J., Lathrop, J.I., Lutz, J.H., Mayordomo, E.: Finite-state dimension. Theoretical Computer Science 310, 1–33 (2004)
7. Doty, D., Moser, P.: Finite-state dimension and lossy decompressors. Technical Report cs.CC/0609096, Computing Research Repository (2006)
8. Fenner, S.A., Lutz, J.H., Mayordomo, E., Reardon, P.: Weakly useful sequences. Information and Computation 197, 41–54 (2005)
9. Hitchcock, J.M.: Fractal dimension and logarithmic loss unpredictability. Theoretical Computer Science 304(1–3), 431–441 (2003)
10. Huffman, D.A.: Canonical forms for information-lossless finite-state logical machines. IRE Trans. Circuit Theory CT-6 (Special Supplement), pp. 41–59, Also available in E.F. Moore (ed.), Sequential Machine: Selected Papers, Addison-Wesley, 1964, pp. 866–871 (1959)

11. Juedes, D.W., Lathrop, J.I., Lutz, J.H.: Computational depth and reducibility. Theoretical Computer Science 132(1–2), 37–70 (1994)
12. Kohavi, Z.: Switching and Finite Automata Theory, 2nd edn. McGraw-Hill, New York (1978)
13. Lathrop, J.I., Lutz, J.H.: Recursive computational depth. Information and Computation 153(2), 139–172 (1999)
14. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and its Applications, 2nd edn. Springer, Berlin (1997)
15. Lutz, J.H.: Almost everywhere high nonuniform complexity. J. Comput. Syst. Sci. 44(2), 220–258 (1992)
16. Lutz, J.H.: The quantitative structure of exponential time. In: Hemaspaandra, L.A., Selman, A.L. (eds.) Complexity Theory Retrospective II, pp. 225–254. Springer, Heidelberg (1997)
17. Schnorr, C.P., Stimm, H.: Endliche Automaten und Zufallsfolgen. Acta. Informatica 1, 345–359 (1972)
18. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal, 27:379–423, 623–656 (1948)
19. Sheinwald, D., Lempel, A., Ziv, J.: On encoding and decoding with two-way head machines. Information and Computation 116(1), 128–133 (1995)
20. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transaction on Information Theory 24, 530–536 (1978)

Abstract Geometrical Computation and the Linear Blum, Shub and Smale Model

Jérôme Durand-Lose[⋆]

Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans,
B.P. 6759, F-45067 ORLÉANS Cedex 2
Jerome.Durand-Lose@univ-orleans.fr
http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose

Abstract. Abstract geometrical computation naturally arises as a continuous counterpart of cellular automata. It relies on signals (dimensionless points) traveling at constant speed in a continuous space in continuous time. When signals collide, they are replaced by new signals according to some collision rules. This simple dynamics relies on real numbers with exact precision and is already known to be able to carry out any (discrete) Turing-computation. The Blum, Shub and Small (BSS) model is famous for computing over $\mathbb{R}$ (considered here as a $\mathbb{R}$ unlimited register machine) by performing algebraic computations.

We prove that signal machines (set of signals and corresponding rules) and the infinite-dimension linear (multiplications are only by constants) BSS machines can simulate one another.

Keywords: Abstract geometrical computation, Analog computation, BSS model, Signal machine.

1 Introduction

There is no agreed analog counterpart of the Church-Turing thesis; relating the models is crucial to understand the differences between the various computing capabilities. For example, Bournez *et al.* related Moore's recursion theory on $\mathbb{R}$ [Moo96], computable analysis [Wei00] and the general purpose analog computer [BH04, BCGH06]. The aim of this paper is to link two analog models of computation. One, abstract geometrical computation deals with regular and automatic drawing on the euclidean plane, while the second, the Blum, Shub and Smale model [BCSS98] relies on algebraic computations over $\mathbb{R}^n$. Let us note that Bournez [Bou99] already provide some relations between linear BSS and Piecewise Constant Derivative systems. The latter also generate Euclidean drawings.

Abstract geometrical computation (ACG) arises from the common use in cellular automata (CA) literature of Euclidean settings to explain an observe dynamics or to design a CA for a particular purpose. But CA operate in discrete time over discrete space, while Euclidean geometry deals with both continuous time and space. This switch of context is justified by the scaling invariance of

⋆ Corresponding author.

CA and comes for our preference and ability for thinking in classical continuous terms rather than in discrete terms (for example just think how recent and complex is discrete geometry compared to the Euclidean one). Abstract geometrical computation works in a continuous setting: discrete signals/particles become dimensionless points; the local function of CA, computing the next state of a cell according to the states of neighbouring cells, is replaced by collision rules: which signals emerges from a collision of signals. Signals and rules define *signal machines* (SM). This recent model, even restricted to rational numbers, is able to carry out any (discrete) Turing-computation [DL06c]. With continuous time, Zeno paradox arises: not only are accumulations possible, but they can be used to decide recursively enumerable problems by using the black-hole principle [DL05, DL06a]. Let us note that if accumulations can be generated at will, they can hardly be foreseen [DL06b]. In this paper, we are not interested by accumulations and the super-Turing capability that they bring forth in the discrete computability. We are interested on considering AGC as an analog model, thus there is no rational number restriction and accumulations are not encompassed (they are considered as divergent computations).

In the Blum, Shub and Smale model (BSS), machines computes over any ring. Roughly speaking, polynomial functions can be performed on variables as well as test (according to some order) for branching. Linear BSS [MM97] is the restriction where it is forbidden to multiply two variables, but it is still allowed to multiply by constants. In the case where the dimension of the input is not bounded or the number of registers needed to compute is not bounded, a *shift* operator is provided on order to access any register (finitely many registers hold non zero values since only finitely many registers can be accessed in finite time). We prove the equivalence of AGC and linear BSS over $\mathbb{R}$ in infinite dimension. For the sake of simplicity, we consider that there is no shift operator but indirect addressing through *addresses* (natural counters, the term address is used to distinguish from *real* registers) and that all operations are carried out on an accumulator; this corresponds to the real number unlimited register machine ($\mathbb{R}$-URM) [Nov95] (the arguments for the full BSS translate to the linear case).

To simulate a lin-$\mathbb{R}$-URM with a SM, the value of each register is encoded as the distance between two signals. A lin-$\mathbb{R}$-URM is considered as an assembly language program and we show how to translate each instruction. Since the reader might not be familiar with ACG, the first and simplest constructions are more detailed to provide examples.

To simulate a SM with a lin-$\mathbb{R}$-URM, a configuration is encoded as a finite sequence of, alternatively, signal value and distance to the next one. Although signal machines work with continuous time, the only important discrete dates are when a collision occurs. The simulation goes from a collision date to the next. This is achieved in three steps: compute the next collision time, then update the distances between the signals and finally carry out the collision(s).

Section 2 gives the definition of both models. Section 3 provides the simulation of any lin-$\mathbb{R}$-URM by a SM while Sect. 4 carries the simulation the other way round. Conclusion, remarks and perspective are gathered in Sect. 5.

2 Definitions

2.1 Abstract Geometrical Computations

In this model, dimensionless objects are moving on the real axis. When a collision occurs they are replaces by others. This is defined by the following machines:

Definition 1. A *signal machine* is defined by (M, S, R) where M (*meta-signals*) is a finite set, S (*speeds*) a mapping from M to $\mathbb{R}$ and R (*collision rules*) a partial mapping from the subsets of M of cardinality at least two into subsets of M (speeds must differ in both domain and range).

The elements of M are called *meta-signals*. Each instance of a meta-signal is a *signal*. The mapping S assigns *speeds* to meta-signals. They correspond to the inverse slopes of the segments in space-time diagrams. The *collision rules*, denoted $\rho^- {\rightarrow} \rho^+$, define what emerges ($\rho^+$) from the collision of two or more signals (ρ^-). Since R is a mapping, signal machines are deterministic. The *extended value set*, V, is the union of M and R plus two symbols: one for void, $\oslash$, and one for an accumulation (which is not addressed here). A *configuration*, c, is a total mapping from $\mathbb{R}$ to V such that the set $\{\, x \in \mathbb{R} \mid c(x) \neq \oslash \,\}$ is finite.

A signal corresponding to a meta-signal μ at a position x, *i.e.* $c(x) = \mu$, is moving uniformly with constant speed $S(\mu)$. A signal must start (resp. end) in the initial (resp. final) configuration or in a collision. These correspond to condition 2 in Def. 2. At a $\rho^- {\rightarrow} \rho^+$ collision signals corresponding to the meta-signals in ρ^- (resp. ρ^+) must end (resp. start) and no other signal should be present (condition 3).

Definition 2. The *space-time diagram* issued from an initial configuration c_0 and lasting for T, is a mapping c from $[0, T]$ to configurations (*i.e.* a mapping from $\mathbb{R} \times [0, T]$ to V) such that, $\forall (x, t) \in \mathbb{R} \times [0, T]$:

1. $\forall t {\in} [0, T]$, $\{\, x \in \mathbb{R} \mid c_t(x) \neq \oslash \,\}$ is finite,
2. if $c_t(x) {=} \mu$ then $\exists t_i, t_f {\in} [0, T]$ with $t_i {<} t {<} t_f$ or $0 {=} t_i {=} t {<} t_f$ or $t_i {<} t {=} t_f {=} T$ s.t.:
   - $\forall t' \in (t_i, t_f)$, $c_{t'}(x + S(\mu)(t' - t)) = \mu$,
   - $t_i = 0$ or $c_{t_i}(x_i) \in R$ and $\mu \in (c_{t_i}(x_i))^+$ where $x_i = x + S(\mu)(t_i - t)$,
   - $t_f = T$ or $c_{t_f}(x_f) \in R$ and $\mu \in (c_{t_f}(x_f))^-$ where $x_f = x + S(\mu)(t_f - t)$;
3. if $c_t(x) {=} \rho^- {\rightarrow} \rho^+ \in R$ then $\exists \varepsilon$, $0 {<} \varepsilon$, $\forall t' {\in} [t - \varepsilon, t + \varepsilon] \cap [0, T]$, $\forall x' {\in} [x - \varepsilon, x + \varepsilon]$,
   - $c_{t'}(x') \in \rho^- {\cup} \rho^+ \cup \{\oslash\}$,
   - $\forall \mu {\in} M$, $c_{t'}(x') {=} \mu \Rightarrow \bigvee \begin{cases} \mu \in \rho^- \text{ and } t' < t \text{ and } x' = x + S(\mu)(t' - t)) \,, \\ \mu \in \rho^+ \text{ and } t < t' \text{ and } x' = x + S(\mu)(t' - t)) \,. \end{cases}$

On space-time diagrams, the traces of signals represent line segments whose directions are defined by $(S(.), 1)$ (1 is the temporal coordinate). Collisions correspond to the extremities of these segments. In the space-time diagrams, time increases upwards.

A configuration is composed of the identities and positions of all the present signals. Since the origin is not relevant (because of the shift invariance), if is enough to have the identities and the distances between signals from left to right. At any time, there are finitely, although unbounded, many signals.

As a computing device, the input is the initial configuration and the output is the final configuration (*e.g.* when no collision can happen).

2.2 Linear Real Number Unlimited Register Machines

We do not use the definition of [BCSS98] (graph with input, output, computation and branch nodes plus a shift node to deal with infinite dimension). Instead, we use the more assembly language like definition of linear $\mathbb{R}$-URM. Each register holds a real number (with exact precision). Inputs as well as outputs are stored in the registers. The machine can add, multiply (by a constant) and copy values. To cope with infinite dimension, address (natural integers) registers are used for indirect addressing. To simplify our constructions we suppose that all real computations are done with one accumulator (which corresponds to a constant slowdown).

Definition 3. A *linear real number unlimited register machine (lin-$\mathbb{R}$-URM)* is described by a sequence of instructions among the following ones:

- `inc` A_i, `dec` A_i and `if` $0{<}A_i$ `goto` n for the addresses, and
- `load` R_i (`load` $R_{(i)}$), `store` R_i (`store` $R_{(i)}$), `add` R_i (`add` $R_{(i)}$), `mul` α, and `if` $0{<}X$ `goto` n for the registers,

where i is a natural integer, α is a constant real number, n is a line number, A_i is an address, R_i is a register and X is the accumulator. The indirect addressing, $R_{(i)}$, corresponds to R_{A_i}.

The first register has number 0 to avoid any addressing problem. All the operations are done on the accumulator; thus there is no second argument. There is no `mul` R_i since multiplication is only by constant (otherwise it would not be linear). To simplify, there is no `add` α, additive constants are supposed to be stored in some registers.

A configuration consists of the line number, the values of the addresses and of the registers. There are finitely many addresses (their number is directly given by the code) and, at any instant, finitely many registers used (but their number is not bounded).

3 Linear $\mathbb{R}$-URM Simulation by Signal Machines

3.1 Encoding a Configuration

A lin-$\mathbb{R}$-URM configuration is composed of a line number, n, and values for addresses, $\{A_i\}_{i\in I}$, and registers, $\{R_i\}_{i\in K}$ (I and K are finite initial segments of $\mathbb{N}$). The set I is a constant of the machine while K may be enlarged during the computation. Since there are finitely many line numbers, each one can be identified by a meta-signal. The rest of the configuration is encoded with speed 0 signals ensuring its stability (parallel signals never interact). Since addresses are only used for indicating registers, they are added as markers on the corresponding registers (again the number of addresses is bounded and as many as needed meta-signals are available from the start).

Registers. A register is encoded as the distance from a base signal to the pairing val signal. There is no absolute scale since signal machines are scale invariant. Two scale signals whose distance amounts for a scale are provided as depicted on Fig. 1. All registers use the same scale. For the value 0, the superposition of base and val is encoded as a single signal nul. This value is never considered in the rest of the paper; the reader is invited to check that it can be easily covered.

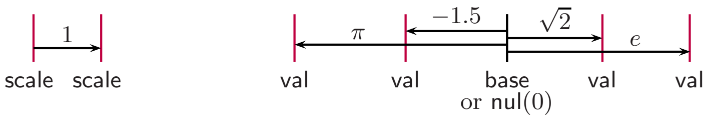


Fig. 1. Encoding: scale and positions of val for values $-\pi$, -1.5, 0, $\sqrt{2}$ and e

The registers are always encoded with the same meta-signals, base and val. They are regularly displayed as depicted on Fig. 2. The signals base are at a distance, say d, one from the next, such that each val is at distance strictly less than $d/2$ from its corresponding base. Not to tangle one register encoding with another one during the computation, each pair is kept away from the others; if a value becomes too large (which is simple to check since sums and multiplications are done on the accumulator), each distance from val to base is scaled down as well as the scale pair to keep the same values.

The accumulator is encoded like a register and is displayed on the left of the registers. An end marker indicates the right limit of the configuration. It is used in order both to prevent signal from drifting forever on the right and to help enlarging the configuration when needed. The line number is encoded as a signal, line$_n$, of negative speed which is about to hit the right scale and start the next iteration. A complete encoding is given on Fig. 2.

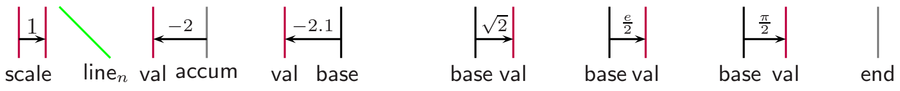


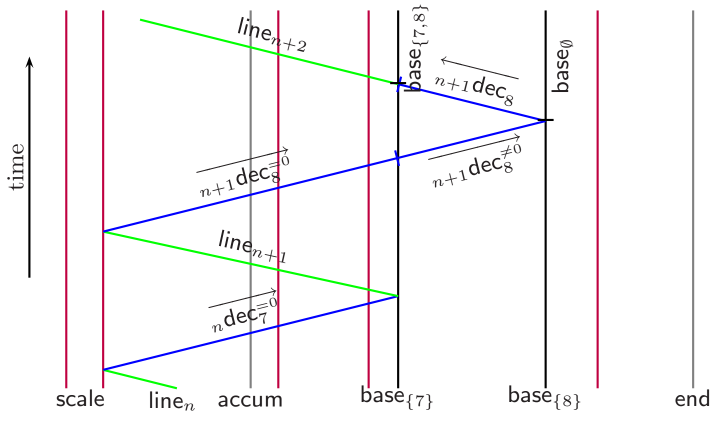
Fig. 2. Scale, line number, accumulator (-2), registers (-2.1, $\sqrt{2}$, $\frac{e}{2}$, $\frac{\pi}{2}$) and end

Addresses. Since they are used to designate registers, they are encoded by marks on the corresponding base's. This is done by replacing base by any value in $\{\text{base}_J\}_{J\subseteq I}$ (there are finitely many such meta-signals). Each i of I must appear in exactly one base$_J$. If needed, dummy null registers are added to cover all the values of the addresses and all registers directly addressed in the code.

3.2 Updating the Configuration

The simulation is as follows: line$_n$ bounces on scale and changes to whatever signal is used to carry out the instruction at line n. After the instruction is carried out, extra signals are disposed of and one signal with the new line number is sent to the scale. It remains to deal independently with each possible instruction.

Address manipulations. These are: increasing or decreasing by one and branch if non null. Decreasing corresponds to getting to the base_J holding i and move the element i to the base on the left except when this register is number 0. To achieve this, signal $\overrightarrow{{}_n\mathsf{dec}_i^{0?}}$ is send to the right. The left subscript n is used to record the line number; it is changed when the operation is performed. The superscript 0? indicates that, as far the computation has gone, the value of the address could still be zero. This signal moves to the right leaving every signal as it is until it reaches the first base_J. If i belongs to J then the address is 0 and the signal goes back on the left as line_{n+1} (lower part of Fig. 3); otherwise it turns to $\overrightarrow{{}_n\mathsf{dec}_i}$ and keep going right until it meets the base_J such that i belongs to J. It then turns to $\overleftarrow{{}_n\mathsf{dec}_i}$ and goes on the left; as soon as it reaches a base_J, it replaces it by $\mathsf{base}_{J\cup\{i\}}$ and turns to line_{n+1} (upper part of Fig. 3).



line_{n+2}
$\mathsf{base}_{\{7,8\}}$
$\overleftarrow{{}_{n+1}\mathsf{dec}_8}$
$\mathsf{base}_\emptyset$
time
$\overrightarrow{{}_{n+1}\mathsf{dec}_8^{=0}}$
$\overrightarrow{{}_{n+1}\mathsf{dec}_8^{\neq 0}}$
line_{n+1}
$\overrightarrow{{}_n\mathsf{dec}_7^{=0}}$
scale
line_n
accum
$\mathsf{base}_{\{7\}}$
$\mathsf{base}_{\{8\}}$
end

Fig. 3. Example of register updating: $\begin{array}{r} n\colon \mathsf{dec}\ A_7 \\ n{+}1\colon \mathsf{dec}\ A_8 \end{array}$

Increasing A_i corresponds to getting to the base_J holding i and move i to the next base on the right (lower part of Fig. 4). When there is no more register on the right, end is reached and used to create a new nul register instead of end and to reposition end one step on the right (upper part of Fig. 4). This time, the meta-signals used are: $\overrightarrow{{}_n\mathsf{inc}_i^s}$ (searching) and $\overrightarrow{{}_n\mathsf{inc}_i}$ plus two extra signals, $\overleftarrow{\mathsf{end}}$ and $\overrightarrow{\mathsf{end}}$ to regenerate end. The last two meta-signals must be three times faster than $\overrightarrow{{}_n\mathsf{inc}_i}$ in order to ensure the positioning of the new end at the same distance.

Branching is very easy, the signal goes on the right, passes the accumulator and get to the first base_J. Depending on whether i belongs to J, it comes back as the following line or the branch line number.

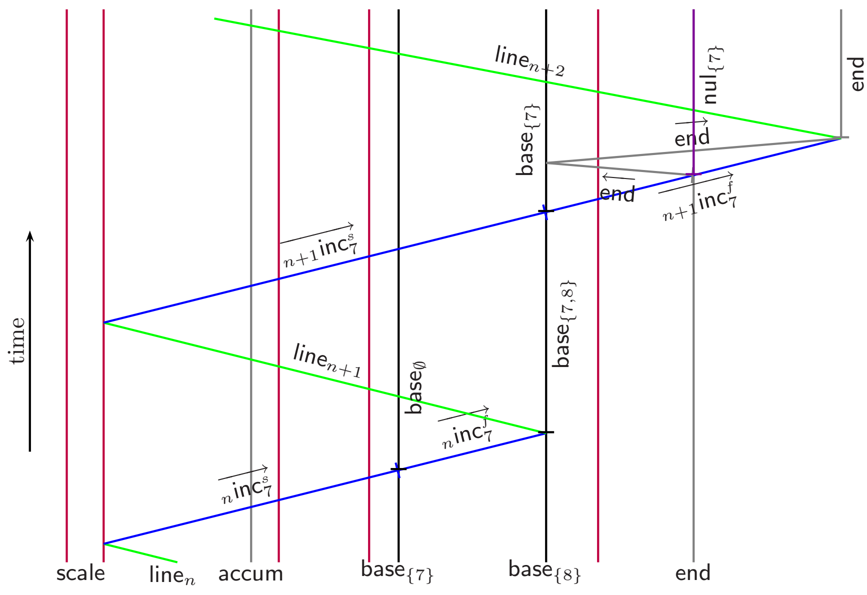

time
scale
line$_n$
accum
base$_{\{7\}}$
base$_{\{8\}}$
end
$\overrightarrow{{}_n\mathsf{inc}^s_7}$
$\overrightarrow{{}_n\mathsf{inc}^f_7}$
base$_\emptyset$
line$_{n+1}$
base$_{\{7,8\}}$
$\overrightarrow{{}_{n+1}\mathsf{inc}^s_7}$
$\overrightarrow{{}_{n+1}\mathsf{inc}^f_7}$
$\overleftarrow{\mathsf{end}}$
$\overrightarrow{\mathsf{end}}$
base$_{\{7\}}$
line$_{n+2}$
nul$_{\{7\}}$
end

Fig. 4. Example of register updating: $\begin{array}{r}n\texttt{:inc}\,A_7\\ n{+}1\texttt{:inc}\,A_8\end{array}$

Registers operations. We first present how to move a value from the accumulator to a given register ($\texttt{load}\,R_i$ is similar). The register can be indicated directly or indirectly. In the first case its number is know directly from the code, it is thus possible to make as many as needed meta-signals to count down from i to 0 (meta-signals are used as a finite unary counter). At each base_J crossing, it is decremented until it reaches 0, the designated register is reached. In the second case, a signal is issued that looks for the base_J such that i belongs to J (as in address manipulation).

The first column of Fig. 5 presents how the copy starts from the accumulator and is stored on the corresponding register. The second column shows the erasing of the previous value of the register. The first row deals with negative values and the second row with positive ones (handling 0 is trivial). What happens on the target register is the superposition of the right of left column and right column. There is no risk of collision with the new val since $\overleftarrow{\mathsf{del}}$ and $\overrightarrow{\mathsf{del}}$ are below set^- and set^+. The value stored is exactly the same since set and ${}_n\mathsf{sto}$ are parallel as well as the pair of set^- (or of set^+).

To handle multiplication, the scale invariance of AGC is used: if starting from two signals at distance 1, they end up at distance α, then starting from two signals at distance d, they end up at distance αd. There are two cases to consider: $\alpha{<}1$ and $1{<}\alpha$ (multiplication by 1 is not very interesting). The space-time diagrams and (pre-computed) speeds are given on Fig. 6.

Let us note that multiplication can produce an overflow: a value so large that it might provoke some entanglement between two registers. This is easy

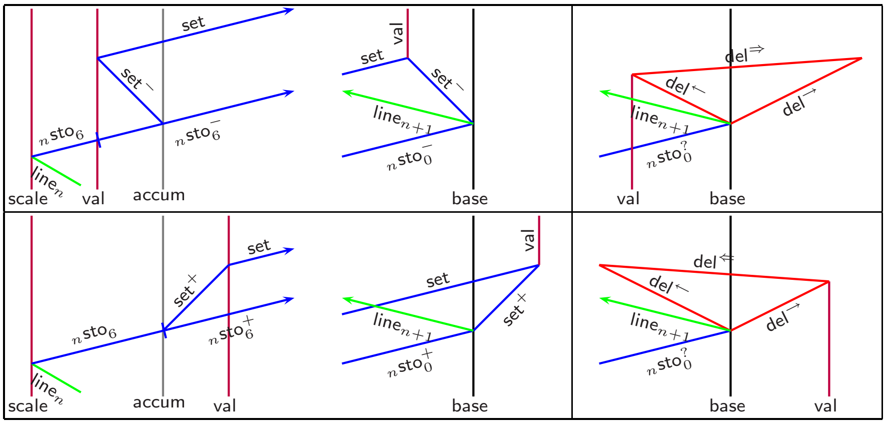


Fig. 5. Various cases to achieve `store` R_6

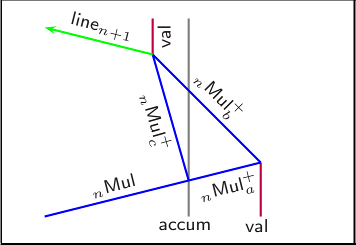
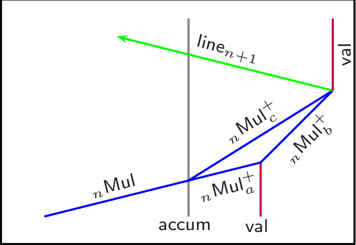
	$\alpha < 1$ (here $\alpha = -\frac{1}{2}$)	$1 < \alpha$ (here $\alpha = 2$)
	 line$_{n+1}$ val $_n$Mul$_b^+$ $_n$Mul$_c^+$ $_n$Mul $_n$Mul$_a^+$ accum val	 line$_{n+1}$ val $_n$Mul$_c^+$ $_n$Mul$_b^+$ $_n$Mul $_n$Mul$_a^+$ accum val
Speed of $_n$Mul$_a^+$:	4	4
Speed of $_n$Mul$_b^+$:	-1	1
Speed of $_n$Mul$_c^+$:	$\frac{4\alpha}{5-4\alpha}$	$\frac{4\alpha}{4\alpha-3}$

Fig. 6. Multiplication scheme

to detect: two dummy bounding signals are added around accum, whenever the multiplication crosses any of them an overflow flag is raised (*i.e.* some signal switches to a given meta-signal). The multiplication is carried out normally, but the val is replaced by over (to distinguish it from any other val). When an overflow occurs, a special subroutine is launched. It scales down by $1/\alpha$ (for `mul` α with $1<\alpha$) all the registers, the accumulator and the scale. This way, all encoded real values are preserved. All the val and accum remain at their positions, only the left scale, over and all val are moved. The above multiplication scheme is used, this time no overflow can happen.

Addition is not presented, let us just say that it works exactly like load except that val (of the accumulator) is used as origin instead of accum, and deletion of the old val is slightly different. Let us note that addition as an overflow detection like multiplication, in such a case, all is scaled by one half.

4 Signal Machines Simulation by Linear $\mathbb{R}$-URM

This construction is less detailed since it relies on classical construction on register machines and the reader should have now a rather clear picture of what both models are (and also because of the lack of room). A SM configuration consists of an alternating sequence of meta-signals and distances, starting and ending with a meta-signal. This is straight forwardly be translated into a sequence of registers (meta-signals are encoded by integers starting at 1) followed by 0's.

Updating is done in three steps: first find the delay to the next collision, then update the distances and finally process the collision(s) (there might be synchronous ones). Finding the delay is just to go through the sequence and consider signals two by two: compute the delay before next collision (if any) and store the minimum. To achieve this, it loops through the configuration encoding (this is easy with indirect addressing; the loop stops at the first 0 for a meta-signal) and computes the collision delay. For the latter, there is a formula depending on the distance and speeds of meta-signals but it is not linear. Nevertheless, there are finitely many meta-signals and their speeds are known from the signal machine; each case is linear. It only remains to branch to the correct case with a big switch/if then else.

If no collision happens then the machine halts. Otherwise, it "advances" the time by the given duration (*i.e.* the distances are updated) then it goes through the configuration again and process each collision, *i.e.* signals at distance 0. There could be more than two signals involved in a collision (but no more than the number of meta-signals). Again, there are finitely many possible collision rules and they are all given by the signal machine. So to find one, a huge switch has to be provided (there is a case for each rule): first consider how many signals are involved then find the corresponding rule. In-coming signals are replaced by out-going. If their numbers are different, a procedure to compress or enlarge the configuration (exactly like one would do inside any array) is used.

5 Conclusion

We have proved that signal machines are equivalent to lin-$\mathbb{R}$-URM and infinite dimensional linear $\mathbb{R}$-BSS. Let us note that SM restricted to rational speeds and positions are equivalent to lin-$\mathbb{Q}$-URM with the same constructions. The number of rules of the simulating SM is up bounded by an exponential in the number of lines of the lin-$\mathbb{R}$-URM while the number of line of the simulating lin-$\mathbb{R}$-URM is linear in the numbers of collisions and rules.

Considering the number of collisions as a complexity measure on SM, in each case, the slowdown is of the order of the number of signals/register, *i.e.* of *space*. Let us remember that all is constructed with lin-$\mathbb{R}$-URM, not linear BSS and full BSS has also a weak model of complexity [Koi93] so we do not go any further on complexity issues here.

Let us note that reversible and conservative signal machines on rational have full Turing-computability. Would reversible and conservative restrictions already be equivalent to linear BSS? In a rational setting, accumulation was used to climb up the arithmetical hierarchy. We believe that in the real setting, they could be used to provide inner multiplication and thus proved that signal machine could simulate the full BSS model. We believe that in such a case BSS would be a strictly less powerful model (unless some limit operator is provided).

References

[BCGH06] Bournez, O., Campagnolo, M.L., Graça, D.S., Hainry, E.: The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 631–643. Springer, Heidelberg (2006)

[BCSS98] Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and real computation. Springer, New York (1998)

[BH04] Bournez, O., Hainry, E.: An analog characterization of elementarily computable functions over the real numbers. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 269–280. Springer, Heidelberg (2004)

[Bou99] Bournez, O.: Some bounds on the computational power of piecewise constant derivative systems. Theory of Computing Systems 32(1), 35–67 (1999)

[DL05] Durand-Lose, J.: Abstract geometrical computation for black hole computation (extended abstract). In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 176–187. Springer, Heidelberg (2005)

[DL06a] Durand-Lose, J.: Abstract geometrical computation 1: embedding black hole computations with rational numbers. Fundamenta Informaticae 74(4), 491–510 (2006)

[DL06b] Durand-Lose, J.: Forcasting black holes in abstract geometrical computation is highly unpredictable. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 644–653. Springer, Heidelberg (2006)

[DL06c] Durand-Lose, J.: Reversible conservative rational abstract geometrical computation is turing-universal. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 163–172. Springer, Heidelberg (2006)

[Koi93] Koiran, P.: A weak version of the Blum, Shub & Smale model. In: 34th Annual Symposium on Foundations of Computer Science (FOCS '93), pp. 486–495. IEEE, Washington (1993)

[MM97] Meer, K., Michaux, C.: A survey on real structural complexity theory. Bulletin of the Belgian Mathematical Society 4, 113–148 (1997)

[Moo96] Moore, C.: Recursion theory on the reals and continuous-time computation. Theoret. Comp. Sci. 162(1), 23–44 (1996)

[Nov95] Novak, E.: The real number model in numerical analysis. J. Complex. 11(1), 57–73 (1995)

[Wei00] Weihrauch, K.: Introduction to computable analysis. In: Texts in Theoretical computer science, Springer, Berlin (2000)

A Continuous Derivative for Real-Valued Functions

Abbas Edalat

Department of Computing, Imperial College London, UK
ae@ic.ac.uk

Abstract. We develop a notion of derivative of a real-valued function on a Banach space, called the L-derivative, which is constructed by introducing a generalization of Lipschitz constant of a map. The values of the L-derivative of a function are non-empty weak* compact and convex subsets of the dual of the Banach space. This is also the case for the Clarke generalised gradient. The L-derivative, however, is shown to be upper semi continuous with respect to the weak* topology, a result which is not known to hold for the Clarke gradient on infinite dimensional Banach spaces. We also formulate the notion of primitive maps dual to the L-derivative, an extension of Fundamental Theorem of Calculus for the L-derivative and a domain for computation of real-valued functions on a Banach space with a corresponding computability theory.

Keywords: L-derivative, Clarke gradient, weak* topology, upper semi-continuity.

This paper is dedicated to the historical memory of Sharaf al-din Tusi (d. 1213), the Iranian mathematician who was the first to use the derivative systematically to solve for roots of cubic polynomials and find their maxima [12,13,15].

1 Introduction

It is well-known that the classical derivative of a real-valued Lipschitz function of a real variable may not always exist and when it does exist it may not give rise to a continuous function. The same is true for the higher dimensional Gâteaux and Fréchet derivatives of a real-valued Lipschitz function defined on a finite dimensional Euclidean space or on a Banach space.

In 1980's, Frank Clarke, motivated by problems in non-smooth analysis and control theory, introduced the notion of generalized gradient of a function, which is now named after him [4]. Clarke's gradient of a locally Lipschitz real-valued function on a Banach space always exists and is a set-valued function: on finite dimensional Euclidean spaces it takes non-empty compact and convex subsets of the Euclidean space as its values and the gradient is upper semi-continuous. On an infinite dimensional Banach space, the Clarke gradient is a non-empty weak* compact and convex subset of the dual of the Banach space. It is however not known if Clarke's gradient is also upper semi-continuous on infinite dimensional Banach spaces [3].

Since the derivative of functions plays a fundamental role in mathematics, one would expect a real interest in a continuous derivative by researchers in computability theory, where the continuity of a function is an essential requirement for its computability [16].

However, surprisingly, no attempt was made to develop a continuous derivative for functions and the work of Clarke went unnoticed by researchers in computable analysis, who have only worked with the classical derivative of functions; see for example [18].

A new approach to differential calculus based on mathematical structures in computer science, called domains [17,14,2,5] was introduced in [9,10] first for real-valued functions of a real variable and then for multivariable functions. The motivation here has arisen from computer science and computable analysis to formulate and use, in particular, a notion of continuous derivative for functions.

In the domain-theoretic framework, a continuous derivative for functions, a corresponding notion of primitive maps, an extension of fundamental theorem of calculus and a domain for differentiable functions have been developed, in a coordinate dependent way, for finite dimensional Euclidean spaces.

In this paper, inspired by the above domain-theoretic framework, we introduce a coordinate free approach to develop the notion of the *L-derivative* of a real-valued function on a Banach space; it is constructed by formulating a generalized Lipschitz property of functions. The local generalized Lipschitz properties of the function, which provide finitary information about the rate of growth of the function in local neighbourhoods, are used to define the L-derivative of the function globally. Like the Clarke gradient, the values of the L-derivative are non-empty weak* compact and convex subsets of the dual of the Banach space.

The L-derivative, developed here from the local to the global and from the discrete to the continuum, is shown to be upper semi-continuous for real-valued locally Lipschitz functions on any Banach space, a result which is not known for the Clarke gradient as we have already mentioned above.

For a C^1 function, i.e., one with a continuous Fréchet derivative, the L-derivative and the Fréchet derivative coincide. More generally, the L-derivative contains the Clarke gradient, and, when they exist, the Gâteaux and Fréchet derivatives.

The L-derivative gives rise to an extension of the Fundamental Theorem of Calculus. The class of functions from the Banach space into the collection of non-empty weak* compact and convex subsets of the dual of the Banach space, which are generated by step functions, is dual via the L-derivative to families of real-valued locally Lipschitz functions on the Banach space. The L-derivative is also employed to construct a domain of computation for real-valued functions on Banach spaces, a structure which supports a computability theory when the space is separable. These results extend those for finite dimensions in [7,11].

In the full version of the paper which contains all proofs [6], we also show that for functions on finite Euclidean spaces, the L-derivative is an element of a countably based continuous domain which can be given an effective structure that characterizes computable functions with computable L-derivatives. Any continuous function and its L-derivative can be obtained as the supremum of an increasing sequence of pairs of finitary and consistent information about the function and its L-derivative.

Although they are defined using very different techniques, we show in the full paper that in finite dimensions the Clarke gradient and the L-derivative coincide. Thus, in finite dimensions the construction of the L-derivative provides a new computable representation for the Clarke gradient.

1.1 Background Definitions

For the remainder of this section we will present the basic background definitions of the various notions of derivative which we will need in this paper.

Let X and Y be Banach spaces and let $U \subset X$ be an open subset. We recall that the (one sided) directional derivative of $f : U \to Y$ at $x \in U$ in the direction $v \in X$ is

$$F'(x; v) = \lim_{t \downarrow 0} \frac{f(x + tv) - f(x)}{t},$$

if the limit exists. If the above directional derivative exists for all $v \in X$, then the *Gâteaux* derivative of f at x, a continuous linear functional from X to Y denoted by $D(f)(x)$, exists if $F'(x; v) = D(f)(x)(v)$ for all $x \in X$.

The *Fréchet* derivative of a map $f : U \to Y$ at $x \in U$, when it exists, is defined as the linear map $T : X \to Y$ with

$$\lim_{\|x-y\| \to 0} \frac{\|f(x) - f(y) - T(x-y)\|}{\|x - y\|} = 0.$$

The linear map T is denoted by $f'(x)$. When the Fréchet derivative exists at x, so does the Gâteaux derivative and they are equal.

From now on we will assume that $Y = \mathbb{R}$. We next aim to define the generalized (Clarke) gradient of a function [4, Chapter two] and explain its properties. Let $f : U \to \mathbb{R}$ be Lipschitz near $x \in U$ and $v \in X$. The *generalized directional derivative* of f at x in the direction of v is

$$f^{\circ}(x; v) = \limsup_{y \to x \;\; t \downarrow 0} \frac{f(y + tv) - f(y)}{t}.$$

Let us denote by X^* the dual of X, i.e. the set of real-valued continuous linear functions on X. Unless otherwise stated we will consider X^* with its weak* topology. Recall that the weak* topology is the weakest topology on X^* in which for any $x \in X$ the map $f \mapsto f(x) : X^* \to \mathbb{R}$ is continuous.

The *generalized gradient* of f at x, denoted by $\partial f(x)$ is the subset of X^* given by

$$\{A \in X^* : f^{\circ}(x; v) \geq A(v) \text{ for all } v \in X\}.$$

It is shown in [4, page 27] that

– $\partial f(x)$ is a non-empty, convex, weak* compact subset of X^*.
– For $v \in X$, we have: $f^{\circ}(x; v) = \max\{A(v) : A \in \partial f(x)\}$.

In finite dimensions, the Clarke gradient is *upper semi-continuous*, i.e. it is continuous with respect to the *upper topology* on the space of the non-empty compact subsets of $\mathbb{R}^n$, where a basic open subset of the upper topology on this space is given by the collection of all nonempty compact subsets of $\mathbb{R}^n$ contained in a given open subset of $\mathbb{R}^n$. It is not known if a similar result holds in infinite dimensions [3], i.e. if the Clarke gradient is continuous with respect to the upper topology on the space of non-empty weak* compact subsets of X^*, where a basic open subset of the upper topology on this space is given by the collection of all nonempty weak* compact subsets of X^* contained in a given weak* open subset of X^*.

We let $\mathbf{C}(X^*)$ denote the dcpo of non-empty, convex and weak* compact subsets of X^* ordered by reverse inclusion.

2 Ties of Functions

The local differential property of a function is formalized in our framework by the notion of an interval Lipschitz constant. Assume $U \subset X$ is an open subset of a Banach space X.

Definition 1. *Let f be a real-valued function with domain dom$(f) \subset U$. We say that f : dom$(f) \to \mathbb{R}$ has* an interval Lipschitz constant *$b \in \mathbf{C}(X^*)$ in a convex open subset $a \subset$ dom(f) if for all $x, y \in a$ we have: $b(x-y) \sqsubseteq f(x) - f(y)$. The* single tie *$\delta(a, b)$ of a with b is the collection of all real-valued partial functions f on U with $a \subset$ dom$(f) \subset U$ which have an interval Lipschitz constant b in a.*

For example, if $X = \mathbb{R}^2$ and b is the compact rectangle $b_1 \times b_2$ (with compact intervals $b_1, b_2 \subset \mathbb{R}$), the information relation above reduces to:

$$b_1(x_1 - y_1) + b_2(x_2 - y_2) \sqsubseteq f(x) - f(y).$$

For any topological space Z and any bounded complete dcpo D with bottom $\perp$, let $Z \to D$ be the bounded complete dcpo of Scott continuous functions from Z to D. The domain of $f : Z \to D$ is defined as dom$(f) = \{x : f(x) \neq \perp\}$. In particular, for any open subset $a \subset Z$ and any $b \in D$, the *single-step function* $a \searrow b : Z \to D$, with $(a \searrow b)(x) = b$ if $x \in a$ and $(a \searrow b)(x) = \perp$ if $x \notin a$, is Scott continuous and has domain a. A *step function* is then the supremum of any finite set of consistent single-step functions. In the sequel, we consider the dcpo $U \to \mathbf{C}(X^*)$ of Scott continuous functions with $U \subset X$ equipped with its the norm topology.

The following proposition justifies our definition of the interval Lipschitz constant. Let a be a convex open subset of X.

Proposition 1. *If $f : a \to \mathbb{R}$ is $C^1(a)$ i.e., f is Fréchet differentiable and $f' : a \to X^*$ is continuous, then the following three conditions are equivalent: (i) $f \in \delta(a, b)$, (ii) $\forall z \in a.\ f'(z) \in b$ and (iii) $a \searrow b \sqsubseteq f'$.*

We will now see that ties have a dual property in relation to step functions of type $U \to \mathbf{C}(X^*)$. For the rest of this section, we assume we are in an infinite dimensional Banach space or in the finite dimensional space $\mathbb{R}^n$ with $n \geq 2$. The case $n = 1$ is completely covered in [9].

Definition 2. *A* tie *of partial real-valued functions on U is any intersection $\Delta = \bigcap_{i \in I} \delta(a_i, b_i)$, for an arbitrary indexing set I. The* domain *of a non-empty tie Δ is defined as dom$(\Delta) = \bigcup_{i \in I} \{a_i \mid b_i \neq \perp\}$.*

If a non-empty tie is given by the intersection of a finite number of single ties, then it gives us a family of functions with a *finite* set of consistent differential properties. Generally, a non-empty tie gives a family of functions with a consistent set of differential properties.

Recall that a function $f : U \to \mathbb{R}$ defined on the open set $U \subseteq X$ is *locally Lipschitz* if it is Lipschitz in a neighbourhood of any point in U.

Proposition 2. *If Δ is a tie and $f \in \Delta$, then f is locally Lipschitz on dom(Δ).*

We now collect some fundamental properties of ties, which we will use later. The next proposition is the key technical result for the development of our theory.

Proposition 3. *For any indexing set I, the family of step functions $(a_i \searrow b_i)_{i\in I}$ is consistent if $\bigcap_{i\in I} \delta(a_i, b_i) \neq \emptyset$.*

Proposition 4. $(\mathbf{T}(U) \setminus \{\emptyset\}, \supseteq)$ *is a dcpo.*

For any topological space Z and any bounded complete dcpo D, let $Z \to_s D$ be the subset of $Z \to D$ consisting of Scott continuous functions which are supremums of step functions, i.e., $f = \sup_{i\in I} a_i \searrow b_i$ for a family $(a_i \searrow b_i)_{i\in I}$ of step functions with a_i an open subset of Z and $b_i \in D$.

Consider $U \to_s \mathbf{C}(X^*)$. Since any open set $a \subset X$ is the union of open balls, we can assume without loss of generality that the open subsets a_i $(i \in I)$ in the expression for f above are convex. It is easy to check that $U \to_s \mathbf{C}(X^*)$ is a dcpo.

We now show that, for any Banach space X, the set of maximal elements of $U \to_s \mathbf{C}(X^*)$ contains the set of functions of type $U \to X^*$, which are continuous with respect to the norm topology on U and X^*. Recall that a metric space is separable if it has a countable dense subset.

Proposition 5

(i) *If $f : U \to X^*$ is continuous with respect to the norm topologies on U and X^*, then $f \in U \to_s \mathbf{C}(X^*)$. Moreover, if X is separable with a countable dense subset $P \subset X$, then f is the lub of single step functions of the form $a \searrow b$ where a is an open ball centred at a point of P with rational radius whereas b is a closed ball centred at a point of P with a rational radius.*

(ii) *If $f : U \to \mathbb{R}$ is continuous with respect to the norm topology on U, then $f \in U \to_s \mathbf{I}\mathbb{R}$. Moreover, if X is separable with a countable dense subset $P \subset X$, then f is the lub of single step functions of the form $a \searrow b$ where a is an open ball centred at a point of P with rational radius whereas b is a rational compact interval.*

We are finally in a position to define the L-primitives of a Scott continuous function; in fact now we can do more and define:

Definition 3. *The* L-primitive map $\int : (U \to_s \mathbf{C}(X^*)) \to \mathbf{T}(U)$ *is defined by*

$$\int f = \bigcap_{a \searrow b \sqsubseteq f} \delta(a, b).$$

We call $\int f$ the L-primitives *of f.*

Proposition 6. *If $f = \sup_{i\in I} a_i \searrow b_i$, then $\int f = \bigcap \delta(a_i, b_i)$.*

Proposition 7. *The L-primitive map is continuous and onto the set of non-empty ties.*

If $X = \mathbb{R}^n$, for $n \geq 2$ or if X is infinite dimensional, the L-primitive map will have the empty tie in its range, a situation which does not occur for $n = 1$. This is similar to the situation in classical analysis in which a continuous vector field in $\mathbb{R}^n$ for $n > 1$ may not be an exact differential.

Example 1. Let $g \in \mathbb{R}^2 \to \mathbf{C}(R^2)$ be the maximal function given by $g(x,y) = (g_1(x,y), g_2(x,y))$ with $g_1(x,y) = 1$ and $g_2(x,y) = x$. Then $\frac{\partial g_1}{\partial y} = 0 \neq 1 = \frac{\partial g_2}{\partial x}$, and it will follow as in classical analysis that $\int g = \emptyset$.

3 The L-Derivative

Given a Scott continuous function $f : U \to \mathbb{R}$, the relation $f \in \delta(a,b)$ provides, as we have seen, finitary information about the local interval Lipschitz properties of f. By collecting all such local information, we obtain the complete differential properties of f, namely its L-derivative.

Definition 4. *The* L-derivative *of a continuous function $f : U \to \mathbb{R}$ is the map*

$$\mathcal{L}f : U \to \mathbf{C}(X^*), \qquad \textit{given by} \qquad \mathcal{L}f = \sup_{f \in \delta(a,b)} a \searrow b.$$

Theorem 1

(i) *The L-derivative is well-defined and Scott continuous.*
(ii) *If $f \in C^1(U)$ then $\mathcal{L}f = f'$.*
(iii) *$f \in \delta(a,b)$ iff $a \searrow b \sqsubseteq \mathcal{L}f$.*

Since the Scott topology refines the upper topology on $\mathbf{C}(X^*)$, we also obtain:

Corollary 1. *The L-derivative of any continuous function $X \to \mathbb{R}$ is upper semi-continuous.* □

We now obtain the generalization of Theorem 1(iii) to ties, which provides a duality between the L-derivative and L-primitives and can be considered as a general version of the Fundamental Theorem of Calculus.

Theorem 2. (Fundamental Theorem of Calculus) *For any $g \in U \to_s \mathbf{C}(X^*)$,*

$$f \in \int g \iff g \sqsubseteq \mathcal{L}f.$$

We will now see that the Gâteaux derivative, if it exists, is always in the L-derivative.

Corollary 2. *The Gâteaux derivative of f at x, when it exists, belongs to the L-derivative. Similarly for the Fréchet derivative.* □

In order to obtain the next corollary we first need the following characterization of the generalized gradient.

Lemma 1. *For any locally Lipschitz function f we have: $A \in \partial f(x)$ iff for all $v \in X$,*

$$\liminf_{\substack{y \to x \\ t \downarrow 0}} \frac{f(y+tv) - f(y)}{t} \leq A(v) \leq \limsup_{\substack{y \to x \\ t \downarrow 0}} \frac{f(y+tv) - f(y)}{t}$$

Corollary 3. *The generalized (Clarke) gradient is contained in the L-derivative.*

We do not know if the L-derivative and the Clarke gradient coincide on an infinite dimensional Banach space. We do know however that in finite dimensions they are the same, as we have shown in the full version of this paper [6].

4 Domain for Lipschitz Functions

We will construct a domain for locally Lipschitz functions and for $C^1(U)$. The idea is to use step functions in $U \to_s \mathbf{I}\mathbb{R}$ to represent the function and step functions in $U \to \mathbf{C}(X^*)$ to represent the differential properties of the function. Note that a continuous partial function f of type $U \to \mathbb{R}$, as we have considered in defining ties of functions in Section 2, can be regarded as an element $\hat{f}$ of $U \to_s \mathbf{I}\mathbb{R}$ with $\hat{f}(x) = f(x)$ if $f(x)$ is defined and $\hat{f}(x) = \bot = \mathbb{R}$ otherwise; we always identify f and $\hat{f}$. Furthermore, a function $f \in U \to \mathbf{I}\mathbb{R}$ is given by a pair of respectively lower and upper semi-continuous functions $f^-, f^+ : U \to \mathbb{R}$ with $f(x) = [f^-(x), f^+(x)]$.

Consider the *consistency* relation

$$\mathsf{Cons} \subset (U \to_s \mathbf{I}\mathbb{R}) \times (U \to_s \mathbf{C}(X^*)),$$

defined by $(f, g) \in \mathsf{Cons}$ if $\uparrow f \cap \int g \neq \emptyset$. For a consistent (f, g), we think of f as the *function part* or the *function approximation* and g as the *derivative part* or the *derivative approximation*. We will show that the consistency relation is Scott closed. The proofs of the rest of results in this section are essentially as in [9] for the case of $X = \mathbb{R}$.

Proposition 8. *Let $g \in U \to_s \mathbf{C}(X^*)$ and $(f_i)_{i \in I}$ be a non-empty family of functions f_i : dom$(g) \to \mathbb{R}$ with $f_i \in \int g$ for all $i \in I$. If $h_1 = \inf_{i \in I} f_i$ is real-valued then $h_1 \in \int g$. Similarly, if $h_2 = \sup_{i \in I} f_i$ is real-valued, then $h_2 \in \int g$.*

Let $R[0,1]$ be the set of partial maps of $[0,1]$ into the extended real line. Consider the two dcpo's $(R[0,1], \leq)$ and $(R[0,1], \geq)$. Define the maps $s : (U \to_s \mathbf{I}\mathbb{R}) \times (U \to_s \mathbf{C}(X^*)) \to (R, \leq)$ and $t : (U \to_s \mathbf{I}\mathbb{R}) \times (U \to_s \mathbf{C}(X^*) \to (R, \geq)$ by

$$s : (f, g) \mapsto \inf\{h : \mathrm{dom}(g) \to \mathbb{R} \mid h \in \int g \;\&\; h \geq f^-\}$$

$$t : (f, g) \mapsto \sup\{h : \mathrm{dom}(g) \to \mathbb{R} \mid h \in \int g \;\&\; h \leq f^+\}.$$

We use the convention that the infimum and the supremum of the empty set are ∞ and $-\infty$, respectively. Note that given a connected component A of dom(g) with $A \cap$ dom$(f) = \emptyset$, then $s(f, g)(x) = -\infty$ and $t(s, f)(x) = \infty$ for $x \in A$. In words, $s(f, g)$ is the least primitive map of g that is greater than the lower part of f, whereas $t(f, g)$ is greatest primitive map of g less that the upper part of f.

Proposition 9. *The following are equivalent:*

(i) $(f, g) \in \mathsf{Cons}$.
(ii) $s(f, g) \leq t(f, g)$.
(iii) *There is a continuous map h : dom$(g) \to \mathbb{R}$ with $g \sqsubseteq \mathcal{L}h$ and $f \sqsubseteq h$ on dom(g).*

Moreover, s and t are well-behaved:

Proposition 10. *The maps s and t are Scott continuous.*

This enables us to deduce:

Corollary 4. *The relation* Cons *is Scott closed.*

We can now sum up the situation for a consistent pair of function and derivative information.

Corollary 5. *Let* $(f, g) \in \mathsf{Cons}$. *Then in each connected component* A *of the domain of definition of* g *which intersects the domain of definition of* f*, there exist two locally Lipschitz functions* $s : A \to \mathbb{R}$ *and* $t : A \to \mathbb{R}$ *such that* $s, t \in \uparrow f \cap \int g$ *and for each* $u \in \uparrow f \cap \int g$*, we have with* $s(x) \le u(x) \le t(x)$ *for all* $x \in A$.

We now can define a basic construct of this paper:

Definition 5. *Define*

$$D^1(U) = \{(f, g) \in (U \to_s \mathbf{I}\mathbb{R}) \times (U \to_s \mathbf{C}(X^*)) : (f, g) \in \mathsf{Cons}\}.$$

From Corollary 4, we obtain:

Corollary 6. *The poset* $D^1(U)$ *is a bounded complete dcpo.*

Proposition 11. *For any* $f \in (U \to \mathbb{R})$ *the element* $(f, \mathcal{L}f)$ *is a maximal element of* $D^1(U)$.

For a locally Lipschitz function $f : U \to \mathbb{R}$ the L-derivative satisfies $\mathcal{L}f(x) \neq \bot$ for all $x \in U$, whereas for a piecewise C^1 function f we further have the property that $\mathcal{L}f(x)$ is maximal except for a finite set of points.

4.1 Computability

Let Z be a topological space with a countable basis M of its open subsets, and D a bounded complete dcpo with a countable subset $E \subset D$. Let $(f_i)_{i\ge 0}$ be an effective enumeration of the class of step functions of $Z \to D$ made from single-step functions $a \searrow b$ where $a \in M$ and $b \in E$. We say $f \in U \to_s D$ is *computable* with respect to this enumeration if there exists a total recursive function $\phi : \mathbb{N} \to \mathbb{N}$ such that $(f_{\phi(n)})_{n\ge 0}$ is an increasing sequence with $f = \sup_{n\ge 0} f_{\phi(n)}$.

When, in addition, Z is locally compact and D is a countably based continuous dcpo, then $Z \to D$ is a countably based bounded complete continuous dcpo, which can be given an effective structure. In this case, we obtain the same class of computable elements with any effective change of a countable basis of D. In general however, the computable elements will depend on the countable subset E.

Suppose now that X is a separable Banach space, with a countable dense set $P \subset X$. Then the collection of open balls centred at points of P with rational radii provides a countable basis of the norm topology on X. We use the rational compact intervals as a countable basis of $\mathbf{I}\mathbb{R}$ and the collection of closed balls of X^* with centres at points P with rational radii as a countable subset of $\mathbf{C}(X^*)$ to generate two countable sets, S_1 and S_2 say, of step functions for the two dcpo's $U \to_s \mathbf{I}\mathbb{R}$ and $U \to_s \mathbf{C}(X^*)$. We then obtain an enumeration $(f_i)_{i\ge 0}$ of S_1 and an enumeration $(g_i)_{i\ge 0}$ of S_2.

By Proposition 5, we know that any continuous function $f : U \to \mathbb{R}$ and any function $g : U \to \mathbf{C}X^*$ continuous with respect to the norm topology on X and X^*, is the supremum of step functions in S_1 and S_2 respectively. We say that f is *computable* with respect to the enumeration $(f_i)_{i\geq 0}$, respectively g is *computable* with respect to $(g_i)_{i\geq 0}$, if f considered as an element of $U \to \mathbf{I}\mathbb{R}$, respectively g considered as an element of $U \to \mathbf{C}(X^*)$, is computable with respect to the enumeration.

The question of decidability of the predicate $(f_i, g_j) \in \mathsf{Cons}$, for $i, j \geq 0$, depends on the structure of the Banach space X under consideration and on the countable subsets $(f_i)_{i\geq 0}$ and $(g_i)_{i\geq 0}$. Assuming that this is indeed decidable, we use an oracle to decide if $(f_i, g_j) \in \mathsf{Cons}$ for $i, j \geq 0$, which enables us to construct an enumeration $(h_i)_{i\geq 0}$ of a countable set, S_3 say, of step functions of $D^1(U)$, where $h_i = (f_{p(i)}, g_{q(i)})$ for $i \geq 0$ with $p, q : \mathbb{N} \to \mathbb{N}$ total recursive functions. By Proposition 5, we know that if $f : U \to \mathbb{R}$ is Fréchet differentiable then (f, f') is the lub of step functions in S_3. We thus say that f and its Fréchet derivative f' are computable with respect to $(h_i)_{i\geq 0}$ if (f, f') considered as a maximal element of $D^1(U)$ is computable with respect to this enumeration.

5 Further Work and Open Problems

As we have shown in the full version of this paper [6], when X is finite dimensional $D^1(U)$ is a countably based continuous Scott domain that can be given an effective structure with respect to which Cons is decidable.

We also see in the full paper that in finite dimensions the L-derivative and the Clarke gradient coincide thus providing a computable representation for the latter in this case.

As already pointed out, it remains an open question if the L-derivative coincides with the Clarke gradient on infinite dimensional Banach spaces. It is also unknown if the Clarke gradient is upper semi-continuous in infinite dimensions, a property which holds for the L-derivative as we have shown in this paper. On the other hand, it will be interesting to see if the L-derivative can be extended to functions from a Banach space to a finite dimensional Banach space, for example to the complex plane, a case which has applications in quantum field theory.

There are quite a few unsolved problems in finite dimensions. For $n = 1$, the algorithm for testing consistency of basis elements in $D^1(U)$ as already mentioned. For $D^2(U)$, consistency on basis elements is decidable but the present algorithm to test it is super-exponential in the total number of single-step functions for the three approximations of the function part, the derivative part and the second derivative part. [1]. Decidability of consistency for $D^m(U)$ when $m > 2$ is unknown. For $n = 2$, consistency on basis elements for $D^1(U)$ is decidable but the algorithm to test it in [10] is super-exponential. The complexity of consistency test in this case is unknown as is the question of decidability of consistency of basis elements for $D^m(U)$ when $m > 1$.

Based on the domain-theoretic framework for differential calculus, one can embark on the task of constructing a domain for orientable Euclidean manifolds, which would extend the set-theoretic model for computational geometry and solid modelling presented in [8] to the piecewise smooth setting.

Acknowledgement

I would like to thank André Lieutier and Dirk Pattinson for reading and checking various parts of this work.

References

1. Abolfathbeigi, S., Mahmoudi, M.: Manuscript in Persian (2003)
2. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. 3, Clarendon Press, Oxford (1994)
3. Clarke, F.H.: Private communications. Summer (2005)
4. Clarke, F.H.: Optimization and Nonsmooth Analysis. Wiley, Chichester (1983)
5. Edalat, A.: Dynamical systems, measures and fractals via domain theory. Information and Computation 120(1), 32–48 (1995)
6. Edalat, A.: A continuous derivative for real-valued functions. Invited submission to New Computational Paradigms (2006) Available at www.doc.ic.ac.uk/∼ae/papers/banachf.ps.
7. Edalat, A., Krznarić, M., Lieutier, A.: Domain-theoretic solution of differential equations (scalar fields). In: Proceedings of MFPS XIX, volume 83 of Electronic Notes in Theoretical Computer Science (2003) Full paper in www.doc.ic.ac.uk/∼ae/papers/scalar.ps
8. Edalat, A., Lieutier, A.: Foundation of a computable solid modelling. Theoretical Computer Science 284(2), 319–345 (2002)
9. Edalat, A., Lieutier, A.: Domain theory and differential calculus (Functions of one variable). Mathematical Structures in Computer Science 14(6), 771–802 (2004)
10. Edalat, A., Lieutier, A., Pattinson, D.: A computational model for multi-variable differential calculus. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 505–519. Springer, Heidelberg (2005)
11. Edalat, A., Pattinson, D.: A domain theoretic account of Picard's theorem. In: Proc. ICALP 2004 LNCS vol. 3142, pp. 494–505 (2004), Full paper in `www.doc.ic.ac.uk/~ae/papers/picard.icalp.ps`
12. Fars, N.: Aspects analytiques dans la mathematique de shraf al-din al-tusi. Historia Sc. 5(1) (1995)
13. Fars, N.: Le calcul du maximum et la 'derive' selon shraf al-din al-tusi. Arabic Sci. Philos. 5(2), 219–237 (1995)
14. Gierz, G., Hofmann, K.H., Keimel, K., Lawson, J.D., Mislove, M., Scott, D.S.: Continuous Lattices and Domains. Cambridge University Press, UK (2003)
15. Hogendijk, J.P.: Shraf al-din al-tusi on the number of positive roots of cubic equations. Fund. Math. 16(1), 69–85 (1989)
16. Pour-El, M.B., Richards, J.I.: A computable ordinary differential equation which possesses no computable solution. Annals Math. Logic 17, 61–90 (1979)
17. Scott, D.S.: Outline of a mathematical theory of computation. In: 4th Annual Princeton Conference on Information Sciences and Systems, pp. 169–176 (1970)
18. Weihrauch, K.: Computable Analysis (An Introduction). Springer, Heidelberg (2000)

Refocusing Generalised Normalisation

José Espírito Santo[⋆]

Departamento de Matemática
Universidade do Minho
Portugal
jes@math.uminho.pt

Abstract. When defined with general elimination/application rules, natural deduction and λ-calculus become closer to sequent calculus. In order to get real isomorphism, normalisation has to be defined in a "multiary" variant, in which reduction rules are necessarily non-local (reason: nomalisation, like cut-elimination, acts at the *head* of applicative terms, but natural deduction focuses at the *tail* of such terms). Non-local rules are bad, for instance, for the mechanization of the system. A solution is to extend natural deduction even further to a *unified calculus* based on the unification of cut and general elimination. In the unified calculus, a sequent term behaves like in the sequent calculus, whereas the reduction steps of a natural deduction term are interleaved with explicit steps for bringing heads to focus. A variant of the calculus has the symmetric role of improving sequent calculus in dealing with tail-active permutative conversions.

Keywords: normalisation, generalised elimination rules, multiarity.

1 Introduction

Natural deduction with general elimination rules is closer to sequent calculus than traditional natural deduction [10]. This paper investigates the exact realization of this claim, and the outcome of such realization, in the context of intuitionistic implicational logic.

In [10] von Plato obtains a perfect correspondence between "fully normal" deductions and cut-free sequent derivations, extending to a *bijection* between natural deductions and a subset of sequent derivations (where cuts are necessarily "right-principal"). In this paper we start by investigating the *isomorphism* between cut-elimination and generalised normalisation. The systems are presented as typed λ-calculi, and a computational reading is present throughout.

Sequent calculus is presented as system λGm, where cuts are "right-principal". In λGm there is the so-called *multiarity* facility, *i.e.* the facility of not naming an active, linearly left-introduced formula [9]. In this system cuts correspond to applicative terms, consisting of a head, a list of arguments, and a "continuation" (or tail).

⋆ The author is supported by FCT through the Centro de Matemática da Universidade do Minho.

In order to get real isomorphism, natural deduction is defined as the system λNm, a multiary extension of von Plato's system. However, normalisation has to be defined with non-local reduction rules. The reason is simple: normalisation, like cut-elimination, acts at the *head* of applicative terms, but natural deduction focuses at the *tail* of such terms. Now, in a multiary system, heads are arbitrarily distant from tails. So, we get isomorphism, but normalisation in the relevant natural deduction system is just a clumsy way of doing cut-elimination. Symmetrically, sequent calculus is the wrong setting for doing tail-active permutative conversions.

A way out of this situation, which is simultaneously the main outcome of proving $\lambda Gm \cong \lambda Nm$, suggested by the analysis of this isomorphism, is to extend natural deduction even further, to a calculus λU that *unifies* λGm and λNm. This *unified calculus* is based on the unification of cut and general elimination.

In the unified calculus all reduction rules are local, and a sequent term behaves like in the sequent calculus, whereas the reduction steps of a natural deduction term are interleaved with explicit steps for bringing heads to focus. This gives an implementation of multiary normalisation with local reduction steps.

The unified calculus seems particularly appropriate for dealing with conversions with are both head-acting and tail-acting. A variant of the calculus is suggested which has the role of improving sequent calculus in dealing with tail-acting permutative conversions.

Structure of the paper: Section 2 presents λGm, λNm and $\lambda Gm \cong \lambda Nm$. Section 3 is the central contribution, presenting the unified calculus λU and its properties and variants. Section 4 concludes.

Notations: Types (=formulas) are ranged over by A, B, C and generated from type variables using the "arrow type" (=implication), written $A \supset B$. Contexts Γ are consistent sets of declarations $x : A$. "Consistent" means that for each variable x there is at most one declaration in Γ. The notation $\Gamma, x : A$ always denotes a consistent union, that is, one that produces a consistent set. Barendregt's variable convention is adopted. In particular, we take renaming of bound variables for granted. Substitution is denoted by $[\_/x]\_$. "s.n." abbreviates "strongly normalising".

2 Isomorphism

Natural deduction with general elimination rules: This system [10] may be presented as a type system for the λ-calculus with generalised application. The latter is the system ΛJ of [6], which we rename here as λg, for the sake of uniformity with the names of other calculi. Terms of λg are given by

$$M, N, P \ ::= \ x \,|\, \lambda x.M \,|\, M(N, x.P)$$

The typing rule for generalised application is

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash M(N, x.P) : C}\ gElim$$

The λg-calculus has two reduction rules

$$\begin{array}{ll} (\beta) & (\lambda x.M)(N, y.P) \to [[N/x]M/y]P \\ (\pi) & M(N, x.P)(N', y.P') \to M(N, x.P(N', y.P')) \ . \end{array}$$

The usual λ-calculus embeds in λg by setting $MN = M(N, x.x)$. Likewise, *modus ponens* (=Gentzen's elimination rule for implication) may be seen as the particular case of $gElim$ where $B = C$ and the rightmost premiss is omitted.

Sequent calculus: We present the system λGm ("G" is after Gentzen). It should be understood as an extension of the λ-calculus whose typing rules define a sequent calculus. As an extension of the λ-calculus, it adds to the application constructor the features of generality and multiarity, to be explained soon. As a sequent calculus, λGm contains, as primitive, cuts of a special form, namely those whose cut-formula in the right premiss is principal in a left-introduction.

There are two sorts of expressions in λGm:

$$\begin{array}{ll} \text{(Terms)} & t, u, v ::= x \,|\, \lambda x.t \,|\, tl \\ \text{(Lists)} & \qquad l ::= u \cdot (x)v \,|\, u :: l \end{array}$$

A term of the form tl is called a *cut*. In general, l has the form $u_1 :: ... :: u_{n-1} :: u_n \cdot (x)v$, for some $n \geq 1$. We regard these expressions as *generalised lists*. In tl, think of t as a function and of l as an expression that provides a non-empty list of arguments for t (this is the multiarity feature), plus some "continuation" $(x)v$, specifying what to do after the last argument is consumed (this is the generality feature).

Define $[u] = u \cdot (x)x$. Expressions of the form $u_1 :: ... :: u_{n-1} :: [u_n]$ are regarded as lists, writen as $[u_1, ..., u_{n-1}, u_n]$, and interpreted in [5] as "applicative contexts". Lists in λGm may be interpreted as generalised applicative contexts.

There are two sorts of sequents in λGm, namely $\Gamma \vdash t : A$ and $\Gamma; A \vdash l : B$. The distinguished position in the antecedent of sequents of the latter kind is named the *stoup*. Typing rules are as follows:

$$\frac{}{\Gamma, x : A \vdash x : A}\ Axiom$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\ Right \qquad \frac{\Gamma \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash u :: l : C}\ plLeft$$

$$\frac{\Gamma \vdash t : A \quad \Gamma; A \vdash l : B}{\Gamma \vdash tl : B}\ Cut \qquad \frac{\Gamma \vdash u : A \quad \Gamma, x : B \vdash v : C}{\Gamma; A \supset B \vdash u \cdot (x)v : C}\ gLeft$$

These rules define a sequent calculus, with a primitive rule of cut, and two sorts of primitive left-introduction rules: general left-introduction ($gLeft$) and principal-linear left-introduction ($plLeft$). The system is such that, in any derivation, the stoup always contains a formula that is *principal and linear*, that is, principal in a left-introduction rule, and introduced without contraction. Given that the cut rule and the left-introduction rules require some of the active formulas and/or the principal formula to be in the stoup, these inference rules are of a particular kind. The cut-formula is always an implication and cut is right-principal. As to left-introduction rules, they are both linear, in the sense that they both introduce without contraction. In addition, rule $plLeft$, by requiring its right active formula to be principal and linear, is of the restricted form identified in [5,1,7].

There are three reduction rules

$$\begin{array}{ll}(\beta 1)\ (\lambda x.t)(u\cdot(y)v) \rightarrow [[u/x]t/y]v & (\pi)\ (tl)l' \rightarrow t(l@l') \\ (\beta 2)\quad (\lambda x.t)(u :: l) \rightarrow ([u/x]t)l & \end{array}$$

where $l@l'$ is the "append" of generalised lists l and l', defined by

$$(u :: l)@l' = u :: (l@l') \qquad (u\cdot(x)v)@l' = u\cdot(x)(vl').$$

Let $\beta = \beta 1 \cup \beta 2$. By *cut-elimination* we mean $\beta\pi$-reduction. A $\beta\pi$-nf is a term where every cut has the form xl. These normal forms correspond exactly to the multiary sequent terms of [9]. In λGm, a λg-term is a term without occurrences of $u :: l$ (hence every cut in a λg-term is a g-application $t(u\cdot(x)v)$).

Multiary natural deduction: We present the system λNm. It should be understood as an extension of the λg-calculus and of natural deduction with general elimination rules. This system has an implementation of the multiarity feature (the ability of forming chains of arguments for a function) within the framework of natural deduction.

Expressions in λNm are given by:

$$\begin{array}{lr}\text{(Terms)} & M, N, P ::= x \mid \lambda x.M \mid app(F, N, (x)P) \\ \text{(Functions)} & F ::= hd(M) \mid FN\end{array}$$

This is a syntax with two syntactic classes: terms and *functions*. A term of the form $app(F, N, (x)P)$ may be called either *gm-application* or *outer application*. Think of this construction as an extension of the generalized application of λg. Indeed, generalized application is recovered as $app(hd(M), N, (x)P)$, because any term M can be coerced to a function $hd(M)$. There is a second kind of application construction, FN, named *inner application* or *mp-application*. Here *mp* is mnemonic of *modus ponens*.

The elements of the second syntactic class are named functions because, in expressions, they only occur in the function position of applications. Application FN is inner because, being a function, occurs in the function position of another application. The general form of a function is $hd(M)N_1...N_{m-1}$, for some $m \geq 1$. A function of the form $hd(M)$ is called a *head*. An intuition about functions is that they are expressions which require an immediate and linear use. On the other hand, in $app(F, N, (x)P)$, the use of the application of F to N, specified by the "continuation" $(x)P$ is required neither to be immediate nor linear.

There are two sorts of sequents in λNm, namely $\Gamma \vdash M : A$ and $\Gamma \rhd F : A$. Typing rules are as follows:

$$\frac{}{\Gamma, x : A \vdash x : A}\ Assumption \qquad \frac{\Gamma \vdash M : A}{\Gamma \rhd hd(M) : A}\ Coercion$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ Intro \qquad \frac{\Gamma \rhd F : A \supset B \quad \Gamma \vdash N : A}{\Gamma \rhd FN : B}\ mpElim$$

$$\frac{\Gamma \rhd F : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash app(F, N, (x)P) : C}\ gmElim$$

In accordance with what was observed before, this is a natural deduction system extending that of von Plato's. The system contains two primitive elimination rules, *general multiary elimination* (or *outer elimination*) and *inner elimination*, the latter being a form of *modus ponens*. There is a further rule, whose instances are called *coercions*, with *coercion formula A*. The general elimination rule is recovered as a combination of outer elimination and coercion. In addition, a sequent of the form $\Gamma \rhd F : A$ occurs in a derivation of the system iff it occurs as the major premiss of an elimination and A is an implication. The *Coercion* rule, then, means that any sequent of the first kind can serve as major premiss of an elimination.

There are three reduction rules: two β-rules

$$\begin{array}{lr} (\beta 1) & app(hd(\lambda x.M), N, (y)P) \to [[N/x]M/y]P \\ (\beta 2) & hd(\lambda x.M)N \to hd([N/x]M)\ , \end{array}$$

and rule (π)

$$\begin{array}{l} app(hd(app(F, N, (x)P))N_1...N_{m-1}, N_m, (y)P') \to \\ \qquad \to app(F, N, (x)app(hd(P)N_1...N_{m-1}, N_m, (y)P'))\ , \end{array}$$

where $m \geq 1$. Let $\beta = \beta 1 \cup \beta 2$. By *gm-normalisation* we mean $\beta\pi$-reduction. Notice that rule $\beta 2$ is a relation on functions. As to rule π, if $F = hd(M)$ and $m = 1$, we recognize the π rule of λg. In the general case, rule π is non-local, because, if we let the redex be $app(F', N_m, (y)P)$, it requires the full inspection of F' until the head emerges. Also a $\beta 2$-reduction step requires the full inspection of function F of the application $app(F, N, (y)P)$ where the $\beta 2$-redex is located.

In λNm, a λg-terms is a term without occurrences of FN (hence every gm-application in a λg-term is a g-application $app(hd(M), N, (x)P)$). M is in $\beta\pi$-nf iff every coercion in M is of the form $hd(x)$. A derivation $\mathcal{D}$ in λNm is $\beta\pi$-normal iff *every coercion formula occurring in* $\mathcal{D}$ *is an assumption.* In particular, this gives von Plato's criterion of normality for λg-terms, because in λg coercion formula = main premiss of elimination.

Remark: Both λGm and λNm are new presentations of the system $\lambda \mathbf{J^m}$ of [3]. A gm-application is written there $t(u_1, [u_2, ..., u_n], (x)v)$. This representation brings to the surface both the head t and the "continuation" $(x)v$. The price to pay for these advantages is that the presentation in *op. cit.* has a hybrid proof-theoretical character. The typing rule of the gm-application constructor in *op. cit.* is an elimination rule, but lists l of a restricted form are primitive. Nevertheless, λGm and λNm inherit the properties of $\lambda\mathbf{J^m}$ [4]:

Theorem 1. *In λGm and λNm, $\beta\pi$-reduction is s.n. on typable terms and confluent.*

Mappings Θ and Ψ: We now define mappings between the set of λGm-terms and the set of λNm-terms. Once and for all, variables and λ-abstractions are mapped identically. The question will always be how to map cuts, left introductions and eliminations.

We start with a mapping $\Psi : \lambda Nm{-}Terms \longrightarrow \lambda Gm{-}Terms$. Let $\Psi(M) = t$, $\Psi(N_i) = u_i$ and $\Psi(P) = v$. The idea is to map, say, $app(hd(M)N_1N_2, N_3, (x)P)$ to $t(u_1 :: u_2 :: u_3 \cdot (x)v)$. This is achieved with the help of an auxiliary function $\Psi : \lambda Nm - Functions \times \lambda Gm - Lists \longrightarrow \lambda Gm - Terms$ as follows:

$$\begin{aligned} \Psi(x) &= x & \Psi(hd(M), l) &= (\Psi M)l \\ \Psi(\lambda x.M) &= \lambda x.\Psi M & \Psi(FN, l) &= \Psi(F, \Psi N :: l) \\ \Psi(app(F, N, (x)P)) &= \Psi(F, \Psi N \cdot (x)\Psi P) \end{aligned}$$

Next we consider a mapping $\Theta : \lambda Gm - Terms \longrightarrow \lambda Nm - Terms$. Let $\Theta(t) = M$, $\Theta(u_i) = N_i$ and $\Theta(v) = P$. The idea is to map, say, $t(u_1 :: u_2 :: u_3 \cdot (x)v)$ to $app(hd(M)N_1N_2, N_3, (x)P)$. This is achieved with the help of an auxiliary function $\Theta : \lambda Nm - Functions \times \lambda Gm - Lists \longrightarrow \lambda Nm - Terms$ as follows:

$$\begin{aligned} \Theta(x) &= x & \Theta(F, u \cdot (x)v) &= app(F, \Theta u, (x)\Theta v) \\ \Theta(\lambda x.t) &= \lambda x.\Theta t & \Theta(F, u :: l) &= \Theta(F\Theta u, l) \\ \Theta(tl) &= \Theta(hd(\Theta t), l) \end{aligned}$$

Theorem 2 (Isomorphism). *Mappings Ψ and Θ are sound, mutually inverse bijections between the set of λGm-terms and the set of λNm-terms. Moreover, for each $R \in \{\beta 1, \beta 2, \pi\}$:*

1. *$M \to_R M'$ in λNm iff $\Psi M \to_R \Psi M'$ in λGm.*
2. *$t \to_R t'$ in λGm iff $\Theta t \to_R \Theta t'$ in λNm.*

The proof follows the pattern of proof of similar results in [2].

3 Unification

A problem of wrong focus: Applicative terms in λGm and λNm have the form of cuts and gm-eliminations

$$t(u_1 :: ... :: u_{m-1} :: u_m \cdot (x)v) \ , \tag{1}$$

$$app(hd(M)N_1...N_{m-1}, N_m, (x)P) \ . \tag{2}$$

respectively. In both cases there is a head, m arguments ($m \geq 1$) and a tail (or continuation). In the first case, the term is split next to the head, with the rest of data organized as a list l; in the second case, the term is split just before the tail, with the rest of data organized as a function F. In the first case, the head is focused, in the second it is the tail that is focused.

Now both cut-elimination and gm-normalisation aim at reducing heads to variables, and are a process of transforming heads. In this respect, the focus of tails is unfortunate and explains the fact that both $\beta 2$ and π are non-local reduction rules in the natural deduction system λNm.

Non-local rules are bad, for instance, for the implementation of λNm, where the search for heads has to be made explicitly. The solution we propose is to extend λNm to a calculus where applicative terms are split at arbitrary position, and not just around the tail. This means that both functions F and lists l are used in the representation of applicative terms, and this representation turns out to unify both cuts and gm-eliminations.

The telescopic effect: The idea of manipulating functions F and lists l in the same system has many motivations. For instance, the intuition about lists l is that they are "applicative contexts", prescribing a linear and immediate use to some expression to be supplied; symmetrically, functions F are expressions which are used in a linear and immediate way (in the function position of some application). But so far functions and lists live in separate systems.

Another motivation is as follows. Let M_0 be the gm-application ([2]), and let $\Theta t = M$, $\Theta u_i = N_i$ and $\Theta v = P$. There are m choices of F, l such that $M_0 = \Theta(F, l)$, ranging from the choice $F = hd(M)N_1 ... N_{m-1}$ and $l = u_m \cdot (x)v$ to the choice $F = hd(M)$ and $l = u_1 :: ... u_{m-1} :: u_m \cdot (x)v$. This last case is particularly important, because the representation of application M_0 as $\Theta(hd(M), l)$, for such l, brings to the surface the head $hd(M)$. In general, we will use pattern matching of gm-application with $\Theta(hd(M), l)$ to obtain the effect of extracting the head of the application, an effect we call the *telescopic* effect. Similarly one extracts the tail of cuts by pattern-matching with $\Psi(F, u \cdot (x)v)$.

The telescopic effect is useful in making global rules look local. This is achieved by manipulating simultaneously, in the meta-language, both functions F and lists l. For instance, reduction rule π in λNm may be defined as follows:

$$(\pi) \qquad \Theta(hd(app(F, N, (x)P)), l) \rightarrow app(F, N, (x)\Theta(hd(P), l)) \ .$$

The calculus we introduce next manipulates expressions $\Theta(F, l)$ formally.

The unified calculus: Expressions in λU are given by:

$$\begin{array}{lrl} \text{(Terms)} & M, N, P & ::= x \,|\, \lambda x.M \,|\, \underline{\theta}(F, L) \\ \text{(Functions)} & F & ::= hd(M) \,|\, FN \\ \text{(Lists)} & L, K & ::= N :: L \,|\, N \cdot (x)P \end{array}$$

$\underline{\theta}(F, L)$ is called a *unified cut*. The symbol $\underline{\theta}$ is a formal counterpart of Θ. In $\underline{\theta}(F, L)$, we say that F is in *focus*. The new typing rule is:

$$\frac{\Gamma \rhd F : A \quad \Gamma ; L : A \vdash B}{\Gamma \vdash \underline{\theta}(F, L) : B}\ uCut$$

In λU, a *sequent term* is a term with no occurrences of FN, *i.e* an elimination-free term, whereas a *natural deduction term* is a term with no occurrences of $N :: L$, *i.e.* a left-introduction-free term. In sequent terms and natural deduction terms, unified cuts have the form

$$ML = \underline{\theta}(hd(M), L) \qquad\qquad app(F, N, (x)P) = \underline{\theta}(F, N \cdot (x)P)$$

respectively. These equations show how unified cut unifies cut and and gm-elimination. Sequent terms (resp. natural deduction terms) dispense with the syntactic class of functions F (resp. lists L) and constitute a copy of λGm-terms (resp. λNm-terms) in λU. Given a λGm-term t (resp. λNm-term M), we denote by t' (resp. M') its copy in λU.

The reduction rules of λU are as follows:

$$\begin{array}{rr}
(\beta 1) & \underline{\theta}(hd(\lambda x.M), N \cdot (y)P) \rightarrow [[N/x]M/y]P \\
(\beta 2) & \underline{\theta}(hd(\lambda x.M), N :: L) \rightarrow \underline{\theta}(hd([N/x]M), L) \\
(\pi) & \underline{\theta}(hd(\underline{\theta}(F, L)), K) \rightarrow \underline{\theta}(F, L@K) \\
(\underline{\psi}) & \underline{\theta}(FN, L) \rightarrow \underline{\theta}(F, N :: L)
\end{array}$$

Let $\beta = \beta 1 \cup \beta 2$. Rules β and π require a head in focus. For this reason, are local transformations. Rule $\underline{\psi}$ is a step towards focusing a head. A λU-term is a $\underline{\psi}$-nf iff it is a sequent term. $\underline{\psi}$-reduction is terminating (it decreases the number of occurrences of FN) and locally confluent. Hence it is confluent. We denote by $\underline{\psi}(M)$ the unique $\underline{\psi}$-nf of a λU-term M. It holds, for all $M \in \lambda Nm$, that

$$\underline{\psi}(M') = \Psi(M)' \ .$$

It is easy to see that sequent terms are closed for $\beta\pi$-reduction, and a λGm-term t $\beta\pi$-reduces in λGm exactly as t' $\beta\pi$-reduces in λU. Let us see what happens when we $\beta\pi$-reduce M' in λU, for $M \in \lambda Nm$.

Proposition 1. *Let $R \in \{\beta 1, \beta 2, \pi\}$.*

1. *In λU, if $M \rightarrow_R M_1$ and $M \rightarrow_{\underline{\psi}} M_2$ then there is M_3 such that $M_2 \rightarrow_R M_3$ and $M_1 \rightarrow^*_{\underline{\psi}} M_3$.*
2. *If $M \rightarrow_R N$ in λNm, then there are M_1, N_1 such that, in λU: $M_1 \rightarrow_R N_1$ and $M \rightarrow^*_{\underline{\psi}} M_1$ and $N \rightarrow^*_{\underline{\psi}} N_1$.*

Theorem 3. *Suppose $M_1 \rightarrow_{R_1} M_2 \rightarrow (\cdots) \rightarrow M_n \rightarrow_{R_n} M_{n+1}$ is a $\beta\pi$-reduction sequence in λNm (hence each $R_i \in \{\beta 1, \beta 2, \pi\}$). Then, the reductions in λU depicted in Fig. 1 hold, when vertical arrows denote $\underline{\psi}$-reduction.*

Proof: By induction on n, using the previous proposition and confluence of $\underline{\psi}$. ■

Regarding Fig. 1 again, we can now compare reduction of M_1 in λNm with reduction of M'_1 in λU. The latter is obtained from the former by interleaving

$$\begin{tikzcd}
M_1 \arrow[r, "R_1"] & M_2 \arrow[r, "R_2"] & M_3 \arrow[r] & (\cdots) \arrow[r] & M_n \arrow[r, "R_n"] & M_{n+1} \\
M_1' \arrow[d, dotted] & M_2' \arrow[d, dotted] & M_3' \arrow[dd, dotted] & (\cdots) & M_n' \arrow[ddd, dotted] & M_{n+1}' \arrow[ddd, dotted] \\
\bullet \arrow[r, "R_1"'] \arrow[dddd, dotted] & \bullet \arrow[d, dotted] & & & & \\
& \bullet \arrow[r, "R_2"'] \arrow[ddd, dotted] & \bullet \arrow[ddd, dotted] & (\cdots) & & \\
& & & & \bullet \arrow[r, "R_n"'] \arrow[dd, dotted] & \bullet \arrow[dd, dotted] \\
& & & & & \\
\underline{\psi}(M_1') \arrow[r, "R_1"] & \underline{\psi}(M_2') \arrow[r, "R_2"] & \underline{\psi}(M_3') \arrow[r] & (\cdots) \arrow[r] & \underline{\psi}(M_n') \arrow[r, "R_n"] & \underline{\psi}(M_{n+1}')
\end{tikzcd}$$

Fig. 1. Normalisation in λU

$\underline{\psi}$-reduction steps. To a possibly non-local reduction step $\rightarrow_{R_i}$ in the former corresponds a necessarily local reduction step $\rightarrow_{R_i}$ in the latter. The interleaved $\underline{\psi}$-reduction steps do explicitly the focusing of heads implicit in the reduction steps at the λNm level. The reduction of $\underline{\psi}(M_1')$ is morally the same as the reduction of $\Psi(M_1)$ in λGm. Fig. 1 is a refinement of the "only if" part of statement 1 in Theorem 2.

Finally, observe that part 1 of Proposition 1 allows the projection of $\beta\pi\underline{\psi}$-reduction sequences of λU into $\beta\pi$-reduction sequences of λGm. So, it is easy to lift Theorem 1 from λGm to λU.

Theorem 4. *$\beta\pi\underline{\psi}$-reduction in λU is s.n. on typable terms and confluent.*

Variant of the unified calculus: Consider a variant of permutative conversion p of $\lambda\mathbf{J^m}$ [3], given here for λGm with the help of telescopic effect:

$$(p) \qquad \Psi(F, u \cdot (x)v) \rightarrow [\Psi(F,[u])/x]v, \text{ if } v \neq x \ .$$

This rule acts on tails, eliminating occurrences of general left-introduction. It is a non-local rule in λGm. A variant of the unified calculus, seen as an extension of λGm, can be defined for tail-active conversions, with terms:

$$t, u, v \ ::= \ x \,|\, \lambda x.t \,|\, \underline{\psi}(f, l)$$

The p-rule now reads $\underline{\psi}(f, u \cdot (x)v) \rightarrow [\underline{\psi}(f,[u])/x]v$. In $\underline{\psi}(f,l)$ the focus is l and a rule $\underline{\theta}$ is needed for bringing continuations to focus: $\underline{\psi}(f, u :: l) \rightarrow \underline{\psi}(fu, l)$.

4 Conclusion

From a logical point of view, λU achieves the same goal as the “uniform” calculus of [8], but with a radically different approach (the latter approach is to extend natural deduction with general elimination and general introduction rules).

It is to be expected that λU admits extensions (encompassing a sequent calculus where cuts are not necessarily right-principal) and further variants. For instance, consider the following rules for λNm, given with telescopic effect:

$$(\mu) \qquad \Theta(F, N \cdot (x)\Theta(hd(x), l)) \to \Theta(FN, l), \text{ if } x \notin l \ .$$

This is a natural deduction variant of rule μ introduced in [9]. Consider the μ-redex. We analyze the tail of the outer applicative term and the head of the inner applicative term. This rules needs a mix of head and tail focus. Maybe a good system for dealing with such rules is a variant of the unified calculus with reduction modulo the equation $\underline{\theta}(FN, L) = \underline{\theta}(F, N :: L)$.

Acknowledgment. The diagram in Fig. 1 was produced with Paul Taylor’s macros.

References

1. Dyckhoff, R., Pinto, L.: Permutability of proofs in intuitionistic sequent calculi. Theoretical Computer Science 212, 141–155 (1999)
2. Espírito Santo, J.: Conservative extensions of the λ-calculus for the computational interpretation of sequent calculus. PhD thesis, University of Edinburgh, (2002) Available at http://www.lfcs.informatics.ed.ac.uk/reports/.
3. Espírito Santo, J., Pinto, L.: Permutative conversions in intuitionistic multiary sequent calculus with cuts. In: Hoffman, M. (ed.) TLCA 2003. LNCS, vol. 2701, pp. 286–300. Springer, Heidelberg (2003)
4. Espírito Santo, J., Pinto, L.: Confluence and strong normalisation of the generalised multiary λ-calculus. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, Springer, Heidelberg (2004)
5. Herbelin, H.: A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 61–75. Springer, Heidelberg (1995)
6. Joachimski, F., Matthes, R.: Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel’s T. Archive for Mathematical Logic 42, 59–87 (2003)
7. Mints, G.: Normal forms for sequent derivations. In: Odifreddi, P., Peters, A.K. (eds.) Kreiseliana, pp. 469–492. Wellesley, Massachusetts (1996)
8. Negri, S., von Plato, J.: Structural Proof Theory. Cambridge (2001)
9. Schwichtenberg, H.: Termination of permutative conversions in intuitionistic gentzen calculi. Theoretical Computer Science, 212 (1999)
10. von Plato, J.: Natural deduction with general elimination rules. Annals of Mathematical Logic 40(7), 541–567 (2001)

The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number*

Michael Fellows[1,2] and Frances Rosamond[1]

[1] University of Newcastle, Callaghan NSW 2308, Australia
{michael.fellows,frances.rosamond}@newcastle.edu.au
[2] Durham University, Institute of Advanced Study,
Durham DH1 3RL, United Kingdom

Abstract. In the framework of parameterized complexity, exploring how one parameter affects the complexity of a different parameterized (or unparameterized problem) is of general interest. A well-developed example is the investigation of how the parameter *treewidth* influences the complexity of (other) graph problems. The reason why such investigations are of general interest is that real-world input distributions for computational problems often inherit structure from the natural computational processes that produce the problem instances (not necessarily in obvious, or well-understood ways). The *max leaf number* of a connected graph G is the maximum number of leaves in a spanning tree for G. Exploring questions analogous to the well-studied case of treewidth, we can ask: how hard is it to solve 3-Coloring or Hamilton Path or Minimum Dominating Set for graphs of bounded max leaf number? We do two things:

(1) We describe much improved FPT algorithms for a large number of graph problems, for input of bounded max leaf number, based on the polynomial-time extremal structure theory associated to the parameter max leaf number.
(2) The way that we obtain these concrete algorithmic results is general and systematic. We describe the approach.

1 Introduction

The analysis of the complexity of problems, for graphs of bounded treewidth, is well-developed and supports many systematic approaches that have developed over a number of years [Cou90, ALS91, Bod96, DF99, Nie06, BK07]. For example, determining whether a graph is 3-colorable can be solved in time $O(n)$ for

* This research has been supported by the Australian Research Council through the Australian Centre for Bioinformatics, by the University of Newcastle Parameterized Complexity Research Unit under the auspices of the Deputy Vice-Chancellor for Research, and by a Fellowship to the Durham University Institute for Advanced Studies. The authors also gratefully acknowledge the support and kind hospitality provided by a William Best Fellowship at Grey College while the paper was in preparation.

graphs of treewidth at most k. In the terminology of parameterized complexity [DF99, FG06], Graph 3-Coloring is *fixed parameter tractable* for the parameter *treewidth*. In this small example, the asymptotic notation conceals serious costs associated to the treewidth bound k, from two sources:

(1) The complexity of computing a tree-decomposition of width k is $O(2^{35k^3}n)$ for an n-vertex graph.
(2) Once the tree-decomposition is obtained, one would then solve the problem by dynamic programming, in time $O(3^k n)$.

Suppose that we wish to solve Graph 3-Coloring for graphs having a different structural restriction — how should this be done? Here we consider the structural parameter of *bounded max leaf number*, where this is defined for a connected graph G as the maximum number of leaves of a spanning tree for G. (We choose this parameter mainly to illustrate the key issues, and because enough is known of the associated P-time extremal structure theory to provide a good example of the general approach. We are not aware of any strong direct applications of bounded max leaf number.)

One way to approach the problem of determining 3-colorability, parameterizing by max leaf number, is to note that graphs of bounded max leaf number exclude a tree minor and therefore have bounded pathwidth, so that the above sketched bounded treewidth approach can be used. This classifies Graph 3-Coloring, parameterized by max leaf number, as FPT, but this is not an efficient algorithm.

We have two objectives in this paper:

(1) We describe *efficient* FPT algorithms for Graph 3-Coloring and many other problems, for input parameterized by max leaf number.
(2) We do so in a way that is very generally systematic, and that "fits" the study of how parameterized structure affects computational complexity in what we term the "ecology" of parameterized complexity. One can view this effort as a kind of *generalized bidimensionality theory* in the sense of Demaine and Hajiaghayi [DFHT05, DH05, DH07].

In the next section, we discuss the basics of parameterized complexity and motivate the general setting for this investigation.

2 Background on Parameterized Complexity and the Complexity Ecology of Parameters

Contemporary sources of introductory material can be found in the survey articles [Ra97, DFS99, Fe02, Nie04], and in the recent books and monographs [DF99, FG06, Nie06].

Parameterized complexity is basically a two-dimensional generalization of the familiar P versus NP framework. In addition to the overall input size n we consider the effects on complexity of a declared secondary "measurement" k (the *parameter*) that generally is used to capture some structure of the input or other aspect of our computational objective (for example, $k = 1/\epsilon$ turns out to be a useful parameterization in the analysis of approximation complexity). Solvability in time $f(k)n^c$ is termed *fixed-parameter tractability* (FPT), where f is some function (usually exponential) and c is a constant independent of k.

Evidence that a parameterized problem is unlikely to admit an FPT algorithm is provided by a strong two-dimensional analog of NP-hardness, termed $W[1]$-hardness. A reference problem complete for $W[1]$ is the k-step halting problem for nondeterministic Turing machines of unlimited nondeterminism and alphabet size. This is obviously solvable by brute force in time $O(n^{O(k)})$. The positive toolkit of FPT turns out to be technically quite rich, and the negative toolkit of $W[1]$-hardness turns out to be widely applicable.

The main motivation for parameterized complexity is that in almost all real world settings and for almost all purposes of computing, the input has "extra structure" that we are able to relevantly capture with the mathematical device of the parameter.

Historically, a key motivating source for parameterized complexity has been the graph minors project of Robertson and Seymour [RS85]. The graph minors structure theory is related to FPT in the following way. The parameterized computational decision problem GRAPH MINOR takes as input graphs G and H and asks whether H is a minor of G (that is, whether a graph isomorphic to H can be obtained from G by contracting edges of a subgraph of G). This is a fundamental problem, naturally parameterized by H.

As far as we know, all of the beautiful structure theory of the graph minors project, *pathwidth*, *treewidth*, and the like, is necessary in order to show that the GRAPH MINOR problem, parameterized by H, is fixed-parameter tractable.

The following lemma codifies how every FPT parameterized problem has a canonically associated structure theory project, via the quest for efficient FPT kernelization bounds.

Lemma 1. *A parameterized problem Π is in FPT if and only if there is a transformation from Π to itself, and a function g, that reduces an instance (x, k) to (x', k') such that:*

(1) the transformation runs in time polynomial in $|(x, k)|$,
(2) (x, k) is a yes-instance of Π if and only if (x', k') is a yes-instance of Π,
(3) $k' \leq k$, and
(4) $|x'| \leq g(k)$.

In the situation described above, we say that we have a *kernelization bound* of $g(k)$. The proof of the above "point of view" on FPT that focuses on P-time kernelization is completely trivial, giving a kernelization bound of $g(k) = f(k)$ for an FPT problem solvable in time $f(k)n^c$. But for many important FPT problems, we can do *much* better, and the "pre-processing" routines that produce small kernels seem to have great practical value [ACFLSS04, Nie04, Nie06, Wei98]. For example, the VERTEX COVER problem can be kernelized in polynomial time to a graph on at most $2k$ vertices [NT75, ACFLSS04, CFJ04]. PLANAR DOMINATING SET also has a problem kernel of linear size [AFN04].

2.1 A Complexity Ecology of Parameters

The extent to which the structure theory of the graph minors project has turned out to be of practical relevance to computing is quite striking. For one example, many naturally occuring databases have bounded treewidth (or bounded hypertreewidth, a related notion) — a matter of immense significance to the realistic assessment of the complexity of database problems [GM99, FFG01, Gr01].

Another example is the problem of Type Checking of programs written in high-level logic-based programming languages such as ML. This problem has been shown to be complete for EXP, and thus "extremely" intractable from the classical point of view. Nevertheless, the ML compilers generally work just fine. The explanation is that most naturally occuring programs have a maximum type-declaration nesting depth k of no more than 5. The FPT type-checking algorithm that runs in time $O(2^k n)$ is entirely adequate in practice. The reason why naturally occuring programs have small nesting depth is that otherwise the programs quickly become incomprehensible to the programmer.

A possible perspective on this quoted from the survey [DFS99]:

> We feel that the parametric complexity notions, with their implicit ultrafinitism, correspond better to the natural landscape of computational complexity, where we find ourselves overwhelmingly among hard problems, dependent on identifying and exploiting thin zones of computational viability. Many natural problem distributions are generated by processes that inhabit such zones themselves (e.g., computer code that is written in a structured manner so that it can be comprehensible to the programmer), and these distributions then inherit limited parameter ranges because of the computational parameters that implicitly govern the feasibility of the generative processes, though the relevant parameters may not be immediately obvious.[1]

We want to know how all the various parameterized structural notions interact with all the other computational objectives one might have. The familiar paradigm of efficiently solving various problems for graphs of bounded treewidth just represents one row of a matrix of algorithmic questions that arise from the relevant parameterized structure theories. In the case of Max Leaf, we investigate how to solve the Independent Set problem, etc., on graphs bounded "max leaf number", exploiting the structure that bounding this parameter yields.

Consider the following table. We use here the shorthand: TW is Treewidth, BW is Bandwidth, VC is Vertex Cover, DS is Dominating Set, G is Genus and ML is Max Leaf. The entry in the 2nd row and 4th column indicates that there is an *FPT* algorithm to optimally solve the Dominating Set problem for a graph G of bandwidth at most k. The entry in the 4th row and second column indicates that it is unknown whether Bandwidth can be solved optimally by an *FPT* algorithm when the parameter is a bound on the domination number of the input.

[1] For a philosophically similar discussion see [Gur89].

Table 1. The Complexity Ecology of Parameters

	TW	BW	VC	DS	G	ML
TW	*FPT*	$W[1]$-hard	*FPT*	*FPT*	?	*FPT*
BW	*FPT*	$W[1]$-hard	*FPT*	*FPT*	?	*FPT*
VC	*FPT*	?	*FPT*	*FPT*	?	*FPT*
DS	?	?	$W[1]$-hard	$W[1]$-hard	?	?
G	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	*FPT*	?
ML	*FPT*	?	*FPT*	*FPT*	*FPT*	*FPT*

Our attention so far has mostly been concerned with:

(1) The diagonal — for example, Treewidth is *FPT* and Bandwidth is $W[1]$-hard — as stand-alone problems, and
(2) The first row.

But if the natural world of complexity "runs" on a commerce of (sometimes rather hidden) structural parameters, then it is important to systematically investigate the entire matrix. The so-called *bidimensionality theory* gives a systematic approach to the first (treewidth) row [DH07].

3 Systematically Attacking a Row

We use the max leaf parameter to show how to systematically attack a row of the "complexity ecology table" (which should not be thought of as limited to the few illustrative examples of problems in the table above — the real table is unbounded). We use the P-time extremal theory approach that is developed in [Pr05, EFLR05] where it is used to give a $3.75k$ P-time kernelization for the parameterized Max Leaf problem. The main point here is how to deploy such P-time kernelization structure theory to prove FPT results in a row of the complexity ecology table. The next two theorems extended and adapted from [EFLR05] illustrate the approach. (We depend heavily and unavoidably on this previous work.)

Theorem 1. *For graphs of max leaf number bounded by k, the minimum domination number can be computed in time $O^*(103^k)$ based on a polynomial-time reduction to a kernel of size at most $7k$.*

Proof. Sketch. Since this is an FPT result, we are necessarily (by Lemma 1) interested in effective kernelization for *this* problem. We must therefore develop a polynomial-time extremal account of the boundary case for the induction.

We take the following hypotheses:

(1) (G, k) is a yes-instance of Max Leaf.
(2) $(G, k + 1)$ is a no-instance of Max Leaf.
(3) There is a witness structure for (1) that satisfies the inductive priorities of the proof of Boundary Lemma II for Max Leaf (Lemma 8 of [EFLR05]).
(4) G is *reduced* according to an admissible set of polynomial time kernelization rules.

Here we must confine the interpretation of *reduced* to P-time reduction rules that are compatible with the new computational objective of computing a minimum dominating set. Many of the structural claims proved in [EFLR05] can now be imported to this new situation, modified in some cases because of changes to the admissible set of reduction rules. To illustrate the point, when proving a kernelization bound for MAX LEAF (as is done in [EFLR05]), one uses reduction rules that can be applied in polynomial time to produce from G a graph G' such that G has a k-leaf spanning tree if and only if G' has a k' leaf spanning tree, where $k' \leq k$. Here, because we are computing a minimum dominating set, we are allowed reduction rules where G has a k-dominating set if and only if G' has a k' dominating set. To the extent that we can find reduction rules for this new computational objective that "mimic" or approximate the ones that were available for the MAX LEAF problem, the structural claims about the kernel still (with some modifications) carry over, and we can conclude similar kernelization bounds for problems in the row of the complexity ecology table that are amenable to this approach. (It turns out that many well-known NP-hard problems are amenable in this way, and this is the main point of the paper.)

The reduction rules shown in Figure 1 below can be used in this way for the MINIMUM DOMINATING SET problem, for graphs of bounded max leaf number.

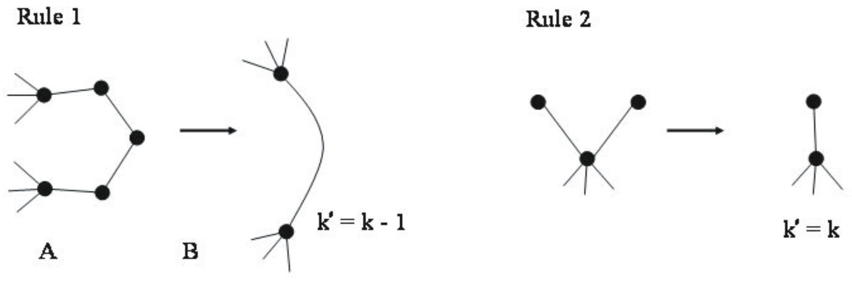


Fig. 1. Reduction rules for minimum domination

The argument for the bound on the kernel size is by minimum counterexample. One of our hypotheses is that (G, k) is a yes-instance for MAX LEAF. We can assume we are given as a witness structure a tree subgraph $T = (V', E')$ of G that has k leaves, and we can also assume that G is connected.

We do not assume that T is a spanning subgraph. (If T is not spanning, then it clearly extends to a spanning tree T' for G that has at least k leaves.)

A counterexample to our theorem would be a graph $G = (V, E)$ such that: (1) (G, k) is a reduced instance of MAX LEAF, (2) (G, k) is a yes-instance of MAX LEAF, (3) $(G, k + 1)$ is a no-instance, and (4) $|G| > 7k$.

Among all such counterexamples, we consider one where the witness subgraph tree T is as small as possible.

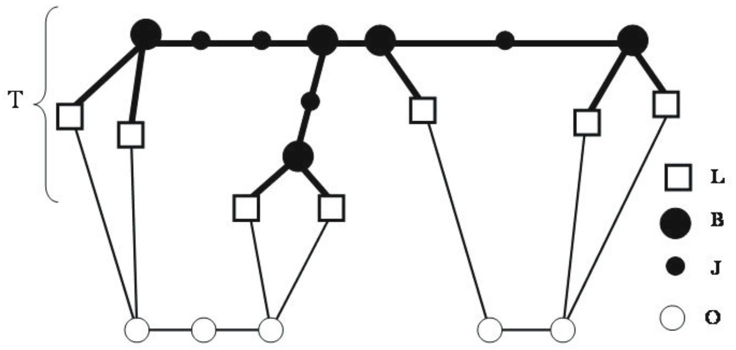


Fig. 2. The witness tree and various sets of vertices

Let $O = V - V'$ be the set of vertices not in the witness subtree T, which we will refer to as *outsiders*. Let L denote the leaves of T, I the internal (non-leaf) vertices of T, $B \subseteq I$ the *branch vertices* of T (the non-leaf, internal vertices of T that have degree at least 3 with respect to T), and let J denote the *subdivider vertices* of T (the non-branch internal vertices of T that have degree 2 with respect to T). See Figure 2 for an illustration of the general situation. One of the key roles of the reduction rules is to bound the number of outsiders (Claim 7 of Lemma 7 of [EFLR05]). In pursuing this sketch, which summarizes much material in [EFLR05], necessarily many details are omitted.

Almost all of the structural claims in the proof of Boundary Lemma II of [EFLR05] carry over (with a few requiring slight modification), yielding a kernel of size at most $7k$. The kernel can be analyzed by means of the algorithm due to Fomin, Kratsch and Woeginger [FKW04], yielding the running time stated for our algorithm. Knowing the domination number of the problem kernel allows us to compute the domination number of the input graph by retracing this information backwards along the kernelization path in polynomial time. □

What was the best previous result for this problem? Using the structure theory of Boundary Lemma II of [EFLR05] we can show that a path decomposition of width at most $g(k) = 20k/3$ can be computed in polynomial time for graphs whose max leaf number is bounded by k. Combining this with the carefully engineered dynamic programming algorithm for Dominating Set in this setting of Telle and Proskurowsky [TP93] (refined by Alber and Niedermeier [AN02]) one would get a "best previous" running time of around $O^*(4^{20k/3})$ or $O^*(10322^k)$.

The following theorem is also reported in [EFLR05], based on essentially the same approach, making use of the reduction rules shown in Figure 3. (Quick sketch: The imported structural claims give a bound of $4.5k$ on the size of a vertex cover for the kernel, which yields the claimed running time by using the algorithm of Chen, Kanj and Xia [CKX05] to analyze the situation.)

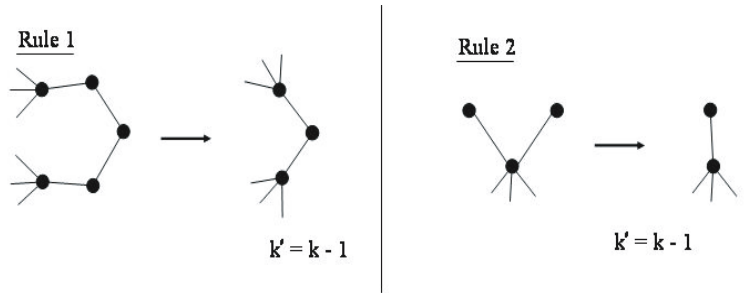


Fig. 3. Reduction rules for maximum independent set

Theorem 2. *For graphs of max leaf number bounded by k, the maximum size of an independent set can be computed in time $O^*(2.972^k)$ based on a polynomial-time reduction to a kernel of size at most $7k$.*

Many other NP-hard problems can be addressed for graphs of bounded max leaf number in much the same way.

Theorem 3. *For graphs of max leaf number bounded by k, it can be determined in $O^*(420.9^k)$ whether the graph is 3-colorable, based on a polynomial-time reduction to a kernel of size at most $5.5k$.*

Proof. The reduction rules: (1) delete vertices of degree 1, and (2) erase vertices of degree 2, are admissible for this problem. This yields an improved bound of $.5k$ over Claim 7 of Lemma 7 of [EFLR05]. The analysis of the kernel for the stated result just tries all possible 3-colorings. □

Graph Hamiltonicity admits the same reduction rules, and thus there is a $5.5k$ kernel for this problem as well. The same statement holds for the Feedback Vertex Set problem.

4 Summary

What we show in this paper is an example of how the structure theory associated (necessarily, by Lemma 1) to a good P-time kernelization result for an FPT problem (such as Max Leaf), can be exploited to give good FPT results for many of the entries in the corresponding "row" of the complexity ecology table. We have obviously picked off the easy examples, where all that is necessary is to identify reduction rules that are similar to the reduction rules used in the inductive proof of a kernelization bound for Max Leaf. Our results are not difficult, but the main point is the overall strategy, which clearly can be deepened. Some of the "columns" of our chosen row are still open. It is unclear how to use the Max Leaf kernel structure for the Bandwidth problem, for example. Since

Bandwidth is NP-hard for trees (with unboundedly many leaves) this may be interesting to resolve.

References

[ACFLSS04] Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: theory and experiments. In: Arge, L., Italiano, G., Sedgewick, R. (eds.) Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX), New Orleans, ACM/SIAM, Proc. Applied Mathematics 115 (January 2004)

[AFN04] Alber, J., Fellows, M., Niedermeier, R.: Polynomial time data reduction for dominating set. Journal of the ACM 51, 363–384 (2004)

[AN02] Alber, J., Niedermeier, R.: Improved Tree Decomposition Based Algorithms for Domination-Like Problems. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 613–627. Springer, Heidelberg (2002)

[ALS91] Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms 12, 308–340 (1991)

[BK07] Bodlaender, H.L., Koster, A.M.: Combinatorial optimisation on graphs of bounded treewidth. The Computer Journal (to appear 2007)

[Bod96] Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small width. SIAM J. Computing 25, 1305–1317 (1996)

[CFJ04] Chor, B., Fellows, M., Juedes, D.: Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 257–269. Springer, Heidelberg (2004)

[CKX05] Chen, J., Kanj, I., Xia, G.: Simplicity is beauty: improved upper bounds for vertex cover. Manuscript communicated by email, April (2005)

[Cou90] Courcelle, B.: The monadic second order logic of graphs I: Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)

[DF99] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)

[DFHT05] Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. Journal of the ACM 52, 866–893 (2005)

[DFS99] Downey, R., Fellows, M., Stege, U.: Parameterized complexity: a framework for systematically confronting computational intractability. In: Graham, R., Kratochvil, J., Nesetril, J., Roberts, F. (eds.) Contemporary Trends in Discrete Mathematics, Proceedings of the DIMACS-DIMATIA Workshop on the Future of Discrete Mathematics, Prague, 1997. AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 49, pp. 49–99 (1999)

[DH05] Demaine, E.D., Hajiaghayi, M.: Bidimensionality: New connections between FPT algorithms and PTASs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), Vancouver, pp. 590–601 (January 2005)

[DH07] Demaine, E.D., Hajiaghayi, M.: The bidimensionality theory and its algorithmic applications. The Computer Journal (to appear)

[EFLR05] Estivill-Castro, V., Fellows, M., Langston, M., Rosamond, F.: Fixed-parameter tractability is P-time extremal structure theory I: The case of max leaf. In: Proceedings of ACiD 2005: Algorithms and Complexity in Durham pp. 1–41 (2005)

[Fe02] Fellows, M.: Parameterized complexity: the main ideas and connections to practical computing. In: Fleischer, R., Moret, B.M.E., Schmidt, E.M. (eds.) Experimental Algorithmics. LNCS, vol. 2547, pp. 51–77. Springer, Heidelberg (2002)

[FFG01] Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 22–32. Springer, Heidelberg (2000)

[FG06] Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)

[FKW04] Fomin, F., Kratsch, D., Woeginger, G.: Exact (exponential) algorithms for the dominating set problem. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Heidelberg (2004)

[GM99] Grohe, M., Marino, J.: Definability and descriptive complexity on databases with bounded treewidth. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 70–82. Springer, Heidelberg (1998)

[Gr01] Grohe, M.: The parameterized complexity of database queries. In: Proc. PODS 2001, pp. 82–92. ACM Press, New York (2001)

[Gur89] Gurevich, Y.: The Challenger-Solver Game: Variations on the Theme of P=? NP, Bulletin EATCS 39 pp. 112–121 (1989)

[GGHNW05] Guo, J., Gramm, J., Hueffner, F., Niedermeier, R., Wernicke, S.: Improved fixed-parameter algorithms for two feedback set problems. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 158–169. Springer, Heidelberg (2005)

[Nie04] Niedermeier, R.: Ubiquitous parameterization — invitation to fixed-parameter algorithms. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 84–103. Springer, Heidelberg (2004)

[Nie06] Niedermeier, R.: Invitation to Fixed Parameter Algorithms (forthcoming). Oxford University Press, Oxford (2005)

[NT75] Nemhauser, G.L., Trotter, L.E.: Vertex packings: structural properties and algorithms. Mathematical Programming 8, 232–248 (1975)

[Pr05] Prieto-Rodriguez, E.: Systematic kernelization in FPT algorithm design. Ph.D. Thesis, School of EE&CS, University of Newcastle, Australia (2005)

[Ra97] Raman, V.: Parameterized Complexity. In: Proceedings of the 7th National Seminar on Theoretical Computer Science, Chennai, India, pp. 1–18 (1997)

[RS85] Robertson, N., Seymour, P.: Graph minors: a survey. In: Anderson, J. (ed.) Surveys in Combinatorics, pp. 153–171. Cambridge University Press, Cambridge (1985)

[TP93] Telle, J.A., Proskurowski, A.: Practical Algorithms on Partial k-Trees with an Application to Domination-Like Problems. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1993. LNCS, vol. 709, pp. 610–621. Springer, Heidelberg (1993)

[Wei98] Weihe, K.: Covering trains by stations, or the power of data reduction. In: Proc. ALEX'98 pp. 1–8 (1998)

Parameterized Complexity and Logic

Jörg Flum

Abteilung für Mathematische Logik, Universität Freiburg, Eckerstr. 1, 79104 Freiburg, Germany
joerg.flum@math.uni-freiburg.de

Abstract. We introduce and discuss some basic concepts of parameterized complexity theory via model-checking problems.

1 Introduction

In parameterized complexity theory the complexity of a computational problem is measured not only in terms of the input size (as it is usually done in classical complexity), but in addition in terms of a parameter. The idea is to choose the parameter in such a way that it can be assumed to be small for the instances one is interested in. A prominent example is the database query evaluation problem (or, more generally, the model-checking problem). An instance of the database query evaluation problem consists of a database and a query. Usually we have a large database and a small query. Therefore, a natural parameter for a parameterized complexity analysis is the size of the query.

Central to parameterized complexity theory is the notion of fixed-parameter tractability. It relaxes the classical notion of tractability by allowing algorithms whose running time can be exponential but only in terms of the parameter. In Section 3 we will see that fixed-parameter tractability yields a notion of tractability appropriate for the database query evaluation problem.

Parameterized complexity theory not only provides methods for proving problems to be fixed-parameter tractable but also gives a framework for dealing with apparently intractable problems. There is a great variety of classes of parameterized intractable problems. In Section 4 we introduce some of these classes by means of logic. These "logical definitions" help to grasp the scope of these classes. Moreover, they allow to obtain information about the parameterized complexity of some problems through the syntactical structure of the defining sentences.

Fixed-parameter tractability is only a general first approximation to feasibilty of parameterized problems. Recently a new and more restrictive class has been introduced. We present it in Section 5 and relate it to a classical approach via limited nondeterminism.

In summary, we introduce some of the main concepts of parameterized complexity via model-checking problems. For detailed expositions of parameterized complexity theory and in particular of the material presented here we refer the reader to [3,8,11].

2 Preliminaries

We recall some notions from logic and fix our notation. A *vocabulary* τ is a finite set of relation symbols. Each relation symbol has an *arity*. A *τ-structure* $\mathcal{A}$ consists of a set A called the *universe*, which we assume to be finite, and an interpretation $R^{\mathcal{A}} \subseteq A^r$ of each r-ary relation symbol $R \in \tau$. For example, we view a *graph* as a structure $\mathcal{G} = (G, E^{\mathcal{G}})$, where E is a binary relation symbol and $E^{\mathcal{G}}$ is an irreflexive and symmetric binary relation on the set G of vertices.

For a τ-structure $\mathcal{A}$ we denote by $\|\mathcal{A}\|$ its *size*, that is, the length of a string encoding $\mathcal{A}$ in a natural way. The number $\|\mathcal{A}\|$ will be within a polynomial factor of the term

$$|\tau| + |A| + \sum_{R \in \tau} |R^{\mathcal{A}}| \cdot \text{arity}(R).$$

Formulas of first-order logic FO of vocabulary τ are built up from atomic formulas $x = y$ and $Rx_1 \dots x_r$ where $x, y, x_1, \dots, x_r$ are variables and $R \in \tau$ is of arity r, using the boolean connectives and existential and universal quantification. For $t \geq 0$ let Π_t denote the class of all first-order formulas of the form

$$\forall x_{11} \dots \forall x_{1k_1} \exists x_{21} \dots \exists x_{2k_2} \ \dots \ Qx_{t1} \dots Qx_{tk_t} \ \psi,$$

where $Q = \exists$ if t is even and $Q = \forall$ otherwise, and where ψ is quantifier-free. The classes Σ_t are defined analogously starting with a block of existential quantifiers.

A variable x is *free* in φ if it has an occurrence in φ that is not in the scope of a quantifier binding x. A *sentence* is a formula without free variables. If φ is a sentence and $\mathcal{A}$ a structure one defines in a natural way whether $\mathcal{A}$ satisfies φ.

The difference between first-order logic and second-order logic is that the latter allows quantification not only over elements of the universe of a structure but also over subsets of the universe and even relations on the universe. For this purpose second-order logic contains, in addition to the symbols of FO, *relation variables*, usually denoted by $X, Y, \dots$. Unary relation variables are also called *set variables*. A second-order formula only containing set variables is *monadic*. The class of all monadic second-order formulas is denoted by MSO.

The set of natural numbers (that is, nonnegative integers) is denoted by $\mathbb{N}$. For a natural number n let $[n] := \{1, \dots, n\}$.

3 Model-Checking and Fixed-Parameter Tractability

The model checking problem $\mathrm{MC(C, L)}$ for a class C of finite structures and a logic L is the problem

MC(C, L)
Input: A structure $\mathcal{A}$ in C and a sentence φ of L.
Question: Does $\mathcal{A}$ satisfy φ?

If C is the class of all finite structures, then we denote $\mathrm{MC(C, L)}$ by MC(L).

The model-checking problem arises very naturally in various areas of computer science. Let us see two examples. In *database theory*, (relational) databases are viewed as structures and (Boolean) queries to the databases as sentences; thus, the query evaluation problem is the model-checking problem for the corresponding query language (the "logic"). The model-checking problem also plays a role in *computer-aided verification*. When verifying that a program or a finite state system has a desired property, the specification of the property in a suitable (modal or temporal) logic will usually be small compared to the large state space of the system. We mostly deal with logics relevant for this first area and do not consider logics like linear temporal logic LTL relevant for computer-aided verification. We only mention that MC(LTL) (= MC(C, LTL), where C is the class of all finite Kripke structures $\mathcal{K}$) is PSPACE-complete and can be solved by an algorithm in time $2^{|\varphi|} \cdot \|\mathcal{K}\|$. Hence, such an algorithm has a running time that is exponential only in the length of φ (the "small parameter") and even linear in the size of $\mathcal{K}$.

It is well-known (see [15]) that the model-checking problem MC(FO) for (the class of all finite structures and) first-order logic FO is PSPACE-complete. Note that in MC(C, L), and in particular in MC(FO), we consider the so called *combined complexity*: both the database and the query are input variables. The usual recursive definition of the set of assignments (defined for all free variables of φ and) satisfying the formula φ yields the following time bound:

Theorem 1 ([15]). $\mathrm{MC(FO)}$ *can be solved in time*

$$O(|\varphi| \cdot |A|^{n_\varphi} \cdot n_\varphi + \|\mathcal{A}\|),$$

where n_φ denotes the maximum number of free variables of a subformula of φ.

Since the query is typically much smaller than the database, the so called *data complexity* has widely been regarded as more meaningful. The data complexity is the complexity of evaluating a query on a database (or structure), when the query is considered as fixed. We write D-MC(C, L) $\in$ COMPL, where COMPL is a complexity class, if the database complexity of the model-checking problem for C and L is in COMPL, which means that for all sentences $\varphi \in$ L there is an algorithm according to COMPL solving the problem

Input:	A structure $\mathcal{A}$ in C.
Question:	Does $\mathcal{A}$ satiyfy φ?

From Theorem 1 we see:

Theorem 2. $\text{D-MC(FO)} \in$ PTIME.

In [13] Papadimitriou and Yannakakis discuss the relevance of the notions of combined complexity and data complexity for the query evaluation problem:

It seems to us that neither of the two notions of complexity is completely satisfactory. On the one hand, combined complexity is rather restrictive because it treats queries and databases as part of the input in the same way, even though the size of the queries is typically orders of magnitude smaller than the size of the database. However, on the other hand, polynomial time in the context of data complexity means time $|A|^{|\varphi|}$, [. . .] *a running time that hardly qualifies as tractable, especially in view of the fact that $|A|$ is typically huge.*

[...] *parametric complexity theory is a productive framework for studying the complexity of query languages, which puts the well-known tractability results of the query languages mentioned above under a different perspective, one that is perhaps more realistic and less confusing and misleading.*

So "what one would like to have is a running time in which $|A|$ is not raised to a power that depends on φ, i.e. the dependence of $|A|$ is of the form $O(|A|^c)$, where c is constant independent of the query φ (and hopefully very small)." However, we allow that the constant hidden in $O(|A|^c)$ depends on φ.

In terms of the following defintion such a running time for $\mathrm{MC}(\mathrm{C}, \mathrm{L})$ means that the model-checking problem is fixed-parameter tractable.

Definition 3
(a) A *parameterized problem* is a pair (Q, κ) consisting of a classical problem Q and a parameterization κ, that is a polynomial time computable function associating with every instance of Q a natural number.
(b) A parameterized problem (Q, κ) is *fixed-parameter tractable* if there is an algorithm deciding whether $x \in Q$ in time

$$f(\kappa(x)) \cdot p(|x|)$$

for some computable function f and polynomial $p \in \mathbb{N}[X]$.

We denote by FPT the class of parameterized problems that are fixed-parameter tractable. We call an algorithm with a running time bounded by $f(\kappa(x)) \cdot p(|x|)$ (for some computable f and $p \in \mathbb{N}[X]$) an *fpt-algorithm*.

We denote by p-$\mathrm{MC}(\mathrm{C}, \mathrm{L})$ the *parameterized model-checking problem for* C *and* L and represent it in the form

p-MC(C, L)	
Input:	A structure $\mathcal{A}$ in C and a sentence φ of L.
Parameter:	$|\varphi|$.
Question:	Does $\mathcal{A}$ satiyfy φ?

So the notation used for p-MC(C, L) shows that we consider the parameterization that assigns to the instance $(\mathcal{A}, \varphi)$ the number $|\varphi|$. Usually, we will represent parameterized problems in this way.

We see that parameterized complexity theory measures the complexity not only in terms of the input size (as it is usually done in classical complexity), but in addition in terms of a parameter. The main intention is to address complexity issues in situations where we know that the parameter is comparatively small. Indeed a running time such as $O(2^k \cdot n)$, where k denotes the parameter and n the size of the instance, can be quite good for small values of k, often better than the polynomial $O(n^2)$.

Example 4. p-$\mathrm{MC}(\mathrm{LTL})$ is fixed-parameter tractable.

Example 5. We consider the model-checking problem for the class of strings. For a finite alphabet Σ, let τ_Σ be the vocabulary that consists of a binary relation symbol $\leq$

and, for each $a \in \Sigma$, a unary relation symbol P_a. A string $\bar{a} = a_1 \dots a_n \in \Sigma^*$ may be represented by the τ_Σ-structure $\mathcal{S} = \mathcal{S}(a_1 \dots a_n)$, where the universe of $\mathcal{S}$ is the set $[n]$, the relation $\leq^{\mathcal{S}}$ is the natural order on $[n]$, and for all $a \in \Sigma$, we have

$$P_a^{\mathcal{S}} := \{i \in [n] \mid a_i = a\}.$$

We denote by MSO the set of formulas of monadic second-order logic. The problem

p-MC(STRING, MSO)	
Input:	An alphabet Σ, a string $\bar{a} \in \Sigma^*$, and an MSO-sentence φ of vocabulary τ_Σ .
Parameter:	$\lvert\varphi\rvert$.
Question:	Does $\mathcal{S}(\bar{a})$ satisfy φ?

is fixed-parameter tractable. In fact, by a well-known result due to Büchi [1], Elgot [5], and Trakhtenbrot [14] there is an algorithm $\mathbb{B}$ associating with every MSO-sentence φ a finite automaton $\mathbb{M}(\varphi)$ such that for all strings $\bar{a}$:

$$\mathbb{M}(\varphi) \text{ accepts } \bar{a} \iff \mathcal{S}(\bar{a}) \text{ satisfies } \varphi.$$

Now the following algorithm $\mathbb{A}$ solves p-MC(STRING, MSO): given $\bar{a} \in \Sigma^*$ and an MSO-sentence φ the algorithm $\mathbb{A}$ first computes the automaton $\mathbb{M}(\varphi)$ and then checks whether $\mathbb{M}(\varphi)$ accepts $\bar{a}$. If f is a computable function bounding the running time of $\mathbb{B}$, then the algorithm $\mathbb{A}$ requires at most $f(|\varphi|) + O(|\mathbb{M}(\varphi)| \cdot |\bar{a}|) \leq f(|\varphi|) + O(f(|\varphi|) \cdot |\bar{a}|) \leq O(f(|\varphi|) \cdot |\bar{a}|)$ steps, thus p-MC(STRING, MSO) $\in$ FPT.

4 Model-Checking and Fixed-Parameter Intractability

We have seen in the preceding section that a parameterized complexity analysis of the model-checking problem is appropriate for some applications we are interested in. Our main example was the database query evaluation problem where usually we have a large database and a small query. We presented two examples of fixed-parameter tractable model-checking problems.

However, Theorem 1 does not qualify p-MC(FO) as fixed-parameter tractable. To deal with parameterized intractable problems a core structural complexity theory has been developed. Unfortunately, it has led to a great variety of parameterized complexity classes. Of course, there are also many classical complexity classes, but NP plays a central role while in parameterized complexity theory there are various important classes of parameterized intractable problems. In this section we define some of the classes by means of logic. These "logical definitions" help to grasp the scope of these classes.

We start by considering two parameterized problems for graphs, the *parameterized clique problem* and the *parameterized dominating set problem* given by

p-CLIQUE	
Input:	A graph $\mathcal{G}$ and $k \in \mathbb{N}$.
Parameter:	k.
Question:	Does $\mathcal{G}$ have a clique of size k?

p-DOMINATING-SET
Input: A graph $\mathcal{G}$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Does $\mathcal{G}$ have a dominating set of size k?

They are reducible to p-MC(FO) as shown by the equivalences:

$$(\mathcal{G}, k) \in p\text{-CLIQUE} \iff \mathcal{G} \models \exists x_1 \ldots \exists x_k \bigwedge_{1 \le i < j \le k} Ex_ix_j\,; \tag{1}$$

$$(\mathcal{G}, k) \in p\text{-DOMINATING-SET}$$
$$\iff \mathcal{G} \models \exists x_1 \ldots \exists x_k \forall y \Big(\textstyle\bigwedge_{1 \le i < j \le k} x_i \ne x_j \wedge \bigvee_{i \in [k]} (y = x_i \vee Ex_iy) \Big). \tag{2}$$

However, we have to be careful with the notion of reduction we use. For example, in general polynomial time reductions do not preserve fixed-parameter tractability. The following notion of fpt-reduction is the one considered in parameterized complexity. It is not hard to show that if (Q, κ) is fpt-reducible to (Q', κ') and $(Q', \kappa') \in$ FPT, then $(Q, \kappa) \in$ FPT.

Definition 6. Let (Q, κ) and (Q', κ') be parameterized problems over the alphabets Σ and Σ', respectively. An *fpt-reduction* from (Q, κ) to (Q', κ') is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:

(a) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.
(b) R is computable by an fpt-algorithm. That is, there is a computable function f and a polynomial $p(X)$ such that $R(x)$ is computable in time $f(\kappa(x)) \cdot p(|x|)$.
(c) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \le g(\kappa(x))$ for all $x \in \Sigma^*$.

We write $(Q, \kappa) \le^{\text{fpt}} (Q', \kappa')$ if there is an fpt-reduction from (Q, κ) to (Q', κ').

Note that the equivalences (1) and (2) above show that p-CLIQUE $\le^{\text{fpt}}$ p-MC(FO) and that p-DOMINATING-SET $\le^{\text{fpt}}$ p-MC(FO). It is not hard to show that p-CLIQUE $\le^{\text{fpt}}$ p-DOMINATINGSET, while it is conjectured that neither p-DOMINATING-SET $\le^{\text{fpt}}$ p-CLIQUE nor p-MC(FO) $\le^{\text{fpt}}$ p-DOMINATING-SET.

The incomparability of fpt-reductions and polynomial time reductions is the source of the richness of parameterized complexity theory.

Consider again the equivalences (1) and (2). The interpretations of the variables $x_1, \ldots, x_k$ constitute the required clique and dominating set, respectively. Perhaps a more natural way to express this in first-order logic is to consider the first-order formulas *clique*(X) and *ds*(X) with a set variable X expressing that X is a clique and a dominating set, respectively:

$$\textit{clique}(X) := \forall y \forall z \Big((Xy \wedge Xz \wedge \neg y = z) \to Eyz \Big);$$
$$\textit{ds}(X) := \forall y \exists z \big(Xz \wedge (y = z \vee Eyz) \big).$$

More generally, let $\varphi(X)$ be any first-order formula with a (second-order) relation variable X of arity r. It *Fagin-defines* the parameterized problem

p-WD$_\varphi$	
Input:	A structure $\mathcal{A}$ and $k \in \mathbb{N}$.
Parameter:	k.
Question:	Is there a subset S of A^r such that $\mathcal{A} \models \varphi(S)$ and $\lvert S \rvert = k$?

Here $\mathcal{A} \models \varphi(S)$ means that $\mathcal{A}$ satisfies φ if X is interpreted by S. (On the class of graphs) p-WD$_{\textit{clique}}$ coincides with p-CLIQUE and p-WD$_{\textit{ds}}$ with p-DOMINATING-SET.

The definition of the classes W[1], W[2], ... of the W-hierarchy reflects the conviction that the computational complexity of the problem p-WD$_\varphi$ is related to its descriptional complexity, that is, to the complexity of the formula $\varphi(X)$:

Definition 7

(a) For $t \geq 1$ let W$[t]$ be the class of parameterized problems fpt-reducible to p-WD$_\varphi$ for some Π_t-formula $\varphi(X)$.

(b) AW$[*]$ is the class of parameterized problems fpt-reducible to p-MC(FO).

As *clique*(X) is a Π_1-formula and *ds*(X) is a Π_2-formula, we see that p-CLIQUE $\in$ W[1] and p-DOMINATING-SET $\in$ W[2]. Moreover, it can be shown that the problems p-CLIQUE and p-DOMINATING-SET are W[1]-complete and W[2]-complete, respectively, under fpt-reductions.

In the formula *ds*(X) Fagin-defining the W[2]-complete dominating set problem the set variable occurs exactly once. One can even show:

Theorem 8 ([4]). *For $t \geq 2$ the class* W$[t]$ *consists of all parameterized problems fpt-reducible to p-WD$_\varphi$, where $\varphi(X)$ is a Π_t-formula with exactly one occurrence of X.*

Remark 9. By Fagin's Theorem every problem (of structures of fixed vocabulary) is definable by a second-order formula of the form $\exists X \psi(X)$, where $\psi(X)$ is a first-order formula and X is a relation variable of some finite arity. For many concrete NP-problems the variable X can be viewed as a solution (or, witness) of the problem. Using so-called Skolem relations one can show that in Fagin's Theorem the formula ψ can be replaced by a Π_2-formula. However note that the use of Skolem relations in our context does not yield a collapse of the W-hierarchy to its second level, as this replacement would yield a non-parameter bounded increase of the parameter.

Many parameterized problems are Fagin-definable. In fact, as shown in Proposition 4.3 of [6], a parameterized problem (Q, κ) with instances of the form $(\mathcal{A}, k)$, where $\mathcal{A}$ is a structure and $k \in \mathbb{N}$ is the parameter (that is, $\kappa(\mathcal{A}, k) = k$), is Fagin-definable if and only if Q is in NP and for some $r \in \mathbb{N}$ we have $k \leq \lvert A \rvert^r$ for the positive instances $(\mathcal{A}, k)$ of Q.

Clearly, we have

$$\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \cdots \subseteq \cdots \subseteq \text{AW}[*].$$

To see that W$[t] \subseteq$ AW$[*]$ consider any Π_t-formula $\varphi(X)$ and for simplicity assume that X is a set variable. For any $k \in \mathbb{N}$ let φ_k be the Σ_{t+1}-sentence

$$\exists x_1 \ldots \exists x_k \Big(\bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \wedge \varphi' \Big),$$

where φ' is obtained from φ by replacing each subformula of the form Xz by $\bigvee_{i\in[k]} x_i = z$. Then for every structure $\mathcal{A}$, there is a subset $S \subseteq A$ with $|S| = k$ such that $\mathcal{A} \models \varphi(S)$ if and only if $\mathcal{A} \models \varphi_k$. Thus the mapping $(\mathcal{A}, k) \mapsto (\mathcal{A}, \varphi_k)$ is an fpt-reduction from p-WD$_\varphi$ to p-MC(FO).

Note that the length of the first (existential) quantifier block in the formulas φ_k depends on k, however, all other quantifier blocks in the φ_k have fixed length, namely the length of the corresponding block in the formula φ. For $t, u \geq 1$ we denote by $\Sigma_{t,u}$ the class of Σ_t-formulas, where all quantifier blocks besides the first one have length $\leq u$. Even though the formulas φ_k which arose from the Π_t-formula $\varphi(X)$ are in $\Sigma_{t+1,u}$ for some $u \geq 1$, one can show:

Theorem 10 ([4,7]). *For $t, u \geq 1$ the parameterized problem p-MC$(\Sigma_{t,u})$ is W$[t]$-complete under fpt-reductions.*

5 Bounded Fixed-Parameter Tractability

Let us emphasize again that one can expect the point of view of parameterized complexity to be appropriate and the techniques of the theory to be useful, only for parameterized problems where it can be assumed that the parameterization takes small values for the instances one is interested in.

Nevertheless, some additional remarks concerning the definition of fixed-parameter tractability are in order. We allow an *arbitrary* computable function f to bound the dependence of the running time of an fpt-algorithm on the parameter, that is, we are quite liberal. Similarly, in classical complexity we allow an *arbitrary* polynomial to bound the running time of a problem in PTIME. However, we expect that natural problems in PTIME have a linear, quadratic, or at most cubic running time. In parameterized complexity we expect to obtain functions f that have reasonable values for small arguments; or more precisely, once we know that a natural problem is in FPT we hope to get, using the combinatorics of the problem appropriately, a function f with this property.

In fact, among the known fixed-parameter tractable problems, problems that require a larger than exponential parameter dependence are rare exceptions. Furthermore, much of the theory is concerned with proving intractability (more precisely, hardness results), and, of course, such results are even stronger for our liberal definition.

However, there are exceptions, the most prominent one being the parameterized model-checking p-MC(STRING, MSO) for monadic second-order logic on the class of strings. We know from Example 5 that it is fixed-parameter tractable. The following result hardly qualifies it as tractable. Call a computable function $f : \mathbb{N} \to \mathbb{N}$ *elementary* if there is an $\ell \in \mathbb{N}$ such that for all $k \in \mathbb{N}$

$$f(k) \leq \text{tower}(\ell, k),$$

where $\text{tower}(0, k) := k$ and $\text{tower}(\ell + 1, k) := 2^{\text{tower}(\ell,k)}$.

Theorem 11 ([10]). *Let f be an elementary function.*

(a) There is no algorithm that for all (u, φ) decides whether $(u, \varphi) \in p$-MC(STRING, MSO) in time $f(|\varphi|) \cdot |u|$ (unless PTIME = NP).

(b) *There is no algorithm that for all instances* (u,φ) *decides whether* $(u,\varphi) \in p\text{-MC}(\textsc{String},\text{FO})$ *in time* $f(|\varphi|)\cdot|u|$ *(unless* $\text{FPT} = \text{AW}[*]$*).*

This raises some doubts about the notion of fixed-parameter tractability. The important fact is that there are reasonable alternatives: one can simply put upper bounds on the growth of the "parameter dependence" f, the most natural being $f \in 2^{O(k)}$. The resulting *bounded fixed-parameter tractability*, or $2^{O(k)}$*-fixed-parameter tractability* contains nearly all of the problems that are "fixed-parameter tractable in practice."

Definition 12. A parameterized problem is $2^{O(k)}$*-fixed-parameter tractable* if there is a computable function $f \in 2^{O(k)}$ and a polynomial $p \in \mathbb{N}[X]$ and an algorithm that, given $x \in \Sigma^*$, decides whether $x \in Q$ in at most $f(\kappa(x))\cdot p(|x|)$ steps.

We denote by $2^{O(k)}$-FPT the class of $2^{O(k)}$-fixed-parameter tractable problems (this class is denoted by EPT in [8,9]).

Example 13. The parameterized satisfiability problem p-SAT for propositional formulas is in $2^{O(k)}$-FPT, where

p-SAT	
Input:	A propositional formula α.
Parameter:	Number of variables of α.
Question:	Is α satisfiable?

As in the unbounded theory, to compare the complexities of parameterized problems that are not $2^{O(k)}$-fixed-parameter tractable, we need a notion of reduction. Again the most fundamental property expected from the notion of reduction is that $2^{O(k)}$-FPT is closed under the corresponding reductions, that is: if (Q,κ) is reducible to (Q',κ') and $(Q',\kappa') \in 2^{O(k)}$-FPT, then $(Q,\kappa) \in 2^{O(k)}$-FPT.

There is a natural notion with this property:

Definition 14. Let (Q,κ) and (Q',κ') be parameterized problems over the alphabets Σ and Σ', respectively. An $2^{O(k)}$*-reduction* from (Q,κ) to (Q',κ') is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:

(a) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.
(b) There is a computable function $f \in 2^{O(k)}$ such that $R(x)$ is computable in time $f(\kappa(x))\cdot|x|^{O(1)}$
(c) There is a $d \in \mathbb{N}$ such that $\kappa'(R(x)) \le d\cdot(\kappa(x) + \log|x|)$.

We write $(Q,\kappa) \le^{2^{O(k)}} (Q',\kappa')$ if there is a $2^{O(k)}$-reduction from (Q,κ) to (Q',κ') and $(Q,\kappa) \equiv^{2^{O(k)}} (Q',\kappa')$ if $(Q,\kappa) \le^{2^{O(k)}} (Q',\kappa')$ and $(Q',\kappa') \le^{2^{O(k)}} (Q,\kappa)$.

Many reductions in parameterized complexity are fpt-reductions and $2^{O(k)}$-reductions. However, there are notable exceptions. For example, we know that $p\text{-MC}(\text{FO}) \le^{\text{fpt}} p\text{-MC}(\textsc{String},\text{FO})$ does not hold (unless $\text{FPT} = \text{AW}[*]$). However:

Theorem 15 ([9]). $p\text{-MC}(\text{FO}) \equiv^{2^{O(k)}} p\text{-MC}(\textsc{String},\text{FO})$.

In the new framework we define the analogue of the W-hierarchy using Theorem 8 as reference:

Definition 16

(a) For $t \geq 2$ the class $2^{O(k)}$-W$[t]$ consists of all parameterized problems $2^{O(k)}$-reducible to p-WD$_\varphi$, where $\varphi(X)$ is a Π_t-formula with exactly one occurrence of X.

(b) $2^{O(k)}$-AW$[*]$ is the class of all parameterized problems $2^{O(k)}$-reducible to p-MC(FO).

Note that we skip the definition of $2^{O(k)}$-W$[1]$, which is more problematic. For example, p-WD$_\varphi \in 2^{O(k)}$-FPT for every Π_1-formula $\varphi(X)$ with one occurrence of X.

Again one can show that p-DOMINATING-SET is $2^{O(k)}$-W$[2]$ complete under $2^{O(k)}$-reductions. Furthermore, as shown by Theorem 15, the problem p-MC(STRING, FO) is $2^{O(k)}$-AW$[*]$-complete under $2^{O(k)}$-reductions. Concerning the complexity of the model-checking for the fragments $\Sigma_{t,u}$ of FO the full analogue of Theorem 10 is valid, namely:

Theorem 17 ([9]). *For $t \geq 2$ and $u \geq 1$ the parameterized problem p-MC$(\Sigma_{t,u})$ is $2^{O(k)}$-W$[t]$ complete under $2^{O(k)}$-reductions.*

Recall that a *tournament* is a directed graph $\mathcal{T} = (T, E^{\mathcal{T}})$ such that for all distinct $u, v \in T$ either $(u, v) \in E^{\mathcal{T}}$ or $(v, u) \in E^{\mathcal{T}}$, but not both. A *dominating set* of $\mathcal{T}$ is a set $S \subseteq T$ such that for all $w \in T \setminus S$ there exists a $v \in S$ with $(v, w) \in E^{\mathcal{T}}$. We consider the following parameterization:

p-TOURNAMENT-DOMINATING-SET
Input: A tournament $\mathcal{T}$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Does $\mathcal{T}$ have a dominating set of k elements?

Theorem 18 ([2,8]). *p-TOURNAMENT-DOMINATING-SET is $2^{O(k)}$-W$[2]$-complete under $2^{O(k)}$-reductions.*

It is well-known and easy to prove that every tournament $\mathcal{T} = (T, E^{\mathcal{T}})$ has a dominating set of size k for every k with $\log |T| \leq k \leq |T|$. Hence, p-TOURNAMENT-DOMINATING-SET has logarithmic parameters, where:

Definition 19. A parameterized problem (Q, κ) has *logarithmic parameters* if for some $c \in \mathbb{N}$ and all x, y with $\kappa(x) > c \cdot \log |x|$ and $\kappa(y) > c \cdot \log |y|$

$$x \in Q \iff y \in Q.$$

Polynomial time reductions and $2^{O(k)}$-reductions coincide on problems with logarithmic parameters as observed by J.A. Montoya (cf. [8]):

Proposition 20. *Let the parameterized problems (Q, κ) and (Q', κ') have logarithmic parameters. Then:*

$$Q \leq^{\text{ptime}} Q' \iff (Q, \kappa) \leq^{2^{O(k)}} (Q', \kappa').$$

This property allows to link the $2^{O(k)}$-theory with the theory on bounded nondeterminism introduced by Papadimitriou and Yannakakis in [12]. In fact they consider the classes $\textsc{Log}[2]$ and $\textsc{Log}[3]$, which they call LOGSNP and LOGNP, respectively, where:

Definition 21. For $t \geq 2$ the class $\textsc{Log}[t]$ consists of all (classical) problems reducible in polynomial time to $\textsc{Log}_\varphi$ where $\varphi(X)$ is a Π_t-formula with exactly one occurrence of the relation variable X, say, of arity r and where

$\textsc{Log}_\varphi$
Input: A structure $\mathcal{A}$ and $k \leq \log |A|$.
Question: Does there exist a subset S of A^r such that $|S| = k$ and $\mathcal{A} \models \varphi(S)$?

For example one can show (cf. [8]) that for arbitrary $t, t' \geq 2$

$$\textsc{Log}[t] = \textsc{Log}[t'] \iff 2^{O(k)}\text{-W}[t] = 2^{O(k)}\text{-W}[t'].$$

Both hierarchies, the $2^{O(k)}$-W-hierarchy and the $\textsc{Log}$-hierarchy, result from an attempt to analyze the complexity of Fagin-defined problems for instances $(\mathcal{A}, k)$ with a "small" k. In the $\textsc{Log}$-hierarchy we assume explicitly that k is small compared with the cardinality of the instance (even $k \leq \log |A|$) and then use the tools from classical complexity theory like polynomial time reductions. In the $2^{O(k)}$-theory the notion of tractability is defined in such a way that it relates to our intuitive concept of tractability only if the parameter is small.

References

1. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6, 66–92 (1960)
2. Downey, R.G., Fellows, M.R.: Parameterized computational feasibility. In: Clote, P., Remmel, J.B. (eds.) Proceedings of Feasible Mathematics II, Birkhäuser, pp. 219–244 (1995)
3. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
4. Downey, R.G., Fellows, M.R., Regan, K.: Descriptive complexity and the W-hierarchy. In: Beame, P., Buss, S (eds.) Proof Complexity and Feasible Arithmetic, volume 39 of AMS-DIMACS Volume Series, AMS, pp. 119–134 (1998)
5. Elgot, J.C.C.: Decision problems of finite automata design and related arithmetics. Transactions of the American Mathematical Society 98, 21–51 (1961)
6. Flum, J., Grohe, M.: Fixed-parameter tractability, definability, and model checking. SIAM Journal on Computing 31(1), 113–145 (2001)
7. Flum, J., Grohe, M.: Model-checking problems as a basis for parameterized intractability. Logical Methods in Computer Science, 1(1) (2004)
8. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
9. Flum, J., Grohe, M., Weyer, M.: Bounded fixed-parameter tractability and $\log^2 n$ nondeterministic bits. Journal of Computer and System Sciences 72, 34–71 (2006)
10. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Annals of Pure. and Applied Logic 130, 3–31 (2004)

11. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford Press (2006)
12. Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the VC-dimension. Journal of Computer and System Sciences 43, 425–440 (1991)
13. Papadimitriou, C.H., Yannakakis, M.: On the complexity of database queries. Journal of Computer and System Sciences 58, 407–427 (1999)
14. Trakhtenbrot, B.: Finite Automata and the Logic of Monadic Predicates. Doklady Akademii Nauk. SSSR 140, 326–329 (1961)
15. Vardi, M.Y.: The complexity of relational query languages. In: Proceedings of the 14th ACM Symposium on Theory of Computing, pp. 137–146 (1982)

Index Sets of Computable Structures with Decidable Theories

Ekaterina B. Fokina[*]

Sobolev Institute of Mathematics
Siberian Branch of the Russian Academy of Sciences
4 Acad. Koptyug avenue
630090 Novosibirsk Russia
e_fokina@math.nsc.ru

Abstract. The index set of computable structures with decidable theory for some fixed infinite language σ^* is m–complete $\Sigma_2^{0,\emptyset^{(\omega)}}$.

1 Introduction

One of the directions of the computable model theory is the study of relations between definability and computability. Investigations on algorithmic complexity of different classes of computable models were done in papers of S.Goncharov, J.Knight, N.Kogabaev, V.Harizanov, R.Miller and many other authors. In the framework of this approach, on the base of the earlier developed methods and of the theory of computable numberings we study the algorithmic complexity of the following natural class of structures: computable structures with decidable theory.

We introduce some basic definitions. We fix a computable Gödel numbering of a language L. Let all structures have universes contained in ω, which we think of as computable sets of constants. A structure $\mathbf{A}$ of the language L is **computable** if its domain is a computable subset of ω and all basic operations and predicates are uniformly computable. We identify formulas with their Gödel numbers. Then computability of a structure is equivalent to the condition that the atomic diagram $\mathcal{D}(\mathcal{A})$ of $\mathbf{A}$ is computable. A structure $\mathbf{B}$ is **decidable** if it's full diagram $\mathcal{FD}(\mathcal{B})$ is computable.

There exists a universal computable enumeration of all computable structures of the fixed predicate language. The **index set** of a structure $\mathcal{A}$ of this language is the set $I(\mathcal{A})$ of all indices of computable (isomorphic) copies of $\mathcal{A}$ in this enumeration. For a class K of structures, closed under isomorphism, the **index set** is the set $I(K)$ of all indices for computable members of K. There are a lot of papers on index sets: [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], etc.

Let Γ be a complexity class (e.g., Σ_2^0). $I(K)$ is m-**complete** Γ if $I(K)$ is Γ and for any $S \in \Gamma$, there is a computable function f such that

$$n \in S \text{ iff } f(n) \in I(K).$$

[*] This work was partially supported by RFBR grant 05-01-00819 and President grant of Scientific Schools of Russia 4413.2006.1.

This condition is equivalent to the condition that there is a uniformly computable sequence $(\mathcal{C}_n)_{n\in\omega}$ for which

$$n \in S \text{ iff } \mathcal{C}_n \in K.$$

In [11] we considered the index set of decidable structures. It was proved that it is m-complete Σ^0_3. In this work we consider the index set of the class of structures with decidable theory. We prove that it is m-complete $\Sigma^{0,\emptyset^{(\omega)}}_2$, that is, Σ^0_2 relative to $\emptyset^{(\omega)}$. To show this, we use the method of reducing the complexity of structures, developed by S.Goncharov and B.Khoussainov in [12]. In Sect.2 and 3 we give the information on Marker's extensions and 1-to-1-representation of Σ^0_2–sets that we need for reducing d-computable structures to computable structures, where d is an arithmetical degree. In Sect.4 we directly prove that the index set has the stated complexity.

2 Marker's Construction

Let L be a finite language with no function symbols. Let $\mathcal{A} = (A, P_0^{n_0}, \ldots, P_m^{n_m})$ be a structure of L. We assume that for every P of this structure the sets P and $A^k \setminus P$ are infinite where k is the arity of P. For each k-ary predicate P of this structure we define $\exists$- and $\forall$-extensions of P.

Marker's $\exists$-extension of P is a $(k+1)$-ary predicate denoted by $P_\exists$ with the following properties. Let X be an infinite set disjoint with A. Then $P_\exists$ satisfies the following conditions:

1. If $P_\exists(a_1, a_2, \ldots, a_k, a_{k+1})$ then $P(a_1, \ldots, a_k)$ and $a_{k+1} \in X$.
2. For every $a_{k+1} \in X$ there exists a unique tuple $(a_1, \ldots, a_k)$ such that $P_\exists(a_1, \ldots, a_k, a_{k+1})$.
3. If $P(a_1, \ldots, a_k)$ then there exists a unique a such that $P_\exists(a_1, a_2, \ldots, a_k, a)$.

Marker's $\forall$-extension of the predicate P is a $(k+1)$-ary predicate $P_\forall$ with the following properties. Let X be an infinite set disjoint with A. Then $P_\forall$ satisfies the following conditions:

1. If $P_\forall(a_1, a_2, \ldots, a_k, a_{k+1})$ then $a_1, \ldots, a_k \in A$ and $a_{k+1} \in X$.
2. For all $(a_1, \ldots, a_k) \in A$ there exists at most one $a_{k+1} \in X$ such that $\neg P_\forall(a_1, a_2, \ldots, a_k, a_{k+1})$.
3. If $P_\forall(a_1, a_2, \ldots, a_k, a_{k+1})$ for all $a_{k+1} \in X$ then $P(a_1, \ldots, a_k)$.
4. For every $a_{k+1} \in X$ there exists a unique tuple $(a_1, \ldots, a_k)$ such that $\neg P_\forall(a_1, \ldots, a_k, a_{k+1})$.

The set X in $\exists$- or $\forall$-extension is called a **fellow of** P.

Definition 1. *Let $\mathcal{A} = (A, P_0^{n_0}, \ldots, P_m^{n_m})$ be a model.*

1. *The model $\mathcal{A}_\exists$ is a model $(A \cup X_0 \ldots \cup X_m, P_0^{n_0+1}, \ldots, P_m^{n_m+1}, X_0, \ldots, X_m)$, where each $P_i^{n_i+1}$, $i = 0, \ldots, m$, is a Marker's $\exists$-extension of $P_i^{n_i}$ such that fellows X_i of distinct predicates are pairwise disjoint sets.*

2. *The model $\mathcal{A}_\forall$ is a model $(A \cup X_0 \ldots \cup X_m, P_0^{n_0+1}, \ldots, P_m^{n_m+1}, X_0, \ldots, X_m)$, where each $P_i^{n_i+1}$, $i = 0, \ldots, m$, is a Marker's $\forall$-extension of $P_i^{n_i}$ such that fellows X_i of distinct predicates are pairwise disjoint sets.*

Theorem 1. *Let $\mathcal{A}_\exists$ and $\mathcal{A}_\forall$ be the Marker's extensions of the model $\mathcal{A}$. Then they satisfy the following properties:*

1. *The model $\mathcal{A}$ is definable in each of the extensions.*
2. *If the theory of the model $\mathcal{A}$ is $\aleph_0$-categorical then so is the theory of each of the extensions.*
3. *If the theory of the model $\mathcal{A}$ is $\aleph_1$-categorical then so is the theory of each of the extensions.*
4. *If the theory of $\mathcal{A}$ is almost strongly minimal then so is the theory of each of the extensions.*
5. *Any automorphism of $\mathcal{A}$ can be extended to automorphisms of each of the extensions.*

Let $\mathcal{A}$ be a structure and w be a word over the alphabet $\{\exists, \forall\}$. We define $\mathcal{A}_w$ by induction. If w is an empty string then $\mathcal{A}_w = \mathcal{A}$. If $w = w'\exists$ or $w = w'\forall$ and $\mathcal{B} = \mathcal{A}_{w'}$ then $\mathcal{A}_{w'\exists} = \mathcal{B}_\exists$ and $\mathcal{A}_{w'\forall} = \mathcal{B}_\forall$. Therefore we have the following corollary:

Corollary 1. *Let $\mathcal{A}$ be a structure and w be a word over the alphabet $\{\exists, \forall\}$. Then*

1. *The model $\mathcal{A}$ is definable in $\mathcal{A}_w$.*
2. *If the theory of the model $\mathcal{A}$ is $\aleph_0$-categorical ($\aleph_1$-categorical) then so is the theory of $\mathcal{A}_w$.*
3. *Any automorphism of $\mathcal{A}$ can be extended to an automorphism of $\mathcal{A}_w$.*

3 On One-to-One Representation of Σ_2^0-Sets

The following definition and lemmas can be found in [12]. We will need them for the proof of the main results of the paper.

Definition 2. *A Σ_2^0-set A is* **one-to-one representable** *if for some computable predicate $Q \subset \omega^3$ the following is true:*

1. *For every $n \in \omega$, $\exists a \forall b Q(n, a, b)$ if and only if $n \in A$.*
2. *For every $n \in \omega$, $\exists a \forall b Q(n, a, b)$ if and only if $\exists^{=1} a \forall b Q(n, a, b)$*[1].
3. *For every b there exists a unique pair $\langle n, a \rangle$ such that $\neg Q(n, a, b)$.*
4. *For every pair $\langle n, a \rangle$ either $\exists^{=1} b \neg Q(n, a, b)$ or $\forall b Q(n, a, b)$.*
5. *For every a there exists a unique n such that $\forall b Q(n, a, b)$.*

Lemma 1. *Let A be a coinfinite Σ_2^0-set that possesses an infinite computable subset S such that $A \setminus S$ is infinite. Then A has a one-to-one-representation.*

[1] $\exists^{=1} x P(x)$ means that there exists a unique x satisfying P.

The definition of a one-to-one-representation of a Σ_2^0-set can be relativized with respect to any oracle X. The relativized version of the lemma will be used in the proofs in the next sections.

Lemma 2. *Let A be a coinfinite $\Sigma_2^{0,X}$-set that possesses an infinite X-computable subset S such that $A \setminus S$ is infinite. Then there exists a X-computable set Q such that Q is a one-to-one-representation of A.*

The following two theorems are corollaries of Lemma 2 and Corollary 1.

Theorem 2. *For every Turing degree d a theory of a model $\mathcal{M}$ is d-decidable if and only if theories of $\mathcal{M}_\forall$ and $\mathcal{M}_\exists$ are d-decidable.*

Proof. According to Theorem 1 the model $\mathcal{M}$ is definable in each of the extensions $\mathcal{M}_\forall$ and $\mathcal{M}_\exists$. Therefore, if $Th(\mathcal{M}_\forall)$ or $Th(\mathcal{M}_\exists)$ is d-decidable then so is $Th(\mathcal{M})$. On the other hand, the properties of $\mathcal{M}_\forall$ or $\mathcal{M}_\exists$ are completely determined by $\mathcal{M}$. Thus, if $Th(\mathcal{M})$ is d-decidable then $Th(\mathcal{M}_\forall)$ and $Th(\mathcal{M}_\exists)$ are d-decidable.

Theorem 3. *Let d be any arithmetical Turing degree. Let $\mathcal{M}_0, \ldots \mathcal{M}_n, \ldots$ be a computable sequence of d'-computable structures, such that in every $\mathcal{M}_i$ for every $P \in \sigma$ there exists an infinite subset $S_{i,P}$, uniformly computable from i and P, such that $P \setminus S_{i,P}$ is infinite. Then there is a computable sequence $(\mathcal{M}_0)_{\forall\exists}, \ldots, (\mathcal{M}_n)_{\forall\exists}, \ldots$ of d-computable models, such that for all i and $P' \in \sigma_{\forall\exists}$ there exists an infinite subset $S_{i,P'} \subseteq P$, uniformly computable from i and P', such that $P' \setminus S_{i,P'}$ is infinite.*

Proof. From [12], the construction of one-to-one-representations may be arranged uniformly for all n. Using the uniform version of the lemma 1 one can construct the sequence $(\mathcal{M}_0)_{\forall\exists}, \ldots, (\mathcal{M}_n)_{\forall\exists}, \ldots$ and show that every $(\mathcal{M}_n)_{\forall\exists}$ is d-computable and every extension of predicates has the desired properties.

For more details on proofs of Sect.2 and 3, see [12], [14].

4 Complexity of Index Set

Let $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n, \ldots$ be a universal computable enumeration of all computable structures of a fixed language σ. We consider the index set of computable structures with decidable theories:

$$DT = \{n \mid \mathrm{Th}(\mathcal{M}_n) \text{ is decidable}\}.$$

Theorem 4. *The index set DT of all computable structures with decidable theories is an m-complete $\Sigma_2^{0,\emptyset^{(\omega)}}$ set in the universal computable enumeration of all computable structures of some infinite language σ^*.*

Lemma 3. $DT \in \Sigma_2^{0,\emptyset^{(\omega)}}$.

Proof. Let $\{\varphi_k\}_{k\in\omega}$ be an enumeration of all formulas with constants $\{c_0, ...\}$. We can compute a computable family of total functions with oracles $f_{n,s}^{\emptyset^{(s)}}(k)$ such that for any number k we have $M_n \models \varphi_k$ iff $f_{n,s}^{\emptyset^{(s)}}(k) = 1$ and φ_k is in the prenex normal form and has s alternations of quantifiers.

We have now that the index set DT of computable structures with decidable theory has the definition: $n \in DT$ iff there exists such x, that:

1. x is a number of a total computable function f_x,
2. for all k, $f_x(k) \in \{0, 1\}$, and
3. for all k, if φ_k is in the prenex normal form, has s alternations of quantifiers and doesn't have constants from $\{c_0, ...\}$, then $\Big(f_x(y) = 1 \leftrightarrow f_{n,s}^{\emptyset^{(s)}}(y) = 1\Big)$.

From this description we have that $DT \in \Sigma_2^{0,\emptyset^{(\omega)}}$.

Proof (of Theorem 4). By Lemma 3, the set $DT \in \Sigma_2^{0,\emptyset^{(\omega)}}$. To prove that DT is m-complete, it is sufficient to construct a computable sequence of computable structures in the infinite language σ^*, such that

$n \in A \Leftrightarrow \mathcal{A}_n$ is computable and its theory is decidable;
$n \notin A \Leftrightarrow \mathcal{A}_n$ is computable but its theory is not decidable,

where A is any $\Sigma_2^{0,\emptyset^{(\omega)}}$ set.

As A is a $\Sigma_2^{0,\emptyset^{(\omega)}}$ set, then there exists a computable with $\emptyset^{(\omega)}$ relation R such that $n \in \overline{A}$ iff $(\exists^\infty z)R(n, z)$. Given such a $\emptyset^{(\omega)}$–computable relation R we define $R^{\emptyset^{(\omega)}\restriction t}(n, z)$ as follows. $R^{\emptyset^{(\omega)}\restriction t}(n, z)$ holds iff we can compute that $R(n, z)$ holds with only information about $\emptyset^{(\omega)}$ restricted up to the set $\emptyset^{(\omega)} \restriction t \rightleftharpoons \{(n, m) \mid m \in \emptyset^{(n)} \& n \leq t\}$.

Let X be a set which is Δ_2^0 and not c.e. We code it into a new relation $Q(n, z')$ that holds iff all the following conditions are true:

1. $z' = \langle z, t, q\rangle$ and
2. $R^{\emptyset^{(\omega)}\restriction t+1}(n, z)$ and
3. $\neg R^{\emptyset^{(\omega)}\restriction t}(n, z)$ and
4. D_q is the set $X \cap \{0, 1, ..., z\}$.

It is easy to see that $(\exists^\infty z)R(n, z)$ iff $(\exists^\infty z)Q(n, z)$. Note, that $Q(n, z)$ iff $S_z^{\emptyset^{(z)}}(n)$ for a computable sequence of relations with oracles.

Let $\sigma_0 = \langle P\rangle$ be a language with only one binary relation symbol. We construct a computable sequence $\{\mathcal{A}_n\}_{n\in\omega}$ of computable structures for a language $\sigma^* = \cup_i(\sigma_0)_{(\forall\exists)^i} \cup \theta$, where θ defines an equivalence relation on the domain of $\mathcal{A}_n$.

We build $\mathcal{A}_n$ by stages as a union of an increasing sequence of computable structures: $\mathcal{A}_n^0 \subseteq \mathcal{A}_n^1 \subseteq ... \subseteq \mathcal{A}_n^i \subseteq \mathcal{A}_n^{i+1} \subseteq ... \subseteq \bigcup_i \mathcal{A}_n^i = \mathcal{A}_n$. For every i, $\mathcal{A}_n^i$ will be a structure of a language $\sigma_i = \cup_{j\leq i}(\sigma_0)_{(\forall\exists)^j} \cup \theta$. To get $\mathcal{A}_n^{i+1}$, we add to $\mathcal{A}_n^i$ a new equivalence class $\mathcal{M}_n^i$ which we define in the following way.

We consider two structures $\mathcal{N}_1$ and $\mathcal{N}_2$ for the language σ_0. $\mathcal{N}_1$ consists of infinitely many cycles of the length 3 and infinitely many cycles of the length 5. Similarly, $\mathcal{N}_2$ consists of infinitely many cycles of the length 4 and infinitely many cycles of the length 5. Here all the cycles are pairwise disjoint.

Let $\mathcal{A}_n^i = \langle A_n^i, \sigma_i \rangle$ be defined. To define $\mathcal{A}_n^{i+1}$ we consider $S_i^{\emptyset^{(i)}}(n)$. If $S_i^{\emptyset^{(i)}}(n)$ holds, then we define $\mathcal{N}_n^i \leftrightharpoons \mathcal{N}_1$, otherwise we let $\mathcal{N}_n^i \leftrightharpoons \mathcal{N}_2$. In any case $\mathcal{N}_n^i$ is a $\emptyset^{(i)}$–computable structure which satisfies the condition of Lemma 2. We let $\mathcal{M}_n^i = (\mathcal{N}_n^i)_{(\forall\exists)^{(i+1)}}$. Thus, $\mathcal{M}_n^i$ is a computable structure for the language $(\sigma_0)_{(\forall\exists)^{i+1}}$. We now let $\mathcal{A}_n^{i+1} = \langle A_n^i \cup M_n^i, \sigma_{i+1} \rangle$, where A_n^i and M_n^i are the universes of $\mathcal{A}_n^i$ and $\mathcal{M}_n^i$ respectively. The formula θ defines an equivalence relation, such that its equivalence classes are the universes of $\mathcal{M}_n^j$, for $j \leq i+1$. All other predicates from σ_{i+1} are true in $\mathcal{A}_n^{i+1}$ if they were true in $\mathcal{A}_n^i$ or $\mathcal{M}_n^i$. For all the tuples, for which any predicate is undefined, define it to be false on this tuple.

We let $\mathcal{A}_n = \bigcup_i \mathcal{A}_n^i$.

Lemma 4. *$n \in A$ iff the theory of $\mathcal{A}_n$ is decidable.*

Proof. The theories of both $\mathcal{N}_1$ and $\mathcal{N}_2$ are decidable. By Theorem 2, the theories of all $\mathcal{M}_n^i$, for all n, i, are also decidable. If $n \in A$, then we add only finitely many of equivalence classes constructed from $\mathcal{N}_2$ by applying Marker's operators $\forall\exists$ sufficiently many times. Therefore, in this case the theory of $\mathcal{A}_n$ is decidable. Otherwise, we add infinitely many of such classes. In this case we can enumerate the set X using $Th(\mathcal{A}_n)$ as an oracle. Namely,

$$x \in X \iff (\exists y, q)(x \in D_q = X \cap \{1, \ldots, y\}) \iff$$

$$\iff (\exists y, q, t)\left(x \in D_q = X \cap \{1, \ldots, y\} \& R^{\emptyset^{(\omega)} \upharpoonright t+1}(n, y) \& \neg R^{\emptyset^{(\omega)} \upharpoonright t}(n, y)\right) \iff$$

$$\iff (\exists z)\left(z_1 \geq x \& Q(z, n)\right), \text{ where } z = \langle z_1, z_2, z_3 \rangle \iff (\exists z)\left(z_1 \geq x \& S_z^{\emptyset^{(z)}}(n)\right)$$

The last expression means that at stage z we add an equivalence class of the type $\mathcal{M}_n^z = (\mathcal{N}_2)_{(\forall\exists)^z}$. By the Corollary 1, we can write a sentence expressing that in $\mathcal{M}_n^z$ there exists a cycle of length 4. Therefore, X is enumerable in $Th(\mathcal{A}_n)$. By our choice, $X \in \Delta_2^0 \setminus \Sigma_1^0$, thus, $Th(\mathcal{A}_n)$ can not be decidable.

This proves the theorem.

References

1. Calvert, W.: The isomorphism problem for classes of computable fields. Archive for Mathematical Logic 75, 327–336 (2004)
2. Calvert, W.: The isomorphism problem for computable Abelian p-groups of bounded length. Journal of Symbolic Logic 70, 331–345 (2005)
3. Calvert, W., Cummins, D., Knight, J.F., Miller, S.: Comparing classes of finite structures. Algebra and Logic 43, 374–392 (2004)

4. Calvert, W., Harizanov, V., Knight, J.F., Miller, S.: Index sets of computable structures. Algebra and Logic 45, 306–325 (2006)
5. Csima, B.F., Montalbán, A., Shore, R.A.: Boolean algebras, Tarski invariants, and index sets, to appear in the Notre Dame Journal of Formal Logic
6. Dobritsa, V.P.: Complexity of the index set of a constructive model. Algebra and Logic 22, 269–276 (1983)
7. Goncharov, S.S., Knight, J.F.: Computable structure and non-structure theorems. Algebra and Logic (English translation) 41, 351–373 (2002)
8. Lempp, S., Slaman, T.: The complexity of the index sets of $\aleph_0$-categorical theories and of Ehrenfeucht theories, to appear in the Advances in Logic In: Proceedings of the North Texas Logic Conference, Contemporary Mathematics, American Mathematical Society. October 8–10, (2004)
9. White, W.: On the complexity of categoricity in computable structures. Mathematical Logic Quarterly 49, 603–614 (2003)
10. White, W.: Characterizations for Computable Structures, PhD dissertation, Cornell University (2000)
11. Fokina, E.: Index sets of decidable models, Sibirsk. Mat. Zh (To appear)
12. Goncharov, S., Khoussainov, B.: Complexity of theories of computable categorical models. Algebra Logic 43(6), 365–373 (2004)
13. Marker, D.: Non-Σ_n-axiomatizable almost strongly minimal theories. J. Symbolic Logic 54, 921–927 (1989)
14. Fokina, E.: On complexity of categorical theories with computable models. Vestnik NGU 5(2), 78–86 (2005)
15. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)
16. Ash, C.J., Knight, J.F.: Computable Structures and the Hyperarithmetical Hierarchy. In: Studies in Logic and the Foundations of Mathematics, vol. 144, North-Holland Publishing Co, Amsterdam (2000)
17. Ash, C.J., Knight, J.F.: Pairs of recursive structures. Ann. Pure Appl. Logic 46(3), 211–234 (1990)
18. Goncharov, S., Harizanov, V., Knight, J., McCoy, C., Miller, R., Solomon, R.: Enumerations in computable structure theory. Ann. Pure Appl. Logic 136(3), 219–246 (2005)

Minimal Representations for Majority Games[⋆]

Josep Freixas[1,3], Xavier Molinero[1,4], and Salvador Roura[2,4]

[1] Escola Politècnica Superior d'Enginyeria de Manresa. E-08242 Manresa, Spain
[2] Universitat Politècnica de Catalunya. CN-Ω, E-08034 Barcelona, Catalonia, Spain
[3] Dept. de Matemàtica Aplicada 3
josep.freixas@upc.edu
[4] Dept. de Llenguatges i Sistemes Informàtics
{molinero,roura}@lsi.upc.edu

Abstract. This paper presents some new results about majority games. Isbell (1959) was the first to find a majority game without a minimum normalized integer representation; he needed 12 voters to construct such a game. Since then, it has been an open problem to find the minimum number of voters of a majority game without a minimum normalized integer representation. Our main new results are:

1. All majority games with less than 9 voters have a minimum integer representation.
2. For 9 voters, there are 14 majority games without a minimum integer representation, but all these games admit a minimum normalized integer representation.
3. For 10 voters, there exist majority games with neither a minimum integer representation nor a minimum normalized integer representation.

Keywords: Simple games, Weighted games, Majority games, Minimum and minimal weighted representations/realizations, Computing games.

1 Introduction

We start by giving some basic definitions on simple games (we refer the interested reader to [16] for a thoroughly presentation). Simple games can be viewed as models of voting systems in which a single alternative, such as a bill or an amendment, is pitted against the status quo.

Definition 1. *A simple game G is a pair (N, W) in which $N = \{1, \ldots, n\}$ for some positive integer n, and W is a collection of subsets of N that satisfies $N \in W$, $\emptyset \notin W$, and the* monotonicity *property: $S \in W$ and $S \subseteq R \subseteq N \Rightarrow R \in W$.*

⋆ This research was partially supported by Grant MTM 2006–06064 of "Ministerio de Ciencia y Tecnología y el Fondo Europeo de Desarrollo Regional" and SGRC 2005-00651 of "Generalitat de Catalunya", and by the Spanish "Ministerio de Ciencia y Tecnología" programme TIN2005-05446 (ALINEX).

Any set of voters is called a *coalition*, the set N is called the *grand coalition*, and the empty set $\emptyset$ is called the *null coalition*. Members of N are called *players* or *voters*, and the subsets of N that are in W are called *winning coalitions*. The intuition here is that a set S is a winning coalition *iff* the bill or amendment passes when the players in S are precisely the ones who voted for it. A subset of N that is not in W is called a *losing coalition*, denoted by L. A *minimal winning coalition* (*maximal losing coalition*), denoted by W^m (L^M), is a winning (losing) coalition all of whose proper subsets (supersets) are losing (winning). Because of monotonicity, any simple game is completely determined by its set of minimal winning coalitions. A voter i is null if $i \notin S$ for all $S \in W^m$.

Before proceeding, we introduce a real–world example (see [15] for an illustration of many real–world examples and [4] for the European Union Council).

Example 1. We shall consider here the composition of the Catalan Parliament today. Six parties got elected members in the last elections, giving rise to the following distribution of the 135 seats:

Party	Name of the Party	Seats
1	Convergència i Unió	48
2	PSC-Ciutadans pel Canvi	37
3	Esquerra Republicana de Catalunya	21
4	Partit Popular	14
5	Iniciativa per Catalunya Verds - EUIA	12
6	Ciutadans-Partido de la Ciudadanía	3

Voters are here parties since they vote by using party whip. Most of the proposals require absolute majority to be passed, i.e., a minimum of 68 votes are needed. Then $W^m = \{\{1,2\},\{1,3\},\{1,4,5\},\{2,3,4\},\{2,3,5\}\}$. Some others proposals need a 2/3-qualified majority to be passed, that is, at least 90 votes. In this case we have $W^m = \{\{1,2,3\},\{1,2,4\},\{1,2,5\},\{1,3,4,5\}\}$. Note that party 6 is null in both games.

Now, let us consider some special types of simple games.

Definition 2. *A simple game* (N,W) *is* strong *if* $S \notin W$ *implies* $N \setminus S \in W$.

A simple game that is not strong is called *weak*. Intuitively speaking, if a game is weak it has too few winning coalitions, because adding sufficiently many winning coalitions would make the game strong. Note that the addition of winning coalitions can never destroy strongness.

Definition 3. *A simple game* (N,W) *is* proper *if* $S \in W$ *implies* $N \setminus S \notin W$.

A simple game that is not proper is called *improper*. An improper game has too many winning coalitions, in the sense that deleting sufficiently many winning coalitions would make the game proper. Note that the deletion of winning coalitions can never destroy properness.

When a game is both proper and strong, a coalition wins *iff* its complement loses. Therefore, in this case we have $|W| = |L| = 2^{n-1}$.

Definition 4. *A simple game is* decisive *(or self–dual, or constant sum) if it is proper and strong.*

Definition 5. *Given a simple game* (N, W)*, its dual game is* (N, W^*)*, where* $S \in W^*$ *if and only if* $N \setminus S \notin W$*.*

That is, winning coalitions in the dual game are just the "blocking" coalitions in the original game. Note that (N, W) is proper *iff* (N, W^*) is strong, and (N, W) is strong *iff* (N, W^*) is proper. As a consequence, we have that a simple game (N, W) is decisive *iff* $W = W^*$.

In the seminal work on game theory by von Neumann and Morgenstern [14] only decisive simple games were considered. Nowadays, many governmental institutions made their decisions through voting rules that are in fact decisive games. If abstention is not allowed (see [6] for voting games with abstention) ties are not possible in decisive games.

This paper is organized as follows. Section 2 is devoted to weighted games, its main properties and to formally introduce several kinds of minimal integer realizations (*representations* in a more general context) for them. It specially focuses in majority games and several open problems related to minimal realizations. There, all proofs have been omitted because of the lack of the space. Section 3 explains the methodology and sketches the main algorithms used to obtain our results. Section 4 provides the most representative cases of minimal realizations. A conclusion section ends the paper.

2 Weighted Games and Majority Games

Weighted simple games (or *weighted games*, for short) are probably the most important kind of simple games.

Definition 6. *A simple game* (N, W) *is* weighted *if there exist a "*quota*"* $q \in \mathbb{R}$ *and a "*weight function*"* $w : N \to \mathbb{R}$ *such that each coalition* S *is winning exactly when the sum of weights of* S *meets or exceeds* q*.*

Any specific example of such a weight function w and quota q are said to *realize* G as a weighted game. A particular realization of a weighted game is denoted $(q; w_1, \ldots, w_n)$, or briefly $(q; w)$. By $w(S)$ we denote $\sum_{i \in S} w_i$. Three parameters can be defined for any realization $(q; w)$ of a weighted game (N, W):

$$T = w(N), \quad a = \min_{S \in W} w(S) \quad \text{and} \quad b = \max_{S \in L} w(S) \ .$$

From the definition of simple game, we have $0 < q \leq T$.

Although a simple game can fail to be proper and fail to be strong, this cannot happen with weighted games. Note that Example 1 with absolute majority is a proper and strong weighted game; for the 2/3-qualified majority, it is a proper and weak weighted game.

Proposition 1. *Any weighted game is either proper or strong.*

It is well–known that any weighted game admits an integer realization (see for instance [3]), that is, a weight function with nonnegative integer values, and a positive integer as quota. Integer realizations naturally arise; just consider the seats distributed among political parties in any voting system.

Henceforth we will only consider the set $\mathcal{I}$ of all integer realizations[1] for every weighted game (N, W). Note that, if $(q; w) \in \mathcal{I}$, then $(c \cdot q; c \cdot w) \in \mathcal{I}$ for every positive c, so that $\mathcal{I}$ is an unbounded *cone of integer values.*

The desirability relation, which goes back at least to Isbell [7] and was later generalized in [9] (see also [10,12]), is useful to define a natural preordering.

Definition 7. *Let (N, W) be a simple game.*

(i) *Player i is* more desirable *than j ($i \succeq j$, for short) in (N, W) if*

$$S \cup \{j\} \in W \;\Rightarrow\; S \cup \{i\} \in W, \qquad \textit{for all } S \subseteq N \setminus \{i, j\}.$$

(ii) *Players i and j are* equally desirable *($i \sim j$, for short) in (N, W) if*

$$S \cup \{i\} \in W \Leftrightarrow S \cup \{j\} \in W, \qquad \textit{for all } S \subseteq N \setminus \{i, j\}.$$

(iii) *Player i is* strictly more desirable *than player j ($i \succ j$, for short) in (N, W) if i is more desirable than j, but i and j are not equally desirable.*

If a weighted game admits $(q; w_1, \ldots, w_n)$ as a realization, then $w_i \geq w_j$ implies $i \succeq j$. Therefore, the desirability relation of weighted games is *complete.* (See [1] for a classification theorem of complete simple games.) Note that $w_i = w_j$ implies $i \sim j$. However, $i \sim j$ does not necessarily imply $w_i = w_j$. But $i \succ j$ does imply $w_i > w_j$. All these comments suggest the following definition.

Definition 8. *A realization $(q; w)$ of a weighted game is said to* preserve types *or to be* normalized *if $w_i = w_j$ whenever $i \sim j$. By $\mathcal{N}$ we denote the set of all normalized realizations of a game.*

From now on, assume w.l.o.g. that $i \succeq i + 1$ for every $i = 1, \ldots, n - 1$. The set of voters admits a partition in t classes $N_1, \ldots, N_t$ such that two voters i and j belong to the same class *iff* $i \sim j$. The extreme cases arise when $t = 1$ (all voters are equally desirable) and $t = n$ (each class reduces to a singleton). We sort the classes from the most desirable (N_1) to the least desirable (N_t).

In the following definitions, let $w \leq w'$ mean $w_i \leq w'_i$ for all $1 \leq i \leq n$.

Definition 9. *A realization $(q; w)$ of a weighted game is called* minimum *if $w \leq w'$ for all realization $(q'; w') \in \mathcal{I}$. By $\mathcal{MI}$ we denote the set of all minimum realizations of a game.*

For instance, one may easily check that the *minimum* realization $(6; 4, 3, 2, 1, 1, 0)$ with just 11 seats is equivalent to the one given in Example 1 with absolute majority.

[1] Since there is no confusion about the used game, we will write $\mathcal{I}$ instead of $\mathcal{I}(N, W)$, also for other sets defined later.

Definition 10. *A realization* $(q; w)$ *of a weighted game has* minimum sum *if* $w(N) \leq w'(N)$ *for all* $(q'; w') \in \mathcal{I}$*. By* $sM\mathcal{I}$ *we denote the set of all minimum sum realizations of a game.*

Definition 11. *A realization* $(q; w)$ *of a weighted game is a* minimum normalized realization *if* $w \leq w'$ *for all* $(q'; w') \in \mathcal{N}$*. By* $M\mathcal{N}$ *we denote the set of all minimum normalized realizations of a game.*

Definition 12. *A realization* $(q; w)$ *of a weighted game is a* minimum sum normalized realization *if* $w(N) \leq w'(N)$ *for all* $(q'; w') \in \mathcal{N}$*. By* $sM\mathcal{N}$ *we denote the set of all minimum sum normalized realizations of a game.*

Note that the quota q for minimum sum realizations must be a. This is why there is no restriction for the quota in the definitions above.

The next proposition summarizes some properties of these types of realizations.

Proposition 2. *For every weighted game,*

(*i*) $M\mathcal{I}$ *has at most one element.*
(*ii*) $sM\mathcal{I}$ *is never empty.*
(*iii*) $M\mathcal{I} \subseteq sM\mathcal{I}$*. Moreover,* $M\mathcal{I} = sM\mathcal{I}$ iff $M\mathcal{I}$ *reduces to a singleton.*
(*iv*) $M\mathcal{N}$ *has at most one element.*
(*v*) $sM\mathcal{N}$ *is never empty.*
(*vi*) $M\mathcal{N} \subseteq sM\mathcal{N}$*. Moreover,* $M\mathcal{N} = sM\mathcal{N}$ iff $M\mathcal{N}$ *reduces to a singleton.*
(*vii*) $M\mathcal{I} \subseteq M\mathcal{N}$*.*

Regarding Proposition 2, it would be interesting to find the *minimum* number of voters for a game in order to obtain: weighted games without a minimum integer realization (i.e., with $M\mathcal{I} = \emptyset$) and, distinguishing among, $|sM\mathcal{I}| = 1$ or $|sM\mathcal{I}| = 2$ or $|sM\mathcal{I}| > 2$. In the same way, it would also be meaningful determining the minimum number of voters for a weighted game without a minimum integer normalized realization, i.e. $M\mathcal{N} = \emptyset$.

In this topic, one of the goals during the last five decades has been to find examples for each of these cases. The construction of all weighted games for $n < 7$ goes as early as 1962 [11]. In 1970 [13] it was found that $M\mathcal{I} \neq \emptyset$ for each weighted game with $n < 8$. Quite recently [5] it has been proved that, for $n = 8$, there are 154 non–isomorphic weighted games such that $M\mathcal{I} \neq \emptyset$, although none of them decisive. However, all these 154 games have a unique normalized realization. Isbell [8] exhibited a remarkable example of a decisive weighted game with 12 voters in which the two affected voters with different weight are not equi–desirable: $(99; 38, 31, 31, 28, 23, 12, 11, 8, 6, 5, 3, 1)$, which also admits the equivalent realization $(99; 37, 31, 31, 28, 23, 12, 11, 8, 7, 5, 3, 1)$. Since 1979 [2], Isbell's example has been very useful in game theory. See [5] for an example of a non–decisive weighted game with $n = 10$ and $M\mathcal{N} = \emptyset$.

Now we introduce the class of games which are the aim of study in this paper.

Definition 13. *A simple game is a* majority game *if it is weighted and decisive.*

From Proposition 1, it follows that there are three kind of weighted games: proper but not strong, strong but not proper, and decisive. In what follows we will only consider decisive games.

We are interested in finding minimal realizations for majority games. The following result provides conditions for a realization to be minimal.

Proposition 3. *Consider a majority game. Any element* $(q; w)$ *in* $\mathcal{MI}$ *satisfies:* $a = q$*,* $b = q - 1$ *and* $T = 2q - 1$*.*

Therefore, the total weight T of every minimal realization of a majority game is an odd number, and its quota is $q = (T + 1)/2$. Henceforth we will represent such a minimal realization omitting the quota, which becomes redundant.

As we claimed, all majority games with $n < 9$ have a minimum realization. On the other hand, Isbell's game (which has $n = 12$) does not have a minimum normalized realization. The goal of the next sections is to find out, with the help of algorithms, majority games without a unique minimal realization and majority games without a unique normalized realization.

3 Algorithm to Classify Majority Games

We implemented several algorithms about majority games, in order to exhaustively study all the games with $n < 10$, and also to find examples with 10 voters depending on its minimal realization. An exhaustive study of games with $n = 10$ was beyond our CPU time possibilities, because the huge number of games (see Table 1 below). All the experiments were done with a processor AMD64X2 4400 (two cores at 2.2 GHz) with 4 Gb of DDR memory with ECC.

Because of the lack of space, we briefly sketch one of the algorithms (see Figure 1 in Appendix). The C++ code can be obtained by asking the authors.

The program uses an integer array $m[n]$, filled with empirical data, with the maximum possible weight for each player when there are n of them. The values for $n = 1, \ldots, 8$ are $1, 1, 2, 3, 5, 9, 18, 42$, respectively.

The main loop considers in increasing order the number of players n. For every such n, and for every sum of weights s between n and $n \cdot m[n]$ also in increasing order, a backtracking searches for all the combinations of n weights with total sum s.

The backtracking searches the combinations in lexicographical order, choosing the weights in increasing order from left to right, and cutting every useless branch. A branch is declared useless when it is known that the weight of every combination reachable from it will be strictly smaller or larger than s.

When we find a combination (we know that its sum of weights is s), we call a routine to check its properties. There, we first compute the minimal winning coalition W^m (also called *representative function*), that is, all the minimum subsets whose sum of weights reaches at least $\lfloor s/2 \rfloor + 1$. Each of these subsets is codified as a bitmap in one integer. Them all, which together form the representative function of the game, are stored in a specific order in a `vector<int>`.

We then check with `M.find(RF)` if the current vector is a new one or if it was already found before. We achieve this by storing every new vector in a `map` (similar to a `set`) data structure. Then,

– For every game with a new representative function, we check if the game verifies $|sM\mathcal{I}| = 2$. Note that, here, we do not have enough information to check whether $|sM\mathcal{N}| = 1$ or $|sM\mathcal{N}| > 1$.
– For every game with an already found representative function, we check if the game verifies: $|sM\mathcal{I}| = 1$, $|sM\mathcal{I}| > 1$, $|sM\mathcal{N}| = 1$, $|sM\mathcal{N}| > 1$, etcetera.

4 Realizations of Majority Games

All possible majority games with less than 10 voters were considered. Table 1 shows the obtained results.

Table 1. Number of Complete Simple Games (CG), Weighted Complete Simple Games (WG), Decisive Games (DG) and Majority Games (MG) with n voters

n	1	2	3	4	5	6	7	8	9
CG	1	3	8	25	117	1171	44313	16175188	–
WG	1	3	8	25	117	1111	29373	2730164	–
DG	1	1	2	3	7	21	135	2470	319124
MG	1	1	2	3	7	21	135	2470	175428

Some observations are worth noting for $n = 9$. In particular, there are just 14 majority games (see Table 2) such that $M\mathcal{I} = \emptyset$ but $|sM\mathcal{I}| > 1$. Muroga *et. al* [13] already found these realizations. Note that $|sM\mathcal{I}| = 2$ for all them. Of course, by adding null voters to these games we obtain new games with these properties.

In Table 3 we give examples of 10 voters with $|sM\mathcal{I}| > 2$, so that 10 voters is sharp to get games with $M\mathcal{I} = \emptyset$ and $|sM\mathcal{I}| > 2$. We have only listed here a small sample because their properties are similar. Thus, things become more

Table 2. All majority games with 9 voters such that $M\mathcal{I} = \emptyset$ and $|sM\mathcal{I}| = 2$. The other minimum sum integer realization are obtained by interchanging encircled weights.

#	T	q	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	#	T	q	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
1	47	24	11	9	6	6	4	4	4	②	①	8	63	32	15	13	9	7	7	5	4	②	①
2	49	25	13	7	6	6	4	4	4	③	②	9	63	32	15	13	10	8	6	4	4	②	①
3	53	27	14	9	⑦	⑥	5	5	3	2	2	10	65	33	13	11	10	8	6	6	⑤	④	2
4	55	28	13	9	7	7	6	4	4	③	②	11	65	33	17	12	8	8	⑦	⑥	3	2	2
5	55	28	13	11	7	7	5	5	4	②	①	12	67	34	16	14	11	9	6	4	4	②	①
6	55	28	13	11	8	6	6	4	4	②	①	13	71	36	17	15	11	9	7	5	4	②	①
7	59	30	17	9	8	⑦	⑥	5	3	2	2	14	75	38	18	16	12	10	7	5	4	②	①

Table 3. Some majority games with 10 voters such that $\mathcal{MI} = \emptyset$ and $|s\mathcal{MI}| > 2$. The other possible minimum sum integer realizations are obtained by interchanging encircled weights.

T,T'	q,q'	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w'_1	w'_2	w'_3	w'_4	w'_5	w'_6	w'_7	w'_8	w'_9	w'_{10}
95	48	22	18	12	12	8	8	8	③	③	1	22	18	12	12	8	8	8	④	②	1
97	49	25	13	12	10	8	8	8	⑥	⑤	2	25	13	12	10	8	8	8	⑦	④	2
99	50	26	14	12	12	8	8	8	⑤	⑤	1	26	14	12	12	8	8	8	⑥	④	1
103	52	24	20	13	13	9	9	8	③	③	1	24	20	13	13	9	9	8	④	②	1
107	54	25	17	13	13	10	8	8	⑥	⑤	2	25	17	13	13	10	8	8	⑦	④	2
111	56	25	17	15	13	12	8	8	⑥	⑤	2	25	17	15	13	12	8	8	⑦	④	2
111	56	26	18	14	14	12	8	8	⑤	⑤	1	26	18	14	14	12	8	8	⑥	④	1
111	56	26	22	14	14	10	10	8	③	③	1	26	22	14	14	10	10	8	④	②	1
111	56	26	22	16	12	12	8	8	③	③	1	26	22	16	12	12	8	8	④	②	1
113	57	24	21	16	16	13	7	6	6	②	②	24	21	16	16	13	7	6	6	③	①
…						…										…					

Table 4. Some majority games with 10 voters *without* minimum integer normalized realization, $\mathcal{MN} = \emptyset$

T,T'	q,q'	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w'_1	w'_2	w'_3	w'_4	w'_5	w'_6	w'_7	w'_8	w'_9	w'_{10}
73	37	19	14	⑪	⑦	5	5	5	3	2	2	19	14	⑫	⑥	5	5	5	3	2	2
75	38	⑮	13	13	7	6	6	4	4	4	③	⑯	13	13	7	6	6	4	4	4	②
77	39	⑯	15	11	9	6	6	4	4	4	②	⑰	15	11	9	6	6	4	4	4	①
77	39	20	13	⑩	⑨	7	7	5	2	2	2	20	13	⑪	⑧	7	7	5	2	2	2
79	40	19	13	10	10	8	⑤	4	4	4	②	19	13	10	10	8	⑥	4	4	4	①
79	40	23	⑫	8	8	7	7	⑤	3	3	3	23	⑬	8	8	7	7	④	3	3	3
81	41	⑮	13	13	9	7	7	6	4	4	③	⑯	13	13	9	7	7	6	4	4	②
81	41	17	⑭	13	11	6	6	4	4	4	②	17	⑮	13	11	6	6	4	4	4	①
81	41	⑱	17	11	9	6	6	4	4	4	②	⑲	17	11	9	6	6	4	4	4	①
81	41	21	11	10	10	8	⑥	4	4	4	③	21	11	10	10	8	⑦	4	4	4	②
…						…										…					

compelling when there are more than 9 voters. Unfortunately, it was not possible for us to study all (majority) games because of the huge number of games and the limitation of our computers, but we could study a sufficiently large subset of such games to find conspicuous examples.

All the examples given in Tables 2 and 3 satisfy $\mathcal{MN} \neq \emptyset$. Table 4 lists some majority games with 10 voters without minimum integer normalized realization. Therefore, 10 is *the least* number of voters required to get such majority games. This contribution concludes the open problem left by Isbell in 1959 [8] when he provided his famous example for 12 voters. All examples in Table 4 share the properties $\mathcal{MN} = \emptyset$ and $|s\mathcal{MN}| = 2$ with Isbell's example.

5 Conclusion

In this paper we have focused in majority games, with the following conclusions:

1. For less than 9 voters, all majority games have a minimum integer realization.
2. For 9 voters, there are exactly 14 majority games without a minimum integer realization, but all these games have a minimum normalized integer realization.
3. For 10 voters:
   – There exist majority games without a minimum integer realization and with more than two minimum sum integer realizations.
   – There exists majority games without a minimum normalized integer realization.

Note that 10 is the smallest number of voters for majority games without a minimum normalized integer realization.

References

1. Carreras, F., Freixas, J.: Complete simple games. Mathematical Social Sciences 32, 139–155 (1996)
2. Dubey, P., Shapley, L.S.: Mathematical properties of the Banzhaf power index. Mathematics of Operations Research 4(2), 99–131 (1979)
3. Freixas, J.: Structure of simple games. PhD thesis, Technical University of Catalonia, Manresa (Barcelona), Spain. In Spanish. Oct (1994)
4. Freixas, J.: The dimension for the European Union Council under the Nice rules. European Journal of Operational Research 156(2), 415–419 (2004)
5. Freixas, J., Molinero, X.: On the existence of a minimum integer representation for weighted voting systems. Annals of Operations Research, (Submitted, October 2006)
6. Freixas, J., Zwicker, W.S.: Weighted voting, abstention, and multiple levels of approval. Social Choice and Welfare 21, 399–431 (2003)
7. Isbell, J.R.: A class of simple games. Duke. Mathematics Journal 25, 423–439 (1958)
8. Isbell, J.R.: On the enumeration of majority games. Mathematical Tables and Other Aids. to Computation 13(65), 21–28 (1959)
9. Maschler, M., Peleg, B.: A characterization, existence proof, and dimension bounds for the kernel of a game. Pacific Journal of Mathematics 18, 289–328 (1966)
10. Muroga, S.: Threshold logic and its applications. Wiley-Interscience, New York, USA (1971)
11. Muroga, S., Toda, I., Kondo, M.: Majority decision functions of up to six variables. Mathematics of Computation 16(80), 459–472 (1962)
12. Muroga, S., Toda, I., Takasu, S.: Theory of majority decision elements. J. Franklin Inst. 271(5), 376–418 (1961)
13. Muroga, S., Tsuboi, T., Baugh, C.R.: Enumeration of threshold funcitons of eight variables. IEEE Trans. Computers 19(9), 818–825 (1970)
14. Von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, Princeton, New Jersey, USA (1944)
15. Taylor, A.D.: Mathematics and Politics. Springer, New York, USA (1995)
16. Taylor, A.D., Zwicker, W.S.: Simple games: desirability relations, trading, and pseudoweightings. Princeton University Press, New Jersey, USA (1999)

Appendix

Figure 1 sketches the used algorithms to compute all given examples.

```
const int N = 11;  // maximum n

typedef vector<int> VI;

map<VI, VI> M;
vector<int> weight(N);
VI RF;
int m[] = { 0, 1, 1, 2, 3, 5, 9, 18, 42, ... };

...

void check(int sum) {

    int quota = sum/2 + 1;
    RF = compute_representative_function(quota);

    if (M.find(RF) == M.end()) {  // RF is new

        M[RF] = weight;  // we store RF in M
        if (is_sMI_2()) print_game("sMI_2");  // case |sMI| = 2

    } else {

        if (is_sMI()) print_game("sMI");  // case |sMI| = 1
        else if (is_Not_sMI()) print_game("Not_sMI");  // case |sMI| > 1

        if (is_sMN()) print_game("sMN");  // case |sMN| = 1
        else if (is_Not_sMN()) print_game("Not_sMN");  // case |sMN| > 1

        ...
    }
}

// Fill the array weight[0..n-1] from the position i,
// using weights less than or equal to m,
// knowing that sum = weight[0] + ... + weight[i-1],
// and in such a way that the total sum of weights will be s.
void backtracking(int n, int i, int m, int sum, int s) {

    if (sum > s) return;
    if (i == n) check(sum);
    else {
        if (sum == s) return;
        int minimum = (s - sum - 1)/(n - i) + 1;
        int maximum = min(m, s - sum - (n - i - 1));
        for (int x = minimum; x <= maximum; ++x) {
            weight[i] = x;
            backtracking(i + 1, x, sum + x, s);
        }
    }
}

int main() {
    ...

    for (int n = 1; n <= N; ++n)
        for (int s = n; s <= n*m[n]; ++s)
            backtracking(n, 0, m[n], 0, s);

    ...
}
```

Fig. 1. Sketch of the used algorithms

Linear Transformations in Boolean Complexity Theory

Joel Friedman[1,2,⋆]

[1] Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, Canada
[2] Department of Mathematics, University of British Columbia, Vancouver, BC V6T 1Z2, Canada
jf@cs.ubc.ca
http://www.math.ubc.ca/~jf

Abstract. We attempt to understand a cohomological approach to lower bounds in Boolean circuits (of [Fri05]) by studying a very restricted case; in this case Boolean complexity is described via the kernel (or nullspace) of a fairly simple linear transformation and its transpose. We look at this linear transformation approach for Boolean functions where we only allow AND gates, which is essentially the SET COVER problem. These linear transformations can recover the linear programming bound. More importantly, we learn that the optimal linear transformation to use can depend on the Boolean function whose complexity we wish to bound; we also learn that infinite complexity (that can occur in AND complexity over arbitrary sets and "bases") appears as a limits of the linear transformation bounds.

Keywords: Boolean circuit complexity, linear transformations, cohomology.

1 Introduction

We describe some new ideas for obtaining lower bounds for the complexity of Boolean function. The general setting of these ideas, described in [Fri05], involves somewhat specialized ideas in the cohomology of Grothendieck topologies (or toposes). However, some very special cases of this setting are quite easy to describe with graph theory and linear transformations; we believe that these cases could shed light on the more general cohomological approach with arbitrary Grothendieck topologies.

We find it hard to understand the extent of the cohomological approach even if we only allow AND gates in our computation (which is essentially the SET COVER problem). Here we show that this linear transformation approach recovers the linear programming bound in SET COVER. But more compelling is the light it sheds on some aspects of the cohomological approach: first, one sometimes needs to vary the formal complexity measure depending on which

⋆ Research supported in part by an NSERC grant.

Boolean function one wants to bound; this contradicts our initial intuition that one should always hope to obtain a formal complexity measure via cohomology that simultaneously gives good bounds on every Boolean function on n variables. Second, in AND complexity functions can have infinite complexity (they can't be written as AND's of the prespecified "base" functions); this infinitely complexity can be recovered in the cohomological approach (and the linear transformation approach and the linear programming bound) by a limit of formal complexity measures.

In this abstract the proofs of Theorems 1 and 3 are omitted.

We thank the referees (unknown to us) for several suggestions and corrections on the manuscript.

1.1 Cohomology in Lower Bounds

Over twenty years ago lower bounds for algebraic decision trees were obtained by counting connected components (and in principle the sum of the Betti numbers) of associated topological spaces (see [DL76, SY82, BO83]). This lead to a hope that problems such as P vs. NP, viewed as lower bound problems in Boolean circuit complexity of a Boolean function, could be studied via cohomology, e.g., the sum of the Betti numbers of a topological space (associated in some way to the function). We are unaware of any essential progress in this direction to date. (But see [Sma87] for a success of algebraic topology and the braid group in another notion of complexity.) In [Fri05] we develop a number of ideas that we hope may eventually give Boolean circuit depth lower bounds; these ideas seem uninteresting unless we (1) generalize the notion of Betti number to dimensions of Ext groups of general sheaves, and (2) replace topological spaces with Grothendieck topologies.

There is a lot of appeal to trying to model Boolean complexity via cohomology over Grothendieck topologies: cohomology is a well studied subject that takes large or infinite dimensional vector spaces and extracts concise, meaningful information, and cohomology has various known connections to combinatorics. We emphasize that Grothendieck topologies provide many examples akin to the theory of finite graphs that ordinary topological spaces miss; also, since we are trying to model finite or discrete computation, it seems natural to look for models in "discrete spaces," which is very different from studying algebraic varieties as was done previously for bounds for algebraic decision trees.

Part of the present difficulty with this approach is that it involves a question in mathematics that has not received much attention. Usually cohomology is used to analyze a particular topology and pair of sheaves (often the first sheaf is the constant sheaf). Here we are asking to choose the topology and sheaves in order to model Boolean complexity, and it seems difficult for us to know how to select or rule out topologies and sheaves from the vast array of possibilities. We are hoping that the special case presented in terms of linear transformation, presumably much easier to analyze fully, will help select and rule out topologies and sheaves for the general cohomological approach.

1.2 Formal Complexity Measures

Let S be a set, and denote by $\mathbb{B}^S = \{0,1\}^S$ the collection of functions from S to $\mathbb{B} = \{0,1\}$ (viewing 1 as TRUE and 0 as FALSE). By a *formal AND measure* we mean a function, h, from $\mathbb{B}^S$ to the non-negative reals such that for all $f, g \in \mathbb{B}^S$ we have

$$h(f \wedge g) \leq h(f) + h(g) \tag{1}$$

(compare the notion of a *formal complexity measure*, e.g., see [Weg87]). Given a subset $\mathcal{S}_1 = \{f_1, \ldots, f_r\}$, let size$(f)$ be the minimum number of elements of $\mathcal{S}_1$ whose conjunction is f; we define size(f) to be infinite if f cannot be expressed as such a conjunction. Equivalently size(f) is the size of the smallest formula computing f via conjunctions of elements of $\mathcal{S}_1$. By induction on "size" we see that for any formal AND measure, h, we have

$$\text{size}(f) \geq h(f)/M, \qquad \text{where } M = \max_i h(f_i).$$

If h also satisfies

$$h(f) = h(\neg f), \tag{2}$$

then similarly we have an $h(f)/M$ lower bound on the size of a formula as before, but where the operations are either conjunctions or the negation of a conjunction; then $\log_2(h(f)/M)$ would bound the formula (or circuit) depth.

The ultimate goal of this project is to obtain formal AND measures that also satisfy equation (2) for circuit depth bounds with unrestricted gates with two inputs. In this work we concern ourselves only with general formal AND measures; however, it does not seem easy just to know which such measures can be constructed via linear transformations arising from vector spaces on graphs.

Given S and $f_1, \ldots, f_r$, determining size(f) is NP-complete; this is essentially the SET COVER problem. It can be approximated (to within $O(\log r)$) by a linear program that we describe below. We shall show that the linear programming bound can be obtained by the cohomological approach of [Fri05]. Furthermore this special case of the cohomologically obtained bounds can be described just with finite directed graphs and vector spaces.

We note that "AND-Circuits," (quite different but) related to the above in the case where S is a Boolean hypercube, have been studied in [AM06].

1.3 The Linear Programming Bound

Given S and $\mathcal{S}_1 \subset \mathbb{B}^S$, determining size$(f)$ is NP-complete, as it is almost a reformulation of SET COVER (see [Vaz01], for example, for SET COVER); indeed, to determine how many $\mathcal{S}_1$ elements we need to obtain f, we may assume $f \leq f_i$ for all i, and then the question is how many $f_i^{-1}(0)$ are required to cover $f^{-1}(0)$. The size complexity is given by the integer program

$$\min \sum_{i\in R} \mu_i, \text{ subject to}$$
$$\sum_{i\in R} \mu_i(1-f_i(s)) \geq 1, \ \forall s \in f^{-1}(0)$$
$$\mu_i = 0,1 \ \ i \in R,$$

where R is the set of i with $f \leq f_i$. A lower bound to this program is given by the "relaxed" linear program where the μ_i are non-negative reals. The gap between the integer and linear program is known to be as high as $\Omega(\log r)$ in certain cases, and never higher (see [Vaz01]). Equivalent to the linear program is its dual,

$$\max \sum_{s\in f^{-1}(0)} \alpha_s, \text{ subject to} \tag{3}$$
$$\sum_{s\in f^{-1}(0)} \alpha_s(1-f_i(s)) \leq 1, \ \forall i \in R \tag{4}$$
$$\alpha_s \geq 0 \ \forall s \in S \tag{5}$$

It is this linear program that we shall derive as a lower bound via linear transformations based on vector spaces on graphs.

2 Linear Transformations

By a *linear transformation*, L, we mean a map $L\colon s(L) \to t(L)$ of finite dimensional real vector spaces ($s(L)$ is called the source of L, $t(L)$ the target). We define the *Betti numbers* of L to be

$$b_0(L) = \dim(\mathrm{Ker}(L)), \quad b_1(L) = \dim(\mathrm{Coker}(L)) = b_0(L^*),$$

(where Ker denotes the kernel or nullspace, and $\mathrm{Coker}(L) = t(L)/\mathrm{Image}(L)$ denotes the cokernel). We define the *cohomological complexity* of L to be

$$\mathrm{cc}(L) = b_0(L) + b_1(L).$$

Roughly speaking our approach will associate to each Boolean function, $g\colon \mathbb{B}^n \to \mathbb{B}$ a linear transformation, L_g, so that

$$h(g) = \mathrm{cc}(L_g)$$

is a formal AND measure. More precisely, we will associate to each g a family of linear transformations, $\mathcal{L}_g$, such that b_0, b_1 of each element of $\mathcal{L}_g$ is the same. We shall need to know a condition on two linear transformation, L, M, to have the same b_0 and b_1.

By a *morphism* of linear transformations, $\phi\colon L \to M$, we mean a pair $\phi = (\phi_s, \phi_t)$ of maps making the diagram

$$\begin{tikzcd} s(L) \arrow[r, "L"] \arrow[d, "\phi_s"'] & t(L) \arrow[d, "\phi_t"] \\ s(M) \arrow[r, "M"] & t(M) \end{tikzcd}$$

commutative, i.e., such that $M\phi_s = \phi_t L$. Two morphisms, ϕ, ν, are *homotopy equivalent* if there is a map $K\colon t(L) \to s(M)$ such that $\phi_s - \nu_s = KL$ and $\phi_t - \nu_t = MK$. Say that ϕ is a *homotopy equivalence* if there is a $\theta\colon M \to L$ such that $\theta\phi$ is homotopy equivalent to the identity on L, and $\phi\theta$ is homotopy equivalent to the identity on M. It is not hard to see that in this case $b_i(L) = b_i(M)$.

We shall associate to g a family of linear transformations, $\mathcal{L}_g$, and homotopy equivalences between any two elements of $\mathcal{L}_g$ (so that b_0, b_1 are the same on all elements of $\mathcal{L}_g$). We shall usually just describe one particular element of $\mathcal{L}_g$, although the proof that h is an AND measure may be easier when one uses the whole family $\mathcal{L}_g$.

A pair of linear transformations

$$A \xrightarrow{\alpha} B \xrightarrow{\beta} C$$

is *exact at* B if $\mathrm{Ker}(\beta) = \mathrm{Image}(\alpha)$. In this case the dimension of B is bounded by the sum of those of A and C, since $B/\mathrm{Ker}(\beta)$ is mapped injectively to C, and the dimension of $\mathrm{Ker}(\beta)$ is that of $\mathrm{Image}(\alpha)$ which is at most that of A.

If $\nu\colon L_1 \to L_2$ is a morphism of linear transformation, then ν_s induces a map $\mathrm{Ker}(L_1) \to \mathrm{Ker}(L_2)$, and ν_t induces a map $\mathrm{Coker}(L_1) \to \mathrm{Coker}(L_2)$. Say that

$$L_1 \xrightarrow{\nu} L_2 \xrightarrow{\mu} L_3$$

is a *short exact sequence* if there exists a linear transformation $\delta\colon \mathrm{Ker}(L_3) \to \mathrm{Coker}(L_1)$ such that

$$0 \longrightarrow \mathrm{Ker}(L_1) \xrightarrow{\nu_s} \mathrm{Ker}(L_2) \xrightarrow{\mu_s} \mathrm{Ker}(L_3)$$

$$\xrightarrow{\delta} \mathrm{Coker}(L_1) \xrightarrow{\nu_t} \mathrm{Coker}(L_2) \xrightarrow{\mu_t} \mathrm{Coker}(L_3) \longrightarrow 0$$

is an exact sequence, i.e., the kernel of each map is the image of the preceding one. (Note that if $L_1 \to L_2 \to L_3$ is a short exact sequence in the usual sense of homological algebra, then it is one in this sense, using the Snake Lemma.) It is easy to see this implies that that the $\mathrm{cc}(L_i)$ satisfy the triangle inequality (i.e., any one is at most the sum of the other two). The proof of equation (1) will be based on a few short exact sequences.

3 Vector Spaces on Graphs

Let H be a directed acyclic graph, possibly with multiple edges. Let $V(H), E(H)$ denote the vertex and edge sets of H; if $e \in E(H)$ with tail u and head v, we write $e = (u, v)$ (this is somewhat abusive, since the graph may have multiple edges). By a *vector space on H*, F, we mean an association to each vertex, v, of H, a vector space Fv, and to each edge $e = (u, v)$ of H a linear transformation $Fe\colon Fv \to Fu$ (note the order is "reversed"). If $U \subset V(H)$, let F_U be F on all vertices in U and edges with both endpoints in U, and otherwise F_U zero.

If F, G are vector spaces on a graph, H, we wish to describe a linear transformation, $\mathcal{D}\mathrm{Hom}(F, G)$, the *derived Hom from F to G*. To do so, note that if $e = (u, v) \in E(H)$ and if $\mathrm{Lin}(Fv, Gv)$ is the vector space of linear transformations from Fv to Gv, there is a natural map $L_1(e)\colon \mathrm{Lin}(Fv, Gv) \to \mathrm{Lin}(Fv, Gu)$ given by $\phi \mapsto (Ge)\phi$; and similarly a map $L_2(e)\colon \mathrm{Lin}(Fu, Gu) \to \mathrm{Lin}(Fv, Gu)$ given by $\phi \mapsto \phi(Fe)$. So if we set

$$S = \bigoplus_{v \in V(H)} \mathrm{Lin}(Fv, Gv),$$

and

$$T = \bigoplus_{e=(u,v) \in E(H)} \mathrm{Lin}(Fv, Gu),$$

it makes to define $L\colon S \to T$ via $L = L_1 - L_2$, where for $i = 1, 2$, L_i is the block matrix given by summing the $L_i(e)$. We define $\mathcal{D}\mathrm{Hom}(F, G)$ to be this linear transformation.

Notice that the kernel of $\mathcal{D}\mathrm{Hom}(F, G)$ can be interpreted as the set of tuples of maps $\mu_v\colon Fv \to Gv$, one for each $v \in V(H)$, such that for every $e = (u, v)$ we have $\mu_u Fe = Ge\mu_v$.

The cohomological interpretation of $\mathcal{D}\mathrm{Hom}(F, G)$ is the morphisms from F to G in the derived category of presheaves (i.e., sheaves with the *topologie grossière*) of the free category associated to the graph H. Since the free category has homological dimension 1, as evident from the resolution by "vertices" and "edges" (see [Fri05]), the elements of the derived category can be taken to live on cochains supported in the zeroth and first position, which amounts to giving a single linear transformation. We further remark that $\mathcal{D}\mathrm{Hom}(F, G)$ has no "preferred definition," but all definitions are homotopy equivalent.

4 AND Measures Via Linear Transformations

Let F, G be vector spaces over a directed, acyclic graph, H. Say that $U \subset V(H)$ is *open* if for all $e = (u, v) \in H$ and $v \in U$ we have $u \in U$. Define the *complexity of U* (with respect to F, G) to be

$$\mathrm{cpx}_{F,G}(U) = \mathrm{cc}(\mathcal{D}\mathrm{Hom}(F, G)). \tag{6}$$

Theorem 1. *For any open sets, U, W, we have*

$$\mathrm{cpx}(U \cap W) \leq \mathrm{cpx}(U) + \mathrm{cpx}(W) + \mathrm{cpx}(V(H)).$$

Problem. Find a simple proof of this theorem.

One can prove Theorem 1 just by using linear algebra, provided one is willing to unwind the sheaf theoretic proof in [Fri06], although we imagine there may be a significantly shorter linear algebraic proof. The sheaf theoretic proof uses the invariance of the cohomological complexity of a matrix under homotopy equivalence, although this may not be necessary in this special case.

Say that $r\colon \mathbb{B}^S \to \mathit{Open}(H)$, a map from Boolean functions on S to open subsets of H, is an *inf-preserving map* if

$$r(f \wedge g) = r(f) \cap r(g)$$

for all $f, g \in \mathbb{B}^S$. Theorem 1 implies the following theorem.

Theorem 2. *Let H be a directed graph, F, G vector spaces on H, and $r\colon \mathbb{B}^S \to \mathit{Open}(H)$ an inf-preserving map.*

$$h(f) = \mathrm{cpx}_{F,G}(r(f)) + \mathrm{cpx}_{F,G}(V(H)) \tag{7}$$

is a formal AND measure, where cpx *is given in equation (6).*

5 LP Bound Via Cohomological AND Measures

Let H be the directed graph with $V(H) = \mathbb{B}^S$ and an edge (f, g) provided $f \leq g$ and $f(s) = g(s)$ for all but exactly one s in S. For any $f, g \in \mathbb{B}^S$ we define $P(f, g)$ to be the set of directed paths of H starting in f and ending in g (viewing a path of length k as a sequence of $k + 1$ vertices, $P(f, f)$ consists of one element, the path of length 0 consisting of the sequence (f)); if $e = (u, v) \in E(H)$, then for any f we have that e determines an *augmentation* map from $P(f, u)$ to $P(f, v)$ by adding v (i.e., in other words, adding the edge, e, if we think of a path as a sequence of edges). Given $f \in \mathbb{B}^S$ we define the vector space, K_f, on H via

$$K_f v = \mathbb{R}^{P(f,v)}$$

for any $v \in V(H)$, and for any $e = (u, v)$ we $K_f e$ is the transpose of the the incidence matrix of the aforementioned augmentation map.

A vector space, F over H is called a *superskyscraper* vector space if $Fe = 0$ for all $e \in E(H)$. Then F is determined (up to isomorphism) by the dimensions of the Fv with v varying over $V(H)$.

Consider the map $r\colon \mathbb{B}^S \to \mathrm{Open}(H)$ given by

$$r(f) = \{g | g \leq f\}.$$

Clearly $r(f \wedge g) = r(f) \cap r(g)$. We define the formal AND measure h as in equation (7) by taking a $g \in \mathbb{B}^S$ and fixing a non-negative integer A_v for each $v \in \mathbb{B}^S$ and setting $G = K_g$ and taking F to be a superskyscraper with $\dim(Fv) = A_v$ for all $v \in V(H)$.

Theorem 3. *Let h as above, and assume that $A_0 = 0$ (the subscript in A_0 is the Boolean function $0 \in V(H)$). Then*

$$h(f) = \sum_{w \not\leq f} A_w |P_f(g, w)|,$$

where $P_f(g, w)$ is the set of paths from g to w such that all but the last vertex (namely w) in the path are $\leq f$ (at all their values).

Problem. Find a simple proof of this theorem.

We claim this recovers the linear programming bound. Indeed, let δ_s for $s \in S$ be the Dirac delta function at s, and let $A_w = 0$ unless $w = g + \delta_s$ for some $s \in g^{-1}(0)$. So $P_f(g, g+\delta_s)$ (with $g(s) = 0$) is one if $g \leq f$ and $g+\delta_s$ is not $\leq f$, and zero otherwise. Theorem 3 shows that

$$\mathrm{cc}(f) = \sum_{s, g(s)=f(s)=0} A_{g+\delta_s}.$$

Denoting $\tilde{A}_s = A_{g+\delta_s}$, we get

$$\mathrm{size}(g) \geq \sum_s \tilde{A}_s / M,$$

for any M with

$$\sum_{s, g(s)=f_i(s)=0} \tilde{A}_s \leq M;$$

this is just the dual linear programming bound (restricted to $\alpha_s = A_s/M$ rational, α_s as in equations (3)–(5)).

We now see, as claimed in the introduction, that the vector space on H, $G = K_g$, depends on the Boolean functions whose size we wish to bound, and that the case of infinite size can be accounted for as a limiting AND measure.

References

[AM06] Arpe, J., Manthey, B.: Approximability of minimum and-circuits. Electronic Colloquium on Compuational Complexity, Report No. 45; also in Dagstuhl Seminar Proceedings 2006:603 (2006)

[BO83] Ben-Or, M.: Lower bounds for algebraic computation trees. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (a.k.a. STOC 1983), pp. 80–86 (1983)

[DL76] Dobkin, D., Lipton, R.J.: Multidimensional searching problems. SIAM J. Comput. 5(2), 181–186 (1976)

[Fri05] Friedman, J.: Cohomology of grothendieck topologies and lower bounds in boolean complexity. (to appear, 2005). http://www.math.ubc.ca/~jf, also http://arxiv.org/abs/cs/0512008

[Fri06] Friedman, J., Cohomology of grothendieck topologies and lower bounds in boolean complexity ii. (to appear, 2006). http://www.math.ubc.ca/~jf, also http://arxiv.org/abs/cs/0604024

[Sma87] Smale, S.: On the topology of algorithms. I. J. Complexity 3(2), 81–89 (1987)

[SY82] Michael Steele, J., Yao, A.C.: Lower bounds for algebraic decision trees. J. Algorithms 3(1), 1–8 (1982)

[Vaz01] Vazirani, V.V.: Approximation algorithms. Springer, Berlin (2001)

[Weg87] Wegener, I.: The complexity of Boolean functions. In: Wiley-Teubner Series in Computer Science, John Wiley & Sons Ltd, Chichester (1987)

Exact Pair Theorem for the ω-Enumeration Degrees

Hristo Ganchev*

Sofia University, Faculty of Mathematics and Informatics
5 James Bourchier blvd. 1164 Sofia, Bulgaria
h.ganchev@gmail.com

Abstract. In the paper the exact pair theorem for the ω-enumeration degrees is proved. As a corollary an exact pair theorem involving the jump operation for the enumeration degrees is obtained.

Mathematics subject classification: 03D30.

Keywords and Phrases: ω-enumeration degrees, enumeration degrees, exact pair, jump.

1 Introduction

The ω-enumeration degrees and the structure $\mathcal{D}_\omega$ are introduced by Soskov in [2]. In order to define it we first consider the set of all denumerable sequences of sets of natural numbers, which we will denote by $\mathcal{S}$. In $\mathcal{S}$ we define the reflexive and transitive relation "$\leq_u$" and the equivalence relation "$\equiv_u$" as follows: for any two sequences $\mathcal{A}, \mathcal{B} \in \mathcal{S}$

$$\mathcal{A} \leq_u \mathcal{B} \iff J_\mathcal{B} \subseteq J_\mathcal{A}, \quad \mathcal{A} \equiv_u \mathcal{B} \iff J_\mathcal{A} = J_\mathcal{B}$$

where $J_\mathcal{A}$ is the set of the Turing degrees of all $X \subseteq \mathbb{N}$ such that $(\forall k)(A_k$ is r.e. in $X^{(k)}$ uniformly in $k)$. Since we have that $A \equiv_u \mathcal{B} \iff \mathcal{A} \leq_u \mathcal{B} \,\&\, \mathcal{B} \leq_u \mathcal{B}$, by factorizing the structure $(\mathcal{S}, \leq_u)$ with respect to $\equiv_u$, we obtain the structure $(\mathcal{D}_\omega, \leq_\omega)$, where the factor relation $\leq_\omega$ is a partial order. By $d_\omega(\mathcal{A})$ we shall denote the equivalence class generated by the sequence $\mathcal{A}$.

In [2] it is shown that $\mathcal{D}_\omega$ is an upper semi-lattice with least element, that the Σ^0_2 ω-enumeration degrees are dense and that there is no minimal ω-enumeration degree. A notion of a jump operator in $\mathcal{D}_\omega$ is also introduced. It is shown that the substructure $\mathcal{D}_1 = \{d_\omega(\mathcal{A}) \mid A_n = \emptyset \text{ for } n > 0\}$ is isomorphic to the semi-lattice $\mathcal{D}_e$ of the enumeration degrees.

From the omitting theorem proved in [3] it follows that every ω-enumeration degree is the greatest lower bound of two ω-enumeration degrees strictly above it, i.e., every ω-enumeration degree has a minimal pair.

In this paper we shall prove that every monotone increasing sequence of ω-enumeration degrees has an exact pair.

* This work was partially supported by Sofia University Science Fund.

Definition 1. *Let $(X, \leq)$ be a partially ordered set and let $a_0 \leq a_1 \leq \dots \leq a_n \leq \dots$ be an increasing sequence in X. We say that $f, g \in X$ form an exact pair for $\{a_n\}_{n<\omega}$ if the following two conditions hold:*

(i) $(\forall n)(a_n \leq f, g)$;
(ii) $x \leq f, g \Longrightarrow \exists n(x \leq a_n)$.

The existence of exact pairs for the Turing degrees and for the enumeration degrees is proved by Spector [4] and Case [1] respectively.

Here we are going to prove the following theorem:

Theorem 1. *Let $\mathbf{a}_0 \leq_\omega \mathbf{a}_1 \leq_\omega \dots \leq_\omega \mathbf{a}_n \leq_\omega \dots$ be an increasing sequence in $\mathcal{D}_\omega$. Then for each $\mathbf{g} \in \mathcal{D}_\omega$, such that $\mathbf{a}_n \leq_\omega \mathbf{g}$, for all n, there is an $\mathbf{f} \in \mathcal{D}_1$ such that $\mathbf{f}, \mathbf{g}$ form an exact pair for the sequence $\{\mathbf{a}_n\}_{n<\omega}$. Even more. For arbitrary k the degrees $\mathbf{f}^{(k)}$, $\mathbf{g}^{(k)}$ form an exact pair for the sequence $\{\mathbf{a}_n^{(k)}\}_{n<\omega}$, where by $\mathbf{x}^{(k)}$ we denote the k-th jump of the degree $\mathbf{x}$ in $\mathcal{D}_\omega$.*

As a corollary we obtain the following exact pair theorem for the enumeration degrees.

Theorem 2. *Let $\mathbf{a}_0 \leq_e \mathbf{a}_1 \leq_e \dots \leq_e \mathbf{a}_n \leq_e \dots$ be an increasing sequence in the semi-lattice of the enumeration degrees $\mathcal{D}_e$. Then for each $\mathbf{g} \in \mathcal{D}_e$, such that $\mathbf{a}_n \leq_e \mathbf{g}$, for all n, there is an $\mathbf{f} \in \mathcal{D}_e$ such that $\mathbf{f}^{(k)}$, $\mathbf{g}^{(k)}$ form an exact pair for the sequence $\{\mathbf{a}_n^{(k)}\}_{n<\omega}$ for arbitrary k, where by $\mathbf{x}^{(k)}$ we denote the k-th enumeration jump of the degree $\mathbf{x}$.*

2 Preliminaries

Let $W_0, W_1, \dots, W_n, \dots$ be a Gödel enumeration of the recursively enumerable sets. We will use the same notation for the enumeration operators, i.e., the operators over sets of natural numbers acting by the rule $W_e(A) = \{x \mid \exists v(\langle x, v\rangle \in W_e \ \&\ D_v \subseteq A)\}$ for each $A \subseteq \mathbf{N}$, where by D_v we shall denote the finite set with canonical index v.

The enumeration jump of a set A is defined as $A' = L_A \oplus \mathbf{N} \setminus L_A$, where $L_A = \{\langle a, x\rangle \mid x \in W_a(A)\}$.

Given a sequence $\mathcal{A} = \{A^n\}_{n<\omega}$ of sets of natural numbers we define its jump sequence $P(\mathcal{A}) = \{P^n(\mathcal{A})\}_{n<\omega}$ as follows:

$$\begin{array}{ll} \text{(i)} & P^0(\mathcal{A}) = A^0 \\ \text{(ii)} & P^{n+1}(\mathcal{A}) = P^n(\mathcal{A})' \oplus A^{n+1}. \end{array}$$

In [3] it is shown that for any two sequences $\mathcal{A}, \mathcal{B}$ we have that $\mathcal{A} \leq_u \mathcal{B} \iff$ there is a recursive function g, such that $A^n = W_{g(n)}(P^n(\mathcal{B}))$ for all natural n (in particular $A^n \leq_e P^n(B)$). This gives us that $\mathcal{A} \equiv_u P(\mathcal{A})$. The jump of a sequence $\mathcal{A} \equiv_u \{P^n(\mathcal{A})\}_{n<\omega}$ is the sequence $\mathcal{A}' = \{P^{n+1}(\mathcal{A})\}_{n<\omega}$. It is true that $J_{\mathcal{A}'} = \{\mathbf{a}' \mid \mathbf{a} \in J_{\mathcal{A}}\}$ and that the jump agrees with the embedding of the e-degrees, so $(A, \emptyset, \emptyset, \dots, \emptyset)' \equiv_u (A', \emptyset, \emptyset, \dots, \emptyset)$.

Now we are ready to show that Theorem 2 follows from Theorem 1.

Proof of Theorem 2. Suppose that we have an increasing sequence of enumeration degrees $\mathbf{a}_0 \leq_e \mathbf{a}_1 \leq_e \cdots \leq_e \mathbf{a}_n \leq_e \ldots$ and let $\mathbf{g}$ be an enumeration degree, such that $\mathbf{g}$ is an upper bound for $\{\mathbf{a}_n\}_{n<\omega}$. Fix a sequence of sets $A_0, A_1, \ldots, A_n, \ldots$ such that $\mathbf{a}_n = d_e(A_n)$ and let G be such that $\mathbf{g} = d_e(G)$. Now for each n we take $\mathcal{A}_n$ to be the sequence $(A_n, \emptyset, \emptyset, \ldots, \emptyset, \ldots)$ and $\mathcal{G}$ to be the sequence $(G, \emptyset, \emptyset, \ldots, \emptyset, \ldots)$. It is clear that $\mathcal{A}_0 \leq_u \mathcal{A}_1 \leq_u \cdots \leq_u \mathcal{A}_n \leq_u \cdots \leq_u \mathcal{G}$. Thus we have an increasing sequence

$$d_\omega(\mathcal{A}_0) \leq_\omega d_\omega(\mathcal{A}_1) \leq_\omega \cdots \leq_\omega d_\omega(\mathcal{A}_n) \leq_\omega \cdots \leq_\omega d_\omega(\mathcal{G})$$

of ω-enumeration degrees.

Now according to Theorem 1 there is an ω-enumeration degree $\mathbf{f} \in \mathcal{D}_1$, such that $\mathbf{f}$ and $d_\omega(\mathcal{G})$ form an exact pair for the sequence $\{d_\omega(\mathcal{A}_n)\}_{n<\omega}$. Since $\mathbf{f} \in \mathcal{D}_1$, then $\mathbf{f}$ is of the form $\mathbf{f} = d_\omega(\mathcal{F})$, where $\mathcal{F} = (F, \emptyset, \emptyset, \ldots, \emptyset, \ldots)$. We claim that $d_e(G)$ and $d_e(F)$ form an exact pair for the sequence $\mathbf{a}_0 \leq_e \mathbf{a}_1 \leq_e \cdots \leq_e \mathbf{a}_n \leq_e \ldots$. Indeed: suppose that some enumeration degree $\mathbf{x}$ has the property $\mathbf{x} \leq d_e(G)$ and $\mathbf{x} \leq d_e(F)$. Let $\mathbf{x} = d_e(X)$ and consider the sequence $\mathcal{X} = (X, \emptyset, \emptyset, \ldots, \emptyset, \ldots)$. We have that $\mathcal{X} \leq_u \mathcal{F}$ and $\mathcal{X} \leq_u \mathcal{G}$ and therefor $\mathcal{X} \leq_u \mathcal{A}_n$ for some n. But then $X \leq_e A_n$, which proves that $d_e(F)$ and $d_e(G)$ are an exact pair for the sequence $\mathbf{a}_0 \leq_e \mathbf{a}_1 \leq_e \cdots \leq_e \mathbf{a}_n \leq_e \ldots$.

According to the definition of the jump operation in $\mathcal{D}_\omega$ we have that the sequences $\mathcal{A}'_n$, $\mathcal{G}'$ and $\mathcal{F}'$ are equivalent to the sequences $(A'_n, \emptyset, \emptyset, \ldots, \emptyset, \ldots)$, $(G', \emptyset, \emptyset, \ldots, \emptyset, \ldots)$ and $(F', \emptyset, \emptyset, \ldots, \emptyset, \ldots)$ respectively. Applying the same reasoning as above, we obtain that $d_e(F')$ and $d_e(G')$ (which are actually $d_e(F)'$ and $d_e(G)'$) are an exact pair for the sequence $\mathbf{a}'_0 \leq_e \mathbf{a}'_1 \leq_e \cdots \leq_e \mathbf{a}'_n \leq_e \ldots$. □

In the rest of the paper we shall present a proof of Theorem 1.

3 Proof of Theorem 1

First we will prove the following omitting theorem:

Theorem 3. *Let $\mathcal{A}_0 \leq_u \mathcal{A}_1 \leq_u \ldots \leq_u \mathcal{A}_n \ldots$ be an increasing sequence of sequences of sets of natural numbers. Let also $\{\mathcal{R}_k\}_{k<\omega}$ be a sequence of elements of $\mathcal{S}$ such that $\mathcal{R}_k \not\leq_u \mathcal{A}_n$, for each k and n. Then there is an $\mathcal{F} \in \mathcal{S}$ such that $\mathcal{A}_n \leq_u \mathcal{F}$, for each n, but $\mathcal{R}_k \not\leq_u \mathcal{F}$ for all k.*

For simplicity we will show how to omit only one sequence $\mathcal{R}$. However, by a simple and standard modification we can use the same technique to omit countably many sequences.

Let us a fix an increasing sequence $\mathcal{A}_0 \leq_u \mathcal{A}_1 \leq_u \cdots \leq_u \mathcal{A}_n \ldots$ of elements of $\mathcal{S}$ and let $\mathcal{R} \in \mathcal{S}$ be such that $\mathcal{R} \not\leq_u \mathcal{A}_n$, for every natural n. We will suppose that $\mathcal{A}_n = \{A^k_n\}_{k<\omega}$ and $\mathcal{R} = \{R^k\}_{k<\omega}$ (we will use upper indexes for coordinates in the rest of the paper). Fix a sequence $g_0, g_1, \ldots, g_n, \ldots$ of all recursive functions.

We will search $\mathcal{F}$ in the form $\mathcal{F} = (f^0, f^1, \ldots, f^n, \ldots)$, where f^n is a mapping from $\mathbf{N}$ in $\mathbf{N}$. We will use the notation $\overrightarrow{f}$ instead of $\mathcal{F}$. We will construct $\mathcal{F}$ by using a forcing technique so we have to define the modelling and forcing relations and the notion of finite parts.

Let us begin with the "finite" parts. We shall call finite part every sequence of the form $\overrightarrow{\tau} = (\tau^0, \tau^1, \dots, \tau^n, \dots)$, where each τ^i is a mapping from of an initial segment of $\mathbf{N}$ into $\mathbf{N}$. By $\mathrm{lh}\,(\tau^i)$ we shall denote the length of the initial segment which is the domain of τ^i. We shall use the notation $\overrightarrow{\tau}$, $\overrightarrow{\rho}$, $\overrightarrow{\delta}$ and $\overrightarrow{\sigma}$ (sometimes with indexes) for finite parts and τ^i, ρ^i, δ^i and σ^i for the respective coordinates.

If n is a natural number, by $\overrightarrow{n}$ we shall denote an increasing sequence $a_0 < a_1 < a_2 < \ldots < a_n$ of $n+1$ natural numbers. We will write $\overrightarrow{n} \preceq \overrightarrow{m}$ if $n \leq m$ and $\overrightarrow{m}$ is an extension of $\overrightarrow{n}$.

Since $\overrightarrow{f}$ has to "encode" in itself all sequences $\mathcal{A}_n$ we will regard only special finite parts which we will call $\overrightarrow{n}$-coding finite parts and which are defined by:

Definition 2. *If $\overrightarrow{n} = a_0 < a_1 < a_2 < \ldots < a_n$ is an increasing sequence of natural numbers and $\overrightarrow{\tau}$ is a finite part, we say that $\overrightarrow{\tau}$ is $\overrightarrow{n}$-coding iff for all i we have that:*

$$a_k < \langle x, k\rangle < \mathrm{lh}\,(\tau^i) \Longrightarrow \tau^i(\langle x, k\rangle) \in A^i_k$$

Now we are ready to define the modelling and forcing relations for the conditions F^k_e (the condition F^k_e will be responsible for what happens to the k-th coordinate when we apply the enumeration operator W_e to it). We begin with the modeling relation since it is more natural.

Definition 3. *Let $\overrightarrow{f} = (f^0, f^1, \dots, f^n, \dots)$ be a sequence of mappings from $\mathbf{N}$ into $\mathbf{N}$. We define the relations $\overrightarrow{f} \models (\neg)F^k_e(x)$ (for each e) using induction on k:*

(i) $\overrightarrow{f} \models F^0_e(x) \iff x \in W_e(\mathrm{Graph}\,(f^0))$.
(ii) $\overrightarrow{f} \models \neg F^0_e(x) \iff \overrightarrow{f} \not\models F^0_e(x)$.
(iii) $\overrightarrow{f} \models F^{k+1}_e(x)$ *if there exist u, u_0 and u_1 such that*
  (1) $\langle x, u\rangle \in W_e$;
  (2) $D_u = D_{u_1} \oplus D_{u_2}$;
  (3) *for each $v \in D_{u_1}$ is either true that $v = 2\langle e_v, x_v\rangle$ and $\overrightarrow{f} \models F^k_{e_v}(x_v)$, or $v = 2\langle e_v, x_v\rangle + 1$ and $\overrightarrow{f} \models \neg F^k_{e_v}(x_v)$;*
  (4) $D_{u_2} \subseteq \mathrm{Graph}\, f^{k+1}$.
(iv) $\overrightarrow{f} \models \neg F^{k+1}_e(x) \iff \overrightarrow{f} \not\models F^{k+1}_e(x)$.

Note that from the definition it follows that do determine whether $\overrightarrow{f} \models (\neg)F^k_e(x)$ it is sufficient to know only the first $k+1$ elements of the sequence $\overrightarrow{f}$, i.e., f^0, f^1,. . ., f^k. Even more. The following proposition is true:

Proposition 1. *Let $\overrightarrow{f}$ be a sequence of mappings from $\mathbf{N}$ in $\mathbf{N}$. Then for each e and each k*

$$\overrightarrow{f} \models F^k_e(x) \iff x \in W_e(P^k(\overrightarrow{f}))$$

Now we will give the definition of the forcing relation.

Definition 4. *Let $\overrightarrow{n}$ be an increasing sequence of $n+1$ natural numbers and let $\overrightarrow{\tau}$ be an $\overrightarrow{n}$-coding sequence of finite parts. We define the relations $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} (\neg)F_e^k(x)$ (for each e) using induction on k:*

(i) $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} F_e^0(x) \iff x \in W_e(\operatorname{Graph} \tau^0)$.
(ii) $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} \neg F_e^0(x) \iff (\forall \overrightarrow{\rho} \supseteq \tau)(\overrightarrow{\rho}$ *is* $\overrightarrow{n}$*-coding* $\Rightarrow \overrightarrow{\rho} \nVdash_{\overrightarrow{n}} F_e^0(x))$.
(iii) $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} F_e^{k+1}$ *if there exist* u, u_0 *and* u_1 *such that:*
  (1) $\langle x, u\rangle \in W_e$;
  (2) $D_u = D_{u_1} \oplus D_{u_2}$;
  (3) *for each* $v \in D_{u_1}$ *is either true that* $v = 2\langle e_v, x_v\rangle$ *and* $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} F_{e_v}^k(x_v)$, *or* $v = 2\langle e_v, x_v\rangle + 1$ *and* $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} \neg F_{e_v}^k(x_v)$;
  (4) $D_{u_2} \subseteq \operatorname{Graph} \tau^{k+1}$.
(iv) $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} \neg F_e^{k+1}(x) \iff (\forall \overrightarrow{\rho} \supseteq \tau)(\overrightarrow{\rho}$ *is* $\overrightarrow{n}$*-coding* $\Rightarrow \overrightarrow{\rho} \nVdash_{\overrightarrow{n}} F_e^{k+1}(x))$.

Again from the definition it follows that the relations $\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} (\neg)F_e^k(x)$ depend only on the first $k+1$ members of the sequence $\overrightarrow{\tau}$.

The next proposition shows that the forcing relations are monotone.

Proposition 2. *Let $\overrightarrow{m}$ and $\overrightarrow{n}$ be two increasing sequences of natural numbers of length $m+1$ and $n+1$ respectively. Let also $\overrightarrow{\tau}$ and $\overrightarrow{\rho}$ be two finite parts. Then the following are true:*

(1) *if $\overrightarrow{\tau}$ and $\overrightarrow{\rho}$ are both $\overrightarrow{n}$-coding and $\overrightarrow{\tau} \subseteq \overrightarrow{\rho}$, then*

$$\overrightarrow{\tau} \Vdash_{\overrightarrow{n}} (\neg)F_e^k(x) \Longrightarrow \overrightarrow{\rho} \Vdash_{\overrightarrow{n}} (\neg)F_e^k(x);$$

(2) *if $\overrightarrow{m} \preccurlyeq \overrightarrow{n}$ and $\overrightarrow{\tau}$ is $\overrightarrow{n}$-coding, then $\overrightarrow{\tau}$ is $\overrightarrow{m}$-coding and*

$$\overrightarrow{\tau} \Vdash_{\overrightarrow{m}} (\neg)F_e^k(x) \Longrightarrow \overrightarrow{\tau} \Vdash_{\overrightarrow{n}} (\neg)F_e^k(x);$$

(3) *if $\overrightarrow{m} \preccurlyeq \overrightarrow{n}$ and $\overrightarrow{\tau}$ is $\overrightarrow{m}$-coding, $\overrightarrow{\rho}$ is $\overrightarrow{n}$-coding and $\overrightarrow{\tau} \subseteq \overrightarrow{\rho}$, then*

$$\overrightarrow{\tau} \Vdash_{\overrightarrow{m}} (\neg)F_e^k(x) \Longrightarrow \overrightarrow{\rho} \Vdash_{\overrightarrow{n}} (\neg)F_e^k(x).$$

Now we are ready to begin the construction. The construction of $\overrightarrow{f}$ will be carried out by steps. At each step s we will construct a finite part $\overrightarrow{\tau}_s$ and a monotone sequence $\overrightarrow{s}$ in such a way that $\overrightarrow{\tau}_s$ is $\overrightarrow{s}$-coding and also that $\overrightarrow{\tau}_s \subseteq \overrightarrow{\tau}_{s+1}$ and $\overrightarrow{s} \preccurlyeq \overrightarrow{s+1}$. Finally we will set $\overrightarrow{f} = \bigcup_{s<\omega} \overrightarrow{\tau}_s$ and we will obtain an infinite increasing sequence $\overrightarrow{\omega} = \bigcup_{s<\omega} \overrightarrow{s}$ of natural numbers.

At step 0 we set $\overrightarrow{\tau}_0 = (\emptyset, \emptyset, \ldots, \emptyset, \ldots)$ and $\overrightarrow{0} = 0$.

Now suppose that $\overrightarrow{\tau}_s$ and $\overrightarrow{s}$ are defined. We divide the step s into three substeps.

First substep. We build an $\overrightarrow{s}$-coding finite part $\overrightarrow{\delta}_s$ such that $\overrightarrow{\tau}_s \subseteq \overrightarrow{\delta}_s$ and $\overrightarrow{\delta}_s$ encodes the first elements of sets A_i^j, for $i, j \leq s$, which are not encoded by $\overrightarrow{\tau}_s$.

Second substep. We build an $\overrightarrow{s}$-coding finite part $\overrightarrow{\sigma}_s \supseteq \overrightarrow{\delta}_s$ such that for each $k \leq s$ and each $\langle e, x\rangle \leq s$ either $\overrightarrow{\sigma} \Vdash_{\overrightarrow{s}} F^k_e(x)$ or $\overrightarrow{\sigma} \Vdash_{\overrightarrow{s}} \neg F^k_e(x)$.

Third substep. We consider the sequence $\{C^n_s\}_{n<\omega}$, where

$$C^n_s=\{x \mid (\exists \overrightarrow{\rho} \supseteq^n \overrightarrow{\sigma}_s)(\overrightarrow{\rho} \text{ is } \overrightarrow{s}\text{-coding } \& \ \rho^0(\mathrm{lh}\,\sigma^0_s) \simeq x \ \& \ \overrightarrow{\rho} \Vdash_{\overrightarrow{s}} F^n_{g_s(n)}(\mathrm{lh}\,\sigma^0_s))\},$$

where $\overrightarrow{\rho} \supseteq^n \overrightarrow{\sigma}_s$ means that $\rho^i \supseteq \sigma^i_s$ for $0 \leq i \leq n$, and $\rho^i = \sigma^i_s$ for $i > n$. The sequence $\{C^n_s\}_{n<\omega}$ is uniformly reducible to the sequence $\mathcal{A}_s$, i.e., $\{C^n_s\}_{n<\omega} \leq_u \mathcal{A}_s$ and therefore $\{C^n_s\}_{n<\omega} \neq \mathcal{R}$ (for a detailed proof of an analogous statement see [3]). Let n be such that $C^n_s \neq R_n$. Then there is a x such that either $x \in C^n_s \ \& \ x \notin R_n$ or $x \notin C^n_s \ \& \ x \in R_n$.

If $x \in C^n_s \ \& \ x \notin R_n$ is the case, we take one $\overrightarrow{\rho} \supseteq^n \overrightarrow{\sigma}_s$ such that $\overrightarrow{\rho}$ is $\overrightarrow{s}$-coding, $\rho^0(\mathrm{lh}\,\sigma^0_s) \simeq x$ and $\overrightarrow{\rho} \Vdash_{\overrightarrow{s}} F^n_{g_s(n)}(\mathrm{lh}\,\sigma^0_s)$, and we set $\overrightarrow{\tau}_{s+1} = \overrightarrow{\rho}$.

If $x \notin C^n_s \ \& \ x \in R_n$ then we set $\overrightarrow{\tau}_{s+1}$ to be an $\overrightarrow{s}$-coding finite part such that $\tau^0_{s+1}(\mathrm{lh}\,\sigma^0_s) \simeq x$. Note that in this case is true that

$$(\forall \overrightarrow{\rho} \supseteq^n \overrightarrow{\tau}_{s+1})(\overrightarrow{\rho} \text{ is } \overrightarrow{s}\text{-coding} \implies \overrightarrow{\rho} \not\Vdash_{\overrightarrow{s}} \mathcal{F}^n_{g_s(n)}(\mathrm{lh}\,\sigma^0_s)).$$

Since the forcing relation $\overrightarrow{\rho} \Vdash_{\overrightarrow{s}} F^n_{g_s(n)}$ depends only on the first $n+1$ elements of $\overrightarrow{\rho}$, we have that

$$(\forall \overrightarrow{\rho} \supseteq \overrightarrow{\tau}_{s+1})(\overrightarrow{\rho} \text{ is } \overrightarrow{s}\text{-coding} \implies \overrightarrow{\rho} \not\Vdash_{\overrightarrow{s}} \mathcal{F}^n_{g_s(n)}(\mathrm{lh}\,\sigma^0_s)),$$

which means that $\overrightarrow{\tau}_{s+1} \Vdash_{\overrightarrow{s}} \neg F^n_{g_s(i)}(\mathrm{lh}\,\sigma^0_s)$.

Finally we set $\overrightarrow{s+1} = \overrightarrow{s} * \max\{\mathrm{lh}\,\tau^i_{s+1} \mid i < \omega\}$, which concludes the construction. In order to complete the proof, we have to prove three properties of $\overrightarrow{f}$.

Claim 1. *$\mathcal{A}_n \leq_u \overrightarrow{f}$, for each n.*

Proof. Let $\overrightarrow{\omega} = a_0, a_1, \ldots, a_i, \ldots$. Then, according to the first substep of each step of the construction, we have that $A^i_n = \{f_i(\langle x, i\rangle) \mid \langle x, i\rangle \geq a_n\}$. □

Claim 2. (Truth lemma) *For each e and each k*

$$\overrightarrow{f} \models (\neg)F^k_e(x) \Leftrightarrow (\exists \overrightarrow{m} \preccurlyeq \overrightarrow{\omega})(\exists \overrightarrow{\tau} \subseteq \overrightarrow{f})(\overrightarrow{\tau} \textit{ is } \overrightarrow{m}\textit{-coding } \& \ \overrightarrow{\tau} \Vdash_{\overrightarrow{m}} (\neg)F^k_e(x)).$$

Proof. This property is assured from the second substeps. We will prove it using induction on k. First consider $k = 0$. The positive equivalence is obvious from the definition of the relations $\models F^0_e$ and $\Vdash_{\overrightarrow{m}} F^0_e$. Now we will prove the equivalence

$$\overrightarrow{f} \models \neg F^0_e(x) \Leftrightarrow (\exists \overrightarrow{m} \preccurlyeq \overrightarrow{\omega})(\exists \overrightarrow{\tau} \subseteq \overrightarrow{f})(\overrightarrow{\tau} \text{ is } \overrightarrow{m}\text{-coding } \& \ \overrightarrow{\tau} \Vdash_{\overrightarrow{m}} \neg F^0_e(x)).$$

The right to left direction follows from the positive equivalence. Let us prove the left to right direction. Suppose that $\overrightarrow{f} \models \neg F^0_e(x)$. Then consider a step $s+1$ such that $s > \langle e, x\rangle$. In the second substep we have constructed an $\overrightarrow{s}$-coding finite part $\overrightarrow{\sigma}_s$ such that either $\overrightarrow{\sigma}_s \Vdash_{\overrightarrow{s}} F^0_e(x)$ or $\overrightarrow{\sigma}_s \Vdash_{\overrightarrow{s}} \neg F^0_e(x)$ and

$\overrightarrow{\sigma}_s \subseteq \overrightarrow{\tau}_{s+1} \subseteq \overrightarrow{f}$. If $\overrightarrow{\sigma}_s \Vdash_{\overrightarrow{s}} F_e^0(x)$ than from the positive equivalence we obtain that $\overrightarrow{f} \models F_e^0(x)$, which contradicts $\overrightarrow{f} \models F_e^0(x)$. Therefore $\overrightarrow{\sigma}_s \Vdash_{\overrightarrow{s}} \neg F_e^0(x)$.

Now suppose that the statement is true for k. First we prove that

$$\overrightarrow{f} \models F_e^{k+1}(x) \Leftrightarrow (\exists \overrightarrow{m} \preccurlyeq \overrightarrow{\omega})(\exists \overrightarrow{\tau} \subseteq \overrightarrow{f})(\overrightarrow{\tau} \text{ is } \overrightarrow{m}\text{-coding } \& \, \overrightarrow{\tau} \Vdash_{\overrightarrow{m}} F_e^{k+1}(x)).$$

For the left to right direction suppose that $\overrightarrow{f} \models F_e^{k+1}(x)$. Then, according to the definition we have that, there are u, u_1 and u_2 such that:

(1) $\langle x, u\rangle \in W_e$;
(2) $D_u = D_{u_1} \oplus D_{u_2}$;
(3) for all $v \in D_{u_1}$ is either true that $v = 2\langle e_v, x_v\rangle$ and $\overrightarrow{f} \models F_{e_v}^k(x_v)$, or $v = 2\langle e_v, x_v\rangle + 1$ and $\overrightarrow{f} \models \neg F_{e_v}^k(x_v)$.
(4) $D_{u_2} \subseteq \operatorname{Graph} f_{k+1}$

Consider (3). According to the induction hypothesis, for each $v \in D_{u_1}$ we can find $\overrightarrow{m}_v \preccurlyeq \overrightarrow{\omega}$ and $\overrightarrow{\rho}_v \subseteq \overrightarrow{f}$, such that $\overrightarrow{\rho}_v$ is $\overrightarrow{m}_v$-coding and

$$v = 2\langle e_v, x_v\rangle \Longrightarrow \overrightarrow{\rho}_v \Vdash_{\overrightarrow{m}_v} F_{e_v}^k(x_v),$$

$$v = 2\langle e_v, x_v\rangle + 1 \Longrightarrow \overrightarrow{\rho}_v \Vdash_{\overrightarrow{m}_v} \neg F_{e_v}^k(x_v).$$

Since $\overrightarrow{m}_v \preccurlyeq \overrightarrow{\omega}$ and $\overrightarrow{\rho}_v \subseteq \overrightarrow{f}$, there is an $\overrightarrow{m} \preccurlyeq \overrightarrow{\omega}$ and a $\overrightarrow{\tau} \subseteq \overrightarrow{f}$, such that for each v, $\overrightarrow{m}_v \preccurlyeq \overrightarrow{m}$ and $\overrightarrow{\rho}_v \subseteq \overrightarrow{\tau}$. Then from the monotonicity of the forcing relation we obtain that for each $v \in D_{u_1}$

$$v = 2\langle e_v, x_v\rangle \Longrightarrow \overrightarrow{\tau} \Vdash_{\overrightarrow{m}} F_{e_v}^k(x_v),$$

$$v = 2\langle e_v, x_v\rangle + 1 \Longrightarrow \overrightarrow{\tau} \Vdash_{\overrightarrow{m}} \neg F_{e_v}^k(x_v),$$

which is exactly point (3) from the definition of the forcing relation $\Vdash_{\overrightarrow{m}} F_e^{k+1}$. We can extend the $k+2$-nd element of $\overrightarrow{\tau}$ in such a way that $\tau^{k+1} \subseteq \overrightarrow{f}_{k+1}$ and $D_{u_2} \subseteq \operatorname{Graph} \tau_{k+1}$. Of course this extension does not afflict the forcing relations that we have satisfied, since they depend only on the first $k+1$ elements of $\overrightarrow{\tau}$.

Thus we obtain that $\overrightarrow{\tau} \subseteq \overrightarrow{f}$, $\overrightarrow{\tau}$ is $\overrightarrow{m}$-coding and $\overrightarrow{\tau} \Vdash F_e^{k+1}(x)$.

For the opposite direction, suppose that there is an $\overrightarrow{m} \preccurlyeq \overrightarrow{\omega}$ and a $\overrightarrow{\tau} \subseteq \overrightarrow{f}$ such that $\overrightarrow{\tau}$ is $\overrightarrow{m}$-coding and $\overrightarrow{\tau} \Vdash_{\overrightarrow{m}} F_e^{k+1}(x)$. Then we have, that there are u, u_1 and u_2 such that

(1) $\langle x, u\rangle \in W_e$;
(2) $D_u = D_{u_1} \oplus D_{u_2}$;
(3) for all $v \in D_{u_1}$ either $v = 2\langle e_v, x_v\rangle$ and $\overrightarrow{\tau} \Vdash_{\overrightarrow{m}} F_{e_v}^k(x_v)$, or $v = 2\langle e_v, x_v\rangle + 1$ and $\overrightarrow{\tau} \Vdash_{\overrightarrow{m}} \neg F_{e_v}^k(x_v)$.
(4) $D_{u_2} \subseteq \operatorname{Graph} \tau_{k+1}$.

Applying the induction hypothesis to (3) we obtain

(1) $\langle x, u\rangle \in W_e$;
(2) $D_u = D_{u_1} \oplus D_{u_2}$;

(3) for all $v \in D_{u_1}$ either $v = 2\langle e_v, x_v\rangle$ and $\overrightarrow{f} \models F^k_{e_v}(x_v)$, or $v = 2\langle e_v, x_v\rangle + 1$ and $\overrightarrow{f} \models \neg F^k_{e_v}(x_v)$;

(4) $D_{u_2} \subseteq \operatorname{Graph}\tau_{k+1} \subseteq f_{k+1}$,

which is exactly what we have to prove. This concludes the proof of the positive equivalence for $k+1$. The proof of the negative equivalence is analogous to that for $k = 0$. $\square$

Claim 3. $\mathcal{R} \not\leq_u \overrightarrow{f}$.

Proof. In order to obtain a contradiction assume that $\mathcal{R} \leq_u \overrightarrow{f}$. Then the sequence $\{f^{0^{-1}}(R_n)\}_{n<\omega}$ is also uniformly reducible to $\overrightarrow{f}$. Therefore there is a recursive function g_s such that $f^{0^{-1}}(R_n) = W_{g_s(n)}(P^n(\overrightarrow{f}))$ for each n, which means

$$x \in f^{0^{-1}}(R_n) \iff \overrightarrow{f} \models F^n_{g_s(n)}(x)$$

Now consider the third substep of the $s+1$-st step. There we have constructed a finite part $\overrightarrow{\tau}_{s+1}$ extending the finite part $\overrightarrow{\sigma}_s$ such that for some i

$$\overrightarrow{\tau}_{s+1} \Vdash_{\overrightarrow{s}} (\neg)F^i_{g_s(i)}(\operatorname{lh}\sigma^0_s) \iff (\neg)(\tau^0_{s+1}(\operatorname{lh}\sigma^0_s) \notin R_i).$$

Now using the Truth Lemma and that $\tau^0_{s+1} \subseteq f^0$ we obtain that

$$\operatorname{lh}\sigma^0_s \in f^{0^{-1}}(R_i) \Rightarrow f^0(\operatorname{lh}\sigma^0_s) \in R_i \Rightarrow \neg(f^0(\operatorname{lh}\sigma^0_s) \notin R_i) \Rightarrow \overrightarrow{f} \models \neg F^i_{g_s(i)}(\operatorname{lh}\sigma^0_s);$$

$$\operatorname{lh}\sigma^0_s \notin f^{0^{-1}}(R_i) \Rightarrow f^0(\operatorname{lh}\sigma^0_s) \notin R_i \Rightarrow \overrightarrow{f} \models F^i_{g_s(i)}(\operatorname{lh}\sigma^0_s),$$

which is a contradiction with $x \in f^{0^{-1}(R_n)} \iff \overrightarrow{f} \models F^n_{g_s(n)}(x)$. This means that the assumption $\mathcal{R} \leq_u \overrightarrow{f}$ leads to a contradiction and therefore we have that $\mathcal{R} \not\leq_u \overrightarrow{f}$. $\square$

Claim 1 and Claim 3 are exactly the properties of $\overrightarrow{f}$ that we had to prove in order to prove Theorem 3 and so this concludes its proof. $\square$

We are ready to prove Theorem 1.

Proof of Theorem 1. Let $\mathbf{a}_1 \leq_\omega \mathbf{a}_2 \leq_\omega \cdots \leq_\omega \mathbf{a}_n \leq_\omega \cdots \leq_\omega \mathbf{g}$ be an infinite increasing sequence of ω-enumeration degrees. Fix an increasing sequence $\mathcal{A}_1 \leq_u \mathcal{A}_2 \leq_u \cdots \leq_u \mathcal{A}_n \leq_u \cdots \leq_u \mathcal{G}$ of elements of $\mathcal{S}$ such that $\mathbf{a}_n = d_\omega(\mathcal{A}_n)$ and $\mathbf{g} = d_\omega(\mathcal{G})$. Since there are only countably many sequences u-reducible to $\mathcal{G}$ then the set $\{\mathcal{X} \in \mathcal{S} \mid \mathcal{X} \leq_u \mathcal{G}\ \&\ \forall n(\mathcal{X} \not\leq_u \mathcal{A}_n)\}$ is denumerable. Let us order its elements into the sequence $\{\mathcal{R}_k\}_{k<\omega}$. Now it is clear that if we take an $\mathcal{F}^* \in \mathcal{S}$ as in Theorem 3, then $d_\omega(\mathcal{F}^*)$ and $d_\omega(\mathcal{G})$ will be an exact pair for the sequence $\{\mathbf{a}_n\}_{n<\omega}$.

Now in order to obtain an $\mathbf{f} \in \mathcal{D}_1$, such that $\mathbf{g}, \mathbf{f}$ form an exact pair for the sequence $\{\mathbf{a}_k\}_{k<\omega}$, we have to use the omitting theorem from [3]:

Theorem 4 (Soskov, Kovachev). *Let $\mathcal{A} \in \mathcal{S}$ and let $\{\mathcal{R}_k\}_{k<\omega}$ be a sequence of elements of $\mathcal{S}$, such that $\mathcal{R}_k \not\leq_u \mathcal{A}$. Then there is a sequence $\mathcal{F} = (F, \emptyset, \emptyset, \ldots, \emptyset, \ldots)$, such that $\mathcal{A} \leq_u \mathcal{F}$ and $\mathcal{R}_k \not\leq_u \mathcal{F}$, for all k.*

Therefore we can take a sequence $\mathcal{F} = (F, \emptyset, \emptyset, \dots, \emptyset, \dots)$, such that $\mathcal{F}^* \leq_u \mathcal{F}$ and $\mathcal{F}$ omits all the sequences which are under $\mathcal{G}$, but are not under $\mathcal{F}^*$. Now set $\mathbf{f} = d_\omega(\mathcal{F})$. It is clear that $\mathbf{f} \in \mathcal{D}_1$ and that $\mathbf{f}$ and $\mathbf{g}$ form an exact pair for the sequence $\{\mathbf{a}_n\}_{n<\omega}$.

For the last part of the theorem is sufficient to prove the following

Proposition 3. *Let $\mathbf{f}, \mathbf{g} \in \mathcal{D}_\omega$ form an exact pair for the increasing sequence $\{\mathbf{a}_n\}_{n<\omega}$ of ω-enumeration degrees. Then $\mathbf{f}'$ and $\mathbf{g}'$ form an exact pair for the sequence $\{\mathbf{a}'_n\}_{n<\omega}$*

Proof. Let $\mathbf{f} = d_\omega(\mathcal{F})$ and $\mathbf{g} = d_\omega(\mathcal{G})$. Then from the definition of the jump operation we have that

$$\mathbf{f}' = d_\omega(\{P^{n+1}(\mathcal{F})\}_{n<\omega}), \quad \mathbf{g}' = d_\omega(\{P^{n+1}(\mathcal{G})\}_{n<\omega})$$

Now suppose that $\mathbf{x} \leq_\omega \mathbf{f}', \mathbf{g}'$ and that $\mathbf{x}$ is the ω-enumeration degree generated by the sequence $(X^1, X^2, \dots, X^k, \dots)$. Since

$$(X^1, X^2, \dots, X^k, \dots) \leq_u \{P^{n+1}(\mathcal{F})\}_{n<\omega},$$

$$(X^1, X^2, \dots, X^k, \dots) \leq_u \{P^{n+1}(\mathcal{G})\}_{n<\omega}$$

we obtain that the sequence $\mathcal{X} = (\emptyset, X^1, X^2, \dots, X^k, \dots)$ is uniformly reducible to the sequences $\mathcal{F}$ and $\mathcal{G}$. Therefore $\mathcal{X} \leq \mathcal{A}_n$, for some n, i.e.,

$$(\emptyset, X^1, X^2, \dots, X^k, \dots) \leq_u (P^0(\mathcal{A}_n), P^1(\mathcal{A}_n), P^2(\mathcal{A}_n), \dots, P^k(\mathcal{A}_n), \dots).$$

Therefore

$$(X^1, X^2, \dots, X^k, \dots) \leq_u (P^1(\mathcal{A}_n), P^2(\mathcal{A}_n), \dots, P^k(\mathcal{A}_n), \dots),$$

which in the terms of the ω-enumeration degrees means that $\mathbf{x} \leq_\omega \mathbf{a}_\mathrm{n}$. $\square$

References

1. Case, J.: Enumeration reducibility and partial degrees. Ann.Math.Log 2, 419–439 (1971)
2. Soskov, I.N.: The ω-enumereation degrees (To appear)
3. Soskov, I.N., Kovachev, B.: Uniform regular enumerations. Math.Struct. in Comp.Sci. 16, 901–924 (2006)
4. Spector, C.: On degrees of recursive unsolvability. Ann. of Math.(2) 64, 581–592 (1956)

Operational Semantics for Positive Relevant Logics Without Distribution

Ying Gao and Jingde Cheng

Department of Information and Computer Sciences, Saitama University,
255 Shimo-Okubo, Sakura-ku, Saitama, 338-8570, Japan
{gaoying,cheng}@aise.ics.saitama-u.ac.jp

Abstract. This paper investigates operational semantics for various positive relevant logics without distribution after the work of Kit Fine in *Models for Entailment*. To invalidate the law of distribution of conjunction over disjunction, we use different types of states to model conjunction and disjunction, respectively. The implication $\rightarrow$ is interpreted by three operations $\oplus, \otimes, \ominus$, instead of one operation '$\cdot$' as in Fine's work.

Keywords: Relevant logics, Operational semantics.

1 Introduction

For relevant logics, there are two kinds of non-algebraic semantics, which are both developed based on Urquhart's semilattice semantics, that is, Routley and Meyer's relational semantics (see [11]), and Kit Fine's operational semantics (see [6]). Routley-Meyer theory uses a ternary relation to interpret implication, and hence can evaluate disjunction by the standard clause. For Fine, implication is modeled using a binary operation, and disjunction is interpreted by use of negation. Actually, these two methods are highly related, that is, the ternary relation can be constructed canonically from a binary operation on sets of formulas; and an operational model can derive an equivalent relational model easily. Since a relation is more flexible than an operation, the ternary relational semantics is used in more situations. But, the relational method will also cause the collapse of some properties of canonical models, which can be revealed by operations. This will make it difficult to generalize Routley and Meyer's semantics to logics without distribution.

Illuminated by Fine's work and in order to overcome this deficiency, we investigate operational semantics for relevant logics without distribution. We use different types of states to model conjunction and disjunction, respectively. Hence both conjunction and disjunction can be modeled by the standard clauses. Canonically, these states are defined as theories and anti-counter-theories. As for implication $\rightarrow$, in the canonical construction of Fine's model, prime theories play an important role such that only one operation '$\cdot$' is sufficient to interpret $\rightarrow$ in relevant logics with distribution. But in the non-distributive case, not every theory can be primed into a prime one as in the distributive case. We show that besides Fine's operation (we denote it as $\oplus$), there are two other binary operations $\otimes$ and $\ominus$, which should be used together to model $\rightarrow$. These operations, $\oplus$, $\otimes$, $\ominus$, are defined canonically on theories and anti-counter-theories.

We are aware that Kripke-style semantics for non-distributive substructural logics, both of a relational and of an operational kind, have been proposed. For example, Girard (see [7]) treat disjunction in a way to rely on the presence of negation. Allwein and Dunn (see [1]) extended Urquhart's representation for lattices and provided a 3-valued relational semantics for linear logic. Hartonas (see [8]) represented general lattices as systems with closure operators, and gave a relational semantics for logics without distribution. In addition, non-standard semantical treatments of disjunction can be found in [3,4,9,10], where implication $\rightarrow$ was modeled by an operation similar to Fine's. Our approach is quite different from these above semantics in that we use two sets of states to invalidate distribution, and make essential use of three operations to evaluate $\rightarrow$ in different types of states.

2 Positive Relevant Logics Without Distribution

We begin with a basic system LB^+[1], which is Meyer-Routley minimal relevant logic B^+ (see [11]) subtracting the law of distribution.

Axioms

A1 $A \rightarrow A$

A2 $A \rightarrow A \vee B$, $B \rightarrow A \vee B$

A3 $A \wedge B \rightarrow A$, $A \wedge B \rightarrow B$

A4 $(A \rightarrow B) \wedge (A \rightarrow C) \rightarrow (A \rightarrow B \wedge C)$

A5 $(A \rightarrow C) \wedge (B \rightarrow C) \rightarrow (A \vee B \rightarrow C)$

Rules

R1 $A, A \rightarrow B \Rightarrow B$ (Modus Ponens)

R2 $A, B \Rightarrow A \wedge B$ (Adjunction)

R3 $A \rightarrow B, C \rightarrow D \Rightarrow (B \rightarrow C) \rightarrow (A \rightarrow D)$ (Affixing).

The following axiom and rule schemes **C1-C7** can be added to LB^+ to obtain various positive relevant logics without distribution.

C1 $(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$

C2 $(A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C))$

C3 $(A \rightarrow B) \rightarrow ((C \rightarrow A) \rightarrow (C \rightarrow B))$

C4 $A \wedge (A \rightarrow B) \rightarrow B$

C5 $A \rightarrow ((A \rightarrow B) \rightarrow B)$

C6 $(A \rightarrow (B \rightarrow C)) \rightarrow (A \wedge B \rightarrow C)$

C7 $A \Rightarrow (A \rightarrow B) \rightarrow B$

Given a logical system L, use $\vdash_L C$ to denote that C is a theorem of L. If L is obvious, the subscript '$_L$' will be omitted.

[1] The notation 'LB^+' is after 'LR' for the logic R without distribution.

3 Basic Frame and Model

A *basic frame* is a triple $(T_1, T_2, \leq)$, where, T_1, T_2 are non-empty sets; $\leq$ is a reflexive, transitive and antisymmetric relation on $T_1 \cup T_2$. A *basic model* is a 4-tuple $(T_1, T_2, \leq, \models)$, where, $(T_1, T_2, \leq)$ is a basic frame; $\models$ is a binary relation from $T_1 \cup T_2$ to the set Sl (sentence letters), such that the following condition and rules hold: $\forall t_1 \in T_1, t_2 \in T_2, p \in Sl$,

Atomic Hereditary Condition
- $t_1 \models p \Leftrightarrow \forall x \in T_2, (t_1 \leq x \Rightarrow x \models p)$,
- $t_2 \models p \Leftrightarrow \exists x \in T_1, (x \leq t_2 \text{ and } x \models p)$.

Evaluation Rules for $\wedge, \vee$

$(\wedge_1)$ $t_1 \models A \wedge B \Leftrightarrow t_1 \models A$ and $t_1 \models B$,
$(\wedge_2)$ $t_2 \models A \wedge B \Leftrightarrow \exists x \in T_1, (x \leq t_2 \text{ and } x \models A \wedge B)$;
$(\vee_1)$ $t_1 \models A \vee B \Leftrightarrow \forall x \in T_2, (t_1 \leq x \Rightarrow x \models A \vee B)$,
$(\vee_2)$ $t_2 \models A \vee B \Leftrightarrow t_2 \models A$ or $t_2 \models B$.

Lemma 1 (Hereditary Condition for $\wedge, \vee$**).** *$\forall t_1 \in T_1, t_2 \in T_2$:*
(1) $t_1 \models C \Leftrightarrow \forall x \in T_2, (t_1 \leq x \Rightarrow x \models C)$,
(2) $t_2 \models C \Leftrightarrow \exists x \in T_1, (x \leq t_2$ and $x \models C)$.

Proof. By induction on the construction of C with **Atomic Hereditary Condition** as the basis. For each of $\wedge, \vee$, it is enough to prove only one direction of (1) or (2).

For $\wedge$, to show right-to-left of (1). Suppose $t_1 \in T_1$ and $t_1 \not\models A \wedge B$, then $t_1 \not\models A$ or $t_1 \not\models B$. By induction hypothesis, $(\exists x \in T_2, t_1 \leq x$ and $x \not\models A)$ or $(\exists x \in T_2, t_1 \leq x$ and $x \not\models B)$, i.e., $\exists x \in T_2, t_1 \leq x, (x \not\models A$ or $x \not\models B)$. If $x \models A \wedge B$, then by evaluation rules $\exists y \in T_1, y \leq x$ and $y \models A \wedge B$, i.e., $y \models A$ and $y \models B$. By induction hypothesis, $x \models A$ and $x \models B$, giving a contradiction. Hence, $x \not\models A \wedge B$.

For $\vee$, to show left-to-right of (2). Suppose $t_2 \in T_2$ and $t_2 \models A \vee B$. Then, $t_2 \models A$ or $t_2 \models B$. By induction hypothesis, $(\exists x \in T_1, x \leq t_2$ and $x \models A)$ or $(\exists x \in T_1, x \leq t_2$ and $x \models B)$. Suppose the former case, by induction hypothesis, $\forall y \in T_2, x \leq y \Rightarrow y \models A$. So, $\forall y \in T_2, x \leq y \Rightarrow y \models A \vee B$. Hence, $x \models A \vee B$ by evaluation rules. □

4 Frame and Model for LB^+

A *frame* for LB^+ is a 7-tuple $(T_1, T_2, \leq, l, \oplus, \otimes, \ominus)$ such that:

(1) $(T_1, T_2, \leq)$ is a basic frame;
(2) $\oplus$ is a binary operation, $\oplus : T_1 \times T_1 \to T_1$, which satisfies,
- $\forall u \in T_1$, $u \oplus l = u$;

(3) $\otimes$ is a binary operation, $\otimes : T_1 \times T_2 \to T_2$, which satisfies,
- $\forall v \in T_2$, $l \otimes v = v$.

(4) $\ominus$ is a binary operation, $\ominus : T_1 \times T_2 \to T_2$, which satisfies:
- $\forall u \in T_1, v \in T_2, l \leq u \ominus v \Leftrightarrow u \leq v$;

(5) $\forall t, u \in T_1, \forall v \in T_2, (u \oplus t \leq v \Leftrightarrow u \leq t \otimes v \Leftrightarrow t \leq u \ominus v)$.

A *model* for LB^+ is a 8-tuple $(T_1, T_2, \leq, l, \oplus, \otimes, \ominus, \models)$, where, $(T_1, T_2, \leq, l, \oplus, \otimes, \ominus)$ is a frame; $(T_1, T_2, \leq, \models)$ is a basic model; $\models$ satisfies $\rightarrow_2$, and only one of $\rightarrow_\oplus, \rightarrow_\otimes, \rightarrow_\ominus$: $\forall t_1 \in T_1, t_2 \in T_2$,

Evaluation Rules for $\rightarrow$

$(\rightarrow_\oplus)$ $t_1 \models A \rightarrow B \Leftrightarrow \forall u \in T_1, (u \models A \Rightarrow u \oplus t_1 \models B)$[2],

$(\rightarrow_\otimes)$ $t_1 \models A \rightarrow B \Leftrightarrow \forall v \in T_2, (t_1 \otimes v \models A \Rightarrow v \models B)$[3],

$(\rightarrow_\ominus)$ $t_1 \models A \rightarrow B \Leftrightarrow \forall u \in T_1, v \in T_2, (t_1 \leq u \ominus v$ and $u \models A \Rightarrow v \models B)$,

$(\rightarrow_2)$ $t_2 \models A \rightarrow B \Leftrightarrow \exists x \in T_1, (x \leq t_2$ and $x \models A \rightarrow B)$.

Given a formula C, C is *valid in a model* $(T_1, T_2, \leq, l, \oplus, \otimes, \ominus, \models)$ iff $l \models C$; C is *valid in a frame* $(T_1, T_2, \leq, l, \oplus, \otimes, \ominus)$ iff C is valid in every model based on this frame; C is *valid*, denoted as $\models C$, iff C is valid in every frame.

Though a model for LB^+ only satisfies one of $\rightarrow_\oplus, \rightarrow_\otimes, \rightarrow_\ominus$, the following propositions show that from any of these rules, we can deduce the others. So, in a model for LB^+, we can interpret implication by all of these rules. This feature is important for LB^+ and its extensions.

Lemma 2 (Hereditary Condition). *$\forall t_1 \in T_1, t_2 \in T_2$:*

(1) $t_1 \models C \Leftrightarrow \forall x \in T_2, (t_1 \leq x \Rightarrow x \models C)$;

(2) $t_2 \models C \Leftrightarrow \exists x \in T_1, (x \leq t_2$ and $x \models C)$.

Proof. By induction on the construction of C with **Atomic Hereditary Condition** as the basis. The induction for $\wedge, \vee$ has been established. Here it suffices to prove right-to-left of (1) for $\rightarrow$. We give proofs by each of $\rightarrow_\oplus, \rightarrow_\otimes, \rightarrow_\ominus$.

By $\rightarrow_\oplus$. First, we prove that $\forall u \in T_1, v \in T_2, (u \models A$ and $v \not\models B \Rightarrow u \ominus v \not\models A \rightarrow B)$. Let $t \in T_1$ and $t \leq u \ominus v$, to show $t \not\models A \rightarrow B$. Since $t \leq u \ominus v$, $u \oplus t \leq v$. By induction hypothesis, $u \oplus t \not\models B$. So, $t \not\models A \rightarrow B$. Since t is arbitrary, $u \ominus v \not\models A \rightarrow B$.

Second, suppose $t \in T_1$ and $t \not\models A \rightarrow B$. So, $\exists u \in T_1$, $u \models A$ and $u \oplus t \not\models B$. By induction hypothesis, $\exists v \in T_2$, $u \oplus t \leq v$ and $v \not\models B$. Hence, $u \ominus v \not\models A \rightarrow B$. Since $u \oplus t \leq v$, $t \leq u \ominus v \in T_2$. Hence, we get the result.

By $\rightarrow_\otimes$. First, we prove that $\forall u \in T_1, v \in T_2, (u \models A$ and $v \not\models B \Rightarrow u \ominus v \not\models A \rightarrow B)$. Let $t \in T_1$ and $t \leq u \ominus v$, to show $t \not\models A \rightarrow B$. Since $t \leq u \ominus v$, $u \leq t \otimes v$. By induction hypothesis, $t \otimes v \models A$. So, $t \not\models A \rightarrow B$. Since t is arbitrary, $u \ominus v \not\models A \rightarrow B$.

Second, suppose $t \in T_1$ and $t \not\models A \rightarrow B$. So, $\exists v \in T_2$, $t \otimes v \models A$ and $v \not\models B$. By induction hypothesis, $\exists u \in T_1$, $u \leq t \otimes v$ and $u \models A$. Hence, $u \ominus v \not\models A \rightarrow B$. Since $u \leq t \otimes v$, $t \leq u \ominus v \in T_2$. Hence, we get the result.

By $\rightarrow_\ominus$. Suppose $t \in T_1$ and $t \not\models A \rightarrow B$. So, $\exists u \in T_1, v \in T_2$, $t \leq u \ominus v$ and $u \models A$, but $v \not\models B$. Then, $u \ominus v \not\models A \rightarrow B$. Otherwise, $\exists x \in T_1, x \leq u \ominus v$ and $x \models A \rightarrow B$. But, $u \models A$ and $v \not\models B$, giving a contradiction. So, $u \ominus v \not\models A \rightarrow B$. Hence, since $t \leq u \ominus v$, we get the result. □

Corollary 1. *$\forall t, t' \in T_1 \cup T_2, t \leq t'$ and $t \models C \Rightarrow t' \models C$.*

[2] Fine used '$\cdot$' to denote $\oplus$. His evaluation rule is $t \models A \rightarrow B \Leftrightarrow \forall u(u \models A \Rightarrow t \cdot u \models B)$.

[3] This evaluation rule is given in [2] (Ch. 8).

Proof. We consider $t \in T_2$ and $t' \in T_1$. Since $t \models C$, $\exists u \in T_1$, $u \leq t$ and $u \models C$. Let $v \in T_2$ and $t' \leq v$, then $u \leq v$. So, $v \models C$. Since v is arbitrary, $t' \models C$. Other cases are similar. □

Corollary 2. *The evaluation rules* $\rightarrow_\oplus$, $\rightarrow_\otimes$, $\rightarrow_\ominus$ *are equivalent.*

Proof. $\rightarrow_\oplus \Rightarrow \rightarrow_\otimes$. First suppose $t \models A \rightarrow B$, $v \in T_2$ and $t \otimes v \models A$, to show $v \models B$. So, $\exists u \in T_1$, $u \leq t \otimes v$ and $u \models A$. Then, $u \oplus t \leq v$ and $u \oplus t \models B$. Hence, $v \models B$. Conversely, suppose $t \not\models A \rightarrow B$, then $\exists u \in T_1$, $u \models A$, but $u \oplus t \not\models B$. So, $\exists v \in T_2$, $u \oplus t \leq v$ and $v \not\models B$. Then, $u \leq t \otimes v$. So, $t \otimes v \models A$.

$\rightarrow_\otimes \Rightarrow \rightarrow_\ominus$. First suppose $t \models A \rightarrow B$, $u \in T_1, v \in T_2, t \leq u \ominus v$ and $u \models A$, to show $v \models B$. Then, $u \leq t \otimes v$. So, $t \otimes v \models A$. Hence, $v \models B$ as required. Conversely, suppose $t \not\models A \rightarrow B$, then $\exists v \in T_2, t \otimes v \models A$, but $v \not\models B$. So, $\exists u \in T_1, u \leq t \otimes v$ and $u \models A$. So, $t \leq u \ominus v$.

$\rightarrow_\ominus \Rightarrow \rightarrow_\oplus$. First suppose $t \models A \rightarrow B$, $u \in T_1$ and $u \models A$, to show $u \oplus t \models B$. Let $v \in T_2$ and $u \oplus t \leq v$, then $t \leq u \ominus v$. So, $v \models B$. Since v is arbitrary, $u \oplus t \models B$. Conversely, suppose $t \not\models A \rightarrow B$, then $\exists u \in T_1, v \in T_2$, $t \leq u \ominus v$ and $u \models A$, but $v \not\models B$. So, $u \oplus t \leq v$. So, $u \oplus t \not\models B$. □

Corollary 3. $l \models A \rightarrow B \Leftrightarrow \forall u \in T_1, (u \models A \Rightarrow u \models B)$.

Proof. This is obvious by $u \oplus l = u$. □

Theorem 1 (Soundness for LB^+). *If* $\vdash C$*, then* $\models C$.

Proof. We give proofs for **A4**,**A5**. For **A4**, suppose $t \in T_1$ and $t \models (A \rightarrow B) \wedge (A \rightarrow C)$. So, $t \models A \rightarrow B$ and $t \models A \rightarrow C$. Suppose further $u \in T_1$ and $u \models A$, then $u \oplus t \models B$ and $u \oplus t \models C$. Hence, $u \oplus t \models B \wedge C$. For **A5**, suppose $t \in T_1$ and $t \models (A \rightarrow C) \wedge (B \rightarrow C)$. So, $t \models A \rightarrow C$ and $t \models B \rightarrow C$. Suppose further $v \in T_2$ and $t \otimes v \models A \vee B$, then $t \otimes v \models A$ or $t \otimes v \models B$. Hence, $v \models C$. □

Now, we give a counter-model for the distribution axiom. Let $M = (T_1, T_2, \leq, l, \oplus, \otimes, \ominus, \models)$, such that:

(1) $T_1 = \{t\}$, $T_2 = \{u, v\}$;
(2) $t \leq u$, $t \leq v$;
(3) $t \models p$, $u \models p, q$, $v \models p, r$.

It is easy to see that M satisfies **Atomic Hereditary Condition**. So, M is a model for LB^+. We show $l \not\models p \wedge (q \vee r) \rightarrow (p \wedge q) \vee r$. By evaluation rules, $u \models q \vee r$ and $v \models q \vee r$. So, $t \models q \vee r$, and then $t \models p \wedge (q \vee r)$. But $t \not\models p \wedge q$. So, $u \not\models p \wedge q$. Since $u \not\models r$, $u \not\models (p \wedge q) \vee r$. Then, $t \not\models (p \wedge q) \vee r$. Hence, $l \not\models p \wedge (q \vee r) \rightarrow (p \wedge q) \vee r$ by **Corollary 3**.

5 Canonical Models and Completeness

5.1 Definitions of a Theory and an Anti-Counter-Theory

We give the following definitions for a logical system L, which is LB^+ or one of its extensions. A *theory* is a set of formulas χ closed under provable L-implication and conjunction, i.e., if $\vdash_L A \to B$ and $A \in \chi$, then $B \in \chi$; and if $A, B \in \chi$, then $A \wedge B \in \chi$. A *counter-theory*[4] (ab. *c-theory*) χ is defined dually, requiring that: if $\vdash_L A \to B$ and $B \in \chi$, then $A \in \chi$; and if $A, B \in \chi$, then $A \vee B \in \chi$. Let Σ be the set of all formulas and $\chi \subseteq \Sigma$, call $\Sigma - \chi$ the complement of χ, and denote it as $-\chi$. The complement of a c-theory is called an *anti-counter-theory* (ab. *a-c-theory*). It is easy to show that χ is an a-c-theory iff it satisfies: if $\vdash_L A \to B$ and $A \in \chi$, then $B \in \chi$; if $A \vee B \in \chi$, then $A \in \chi$ or $B \in \chi$. It is easy to show that for every theory χ, $A \wedge B \in \chi$ iff $A \in \chi$ and $B \in \chi$; for every a-c-theory χ, $A \vee B \in \chi$ iff $A \in \chi$ or $B \in \chi$.

For a logical system L, the set of all theories is denoted as $Th(L)$, and the set of all a-c-theories as $aTh(L)$. We denote the set of all theorems of L as $l(L)$, or simply l. Then it is obvious that $l \in Th(L)$.

5.2 Canonical Definitions of Operations $\oplus, \otimes, \ominus$

For a logical system L, define three operations on sets of formulas, $t, u, v \subseteq \Sigma$:

(1) $u \oplus t = \{B : \exists A, A \to B \in t \text{ and } A \in u\}$,
(2) $t \otimes v = \{A : \forall B, A \to B \in t \Rightarrow B \in v\}$[5],
(3) $u \ominus v = \{C : \forall A, B, A \in u \text{ and } \vdash_L C \to (A \to B) \Rightarrow B \in v\}$.

Lemma 3. *If $t, u \in Th(L), v \in aTh(L)$, then $u \oplus t \in Th(L)$, $t \otimes v, u \ominus v \in aTh(L)$.*

Proof. We show $u \ominus v \in aTh(L)$ as an example. First, suppose $\vdash_L C \to C'$ and $C' \notin u \ominus v$, i.e., $\exists A \in u, B \notin v$, $\vdash_L C' \to (A \to B)$. So, $\vdash_L C \to (A \to B)$. Hence $C \notin u \ominus v$. Second, suppose $C_1, C_2 \notin u \ominus v$, then $\exists A_1, A_2 \in u, B_1, B_2 \notin v$, $\vdash_L C_1 \to (A_1 \to B_1)$ and $\vdash_L C_2 \to (A_2 \to B_2)$. Since $\vdash_L A_1 \wedge A_2 \to A_1$, $\vdash_L (A_1 \to B_1) \to (A_1 \wedge A_2 \to B_1)$. Since $\vdash_L B_1 \to B_1 \vee B_2$, $\vdash_L (A_1 \wedge A_2 \to B_1) \to (A_1 \wedge A_2 \to B_1 \vee B_2)$. Then $\vdash_L C_1 \to (A_1 \wedge A_2 \to B_1 \vee B_2)$. Similarly, $\vdash_L C_2 \to (A_1 \wedge A_2 \to B_1 \vee B_2)$. So, $\vdash_L C_1 \vee C_2 \to (A_1 \wedge A_2 \to B_1 \vee B_2)$. Since $A_1 \wedge A_2 \in u$ and $B_1 \vee B_2 \notin v$, $C_1 \vee C_2 \notin u \ominus v$. □

5.3 Properties of Operations $\oplus, \otimes, \ominus$

Lemma 4. *$\forall t, u, v \in Th(L) \cup aTh(L)$, $u \oplus t \subseteq v \Leftrightarrow u \subseteq t \otimes v \Leftrightarrow t \subseteq u \ominus v$.*

Proof. For $u \oplus t \subseteq v \Rightarrow u \subseteq t \otimes v$, suppose $A \in u$ and $A \notin t \otimes v$, then $\exists B \notin v$, $A \to B \in t$. So, $B \notin u \oplus t$. But since $A \in u$, $B \in u \oplus t$, giving a contradiction.

For $u \subseteq t \otimes v \Rightarrow t \subseteq u \ominus v$, suppose $C \in t$ and $C \notin u \ominus v$, then $\exists A \in u, B \notin v$, $\vdash_L C \to (A \to B)$. So, $A \in t \otimes v$ and $A \to B \in t$. Then $B \in v$, giving a contradiction.

For $t \subseteq u \ominus v \Rightarrow u \oplus t \subseteq v$, suppose $B \in u \oplus t$, then $\exists A \in u$, $A \to B \in t$. So, $A \to B \in u \ominus v$. Since $\vdash_L (A \to B) \to (A \to B)$, $B \in v$. □

[4] This name comes from Dunn [5].
[5] This canonical definition is given in [2] (Ch. 8).

Lemma 5. *$\forall t, u, v \in Th(L) \cup aTh(L)$, (1) $u \oplus l = u$; (2) $l \otimes v = v$; (3) $l \subseteq u \ominus v \Leftrightarrow u \subseteq v$.*

Proof. The proof is by definitions of $\oplus, \otimes, \ominus$, and properties of theories and a-c-theories. We show (3) as an example. Suppose $l \subseteq u \ominus v$ and $A \in u$. Since $A \to A \in l$, $A \to A \in u \ominus v$. By $\vdash_L (A \to A) \to (A \to A)$, $A \in v$ as required. Conversely, suppose $C \in l$ and $C \notin u \ominus v$. Then $\exists A \in u, B \notin v, \vdash_L C \to (A \to B)$. So, $A \to B \in l$, i.e., $\vdash_L A \to B$. Then, $B \in u$. Since $u \subseteq v$, $B \in v$, giving a contradiction. □

5.4 Canonical Hereditary Condition and Evaluation Rules

Lemma 6. *$\forall t_1 \in Th(L), t_2 \in aTh(L)$,*
(1) $C \in t_1 \Leftrightarrow \forall x \in aTh(L), (t_1 \subseteq x \Rightarrow C \in x)$,
(2) $C \in t_2 \Leftrightarrow \exists x \in Th(L), (x \subseteq t_2$ and $C \in x)$.

Proof. It suffices to show right-to-left of (1) and left-to-right of (2).

For (1), suppose $t_1 \in Th(L)$ and $C \notin t_1$, let $x = -\{C' :\vdash_L C' \to C\}$. Then $C \notin x$. Also, $t_1 \subseteq x$, otherwise, $\exists C' \in t_1$ and $C' \notin x$, i.e., $\vdash_L C' \to C$, then $C \in t_1$, giving a contradiction. We show $x \in aTh(L)$, i.e., $-x = \{C' :\vdash_L C' \to C\}$ is a c-theory. First, suppose $\vdash_L C_1 \to C_2$ and $C_2 \in -x$. So, $\vdash_L C_2 \to C$. Then, $\vdash_L C_1 \to C$. Hence $C_1 \in -x$. Second, suppose $C_1, C_2 \in -x$. Then $\vdash_L C_1 \to C$ and $\vdash_L C_2 \to C$. So, $\vdash_L C_1 \vee C_2 \to C$. Hence, $C_1 \vee C_2 \in -x$.

For (2), suppose $t_2 \in aTh(L)$ and $C \in t_2$, let $x = \{C' :\vdash_L C \to C'\}$. Then $C \in x$. Also, $x \subseteq t_2$, since $C \in t_2$. We show $x \in Th(L)$. First, suppose $\vdash_L C_1 \to C_2$ and $C_1 \in x$, then $\vdash_L C \to C_1$. So, $\vdash_L C \to C_2$. Hence $C_2 \in x$. Second, suppose $C_1, C_2 \in x$, then $\vdash_L C \to C_1$ and $\vdash_L C \to C_2$. So, $\vdash_L C \to C_1 \wedge C_2$. Hence, $C_1 \wedge C_2 \in x$. □

Lemma 7. *$\forall t \in Th(L)$,*
(1) $A \to B \in t \Leftrightarrow \forall u \in Th(L)$, $A \in u \Rightarrow B \in u \oplus t$,
(2) $A \to B \in t \Leftrightarrow \forall v \in aTh(L)$, $A \in t \otimes v \Rightarrow B \in v$,
(3) $A \to B \in t \Leftrightarrow \forall u \in Th(L), v \in aTh(L), t \subseteq u \ominus v$ and $A \in u \Rightarrow B \in v$.

Proof. For each equivalence, it suffices to show right-to-left. The other direction can be obtained by definitions of $\oplus, \otimes, \ominus$.

For (1), suppose $A \to B \notin t$, let $u = \{A' :\vdash_L A \to A'\}$, then $A \in u \in Th(L)$. If $B \in u \oplus t$, then $\exists A' \in u$ and $A' \to B \in t$. Then, $\vdash_L (A' \to B) \to (A \to B)$. So, $A \to B \in t$, giving a contradiction. Hence, $B \notin u \oplus t$.

For (2), suppose $A \to B \notin t$, let $v = -\{B' :\vdash_L B' \to B\}$, then $B \notin v \in aTh(L)$. If $A \notin t \otimes v$, then $\exists B' \notin v$ and $A \to B' \in t$. Then, $\vdash_L B' \to B$. So, $\vdash_L (A \to B') \to (A \to B)$. So, $A \to B \in t$, giving a contradiction. Hence, $A \in t \otimes v$.

For (3), suppose $A \to B \notin t$, let $u = \{A' :\vdash_L A \to A'\}$ and $v = -\{B' :\vdash_L B' \to B\}$. Then, $A \in u \in Th(L)$ and $B \notin v \in aTh(L)$. If $t \not\subseteq u \ominus v$, then $\exists C \in t$, but $C \notin u \ominus v$. So, $\exists A' \in u, B' \notin v, \vdash_L C \to (A' \to B')$. So, $\vdash_L (A' \to B') \to (A \to B)$. So, $\vdash_L C \to (A \to B)$. So, $A \to B \in t$, giving a contradiction. Hence, $t \subseteq u \ominus v$. □

5.5 Canonical Models and Completeness

For a logical system L, let $\subseteq$ be set inclusion relation, and $\oplus, \otimes, \ominus$ be canonically defined above, define:

- the *canonical basic frame* as $(Th(L), aTh(L), \subseteq)$,
- the *canonical frame* for LB^+ as $(Th(LB^+)$, $aTh(LB^+)$, $\subseteq$, $l(LB^+)$, $\oplus$, $\otimes$, $\ominus)$.

For any $\chi \in Th(L) \cup aTh(L)$ and any formula C, let $\chi \models C$ iff $C \in \chi$, define:

- the *canonical basic model* as $(Th(L), aTh(L), \subseteq, \models)$,
- the *canonical model* for LB^+ as $(Th(LB^+)$, $aTh(LB^+)$, $\subseteq$, $l(LB^+)$, $\oplus$, $\otimes$, $\ominus$, $\models)$.

By definitions and lemmas in the above subsections, it is easy to show that these canonical frames and models deserve their names.

Lemma 8. *The above canonical frames and models are well defined.*

Now, if C is not a theorem of L, then $C \notin l(L)$, i.e., C is not valid in the canonical model of L. So we can obtain the following completeness results by proving the contrapositive.

Theorem 2 (Completeness for LB^+). *If $\models C$, then $\vdash_{LB^+} C$.*

6 Extensions of LB^+

In this section, we consider extensions to various positive relevant logics without distribution. For each of **C1-C7**, we give corresponding semantical condition on models so that soundness and completeness are maintained. Since implication $\rightarrow$ can be interpreted by three operations $\oplus, \otimes, \ominus$, there may be a number of semantical conditions, which are equivalent to each other (guaranteed by the condition: $\forall t, u \in T_1, \forall v \in T_2, u \oplus t \leq v \Leftrightarrow u \leq t \otimes v \Leftrightarrow t \leq u \ominus v$), corresponding to one given axiom or rule.

Theorem 3. *The extension of LB^+ obtained by adding axiom or rule* **Ci** *is sound and complete with respect to the class of models $(T_1, T_2, \leq, l, \oplus, \otimes, \ominus, \models)$, which satisfy condition* **Di** *(with subscripts)*[6].

D1 $\forall t \in T_1, v \in T_2, t \otimes v \leq t \otimes (t \otimes v)$
D2 $\forall t, u \in T_1, v \in T_2, (u \oplus t) \otimes v \leq t \otimes (u \otimes v)$
D3 $\forall t, u \in T_1, v \in T_2, (u \oplus t) \otimes v \leq u \otimes (t \otimes v)$
D4$_1$ $\forall t \in T_1, v \in T_2, t \leq v \Rightarrow t \leq t \otimes v$
D4$_2$ $\forall t \in T_1, v \in T_2, t \leq v \Rightarrow t \leq t \ominus v$
D5$_1$ $\forall t, u \in T_1, v \in T_2,\ u \leq t \otimes v \Rightarrow t \leq u \otimes v$

[6] Please note that there may be other corresponding conditions for each of **C1-C7** besides those listed here. Especially, Fine's semantical conditions with the operation $\cdot$ in [6] work well on our models (one should permute 1-place and 2-place parameters of $\cdot$).

D5$_2$ $\forall t, u \in T_1, v \in T_2,\ t \leq u \ominus v \Rightarrow u \leq t \ominus v$
D5$_3$ $\forall t \in T_1, v \in T_2, t \otimes v \leq t \ominus v$
D6 $\forall u \in T_1, v \in T_2, u \ominus v \leq u \ominus (u \ominus v)$
D7$_1$ $\forall t \in T_1, v \in T_2, t \leq v \Rightarrow l \leq t \otimes v$
D7$_2$ $\forall t \in T_1, v \in T_2, t \leq v \Rightarrow t \leq l \ominus v$

Proof. For soundness, we take an arbitrary model and assume it satisfies **Di** (with subscripts). Then we demonstrate **Ci** is valid in this model. Completeness is proved by showing the canonical model for an extension with **Ci** must satisfy **Di** (with subscripts). We give proofs for some lists as examples.

C1. For soundness, suppose $t \in T_1$ and $t \models (A \to B) \wedge (B \to C)$, then $t \models A \to B$ and $t \models B \to C$. Suppose $v \in T_2$ and $t \otimes v \models A$. So, $t \otimes (t \otimes v) \models A$. Then, $t \otimes v \models B$. So, $v \models C$ as required.

For completeness, suppose $A \in t \otimes v$ and $A \notin t \otimes (t \otimes v)$, then $\exists B \notin t \otimes v$, $A \to B \in t$. Then, $\exists C \notin v$, $B \to C \in t$. So, $(A \to B) \wedge (B \to C) \in t$. Then, $A \to C \in t$. Since $A \in t \otimes v$, $C \in v$, giving a contradiction.

C2. For soundness, suppose $t \in T_1$ and $t \models A \to B$, $u \in T_1$ and $u \models B \to C$, to show $u \oplus t \models A \to C$. Suppose further $v \in T_2$ and $(u \oplus t) \otimes v \models A$. So, $t \otimes (u \otimes v) \models A$. Then, $u \otimes v \models B$. Then, $v \models C$ as required.

For completeness, suppose $A \notin t \otimes (u \otimes v)$, then $\exists B \notin u \otimes v$, $A \to B \in t$. So, $\exists C \notin v$, $B \to C \in u$. So, $(B \to C) \to (A \to C) \in t$. Then, $A \to C \in u \oplus t$. Since $C \notin v$, $A \notin (u \oplus t) \otimes v$.

C3. For soundness, suppose $t \in T_1$ and $t \models A \to B$, $u \in T_1$ and $u \models C \to A$, to show $u \oplus t \models C \to B$. Suppose further $v \in T_2$ and $(u \oplus t) \otimes v \models C$. So, $u \otimes (t \otimes v) \models C$. So, $t \otimes v \models A$. So, $v \models B$ as required.

For completeness, suppose $C \notin u \otimes (t \otimes v)$. Then, $\exists A \notin t \otimes v$, $C \to A \in u$. Then, $\exists B \notin v$, $A \to B \in t$. So, $(C \to A) \to (C \to B) \in t$. Then, $C \to B \in u \oplus t$. Since $B \notin v$, $C \notin (u \oplus t) \otimes v$.

C4 by **D4**$_1$**.** For soundness, suppose $t \in T_1$ and $t \models A \wedge (A \to B)$, then $t \models A$ and $t \models A \to B$. Let $v \in T_2$ and $t \leq v$, to show $v \models B$. So, $t \leq t \otimes v$. Then, $t \otimes v \models A$. So, $v \models B$. Since v is arbitrary, $t \models B$ as required.

For completeness, suppose $A \in t$ and $A \notin t \otimes v$. Then, $\exists B \notin v$, $A \to B \in t$. So, $A \wedge (A \to B) \in t$. Then, $B \in t$. Since $t \subseteq v$, $B \in v$, giving a contradiction.

C5 by **D5**$_1$**.** For soundness, suppose $t \in T_1$ and $t \models A$, $v \in T_2$ and $t \otimes v \models A \to B$, to show $v \models B$. Then $\exists u \in T_1$, $u \leq t \otimes v$ and $u \models A \to B$. So, $t \leq u \otimes v$. Then, $u \otimes v \models A$. Hence, $v \models B$, as required.

For completeness, suppose $A \in t$, but $A \notin u \otimes v$. Then, $\exists B \notin v$, $A \to B \in u$. Since $u \subseteq t \otimes v$, $A \to B \in t \otimes v$. Since $(A \to B) \to B \in t$, $B \in v$, giving a contradiction.

C6. For soundness, suppose $t \in T_1$ and $t \models A \to (B \to C)$, and $u \in T_1, v \in T_2$, $t \leq u \ominus v$ and $u \models A \wedge B$, to show $v \models C$. Then, $u \models A$ and $u \models B$. Since $u \ominus v \leq u \ominus (u \ominus v)$, $t \leq u \ominus (u \ominus v)$. So, $u \ominus v \models B \to C$. Since $u \ominus v \leq u \ominus v$, $v \models C$ as required.

For completeness, suppose $C' \in u \ominus v$ and $C' \notin u \ominus (u \ominus v)$, then $\exists A \in u, A' \notin u \ominus v$, $\vdash_L C' \to (A \to A')$, Then $\exists B \in u, C \notin v$, $\vdash_L A' \to (B \to C)$. So, $\vdash_L C' \to (A \to (B \to C))$. Then, $\vdash_L C' \to (A \wedge B \to C)$ and $A \wedge B \in u$. Since $C' \in u \ominus v$, $C \in v$, giving a contradiction.

C7 by **D7$_1$.** For soundness, suppose $l \models A$, $t \in T_1$ and $t \models A \to B$. Let $v \in T_2$ and $t \leq v$, to show $v \models B$. So, $l \leq t \otimes v$. Then, $t \otimes v \models A$. So, $v \models B$. Since v is arbitrary, $t \models B$ as required.

For completeness, suppose $A \in l$ and $A \notin t \otimes v$, then $\exists B \notin v$, $A \to B \in t$. So, $(A \to B) \to B \in l$, i.e., $\vdash_L (A \to B) \to B$. So, $B \in t$. Since $t \subseteq v$, $B \in v$, giving a contradiction. □

7 Concluding Remarks

In this paper, we presented operational semantics for various positive relevant logics without distribution. Our work followed and extended Fine's work. We showed that three different binary operations $\oplus, \otimes, \ominus$ can be used together to model implication $\to$.

There are several topics that can be further developed. First, it is expected to generalize our operational semantics to treat other substructural logics without distribution. Second, since the canonical model for LB^+ satisfies: $\forall t, u, v \in Th(L) \cup aTh(L)$, $u \oplus t \subseteq v \Leftrightarrow u \subseteq t \otimes v \Leftrightarrow t \subseteq u \ominus v$, we can use a ternary relation R, instead of these three operations, to evaluate $\to$. Canonically, $Rtuv$ can be defined as: $\forall A, B, (A \to B \in t$ and $A \in u \Rightarrow B \in v)$, i.e., $u \oplus t \subseteq v$, i.e., $u \subseteq t \otimes v$, i.e., $t \subseteq u \ominus v$. Then, we can define an equivalent relational model for LB^+ with $\to$ modeled by R. Actually, we will obtain a new relational model, which retains and extends most features of Routley-Meyer semantics for relevant logic, including semantical conditions for **C1-C7**. But, since all these results are not easy to obtain, we will leave this topic for another occasion.

References

1. Allwein, G., Dunn, J.M.: Kripke Models for Linear Logic. Journal of Symbolic Logic 58, 514–545 (1993)
2. Brady, R.T. (ed.): Relevant Logics and their Rivals, Volume 2, Ashgate Publishing Company (2003)
3. Dosen, K.: Sequent-Systems and Groupoid Models. I. Studia Logica 47, 353–385 (1988)
4. Dosen, K.: Sequent-Systems and Groupoid Models. II. Studia Logica 48, 41–65 (1989)
5. Dunn, J.M.: Positive Modal Logic. Studia Logica 55, 301–317 (1995)
6. Fine, K.: Models for Entailment. Journal of Philosophical Logic 3, 347–372 (1974)
7. Girard, J.Y.: Linear Logic. Theoretical Computer Science 50, 1–102 (1987)
8. Hartonas, C.: Duality for Lattice-Ordered Algebras and for Normal Algebraizable Logics. Studia Logica 58, 403–450 (1997)
9. Ono, H., Kimori, Y.: Logics without the Contraction Rule. Journal of Symbolic Logic 50, 169–201 (1985)

10. Ono, H.: Semantics for Substructural Logics. In: Schroeder-Heister, P., Dosen, K. (eds.) Substructural Logics, pp. 259–291. Oxford University Press, Oxford (1993)
11. Routley, R., Meyer, R.K.: The Semantics of Entailment. III. Journal of Philosophical Logic 1, 192–208 (1972)
12. Routley, R., Plumwood, V., Meyer, R.K., Brady, R.T.: Relevant Logics and their Rivals, Volume 1, Ridgeview Publishing Company (1982)

Multi-valued Logics, Effectiveness and Domains

Giangiacomo Gerla

Department of Mathematics and Computer Science,
University of Salerno
Via Ponte don Melillo 84084,
Fisciano (SA) Italy
gerla@unisa.it

Abstract. Effective domain theory is applied to fuzzy logic to give suitable notions of semi-decidable and decidable L-subset. The connection with the notions of fuzzy Turing machines and fuzzy grammar given in literature is also investigated. This shows the inadequateness of these definitions and the difficulties in formulating an analogue of Church Thesis for fuzzy logic.

Keywords: Multi-valued logic, Fuzzy logic, Computability, Domain theory, Fuzzy grammar, Fuzzy Turing Machine, Church Thesis.

1 Introduction

Fuzzy logic is a promising chapter of multi-valued logic whose basic ideas have been formulated by L. A. Zadeh, J. A. Goguen, J. Pavelka and others (see, for example, [5], [18] and [13]) and successively investigated by several authors (see, for example, [7], [11], [6], [4],[12]). The aim of such a logic is to formalize the *"approximate reasoning"* we use in everyday life where vague notions, such as *big*, *slow*, *near*, are constantly involved. This leads to define a deduction operator associating every fuzzy subset of axioms with the related fuzzy subset of logical consequences. Now it is evident that a basic task for fuzzy logic is to exhibit the effectiveness of its deduction apparatus. In particular, it is important to prove that the fuzzy subset of consequences of a "decidable" fuzzy subset of axioms is "effectively enumerable". To do this we have to give adequate definitions of "effective enumerability" and "decidability" for fuzzy subsets.

On the other hand, in my opinion the phenomenon of the vagueness leads to assume that the set of truth values is a continuum. More precisely, density is suggested by the existence of intermediate values. To give an example, assume that the atomic formula Big(a) is evaluated λ, that Big(b) is evaluated μ and $\lambda < \mu$. Then we cannot exclude the existence of an object d such that Big(d) has a truth value between λ and μ. Also, completeness is suggested by the fact that the quantifiers are interpreted by the least upper bound and the greatest lower bound operators. Moreover, the existence of the least upper bounds is necessary to fuse the different valuations given by different proofs of a given formula (see [13]).

Once we accept the hypothesis that the set of truth values is a continuum, the notion of effectiveness has to be based on endless effective approximation algorithms (as in recursive analysis) and not on algorithms converging in finite steps (as in recursive arithmetics). So, as proposed in [3], a natural framework to define the notion of effectiveness in multi-valued logic is the theory of effective domains (see also [1], [2]). Obviously, this is not the unique possible choice and it is possible to refer to the vast and interesting literature concerning a constructive approach to the continuum.

In this paper we compare the domain-based definition of semi-decidability for fuzzy subsets with the definitions given in literature based on the notions of fuzzy grammar and fuzzy Turing machine. This comparison proves that these definitions are not adequate and it shows the difficulties in formulating an analogue of Church Thesis for fuzzy logic. Also, it emphasizes an open question: to find adequate definitions of multi-valued Turing machine and multi-valued grammar.

2 Preliminaries: Effective Lattices and Semi-decidable Elements

In this paper L always denotes a complete lattice with minimum 0 and maximum 1. Given $x, y \in L$, we say that x is *way below* y and we write $x \ll y$ provided that, for every nonempty upward directed subset A of L

$$y \leq \sup A \Rightarrow \text{ there is } a \in A \text{ such that } x \leq a.$$

Definition 1. *A based continuous lattice, in brief a based lattice, is a structure $(L, \leq, B)$ where L is a complete lattice and B, the basis, is a subset of L containing 0, closed with respect to $\vee$ and $\wedge$ and such that, for every $x \in L$,*

$$x = \sup(\{b \in B : b \ll x\}). \tag{1}$$

We have that $x \ll y$ entails $x \leq y$. If L is a finite chain and we set $B = L$, then $(L, \leq, B)$ is a based lattice such that

$$x \ll y \Leftrightarrow x \leq y.$$

If L is a complete chain and B a dense subset of L, then $(L, \leq, B)$ is a based lattice such that

$$x \ll y \Leftrightarrow \text{ either } x = 0 \text{ or } x < y.$$

Definition 2. *An effective continuous lattice (see [15]), in brief an effective lattice, is a based lattice $(L, \leq, B)$ with an enumeration $(b_n)_{n\in\mathbb{N}}$ of B such that*

- *the relation $\{(n, m) \in \mathbb{N}^2 : b_n \ll b_m\}$ is recursively enumerable*
- *two recursive maps* $\text{join} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, *and* $\text{meet} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *exist such that*

$$b_n \vee b_m = b_{\text{join}(n,m)} \quad , \quad b_n \wedge b_m = b_{\text{meet}(n,m)}.$$

In brief, an effective lattice is a based lattice such that in B the relation $\ll$ is recursively enumerable and $\vee$, $\wedge$ are computable operations.

It is evident that every finite chain L is an effective lattice with respect to $B = L$. The interval $U = [0, 1]$ is an effective lattice whose basis is the set U_Q of rational numbers in U.

Definition 3. *We say that an element x in an effective lattice $(L, \leq, B)$ is semi-decidable if the cut $\{n \in \mathbb{N} : b_n \ll x\}$ is recursively enumerable.*

In particular, every $b \in B$ is semi-decidable and 1 is semi-decidable, too. In a finite lattice all the elements are semi-decidable. If $L = U$, then

$$x \text{ is semi-decidable} \Leftrightarrow \{r \in U_Q : r < x\} \text{ is recursively enumerable.}$$

Proposition 1. *Let $(L, \leq, B)$ be an effective lattice, then the following are equivalent:*

i) *x is semi-decidable,*
ii) *a recursive map f exists such that $(b_{f(n)})_{n\in\mathbb{N}}$ is $\ll$-preserving and*

$$x = \sup_{n\in\mathbb{N}} b_{f(n)}, \tag{2}$$

iii) *a recursive map f satisfying (2) exists such that $(b_{f(n)})_{n\in\mathbb{N}}$ is order-preserving,*
iv) *a recursive map f satisfying (2) exists.*

3 Decidable Elements

To define the notion of decidability we need to dualize some of the definitions in the previous sections. Given a lattice $(L, \leq)$, we denote by $(L, \leq_{\mathrm{d}})$ its dual. Any order-theoretical concept in $(L, \leq)$ is associated with its dual, i.e. the same concept interpreted in $(L, \leq_{\mathrm{d}})$. As an example, we say that y is *way above* x and we write $x \ll^{\mathrm{d}} y$ in the case y is way below x in $(L, \leq_{\mathrm{d}})$. Then $x \ll^{\mathrm{d}} y$ if, for every downward directed subset A of L,

$$x \geq \inf A \Rightarrow \text{ there exists } a \in A \text{ such thath } y \geq a.$$

Obviously $x \ll^{\mathrm{d}} y$ entails $y \leq x$. If L is a finite chain,

$$x \ll^{\mathrm{d}} y \Leftrightarrow y \leq x.$$

If L coincides with U, then

$$x \ll^{\mathrm{d}} y \Leftrightarrow \text{either } y = 1 \text{ or } y < x.$$

Definition 4. *A structure $(L, \leq, B, \underline{B})$ is called an effective above-below lattice, in brief ab-lattice, provided that both the structures $(L, \leq, B)$ and $(L, \leq_{\mathrm{d}}, \underline{B})$ are based (effective) continuous lattices. In such a case we say that $B = (b_n)_{n\in\mathbb{N}}$ is the basis and $\underline{B} = (\underline{b}_n)_{n\in\mathbb{N}}$ the dual basis of $(L, \leq_{\mathrm{d}}, B, \underline{B})$.*

The term *above-below* is due to the fact that, for every $x \in L$,

$$x = \sup\{b \in B : b \ll x\} = \inf\{\underline{b} \in \underline{B} : x \ll^{\mathrm{d}} \underline{b}\}.$$

So, we can approximate every element both from below and from above. If the approximation process is effective, we obtain the notion of decidable element.

Definition 5. *Given an ab-lattice $(L, \leq, B, \underline{B})$, we say that x is decidable if x is semi-decidable both in $(L, \leq, B)$ and in $(L, \leq_{\mathrm{d}}, \underline{B})$, i.e. if both the cuts*

$$\{n \in \mathbb{N} : b_n \ll x\} \quad ; \quad \{n \in \mathbb{N} : x \ll^{\mathrm{d}} \underline{b}_n\}$$

are recursively enumerable.

Trivially, in all the *ab*-lattices both 0 and 1 are decidable. The proof of the following proposition is an immediate consequence of Proposition 1.

Proposition 2. *Given an element x of an effective ab-lattice, the following are equivalent:*

i) x is decidable
ii) two total recursive functions $h : \mathbb{N} \to \mathbb{N}$, $k : \mathbb{N} \to \mathbb{N}$ exist such that $(b_{h(n)})_{n \in \mathbb{N}}$ is $\ll$-preserving, $(\underline{b}_{k(n)})_{n \in \mathbb{N}}$ is $\ll^{\mathrm{d}}$-reversing and

$$\sup_{n \in \mathbb{N}} b_{h(n)} = x = \inf_{n \in \mathbb{N}} \underline{b}_{k(n)} \tag{3}$$

iii) two total recursive functions $h : \mathbb{N} \to \mathbb{N}$, $k : \mathbb{N} \to \mathbb{N}$ exist such that (3) is satisfied, $(b_{h(n)})_{n \in \mathbb{N}}$ is order-preserving and $(\underline{b}_{k(n)})_{n \in \mathbb{N}}$ is order-reversing
iv) a nested effectively computable sequence $([b_{h(n)}, \underline{b}_{k(n)}])_{n \in \mathbb{N}}$ of intervals exists such that

$$\{x\} = \bigcap_{n \in \mathbb{N}} [b_{h(n)}, \underline{b}_{k(n)}].$$

An easy way to obtain *ab*-lattices is by an involution in L.

Definition 6. *A structure $(L, \leq, -, B)$ is an effective lattice with an involution if $(L, \leq, B)$ is an effective lattice and $-$ is an involution such that $\{(n, m) \in \mathbb{N} \times \mathbb{N} : -b_n \ll -b_m\}$ is recursively enumerable.*

In the case $L = \{\lambda_0, ..., \lambda_n\}$ is a finite chain where $0 = \lambda_0 < ... < \lambda_n = 1$, there is a unique involution $\neg$ defined by setting $\neg(\lambda_i) = \lambda_{n-i}$. In the case L is the interval U, an involution $\neg$ is obtained by setting $\neg(\lambda) = 1 - \lambda$.

Since an involution is an isomorphism between L and its dual and since an isomorphism preserves the definable relations, we have that, for every $x \in L$:

$$x \ll^{\mathrm{d}} y \;\Leftrightarrow\; -y \ll -x.$$

The proof of the following proposition is trivial.

Proposition 3. *Let $(L, \leq, B, -)$ be an effective lattice with an involution and set $\underline{B} = (\underline{b}_n)_{n\in\mathbb{N}}$ where $\underline{b}_n = -b_n$. Then $(L, \leq, B, \underline{B})$ is an effective ab-lattice. Moreover,*

x is decidable $\Leftrightarrow$ both x and $-x$ are semi-decidable.

This proposition entails that a finite chain L is an effective *ab*-lattice in which $B = \underline{B} = L$ and in which all the elements are decidable. The interval U is an effective *ab*-lattice in which $B = \underline{B} = U_Q$. In such a case an element x is decidable provided that both the sections $\{r \in U_Q : r < x\}$ and $\{r \in U_Q : x > r\}$ are recursively enumerable, i.e. x is a recursive real number.

4 The Effective Lattice of the *L*-Subsets of a Given Set

Let S be a nonempty set. Then we call *L-subset* of S every element in the direct power L^S. We denote by $\cup$ and $\cap$ the lattice operations in L^S and we call these operations *union* and *intersection*, respectively. Then the union and intersection operations are defined by setting, for every $s_1, s_2 \in L^S$ and $x \in S$,

$$(s_1 \cup s_2)(x) = s_1(x) \vee s_2(x) \ \ ; \ \ (s_1 \cap s_2)(x) = s_1(x) \wedge s_2(x).$$

In an analogous way we define the infinitary unions and intersections. If $L = U$ an $L-$subset is also called *fuzzy subsets* of S. In the case an involution $\neg : L \to L$ is defined in L, then we call *complement* the corresponding operation in L^S. Then, the complement of an L-subset s, is the L-subset $-s$ defined by setting $(-s)(x) = \neg s(x)$. The elements in L are interpreted as truth values in a multi-valued logic where 0 is interpreted as "true" and 1 as "false". An L-subset is interpreted as a generalized characteristic function to represent the extension of a vague predicate. So, for every $x \in S$, $s(x)$ is the *membership degree* of x to s. We call *crisp* an L-subset s such that $s(x) \in \{0, 1\}$ for every $x \in S$. Given $X \in P(S)$, the *characteristic function* of X is the map $c_X : S \to L$ defined by setting $c_X(x) = 1$ if $x \in X$ and $c_X(x) = 0$ otherwise. We can identify the classical subsets of S with the crisp L-subsets of S via the characteristic functions.

Given an L-subset s, we set $\mathrm{Supp}(s) = \{x \in S : s(x) \neq 0\}$ and $\mathrm{Cosp}(s) = \{x \in S : s(x) \neq 1\}$. We say that s is *finite* (*co-finite*) provided that $\mathrm{Supp}(s)$ ($\mathrm{Cosp}(s)$, respectively) is finite. We call finite also the empty set and co-finite the whole set S. The classes of finite and co-finite L-subsets of S are denoted by $\mathrm{Fin}(L^S)$ and $\mathrm{Cof}(L^S)$, respectively. Obviously, if a negation is defined in L, then an L-subset is finite if and only if its complement is co-finite.

In the following we assume that S admits a code. This enables us to identify S with the set of natural numbers and to prove the following theorems (see [3]).

Theorem 1. *Let $(L, \leq, B)$ be an effective lattice. Then the class L^S of L-subsets of S is an effective lattice admitting as a basis the class $\mathrm{Fin}(B^S)$ of finite L-subsets of S with values in B. Also, for every s_1 and s_2 in L^S,*

$$s_1 \ll s_2 \Leftrightarrow s_1 \textit{ is finite and } s_1(x) \ll s_2(x) \textit{ for every } x \in S.$$

Observe that, by definition, an L-subset s is semi-decidable provided that

$$\{n \in \mathbb{N} : b_n \ll s\} = \{n \in \mathbb{N} : b_n(i) \ll s(i) \text{ for every } i \in \mathrm{Supp}(b_n)\}.$$

is a recursively enumerable set. There are simple characterizations of the semi-decidable L-subsets.

Theorem 2. *Let $(L, \leq, B)$ be an effective continuous lattice and $s \in L^S$. Then the following are equivalent:*

i) s is semi-decidable

ii) a recursive function $h : S \times \mathbb{N} \to B$ exists which is $\ll$-increasing with respect to n such that

$$s(x) = \sup_{n \in \mathbb{N}} h(x, n)$$

iii) a recursive function $h : S \times \mathbb{N} \to B$ exists which is increasing with respect to n such that

$$s(x) = \sup_{n \in \mathbb{N}} h(x, n).$$

The following proposition enables us to define the notion of decidable L-subset.

Proposition 4. *Let $(L, \leq, B, \underline{B})$ be an effective ab-lattice. Then L^S is an effective ab-lattice with dual basis the class $\mathrm{Cof}(B^S)$ of co-finite L-subsets of S with values in B. If $(L, \leq, B, -)$ is an effective lattice with an involution, then L^S is an effective lattice with the complement as an involution.*

Trivially, if L is an effective lattice with an involution, then
s is decidable $\Leftrightarrow$ both s and its complement $-s$ are semi-decidable.

5 The Main Cases

In this section we will consider two cases which are basic ones in fuzzy logic: the finite chains and the interval U. Observe that in these cases the proposed notions of semi-decidability and decidability for fuzzy subsets are in accordance with the ones given in [1] and [2].

Proposition 5. *Let L be a finite chain. Then the class L^S of L-subsets of S is an effective lattice with the complement as an involution and therefore it is an effective ab-lattice. Its basis is the class $\mathrm{Fin}(L^S)$ of finite L-subsets of S, its dual basis is the class $\mathrm{Cof}(L^S)$ of co-finite L-subsets of S. Also*
$s_1 \ll s_2 \Leftrightarrow s_1 \subseteq s_2$ *and* s_1 *is finite*
and
$s_1 \ll^{\mathrm{d}} s_2 \Leftrightarrow s_1 \subseteq s_2$ *and* s_2 *is co-finite.*

In particular, the class $P(S)$ of subsets of S is an effective lattice with an involution whose basis is the class of finite subsets and whose dual basis is the class of co-finite subsets of S. Also

$X_1 \ll X_2 \Leftrightarrow X_1 \subseteq X_2$ and X_1 is finite
and
$X_1 \ll^{\mathrm{d}} X_2 \Leftrightarrow X_1 \subseteq X_2$ and X_2 is co-finite.
Moreover, the proposed notions of decidability and semi-decidability coincide with the classical ones.

Proposition 6. *Let L be a finite chain. Then an L-subset s is semi-decidable if and only if there is a recursive function $h : S \times \mathbb{N} \to L$ increasing with respect to the second variable such that*

$$s(x) = \max_{n \in \mathbb{N}} h(x, n).$$

Moreover, s is decidable if and only if s is a recursive function.

The following proposition shows that, in the case of a finite chain, the proposed definition of semi-decidability is the only possible extension of the classical one such that

- the constant L-subsets are semi-decidable
- the union of two semi-decidable L-subsets is semi-decidable
- the intersection of two semi-decidable L-subsets is semi-decidable.

To show this, given an L-subset s, we call *closed λ-cut of s* the subset $C(s, \lambda) = \{x \in S : s(x) \geq \lambda\}$ where $\lambda \in L$. The equation

$$s(x) = \textstyle\bigcup_{\lambda \in L} \lambda \wedge C(s, \lambda)$$

shows that the lattice of the L-subsets is the lattice generated by the constant L-subsets and the crisp L-subsets.

Proposition 7. *Let L be a finite chain. Then, the following are equivalent:*

i) s is a semi-decidable L-subset
ii) all the cuts of s are recursively enumerable.

As a consequence, the lattice of the semi-decidable L-subsets is the lattice generated by the recursively enumerable subsets and the constant L-subsets.

In the case $L = U$ we can prove a proposition similar to Proposition 5.

Proposition 8. *The class of fuzzy subsets of S is an effective lattice with the complement as an involution. The basis is the class $\mathrm{Fin}({U_Q}^S)$ of finite fuzzy subsets of S with rational values. The dual basis is the class $\mathrm{Cof}({U_Q}^S)$ of co-finite fuzzy subsets of S with rational values. Moreover*

$$s_1 \ll s_2 \;\Leftrightarrow\; s_1 \textit{ is finite and } s_1(x) < s_2(x) \textit{ for every } x \in \mathrm{Supp}(s_1).$$

$$s_1 \ll^{\mathrm{d}} s_2 \;\Leftrightarrow\; s_2 \textit{ is co-finite and } s_1(x) < s_2(x) \textit{ for every } x \in \mathrm{Cosp}(s_1).$$

Unfortunately, we cannot extend Proposition 7 to this case since a closed cut of a semi-decidable L-subset is not necessarily recursively enumerable. More precisely, we have the following proposition whose proof is an immediate consequence of a series of interesting results about the effectiveness in multi-valued logic (see for example [7] and [10]).

Theorem 3. *A subset of S is a closed cut of a semi-decidable fuzzy subset iff it belongs to the Σ_2-level of the arithmetical hierarchy.*

Observe that such a theorem gives an explanation of an apparent contradiction. In fact the scholars interested in multi-valued logic claim that such a logic is not effective since the set Val of valid formulas is not effective at all (see for example [7]). At the same time it is possible to prove that the L-subset of theorems of a decidable L-theory is semi-decidable and therefore, that the fuzzy set lt of the logically true sentence is semi-decidible (see [1]). This apparent contrast depends on the fact that Val is a cut of lt and, as claimed in Theorem 3, it is not surprising that lt is semi-decidable and that such a cut is not recursively enumerable.

6 Fuzzy Machines and Fuzzy Grammars

A basic question is whether our definition of recursive enumerability is the correct formal counterpart of the intuition and experience of fuzzy people about fuzzy computability. In other words:
Is our definition a reasonable proposal for a "Church Thesis" in multi-valued logic ?

As an attempt to face this question, we consider the notions of L-grammar and L-Turing machine given in literature. To do this, we assume that in the effective lattice L an operation $\otimes$ is defined to interpret the conjunction and that $\otimes$ is order-preserving, associative, commutative and such that $x \otimes 1 = x$ for every $x \in L$. We assume also that $\otimes$ is recursive on the basis B. These conditions are satisfied in all the main multi-valued logics. Firstly, we recall the notion of L-grammar (see [8] and ([9]))

Definition 7. *An L-grammar is a structure $G = (T, I, \mu, s)$ where:*
- *T is a finite set and $I \subset T$,*
- *$\mu : T^+ \times T^+ \rightarrow B$ is a finite L-subset (the L-subset of productions)*
- *$s \in T - I$ (the start symbol).*

Given $\lambda \neq 0$ and two words w, w', we say that w' is *directly derivable from w with degree λ* if $x, y \in T^+$ and $a, b \in T^*$ exists such that $w = axb$, $w' = ayb$ and $\lambda = \mu(x, y)$. We say that a sequence $\pi = (w_1, ..., w_p, \lambda_1, ..., \lambda_{p-1})$ is a *derivation for w at degree* $\lambda(\pi) = \lambda_1 \otimes ... \otimes \lambda_{p-1}$ provided that $w_1 = s$, $w_p = w$ and, for $i = 1, 2, ..., p-1$, the word w_{i+1} is directly derivable from w_i with degree λ_i. Since it is possible that there are different derivations of the same word, the L-language generated by an L-grammar is defined as follows.

Definition 8. *Let $G = (T, I, \mu, s)$ be an L-grammar, then the L-language generated by G is the L-subset $s : I^+ \rightarrow L$ defined by*

$$s(w) = \sup\{\lambda(\pi) : \pi \text{ is a derivation of } w\}. \tag{4}$$

There are various attempts to formalize of the notion of fuzzy algorithms in terms of Turing machines. The first ones are dated in late 1960s when this notion was introduced by L. A. Zadeh (see [17]). The following definition is an obvious extension of the one proposed by E. S. Santos in [14] (see also [16]).

Definition 9. *An L-Turing machine is a structure $F = (S, T, I, b, q_0, q_f, \mu, \otimes)$, where*

- S is the finite set of states;
- T is the finite set of tape symbols;
- $I \subset T$ is the set of input symbols ;
- μ is an L-subset of $S \times T \times S \times T \times \{-1, 0, 1\}$ with values in B (we call L-transition function)
- $b \in T - I$ is the blank symbol;
- q_0 and q_f are the initial and accepting states, respectively.

Symbol -1 (+1) denotes a move by one cell to the left (right) and 0 denotes no move. The tape symbols can be printed on a tape that has a left-most cell but is unbounded to the right. A *move* is an element $m = (s_1, t_1, s_2, t_2, d)$ in $S \times T \times S \times T \times \{-1, 0, 1\}$ and this move is realized provided that if the current state is s_1 and the tape symbol scanned by the machine's head is t_1, then F will enter the new state s_2, the new tape symbol t_2 will rewrite the previous symbol t_1, and the tape head will move in accordance with d. The value $\mu(m)$ is a valuation of correctness (possibility) of the move m. The notion of computation is defined as usual with the help of instantaneous descriptions (IDs). An instantaneous description Q_t of F working on input w at time t is a unique description of the machine's tape, of its state and of the position of the machine's head after performing its *tth* move on input w.

Definition 10. *If Q_t and Q_{t+1} are two* ID*s we denote by $D(Q_t, Q_{t+1})$ the last upper bound of the set of correctness degrees $\mu(m)$ of the moves m leading from Q_t to Q_{t+1}.*

We can interpret $D(Q_t, Q_{t+1})$ as the valuation in a multi-valued logic of the claim "there is a correct move leading from Q_t to Q_{t+1}". Observe that if no move exists leading from Q_t to Q_{t+1} then $D(Q_t, Q_{t+1}) = 0$, otherwise $D(Q_t, Q_{t+1})$ is a maximum and we can calculate it in an effective way. On input w whose length is n, the machine starts its computation in an initial ID, we denote by $Q(w)$, describing the tape holding a string of n input symbols (the so-called input string, or input word), one symbol per cell starting with the leftmost cell. All cells to the right of the input string are blank. The head is scanning the leftmost cell and the current state is q_0. From this ID the computation proceeds to an ID, we denote by Q_1 which is reachable in one step from Q_0, etc.

Definition 11. *A computation is a sequence $Q_0, ...Q_k$ of* ID*s. We extend the function D to any computation $Q_0, ..., Q_k$, by setting*
$D(Q_0, ..., Q_k) = D(Q_0, ..., Q_{k-1}) \otimes D(Q_{k-1}, Q_k)$.
Moreover, if Q and Q^* are two ID*s, we set*
$d(Q, Q^*) = \sup\{D(Q_0, Q_1, ..., Q_t) : Q_0 = Q, Q_t = Q^*\}$.

We can interpret $D(Q_0, ..., Q_k)$ as the valuation in a multi-valued logic of the claim "the computation $Q_0, ...Q_k$ is correct" and $d(Q, Q^*)$ as the valuation of the claim "there is a correct computation leading from Q to Q^*".

Definition 12. *Let F be an L-Turing machine and $w \in I^+$. Then we say that $Q_0, Q_1 ... Q_k$ is an accepting computation for w, if $Q_0 = Q(w)$ and Q_k is an* ID *containing the accepting state q_f. Moreover, the L-language accepted by F is the L-subset $e : I^+ \to L$ of I^+ defined by setting*

$$e(w) = \sup\{d(Q(w), Q^*) : Q^* \textit{ is an accepting } \mathrm{ID} \textit{ for } w\}. \tag{5}$$

The following theorem shows that the notion of effectiveness for multi-valued logic proposed in this paper is in accordance with just given notions of L-grammar and L-Turing machine.

Theorem 4. *Let s be an L-language either generated by an L-grammar or accepted by an L-Turing machine. Then s is a semi-decidable L-subset.*

Proof. Assume that s is generated by an L-grammar and therefore that s satisfies (4). Then, since for every input w we can enumerate in an effective way the class of derivations for w, s is semi-decidable. A similar argument holds true for the L-Turing machines.

The following theorem shows that, in the case L is a finite chain, the domain-based, the grammar-based and the machine-based notions of effectiveness all coincide.

Theorem 5. *Assume that L is a finite chain and let s be an L-subset. Then the following are equivalent:*

- *s is semi-decidable,*
- *there is a suitable L-grammar able to generate s,*
- *there is a suitable L-Turing machine able to accept s.*

Proof. Let L be the finite chain whose elements are $0 = \lambda_0 < ... < \lambda_n = 1$ and assume that s is semi-decidable. Then all the cuts $C(s, \lambda_i)$ of s are recursively enumerable. For every $0 < i \leq n$, let $G_i = (T, I, M_i, s)$ be a grammar able to generate $C(s, \lambda_i)$ where, as usual, $M_i \subseteq T^+ \times T^+$. Denote by G the L-grammar (T, I, μ, s) obtained by setting $\mu(x) = \sup\{\lambda_i : x \in M_i\}$ and assume that $\otimes$ is the minimum. Then it is easy to see that the L-language generated by such a machine coincides with s.

Likewise, denote by F_i a Turing machine $(S, T, I, b, q_0, q_f, M_i)$ able to accept $C(s, \lambda_i)$ where $M_i \subseteq S \times T \times S \times T \times \{-1, 0, 1\}$. Let F be the L-Turing machine $(S, T, I, b, q_0, q_f, \mu, \wedge)$ such that $\mu(x) = \sup\{\lambda_i : x \in M_i\}$. Then it is easy to see that the L-subset accepted by F coincides with s.

In order to examine the case L infinite, it is useful the following interesting proposition whose proof is in [16].

Proposition 9. *Let $(M, \otimes, 1)$ be a finitely generated sub-monoid of $(L, \otimes, 1)$. Then every nonempty subset of M admits a maximal element. If L is totally ordered, every nonempty subset of M admits a maximum. As a consequence, for every word w, the supremum in (4) and in (5) is a maximum.*

The following theorem shows that the L-languages generated by an L-grammar (accepted by an L-Turing machine) satify very particular properties.

Theorem 6. *Assume that L is totally ordered and let s be an L-language generated by an L-grammar (accepted by an L-Turing machine). Then the values assumed by s are in B and all the closed λ-cuts with $\lambda \in B$ are recursively enumerable.*

Proof. Assume that s is generated by an L-grammar and therefore, by Proposition 9, that $s(w) = \max\{\lambda(\pi) : \pi \text{ is a derivation of } w\}$. Then,

$C(s, \lambda) = \{x \in S : \text{there is a derivation } \pi \text{ such that } \lambda(\pi) \geq \lambda\}$

and therefore, to prove that $C(s, \lambda)$ is recursively enumerable, it is sufficient to observe that that the map $\lambda(w)$ is effectively computable and that the relation $\lambda(\pi) \geq \lambda$ is decidable.

In the case of the L-Turing machines we can go on in a similar way.

Such a theorem shows that Theorem 5 cannot be extended to the case L is an infinite chain. For example, if $L = U$, then every semi-decidable L-language assuming irrational values gives an example of semi-decidable L-subset such that there is no L-grammar able to generate it and no L-Turing machine able to accept it. A more interesting example is furnished in the following theorem.

Theorem 7. *Assume that L is the effective lattice defined by the interval U and let* big : $I^+ \to U$ *be the fuzzy subset of the "big words" defined by setting*

$$\text{big}(w) = 1 - 1/length(w) \tag{6}$$

for every word w. Then big *is a decidable L-language such that*

- big *assumes only rational values*
- *the cuts of* big *are all decidable*
- *no L-grammar is able to generate big*
- *no L-Turing machine is able to accept big.*

Proof. The proof is trivial. We observe only that, since there is no maximum in the co-domain of big, no L-grammar is able to generate big and no L-Turing machine is able to accept big.

Since should be hard to deny that big is decidable from an intuitive point of view, we can conclude that the existing definitions of L-grammar and L-Turing machine are not adequate. This corroborates the domain-based definition of the effectiveness for multi-valued logic and a formulation of the following *"Church Thesis"* for fuzzy set theory: *the domain-based definitions give the adequate formalization of the intuition and experience of fuzzy people about the effectiveness in the fuzzy framework.* Once we accept such a thesis, it is an open question to find adequate definitions of multi-valued Turing machine and multi-valued grammar.

References

1. Biacino, L., Gerla, G.: Fuzzy logic, continuity and effectiveness. Archive for Mathematical Logic 41, 643–667 (2002)
2. Gerla, G.: Decidability, partial decidability and sharpness relation for L-subsets. Studia Logica 46, 227–238 (1987)
3. Gerla, G.: Effectiveness and Multivalued Logics. Journal of Symbolic Logic 71, 137–162 (2006)
4. Gerla, G.: Fuzzy logic: Mathematical tools for approximate reasoning. Kluwer Academic Publishers, Dordrecht (2001)
5. Goguen, J.A.: The logic of inexact concepts. Synthese 19, 325–373 (1968/69)
6. Gottwald, S.: A treatise on many-valued logics. Research Studies Press, Baldock (2000)
7. Hájek, P.: Metamathematics of fuzzy logic. Kluwer Academic Publishers, Dordrecht (1998)
8. Lee, E.T., Zadeh, L.A.: Note on fuzzy languages. Information Science 1, 421–434 (1969)
9. Mizumoto, M., Toyoda, J., Tanaka, K.: $\mathbb{N}$-fold fuzzy grammars. Information Science 5, 25–43 (1973)
10. Montagna, F.: Three complexity problems in quantified fuzzy logic. Studia Logica 68, 143–152 (2001)
11. Mundici, D., Cignoli, R., D'Ottaviano, I.: Algebraic foundations of many-valued reasoning. Kluwer Academic Publishers, Dordrecht (2000)
12. Novak, V., Perfilieva, I., Mockor, J.: Mathematical principles of fuzzy logic. Kluwer Academic Publishers, Dordrecht (2000)
13. Pavelka, J.: On fuzzy logic I: Many-valued rules of inference. Zeitschrift für Mathemathische Logik und Grundlagen der Mathematik 25, 45–52 (1979)
14. Santos, E.S.: Max-Product Machines. J. of Math. Anal. Appl. 37, 677–686 (1972)
15. Smyth, M.: Effectively given domains. Theoretical Computer Science 5, 257–274 (1977)
16. Wiedermann, J.: Fuzzy Turing Machines revised. Inform. and Computing 21, 1–13 (2002)
17. Zadeh, L.A.: Fuzzy algorithms. Information and Control 5, 62–94 (1968)
18. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning I, II, III, Information Sciences, 8, 9 pp. 199–275, pp. 301–357, pp. 43–80 (1975)

Internal Computability

Guido Gherardi

Dipartimento di Scienze Matematiche e Informatiche
"R. Magari", Università di Siena
Pian dei Mantellini 44 - 53100 Siena Italy
gherardi3@unisi.it

Abstract. We extend the notion of (TTE-)computability to nonstandard universes by the traditional method of enlarging universes through ultrafilters. In this way a nonstandard notion of effectivity is obtained.

Keywords: Computable Analysis, Nonstandard Analysis, Constructive Mathematics, Type-2 Theory of Effectivity, Theory of Representations.

The objects investigated in computable analysis are elements of usual mathematical universes. Through the notions of universe embedding and enlargement, these objects can be extended within universes containing nonstandard elements. By the transfer principle, which is fundamental in nonstandard analysis, all ordinary properties of computation hold also in nonstandard universes. In this way we obtain an extension of the notion of computability to $^*\mathbb{R}$ and to other nonstandard amplifications of ordinary mathematical structures.

Coherently, a possible notion of *nonstandard effectiveness* is obtained. By this, we can then provide nonstandard computable versions of classical theorems of nonstandard analysis, as the ordinary theory of representations does for standard mathematics.

Nevertheless, some differences are evident. By our methods, we will define "nonstandardly" computable functions, but the proof of their existence is indirect, since it is given only by analogy with usual computation. In general, the outputs of these functions will be nonstandard, thus objects which are not concretely constructed. This is a characteristic of nonstandard mathematics, which use entities, like particular types of numbers, whose individual essence is known only theoretically.

As a first application, we give nonstandard computable versions of classical results concerning the spillover principle.

Further, we show how this notion of nonstandard effectivity may fit to the intuitive constructive method recently used by D. Ross ([13]) to give a nonstandard proof of an existence result by E. Bishop and H. Cheng ([2]).

The aim of the present work is to introduce nonstandard computability. An inquiry into the possible relations of this notion with the constructive approaches to nonstandard analysis (see for example [10] and [14]) is not within the scope of this paper.

1 Basic Tools from Computable Analysis

We use the TTE-approach to computable analysis ([15]).

For the theory of representations, we use the Baire space $\mathbb{B}$, which is the ordinary metric space on $\mathbb{N}^{\mathbb{N}}$.

Given any sequence $p \in \mathbb{B}$, we write $n \lhd p$ when $n \in \text{range}(p)$.

Definition 1. ***Representations and realizations.*** *A* representation δ *of a set* Y *is a surjective function* $\delta :\subseteq \mathbb{B} \to Y$*. The pair* (δ, Y) *is then said to be a* represented set.

Let (δ_1, Y_1)*,* (δ_0, Y_0) *be two represented sets and let* $f :\subseteq Y_1 \rightrightarrows Y_0$ *be a multifunction.*

Then $F :\subseteq \mathbb{B} \to \mathbb{B}$ *is a* (δ_1, δ_0)*-realization of* f *if and only if* $\delta_0 \circ F(p) \in f\{\delta_1(p)\}$ *for all* $p \in \mathbb{B}$ *such that* $\delta_1(p) \in \text{dom}(f)$*.*

The multifunction f *is* (δ_1, δ_0)-computable *if and only if it has a computable* (δ_1, δ_0)*-realization.*

If Y_0 *is enumerable, we may denote the elements in* Y *through* $\mathbb{N}$*. The notions of realizations and computability are then extended coherently.*

Definition 2. ***Computable metric space.*** *A* computable metric space $\mathbf{Y} = (Y, d, \nu)$ *is a* 3*-tuple where* (Y, d) *is a nonempty complete metric space and:*

- $\nu : \mathbb{N} \to Y$ *is a dense sequence in* Y*;*
- *the distance function* d *is* (ν, ν, ρ)*-computable, where* ρ *is the standard representation of* $\mathbb{R}$ *(see [15]).*

Let then $(I_n)_{n \in \mathbb{N}}$ *be an obvious enumeration of all* (sub)basic open balls $B(c, \alpha)$*, for* $c \in \text{range}(\nu)$ *and* $\alpha \in \mathbb{Q}^+$*. The* standard representation $\delta_{\mathbf{Y}}$ *of* Y *is so defined:*

$$\delta_{\mathbf{Y}}(p) = y \in Y \iff (y \in I_n \leftrightarrow n + 1 \lhd p) \ .$$

In the following, when we speak of computability of some multifunctions $f :\subseteq Y \rightrightarrows \mathbb{N}$, we assume Y to be represented by $\delta_{\mathbf{Y}}$, whereas the elements in $\mathbb{N}$ are denoted by themselves, coherently to Definition 1. This is a very natural way to intend the computability of these multifunctions when they are supposed to give their outputs in a finite amount of time.

Definition 3. ***Closed and compact sets representations.*** *Given any computable metric space* $\mathbf{Y} = (Y, d, \nu)$*, define the following representation of the collection* $\mathcal{A}(Y)$ *of the closed subsets* $A \subseteq Y$*: for* $p \in \mathbb{B}$ *and* $A \in \mathcal{A}(Y)$*,*

$$\psi_{-}(p) = A \iff \sim A = \bigcup_{n+1 \lhd p} I_n \ .$$

Further we represent the class $\mathcal{K}(Y)$ *of compact subsets of* Y *in a natural way:* $p \in \mathbb{B}$ *is a* κ_{mc}*-name of* $K \in \mathcal{K}(Y)$ *if* p *enumerates all finite minimal covers of* K*, where* $I_{n_1}, ..., I_{n_k}$ *is a minimal cover of* K *if* $K \subseteq I_{n_1} \cup ... \cup I_{n_k}$*, and* $K \cap I_{n_i} \neq \emptyset$ *for* $1 \leq i \leq k$ *(recall that* $I_{n_1}, ..., I_{n_k}$ *are (sub)basic open balls).*

For the representations ψ_-, κ_{mc} see [4], where different symbols are used. Given computable metric spaces $\mathbf{Y}_1, \mathbf{Y}_0$ we let $\delta^{\rightarrow}_{\mathbf{Y}_1,\mathbf{Y}_0}$ be an ordinary representation of continuous functions $f : Y_1 \to Y_0$ such that the *apply function* $(f, y) \mapsto f(y)$, for $y \in Y_1$, is computable (see [15]).

2 Nonstandard Analysis

We give the fundamental concepts of non standard analysis. Our presentation cannot be exhaustive, but we refer the reader to [8] and [9] for a wider introduction to the subject.

Definition 4. ***Universes.*** *Let* $\mathbb{R} \subseteq X$*. The* n-th cumulative power set $\mathbb{U}_n(X)$ *of* X *is defined inductively by:*

$$\mathbb{U}_0(X) = X \ ,$$
$$\mathbb{U}_{n+1}(X) = \mathbb{U}_n(X) \cup \wp(\mathbb{U}_n(X)) \ ,$$

so that $\mathbb{U}_0(X) \subseteq \mathbb{U}_1(X) \subseteq ... \subseteq \mathbb{U}_{n+1}(X) \subseteq ....$ *The* superstructure *(or* universe*)* $\mathbb{U}(X)$ *over* X *is defined as:*

$$\mathbb{U}(X) = \bigcup_{n \in \mathbb{N}} \mathbb{U}_n(X) \ .$$

We assume that the elements of X *are* atomic*, thus if* $b \in X$ *there is no* $a \in \mathbb{U}(X)$ *such that* $a \in b$*.*

2.1 The Language of a Universe

As part of model theory, nonstandard analysis is based on the expressivity of languages about $\mathbb{U}(X)$. We follow [8] for the definition of the language $\mathcal{L}_X$. Besides variables, which are $\mathcal{L}_X$-terms, all objects in $\mathbb{U}(X)$ are constants $\mathcal{L}_X$-terms, and if t, s are $\mathcal{L}_X$-terms, then $t(s)$ is an $\mathcal{L}_X$-term. Obviously, this procedure can bring to undefined terms (thus terms with no denotation) like $\tan(\pi/2)$ or $2(\sqrt{\ })$. Natural conditions for distinguishing defined and undefined terms are then given in [8] for $\mathcal{L}_X$-terms.

Definition 5. ***$\mathcal{L}_X$-formulas***

- *atomic formulas are of the form* $t = s$ *and* $t \in s$ *for* t,s $\mathcal{L}_X$*-terms;*
- *if* φ, ψ *are* $\mathcal{L}_X$*-formulas, the same are* $\neg\varphi$*,* $\varphi \wedge \psi$*,* $\varphi \vee \psi$*,* $\varphi \to \psi$*,* $\varphi \leftrightarrow \psi$*;*
- *if* φ *is a* $\mathcal{L}_X$*-formula, the same are* $(\forall x \in t)\varphi$ *and* $(\exists x \in t)\varphi$*, where* t *is any* $\mathcal{L}_X$*-term and* x *is a variable that does not occur in* t*.*

A formula φ is *defined* if and only if all its closed terms are defined.
The conditions for a *defined closed* formula φ to *be true* in $\mathbb{U}(X)$ (for short, $\mathbb{U}(X) \models \varphi$) are given by induction in a quite natural way. The most sensitive case are the quantifiers:

– $\mathbb{U}(X) \models (\forall x \in t)\varphi$ if and only if t names an element (set) $A \in \mathbb{U}(X) \setminus X$ and for all $a \in A$, $\mathbb{U}(X) \models \varphi(a)$ is true, whenever $\varphi(a)$ is defined.

The case of the existential quantifier is analogous. Again, we refer the reader to [8] for further details.

2.2 Universe Embeddings

Given any two universes $\mathbb{U}(X)$, $\mathbb{U}(Y)$, let $* : \mathbb{U}(X) \to \mathbb{U}(Y)$ be a 1-1 map. For each $a \in \mathbb{U}(X)$ we write ${}^*a$ for $*(a)$ as usually done in the literature.

Definition 6. *If φ is a $\mathcal{L}_X$-formula, the* $*$-transform *${}^*\varphi$ of φ is the $\mathcal{L}_Y$-formula obtained by replacing in φ each $a \in \mathbb{U}(X)$ by ${}^*a \in \mathbb{U}(Y)$.*

Definition 7. *Given any two universes $\mathbb{U}(X)$, $\mathbb{U}(Y)$, a* universe embedding *of $\mathbb{U}(X)$ to $\mathbb{U}(Y)$ is a 1-1 map $* : \mathbb{U}(X) \to \mathbb{U}(Y)$ satisfying the* ***transfer principle****:*

– *For any $\mathcal{L}_X$-formula φ: $\mathbb{U}(X) \models \varphi$ if and only if $\mathbb{U}(Y) \models {}^*\varphi$.*

We know through Łos theorem that by using a special ultrafilter we can "*construct*" a proper extension $\mathbb{U}(Y)$ of $\mathbb{U}(X)$ such that $* : \mathbb{U}(X) \to \mathbb{U}(Y)$ satisfies the transfer principle. For the sake of simplicity, we can identify a standard object a with ${}^*a$, unless it is a function or a set and this is relevant in the given context.

The new universe is called an *enlargement* of $\mathbb{U}(X)$ and is denoted by $\mathbb{U}({}^*X)$, where $X \subsetneq {}^*X$. Since $\mathbb{U}({}^*X)$ contains nonstandard objects, we may call it a *nonstandard* universe. More remarkably, $A \subsetneq {}^*A$ when $A \in \mathbb{U}(X)$ is infinite. Thus the extension of any infinite standard set in the new universe contains nonstandard elements. This holds in particular for natural numbers, real numbers, computable functions, representations of a given set, and so on. The elements of ${}^*\mathbb{N}$ and ${}^*\mathbb{R}$ are called *hypernatural* and *hyperreal* numbers, respectively.

Definition 8. ***Unlimited and infinitesimal numbers.*** *A hyperreal number $a \in {}^*\mathbb{R}$ is* unlimited *if for any $n \in \mathbb{N}$: $n < |a|$. A hyperreal which is not unlimited is* limited.

An element $a \in {}^*\mathbb{R}^+$ is infinitesimal *if for any $n \in \mathbb{N}$: $|a| < \frac{1}{n}$.*

"$\mathbb{N}^\infty$" will denote the set of unlimited natural numbers, thus $\mathbb{N}^\infty = {}^*\mathbb{N} \setminus \mathbb{N}$.

We will denote hypernatural numbers by capital letters, usually N, M, K, but we prefer lower-case letters m,n,k for hypernaturals which are known to be standard.

Definition 9. *Let $(Y, d) \in \mathbb{U}(X)$ be a metric space and let $a, b \in {}^*Y$.*

We write $a \simeq b$ if $d(a, b)$ is infinitesimal.

The element $b \in {}^*Y$ is near-standard *if a is standard, thus $a \in Y$. In this case a can be denoted by $st(b)$, as the* standard part *of b.*

The fundamental theorems about near-standard elements are the followings:

Theorem 1. *Given a metric space (Y, d), a real function $f : Y \to \mathbb{R}$ is continuous if and only if for any given $a \in Y$:*

– *if $y \in {}^*Y$ and $y \simeq a$, then ${}^*f(y) \simeq {}^*f(a) = f(a)$.*

Theorem 2. ***Robinson's compactness criterion.*** *A metric space (Y, d) is compact if and only if all $y \in {}^*Y$ are near-standard.*

3 Internally Computable Functions

Definition 10. ***Internal and external objects.*** *An entity $a \in \mathbb{U}({}^*X)$ is* internal *if for some standard set $B \in \mathbb{U}(X)$, $a \in {}^*B$.*
All other entities in $\mathbb{U}({}^*X)$ are external.

Internal entities are those in $\mathbb{U}({}^*X)$ which are ruled by the transfer principle. More precisely

$$\mathbb{U}(X) \vDash (Qx \in a)\varphi \Longleftrightarrow \mathbb{U}({}^*X) \vDash (Qx \in {}^*a)^*\varphi$$

for $Q \in \{\forall, \exists\}$, $a \in \mathbb{U}(X)$, and $\varphi(x)$ any $\mathcal{L}_X$-formula. Thus, the standard elements in $\mathbb{U}(X)$ and the internal elements in $\mathbb{U}({}^*X)$ satisfy the same $\mathcal{L}_X$-properties. For this reason, we can use the expressions "internal" or "internally" to denote nonstandard extensions of standard notions. For example if $(x_n)_{n \in \mathbb{N}}$ is a sequence, it is uniquely extended to an *internal* sequence $(x'_N)_{N \in {}^*\mathbb{N}}$ such that $x'_n = x_n$ for all standard n. Moreover if $x_n \to x$, then $x_N \simeq x_M \simeq x$ for all $N, M \in \mathbb{N}^\infty$ (this fact will be used in the proof of Theorem 8).

Definition 11 is based on the same natural extension principle.

On the other hand, external entities need not to respect the transfer principle, and are more difficult to be handled.

By transfer principle, any member of an internal set is internal. Indeed for all B standard, $B \subseteq \mathbb{U}_n(X)$ for some $n \in \mathbb{N}$. Therefore $a \in A \in {}^*B$ implies $a \in {}^*\mathbb{U}_{n+1}(X)$.

Definition 11. ***Internally computable functions.*** *By "Comp" we denote the set of all (TTE-)computable functions $F :\subseteq \mathbb{B} \to Z$, where $Z \in \{\mathbb{B}, \mathbb{N}\}$.*

Since *Comp and its members are internal, such members (most of them are nonstandard) are said to be internally computable.

Let $(Y_1, \delta_1), (Y_0, \delta_0) \in \mathbb{U}(X)$ be represented sets, and let $\mathcal{S}(Y_1, \delta_1, Y_0, \delta_0)$ be the set of all (δ_1, δ_0)-computable multifunctions $f :\subseteq Y_1 \rightrightarrows Y_0$. Then any $f \in {}^*\mathcal{S}(Y_1, \delta_1, Y_0, \delta_0)$ is said to be internally (δ_1, δ_0)-computable *(this definition is given coherently also for Y_0 enumerated).*

Since internal computable functions are internal objects, by transfer principle they satisfy the translations of all $\mathcal{L}_X$-properties satisfied by usual computable functions. In this way an extension of the notion of (computable) realization of mathematical functions for the case of internally represented or enumerated sets is obtained for free. A nonstandard version of the Main Theorem ([15]) is then given coherently. Further, we can prove the existence of a certain internally computable (multi)function, if we know that there is a standard (multi)function doing something similar. On the other hand, we can prove the existence of a standard computable (multi)function, provided that we know that there is an internally computable (multi)function that behaves similarly. In this way it is possible in principle to give (shorter?) nonstandard proofs of the existence of standard computable functions.

Nevertheless, the fundamental concepts of nonstandard analysis are external: standard element, limited, unlimited, infinitesimal... Therefore, to provide nonstandard computable versions of classical nonstandard theorems, like is currently done in computable analysis for classical mathematics, we need to consider the behavior of internally computable functions on external subsets of their domains. The idea is then to extend the notion of internal realization to the external subsets in the the following way. Let $(Y_1, \delta_1), (Y_0, \delta_0)$ be represented sets and let $Y_1' \subseteq {}^*Y_1, Y_0' \subseteq {}^*Y_0$ be possibly external. An internally computable function $F :\subseteq {}^*\mathbb{B} \to {}^*\mathbb{B}$ will be an *internal* (δ_1, δ_0)*-representation* of a nonstandard (multi)function $f :\subseteq Y_1' \rightrightarrows Y_0'$ if all ${}^*\delta_1$-names of any $y \in \mathrm{dom}(f)$ are mapped by F to some ${}^*\delta_0$-name of some $z \in f\{y\}$. In this case we say that f is *internally* (δ_1, δ_0)*-computable.* For Y_0 enumerable, analogous notions are immediately obtained.

3.1 Computability Properties of Spillover

We analyze some applications of internal computability to the basic nonstandard principle of spillover. The following theorem is a fundamental classical result:

Theorem 3. ***Robinson's Sequential Lemma.*** *Let $(s_N)_{N \in {}^*\mathbb{N}}$ be an internal sequence in ${}^*\mathbb{R}$ with $s_n \simeq 0$ for all $n \in \mathbb{N}$. There is then $M \in \mathbb{N}^\infty$ such that $s_N \simeq 0$ for all $N \leq M$.*

A proof of this result can be found in [11]. A different proof, which is more suitable for an "internal effectivization", has been given in [9]. Obviously, some adaptations must be made in order to let the hidden computational structure of the proof appear. Nevertheless, this is enough to let us think that the notion of internal computability is sometimes actually used in nonstandard analysis, even if in an intuitive and unexplicit form. This should not be a surprise, since it simply extends the notion of constructibility to the nonstandard world.

Theorem 4. *There is an internally computable multifunction f mapping each internal sequence $(s_N)_{N \in {}^*\mathbb{N}}$ in ${}^*\mathbb{R}$ with $s_n \simeq 0$ for all $n \in \mathbb{N}$ to hypernatural numbers $M \in \mathbb{N}^\infty$ such that $s_N \simeq 0$ for all $N \leq M$.*

Proof. For any $k \in \mathbb{N}$, consider the computable function which is described by the following program:

Stage $m \leq k$) If $m = k$ then output $k-1$. If $m < k$, test whether $s_m < \frac{1}{m-1}$. If this is the case, go to the next stage. But if $s_m > \frac{1}{m}$ then output $m-1$. Choose a priority rule in case both conditions are satisfied.

By transfer principle there is a similar internally computable function for a fixed $K \in \mathbb{N}^\infty$.

We show that the function so defined works for any hypersequence $(s_N)_{N \in {}^*\mathbb{N}}$ with $s_n \simeq 0$ for all $n \in \mathbb{N}$.

Notice that for all $n \in \mathbb{N}$: $s_n < \frac{1}{n}, \frac{1}{n-1}$. If the same property holds for all $M \leq K$, then for all such M: $s_M \simeq 0$. Indeed on the one hand, by hypothesis, $s_n \simeq 0$ for $n \in \mathbb{N}$, and on the other hand $\frac{1}{M-1} \simeq 0$ for $M \in \mathbb{N}^\infty$. Therefore, $K-1$ is a reliable output.

Otherwise, suppose that there is an $M < K-1$ such that $s_M \geq \frac{1}{M-1}$ or $s_M \geq \frac{1}{M}$. Thus $s_M \geq \frac{1}{M}$ anyway. In this case s_M is not supposed to be infinitesimal. Anyway, by transfer principle there is a least M such that $s_M \geq \frac{1}{M}$. This means that for all $I < M$: $s_I < \frac{1}{I} \simeq 0$. Therefore $M-1$ is anyway a suitable output. □

The adaptations introduced in the proof given in [9] are motivated essentially by the undecidability of $=$ in $\mathbb{R}$, and then in ${}^*\mathbb{R}$.

We now consider another example of spillover from the real field (see [8]):

Theorem 5. *Let an internal set $A \subseteq {}^*\mathbb{R}$ be given. If A has arbitrarily large limited members, then it has a positive unlimited member.*

Classically, this is a direct consequence of the internal Dedekind completeness. For the internally computational version, we consider only a special case of the statement, as commonly happens in standard computable analysis: very often only specific restrictions of a classical result are proved to be computable.

We use the representation $\rho_>$ from [15], for which a real number is denoted by all its strictly larger rational upper bounds.

Theorem 6. *There is an internally $(\psi_-, \rho_>)$-computable function f mapping any internal closed set $A \subseteq {}^*\mathbb{R}$ with arbitrarily large limited members to a positive unlimited member in it.*

Proof. Consider the following computable function. Fix $k \in \mathbb{N}$. Let then (any ψ_--name of) any closed set $C \subseteq \mathbb{R}$ be given. Observe that $C \cap [-k, k] = B$ is compact, and so $\max(B)$ exists. Moreover, the interval $[-k, k]$ is ψ_--computable, and so we can compute a ψ_--name of B, by Lemmas 5.1.10 and 5.1.13 in [15].

The set B is included in $[-k, k]$, and so by Definition 5.2.1 and Lemma 5.2.6 in [15], we can computably enumerate all $\alpha \in \mathbb{Q}$ with $\max(B) < \alpha$.

By transfer principle, there is an analogous internally computable function for any fixed $K \in \mathbb{N}^\infty$. Let then $A \subseteq {}^*\mathbb{R}$ closed and let $A \cap [-K, K] = D$. Then $\max(D)$ can be internally computably approximated by its strictly larger rational upper bounds and it is unlimited, since D contains arbitrarily large finite elements. □

3.2 Ross's Generalization of Bishop-Cheng Theorem

Given a Hausdorff space (Y, Υ), let $C_\mathcal{K}(Y)$ be the vector space of real continuous functions on Y with compact support, and $C^+_\mathcal{K}(Y)$ be the subvector space of the nonnegative elements of $C_\mathcal{K}(Y)$.

If $f \in C_\mathcal{K}(Y)$ and $g \in C^+_\mathcal{K}(Y)$, the function f is said to be *dominated* by g if $g(y) = 1$ whenever $f(y) \neq 0$, for all $y \in Y$.

In [13] the following theorem is proved by nonstandard reasoning:

Theorem 7. *Let Y be a Hausdorff space. Suppose that T is a nonnegative linear functional on $C_{\mathcal{K}}(Y)$, that $(f_n)_{n\in\mathbb{N}}$ is a sequence from $C^+_{\mathcal{K}}(Y)$ and that $\sum_{n=1}^{\infty} Tf_n < Tf_0$. If f_0 is dominated, there exists $y \in Y$ with $\sum_{n=1}^{\infty} f_n(y) < f_0(y)$.*

This statement generalizes an earlier result of constructive analysis by Bishop and Cheng ([2]), which was formulated for the case of locally compact metric spaces. Their original constructive proof is very difficult, and also the relatively simpler proofs given in [1] and [6] are not trivial to understand. D.A. Ross has recently provided two very simple nonstandard proofs of the more general Theorem 7 (a proof for the particular case of compact metric spaces was already included in [12]). About the second proof he gives, the author states that it is the kind of argument that a nonstandard analyst might call "constructive modulo an ultrafilter", as it does in fact produce (in some sense) a point y. Indeed the proof uses the extension to nonstandard entities of the principle, acceptable constructively (see [5]), that if $h \in C_{\mathcal{K}}(Y)$ and $Th < 0$, then a y with $h(y) < 0$ can be produced. This can be considered therefore as a nonstandard constructive method, and this approach to nonstandard constructiveness is quite close to our approach to nonstandard computability.

The second proof by Ross is interesting also for our investigation on internal computability, since it shows implicitly the existence of a certain internally computable multifunction, for the case of (locally compact) computable metric spaces (where any function with compact support is dominated and concrete representations of points are given). We obtain in this way a computable version of the original Bishop-Cheng Theorem. Some little addition in the proof of Ross must be made to reveal the hidden internally computable function described, as one gives evidence to what is computably implicit, but the whole reasoning can be strictly followed.

Definition 12. *For $\mathbf{Y} = (Y, d, \nu)$ a computable metric space, define the following representation $\delta^{\mathcal{K}}_{\mathbf{Y}}$ of $C_{\mathcal{K}}(Y)$:*

$$\delta^{\mathcal{K}}_{\mathbf{Y}}\langle q, r\rangle = f \in C_{\mathcal{K}}(Y) \iff \delta^{\rightarrow}_{\mathbf{Y},\mathbb{R}}(q) = f \wedge \kappa_{mc}(r) = K$$

where $q, r \in \mathbb{B}$ and K is the compact support of f.

When searching for internally computable versions of classical results, one of the major problems to face is the *external* nature of the basic notions of nonstandard analysis (like for example the concept of infinitesimal). As internal objects, the functions in ${}^*Comp$ cannot explicitly separate the external objects from the internal ones. Therefore some strategies must be adopted, and we "cheat" someway. In general, we define an internally computable function such that its outputs denote objects which satisfy our requirements (for example they are infinitesimals or standard numbers) but this meaningful interpretation of the outputs is possible only for "external observers". We did this in proofs of Theorems 4 or 6. Another method we suggest, which extends the previous one and may be more widely applicable, is that of obtaining more redundant outputs, with much useless information. For example, an output can be a list of labelled individuals, most of them are of no interest for us. Nevertheless, the relevant ones are supposed to be identified by labels satisfying precise requirements. Such requirements may involve external notions. Preferably, the labels could have finite length, so that they could (theoretically) be interpreted by an external observer in a single step, and the relevant solutions could be then immediately individuated.

The multifunction described in the following proof outputs lists of individuals, and uses standard natural numbers to label interesting elements, whereas the unlimited numbers are used to label irrelevant entities.

Theorem 8. ***Internally computable Bishop-Cheng Theorem***
Let $\mathbf{Y} = (Y, d, \nu)$ be a locally compact computable metric space, and T a non-negative linear functional on $C_{\mathcal{K}}(Y)$.

There is an internally computable multifunction which outputs internal sequences of pairs (y, N) for $y \in {}^*Y$ and $N \in {}^*\mathbb{N}$ with the following properties. If $(f_n)_{n\in\mathbb{N}}$ is a sequence from $C^+_{\mathcal{K}}(Y)$ such that $\sum_{n=1}^{\infty} Tf_n < Tf_0$, then for all pairs (y, N) with N standard: $\sum_{n=1}^{\infty} f_n(st(y)) < f_0(st(y))$. Moreover (in each output sequence) the set of enumerated pairs (y, N) with N standard is nonempty.

Proof. For brevity, for $m \in \mathbb{N}$, let:

$$\varphi_m(x) = \sum_{n=1}^{m} f_n(x) - f_0(x), \qquad \varphi(x) = \sum_{n=1}^{\infty} f_n(x) - f_0(x) \ ,$$

$$\alpha_m = \sum_{n=1}^{m} Tf_n - Tf_0 \qquad \alpha = \sum_{n=1}^{\infty} Tf_n - Tf_0 \ .$$

Let g dominate f_0, and let K be the compact support of f_0 (then $g = 1$ on K). Fix $M \in {}^*\mathbb{N}^{\infty}$. By hypothesis $\sum_{n=1}^{\infty} Tf_n < Tf_0$, and so $\alpha_m \to \alpha \in \mathbb{R}^-$. Hence ${}^*T\varphi_M = \alpha_M \simeq \alpha < 0$. Therefore for some (standard) $\epsilon > 0$, ${}^*T\varphi_M < -\epsilon Tg$, thus ${}^*T(\varphi_M + \epsilon {}^*g) < 0$. By nonnegativity of T there is some $y \in {}^*Y$ for which $\varphi_M(y) + \epsilon {}^*g(y) < 0$, thus $\varphi_M(y) < -\epsilon {}^*g(y)$. Since all f_n are non negative, then ${}^*f_0(y) \neq 0$, hence ${}^*g(y) = 1$ and $\varphi_M(y) < -\epsilon$. In other words, there is $y \in {}^*K$ such that

$$\varphi_M(y) < -\frac{1}{n} \qquad \text{for some standard } n \in \mathbb{N} \tag{1}$$

and in particular we can let n to be the smallest. Let a $\delta^K_{\mathbf{Y}}$-name of f_0 be given. By Theorem 3.8 and Proposition 4.2 in [4], in the standard world we can computably find a dense set A in K (which depends in general on the name of K). Moreover, by apply function, given any $m \in \mathbb{N}$ one can computably check for each $z \in A$ whether there is an $n \in \mathbb{N}$ such that $\varphi_m(z) < -\frac{1}{n}$. Suppose that for some z such n exists; there is then a smallest such n. There is a computable multifunction that maps z to n or $n-1$, as it looks computably for some n' such that $-\frac{1}{n'-1} < \varphi_m(z) < -\frac{1}{n'+1}$. We enumerate then all such pairs (z, n') in the output (where z is coded by a $\delta_{\mathbf{Y}}$-name and n' can denote itself).

By transfer principle, there is an analogous internal computable function which executes the test relatively to φ_M, for all points $z \in {}^*A$ and all $-\frac{1}{N}$ with $N \in {}^*\mathbb{N}$, and which lists all suitable pairs (z, N).

By transfer principle, without loss of generalization, the y satisfying (1) belongs to ${}^*A$. Then either (y,n) or $(y,n-1)$ are enumerated in the output for a standard n. We show then that $st(y)$ exists and satisfies the requirements. Indeed $y \in {}^*A \subseteq {}^*K$, thus y is near-standard by Robinson's compactness criterion, which means that $st(y)$ exists. Therefore, by continuity of φ_m for any standard $m > 0$, $\varphi_m(st(y)) \simeq {}^*\varphi_m(y) \leq \varphi_M(y) < -\frac{1}{n}$. Let $a = st(y)$. By transfer principle, for all standard $m \in \mathbb{N}$, $\mathbb{U}(X) \vDash \varphi_m(a) < -\frac{1}{n}$. Taking the supremum over all standard m, $\mathbb{U}(X) \vDash \varphi(a) \leq -\frac{1}{n} < 0$. □

We conclude this preliminary investigation on internal computability with an observation: nonstandard real numbers can be constructed through equivalence classes on Cauchy sequences modulo an ultrafilter. In particular, $\mathbb{R}$ and ${}^*\mathbb{R}$ have the same cardinality and a (concrete) representation $\delta :\subseteq \mathbb{B} \to {}^*\mathbb{R}$ is possible. Are there meaningful examples of such representations?

References

1. Bishop, E.A., Bridges, B.S.: Constructive Analysis. Springer, Heidelberg (1985)
2. Bishop, E.A., Cheng, H.: Constructive measure theory. Mem. American Mathematical Society. vol. 116 (1972)
3. Brattka, V.: Effective Borel measurability and reducibility of functions. Mathematical Logic Quarterly. 51, 19–44 (2005)
4. Brattka, V., Presser, G.: Computability on subsets of metric spaces. Theoretical Computer Science. 305, 43–76 (2003)
5. Bridges, D.S.: Constructive Functional Analysis. Pitman (1979)
6. Chan, Y.K.: A short proof of an existence theorem in constructive measure theory. Proceedings of the American Mathematical Society. 48, 435–437 (1975)
7. Friedman, H.: 6:Undefinability/Nonstandard Models, http://www.cs.nyu.edu/pipermail/fom/1997-November/000278.html
8. Goldblatt, R.: Lectures on the Hyperreals. Springer, Berlin-Heidelberg-New York (1998)
9. Loeb, P.A., Wolff, M. (eds.): Nonstandard Analysis for the Working Mathematician. Kluwer, Dordrecht-Boston-London (2000)
10. Palmgren, E.: Developements in constructive nonstandard analysis. The Bullettin of Symbolic Logic 4, 233–273 (1998)
11. Robinson, A.: Non-Standard Analysis. North-Holland, Amsterdam (1966)
12. Ross, D.A.: The constructive content of nonstandard measure existence proof - is there any? In: Berger, U., Osswald, P., Schuster, P. (eds.) Reunititing the antipodes - Constructive and Nonstandard Views of the Continuum, pp. 229–239. Kluwer Academic Publishers, Boston (2001)
13. Ross, D.A.: A nonstandard proof o a lemma from constructive measure theory. Mathematical Logic Quartely 52, 494–497 (2006)
14. Tanaka, K.: Nonstandard Analysis in WKL0. Mathematical Logic Quarterly vol. 43 (1997)
15. Weihrauch, K.: Computable Analysis. Springer, Heidelberg (2000)

Post's Problem for Ordinal Register Machines

Joel D. Hamkins[1] and Russell G. Miller[2,⋆]

[1] Department of Mathematics, College of Staten Island, and Doctoral Program in Mathematics, The CUNY Graduate Center

[2] Department of Mathematics, Queens College, and Doctoral Program in Computer Science, The CUNY Graduate Center

jhamkins@gc.cuny.edu

Russell.Miller@qc.cuny.edu

Abstract. We study Post's Problem for the ordinal register machines defined in [6], showing that its general solution is positive, but that any set of ordinals solving it must be unbounded in the writable ordinals. This mirrors the results in [3] for infinite-time Turing machines, and also provides insight into the different methods required for register machines and Turing machines in infinite time.

Keywords: computability, ordinal computability, ordinal register machine, Post's Problem.

1 Definitions

Ordinal register machines, or ORM's, are defined and described by Koepke and Siders in [6]. They generalize the traditional finite-time register machines: the registers are now allowed to contain any ordinal value, not just natural numbers, and the program runs through ordinal time, with its state at limit-ordinal stages determined in a natural way by taking liminf's of the cells and states at the preceding stages. Koepke and Siders proved that the sets of ordinals computable by an ORM, with finitely many ordinal parameters, are precisely the constructible sets of ordinals, i.e. those lying in Gödel's constructible universe L.

Since the ordinal register machine programs are finite, they each refer to only finitely many registers, and the memory used by any ORM algorithm is correspondingly limited to these fixed finite number of ordinal values at any time. This contrasts with the situation for the infinite time Turing machines of Hamkins and Lewis [2] and for the ordinal Turing machines of Koepke [5], where the algorithms can store information stretching out on an transfinite tape.

The original version of Post's Problem applied to finite-time Turing machines. It asked whether there exists a computably enumerable set A which is neither computable nor complete. That is, it required $\emptyset <_T A <_T \emptyset'$, where $\emptyset'$ is the

⋆ The authors were partially supported by several grants from the Research Foundation of CUNY. In addition, they are thankful for support provided by the Hausdorff Center in Mathematics at the University of Bonn in connection with the Bonn International Workshop on Ordinal Computability 2007.

jump of the empty set, or equivalently the Halting Problem for finite-time Turing machines. Post's Program for solving this problem was to discover a nonvacuous property of c.e. sets, expressible using only the containment relation $\subseteq$, which would guarantee that A was incomplete and noncomputable.

Post's Program was the genesis of the notions of simple, hypersimple, and hyperhypersimple sets, all of which properties Post originally hoped would fulfill his program. In fact, none of these properties implies incompleteness, and Post did not live to see the solution of the problem that bears his name. Post's Program was completed by Harrington and Soare [4] in 1991, but his Problem was solved much earlier, in 1956 and 1957, with the invention of the finite injury priority method (independently) by Friedberg [1] and Muchnik [8]. A good description of this method appears in section VII.2 of [10].

The notion of a computably enumerable subset of ω extends naturally to our context: a set of ordinals is *ORM-enumerable*, or *semidecidable*, if it is the domain (equivalently, the range) of some ORM-computable function. Shortly we will also define the jump operation for ORM's. Then we will ask the analogue of Post's Problem for sets of ordinals under computation by ORM's.

An ordinal α is ORM-*writable* if there is an ordinal register machine which, on input 0, halts and outputs α. Briefly, $\varphi_e(0)\downarrow = \alpha$ for some $e \in \omega$. Also, an ordinal σ is ORM-*clockable* if some computation $\varphi_e(0)$ halts after exactly σ steps. Notice that it is ORM-computable whether σ is clockable, since we can run all computations $\varphi_e(0)$ for σ-many steps and check whether any of them halted after exactly σ steps. On the other hand, the set of writable ordinals is ORM-enumerable, but not ORM-computable, essentially because a computation which halts after a very long number of steps could still have a relatively small ordinal as its output.

We write $\varphi_{e,\sigma}(\alpha)\downarrow$ to signify that the program φ_e converges on input α in *fewer than* σ steps. This notation is different from the common usage in finite-time computability theory, where $\varphi_{e,s}(n)\downarrow$ denotes convergence in $\leq s$ steps; our way is more appropriate in a context where we must deal with limit ordinals.

The *weak jump* $\emptyset^\Diamond$ of the empty set is the set

$$\emptyset^\Diamond = \{e \in \omega : \varphi_e(0)\downarrow\}.$$

We have approximations to the weak jump:

$$\emptyset^\Diamond_\sigma = \{e : \varphi_{e,\sigma}(0)\downarrow\};$$

these are nested upwards, and are computable uniformly in σ. Also notice that σ is clockable iff $\emptyset^\Diamond_\sigma \neq \emptyset^\Diamond_{\sigma+1}$, and that for limit ordinals λ, $\emptyset^\Diamond_\lambda = \cup_{\sigma<\lambda}\emptyset^\Diamond_\sigma$. (This would fail if we had kept the finite-time notation for $\varphi_{e,\lambda}(\alpha)\downarrow$.) We also have the *strong jump* $\emptyset^\blacklozenge$ of $\emptyset$ and its computable approximations $\emptyset^\blacklozenge_\sigma$:

$$\emptyset^\blacklozenge = \{\langle e, \alpha\rangle : \varphi_e(\alpha)\downarrow\} \subset \omega \times \mathrm{ON} \qquad\qquad \emptyset^\blacklozenge_\sigma = \{\langle e, \alpha\rangle : \varphi_{e,\sigma}(\alpha)\downarrow\}.$$

The strong jump is the actual halting problem in ORM-computability; the weak jump, roughly analogous to the jump in finite-time computability, is just the most convenient way to diagonalize and build a noncomputable set. In finite time, of course, the jump and the halting problem are computably isomorphic, but in our context this is no longer true; indeed $\emptyset^{\Diamond} <_{ORM} \emptyset^{\blacklozenge}$, under the standard definition of oracle computability for ordinal register machines.

The version of the following lemma for infinite time Turing machines was a significant result, proved by Philip Welch in [11], but in the ordinal register machine context we may observe it easily.

Lemma 1. *For ordinal register machines, the supremum γ of the clockable ordinals equals the supremum λ of the writable ordinals.*

Proof. Every clockable ordinal α is writable: just run the computation $\varphi_p(0)$ which halts after α steps, adjusting the machine so that at each step, it increments the ordinal in a new step register. When $\varphi_p(0)$ halts, transfer the contents of the step register to the output register and then halt. Thus $\gamma \leq \lambda$. Conversely, if $\alpha = \varphi_q(0)$ is writable, then $\varphi_q(0)$ takes at least α-many steps to halt, since after β steps, no register can contain any ordinal $> \beta$. Hence $\lambda \leq \gamma$. □

2 Post's Problem

Now we begin to consider Post's Problem. First we ask whether there exist relatively simple ORM-enumerable sets (subsets of ω, for instance) which are noncomputable and incomplete. The answer is no.

Theorem 1. *No subset $C \subseteq \omega$ satisfies $\emptyset <_{ORM} C <_{ORM} \emptyset^{\Diamond}$. Indeed, the same holds for subsets $C \subseteq \rho$, for any writable ordinal ρ.*

Proof. Consider a set $C \subseteq \rho$ with $\emptyset \leq_{ORM} C \leq_{ORM} \emptyset^{\Diamond}$ and ρ writable. This part of the proof is similar to that of Theorem 2.1 in [3], the corresponding result for infinite-time Turing machines.

Recall that $\emptyset^{\Diamond}_{\sigma} = \{e \in \omega : \varphi_{e,\sigma}(0)\downarrow\}$. This is a computable enumeration of the semidecidable set $\emptyset^{\Diamond}$. By assumption there is a program q such that $\varphi_q^{\emptyset^{\Diamond}}$ computes the characteristic function of C. This gives us a computable approximation to C:

$$C_\sigma = \{\beta < \rho : (\exists \delta \geq \sigma)[\varphi_{q,\delta}^{\emptyset^{\Diamond}_{\sigma}}(\beta)\downarrow = 1 \ \&\ (\forall \theta)[\sigma < \theta \leq \delta \implies \emptyset^{\Diamond}_{\sigma} = \emptyset^{\Diamond}_{\theta}]]\}.$$

Since $\varphi_q^{\emptyset^{\Diamond}}$ is the characteristic function of C, it must be total. Indeed, since $\emptyset^{\Diamond}_{\gamma} = \emptyset^{\Diamond}$, we must have $C_\sigma = C$ for all $\sigma \geq \gamma$. Therefore we can compute, uniformly in β and σ, whether $\beta \in C_\sigma$: having written ρ and checked that $\beta < \rho$, just run $\varphi_q^{\emptyset^{\Diamond}_{\sigma}}(\beta)$ until we reach a stage $\delta \geq \sigma$ such that either $\emptyset^{\Diamond}_{\delta} \neq \emptyset^{\Diamond}_{\sigma}$ (so $\beta \notin C_\sigma$) or $\varphi_{q,\delta}^{\emptyset^{\Diamond}_{\sigma}}(\beta)\downarrow$. If we find that this computation halts and outputs 1, before the approximation to $\emptyset^{\Diamond}$ changes, then $\beta \in C_\sigma$; if it outputs a different value, then $\beta \notin C_\sigma$. (In finite-time computability, the analogous process is the construction of a computable approximation to an arbitrary set $\leq_T \emptyset'$.)

Lemma 2. *With this approximation, if $C_\sigma \neq C_{\sigma+1}$, then $\sigma+1$ is clockable (and hence σ is writable).*

Proof. If $\emptyset^\Diamond_\sigma \neq \emptyset^\Diamond_{\sigma+1}$, then some computation $\varphi_e(0)$ converged in $(\sigma+1)$-many steps, so $(\sigma+1)$ is clockable. Otherwise, the definition of C_σ shows that $C_\sigma = C_{\sigma+1}$, as follows. To check whether some $\beta < \rho$ lies in C_σ, we find the least $\delta \geq \sigma$ for which either $\emptyset^\Diamond_\sigma \neq \emptyset^\Diamond_\delta$, or $\varphi^{\emptyset^\Diamond_\sigma}_{q,\delta}(\beta)\downarrow$. If $\delta \geq \sigma+1$, then we find the same δ when checking whether $\beta \in C_{\sigma+1}$, so the answer is the same. If $\delta = \sigma$, then $\varphi^{\emptyset^\Diamond_\sigma}_{q,\sigma}(\beta)\downarrow$, and hence $\varphi^{\emptyset^\Diamond_{\sigma+1}}_{q,\sigma+1}(\beta)\downarrow = \varphi^{\emptyset^\Diamond_\sigma}_{q,\sigma}(\beta)$ as well, since the oracle has not changed. □

We now consider two cases. First, if there exists some $\beta < \gamma(=\lambda)$ such that $C_\beta = C$, then C is ORM-computable. To compute C, we run some fixed program $\varphi_p(0)$ which writes the least writable ordinal $\delta \geq \beta$ and then computes C_δ. By Lemma 2, $C_\sigma = C_\beta = C$ for every σ with $\beta \leq \sigma < \delta$. But then $\emptyset^\Diamond_\delta = \emptyset^\Diamond_\beta$ as well, since $\emptyset^\Diamond_\delta$ contains those programs which halt *before* stage δ, and so the definition of C_σ shows that $C_\delta = C_\beta = C$. Hence we have computed C.

Otherwise there is no such β, and in this case $\emptyset^\Diamond \leq_{ORM} C$, since with a C-oracle we can search for the least σ such that $C_\sigma = C$. (Here we need to know that C is contained in ρ, as assumed by the theorem. We can write ρ, and then to verify that $C = C_\sigma$, we need only check that all $\alpha < \rho$ lie in C iff they lie in C_σ.) But by assumption, the σ we find is $\geq \gamma$ (in fact precisely γ), and when we find it, we write it on the output tape and halt. Thus γ is C-writable, and with γ it is easy to compute $\emptyset^\Diamond$. □

So Post's Problem has a negative solution when we restrict to sets C such that the supremum of C is writable. This is a large class of sets: it includes every non-cofinal subset of γ. However, without this restriction, the same problem has a positive solution. We prove this by a construction in the style of Friedberg and Muchnik. First we give a necessary lemma.

Lemma 3 (Reflection Lemma for ORM's). *Suppose that $\varphi^A_e(x)\downarrow = 0$, where x is a writable ordinal and A is a semidecidable set of ordinals, with ORM-computable enumeration $\langle A_\sigma \rangle$ such that $A_\gamma = A$. (This means that $A_\sigma \subseteq A_\tau$ for all $\sigma < \tau$, that $A_\beta = \cup_{\sigma<\beta} A_\sigma$ for limit ordinals β, and that there is a computable function $f(\alpha,\sigma)$ with value 1 if $\alpha \in A_\sigma$ and 0 if not.) Assume also that every σ with $A_\sigma \neq A_{\sigma+1}$ is clockable. Then for every β less than the supremum γ of the clockable ordinals, there exists a clockable limit ordinal $\tau > \beta$ such that $\varphi^{A_\tau}_{e,\tau}(x)\downarrow = 0$.*

The same would hold if we replaced "clockable" by "writable" throughout the last paragraph. However, clockability will be the property we need.

Proof. Our proof mirrors that of the Reflection Lemma for infinite-time Turing machines (Lemma 4.3 in [3]). Consider the algorithm which, on input 0, writes x and then searches and outputs the least nonclockable ordinal $\sigma > \beta$ such that $\varphi^{A_\sigma}_{e,\sigma}(x)\downarrow = 0$.

Now $A_\gamma = A$, and since $\varphi_e^A(x)\downarrow= 0$, there are plenty of ordinals $\sigma > \gamma > \beta$ for which $\varphi_{e,\sigma}^{A_\sigma}(x)\downarrow= 0$. But our algorithm runs on input 0 with no oracle, so its own halting time is clockable and greater than its output. Therefore there must exist a nonclockable stage σ between β and γ such that $\varphi_{e,\sigma}^{A_\sigma}(x)\downarrow= 0$. Let τ be the least clockable ordinal $> \sigma$. Then τ is a limit ordinal and $A_\sigma = A_\tau$, by our condition on changes to the enumeration of A, so this τ satisfies the lemma. □

Theorem 2. *There exist ORM-incomparable enumerable sets A and B of ordinals. It follows that $\emptyset <_{ORM} A <_{ORM} \emptyset^\Diamond (<_{ORM} \emptyset^\blacklozenge)$, and likewise for B.*

Proof. We build the ORM-enumerable sets A and B as follows, to satisfy the usual Friedberg-Muchnik requirements for all $e \in \omega$:

$$\mathcal{R}_e : \ (\exists x_e)[\varphi_e^A(x_e)\downarrow= 0 \text{ iff } x_e \in B] \qquad \mathcal{S}_e : \ (\exists y_e)[\varphi_e^B(y_e)\downarrow= 0 \text{ iff } y_e \in A].$$

These have the standard priority ranking: each $\mathcal{R}_e$ has higher priority than $\mathcal{S}_e$, which has higher priority than $\mathcal{R}_{e+1}$. At each stage σ we will have an approximation $x_{e,\sigma}$ to x_e, which converges to x_e as σ grows, and similarly for y_e.

Set $A_0 = B_0 = \emptyset$, and start with approximation $x_{e,0} = y_{e,0} = e$ to the witness elements x_e and y_e. We will redefine $x_{e,\sigma+1} \neq x_{e,\sigma}$ at only finitely many stages σ, namely those stages at which $\mathcal{S}$-requirements of higher priority than $\mathcal{R}_e$ act, and the same holds symmetrically for $y_{e,\sigma}$.

If τ is a limit ordinal, define $A_\tau = \cup_{\sigma<\tau} A_\sigma$, with $x_{e,\tau} = \lim_{\sigma\to\tau} x_{e,\sigma}$, and similarly for B_τ and $y_{e,\tau}$. (Notice that each $x_{e,\sigma}$ is eventually constant as σ approaches τ, so these limits exist.)

For successor ordinals $\tau = \sigma+1$, if σ is either unclockable or a successor ordinal itself, then we preserve the settings at stage $\sigma + 1$: $A_{\sigma+1} = A_\sigma$, $x_{e,\sigma+1} = x_{e,\sigma}$, and so on. Only if σ is a clockable limit ordinal do we act at the successor stage $\sigma + 1$, as follows.

At such a stage $\sigma + 1$, we fix the highest-priority requirement, say $\mathcal{R}_e$, which requires attention, by which we mean that $\mathcal{R}_e$ is not yet satisfied and $x_{e,\sigma} \le \sigma$ and $\varphi_{e,\sigma}^{A_\sigma}(x_{e,\sigma})\downarrow= 0$. We act by enumerating $x_{e,\sigma}$ into $B_{\sigma+1}$, with $A_{\sigma+1} = A_\sigma$. We then set $x_{i,\sigma+1} = x_{i,\sigma}$ for all $i \in \omega$, and $y_{i,\sigma+1} = y_{i,\sigma}$ for all $i < e$. The lower-priority $\mathcal{S}$-requirements are injured at this stage, for we redefine

$$y_{e+j,\sigma+1} = y_{e+j,\sigma} + (x_{e,\sigma} + \sigma) + j + 1$$

for each $j \in \omega$. Notice that since $x_{e,\sigma} \le \sigma$, we will have $B_{\sigma+1} \subseteq (\sigma + 1)$ (by induction on the preceding stages). If there is no requirement $\mathcal{R}_e$ which requires attention at stage $\sigma + 1$, then we preserve all settings from stage σ.

If the highest-priority requirement requiring attention at stage $\sigma + 1$ is an $\mathcal{S}$-requirement, say $\mathcal{S}_e$, we simply interchange A with B and the x-witnesses with the y-witnesses in the above paragraph. At this stage, we preserve $y_{i,\sigma+1} = y_{i,\sigma}$ for all i, and $x_{i,\sigma+1} = x_{i,\sigma}$ for all $i \le e$, with

$$x_{e+j,\sigma+1} = x_{e+j,\sigma} + (y_{e,\sigma} + \sigma) + j$$

for each $j > 0$ in ω, but not for $j = 0$. This reflects the fact that $\mathcal{R}_e$ has higher priority than $\mathcal{S}_e$. Otherwise, the entire process is symmetric in A and B.

The point of the redefinition of the y-elements when we satisfy $\mathcal{R}_e$ is that since the computation $\varphi_e^{A_\sigma}(x_{e,\sigma}) = 0$ converged in $\leq \sigma$ steps, the only oracle questions it can have asked involved membership of ordinals $\leq x_{e,\sigma} + \sigma$ in the oracle set. The elements which may later enter A are the witness elements $y_{i,\rho}$ at stages $\rho > \sigma$. By redefining $y_{e+j,\sigma+1} > x_{e,\sigma} + \sigma$ for all $j \geq 0$ at stage $\sigma + 1$, we ensure that any of these elements that later enters A will not change the oracle computation $\varphi_e^A(x_{e,\sigma}) = 0$, since the computation cannot have asked whether such large elements were in A. (It is possible for $y_{e+j,\rho}$ to be redefined yet again at a stage $\rho + 1 > \sigma + 1$, but our formula also ensures that it is always redefined to be larger than it had been before.) Of course, if a higher-priority y-witness element later enters A, it could change this computation, but the usual finite-injury argument shows that eventually we will reach a stage after which no higher-priority requirement acts again. So, using induction on the requirements according to their priority, we see that for every e, the witness elements $x_e = \lim_\sigma x_{e,\sigma}$ and $y_e = \lim_\sigma y_{e,\sigma}$ exist, and that if $x_e \in B$, then $\varphi_e^A(x_e)\!\downarrow = 0$, and symmetrically.

A further argument is necessary for the converse, using the Reflection Lemma 3. The point of restricting our construction to clockable stages was to ensure that every witness element $x_{e,\sigma}$ and $y_{e,\sigma}$ at every stage is a writable ordinal, i.e. equal to $\varphi_p(0)$ for some program p. The clockable limit stages were chosen precisely because, being clockable, they were writable, as are their successors and the stage 0. (Also, clockability is easier to check than writability!) Then, if we redefined any $x_{i,\sigma+1}$ at stage $\sigma + 1$, we set it equal to a sum of writable ordinals, and similarly for $y_{e,\sigma+1}$. So, by induction on stages, all witness elements are writable, and thus all elements of A and B are writable. But we can write the stage at which a writable ordinal enters a semidecidable set, so $A_\gamma = A$ and $B_\gamma = B$, as required by the Reflection Lemma.

Now suppose that x_e never entered B. Then for all sufficiently large clockable limit ordinals δ, $\varphi_{e,\delta}^{A_\delta}(x_e)$ either diverges or converges to a nonzero value, so by the Reflection Lemma, the full computation $\varphi_e^A(x_e)$ cannot converge to 0. Thus again x_e witnesses that $\varphi_e^A \neq B$, so $B \not\leq_{ORM} A$. A symmetric result holds for y_e, so A and B are ORM-incomparable sets.

We have called this process a construction, and indeed it enumerated elements into A and B, without ever removing them. Nevertheless, it remains to show that A and B are actually ORM-enumerable, of course, since the description above did not use ORM's. The heart of the proof of Theorem 2 is the following lemma.

Lemma 4. *There exists an ORM which decides, for arbitrary ordinal inputs α and σ, whether $\alpha \in A_{\sigma+1} - A_\sigma$; and similarly for B. We refer to the algorithms for these machines as the* entry algorithms *for A and B.*

Proof. We use an ordinal stack machine, which is a special type of ordinal register machine. In an ordinal stack machine, along with finitely many registers, we have finitely many *stacks*, each of which (at any single stage of operation) consists of a descending (hence finite) sequence of ordinals. We can push a new ordinal from a register onto one of these stacks, provided that it is strictly less than all ordinals currently on the stack; and we can pop the smallest ordinal off of any stack and transfer it to a register, where it can be compared to the contents of other registers, etc., by the usual register operations. In [6], Koepke and Siders use the Cantor normal form of an ordinal to prove that the functions on ordinals computable by ordinal stack machines are precisely those computable by regular ORM's, so we may prove our lemma by giving two stack-machine programs (one for A, one for B) which answer the question for arbitrary α and σ.

Our stack machine accepts the inputs α and σ. It immediately pushes σ on top of its stage stack, and pushes the ordinal $(\omega^\sigma + \alpha)$ on top of its input stack. These will stay on these stacks for the rest of the operation of this machine, but occasionally the entry algorithm will call itself and push smaller ordinals above them, which are then popped off the stack when the subroutine ends. Of course, we have access to the value σ from the top of the stage stack any time we need it, and from the two stacks together we can also compute the value of α whenever we need it.

The reason for not simply pushing α onto the input stack is that the entry algorithm may later call itself, with inputs τ and β. We will ensure that $\tau < \sigma$, but we may have to allow $\beta \geq \alpha$, in which case we could not push β onto a stack above α. However, pushing $\omega^\tau + \beta$ onto the stack above $\omega^\sigma + \alpha$ will be legal, because we will always have $\beta < \sigma$.

We give the details for the entry algorithm which decides whether α entered B at stage $\sigma + 1$. The entry algorithm for A is quite similar, of course, and in fact is used by the algorithm for B. Given α and σ, we execute the following steps.

1. If $\alpha \geq \sigma$, or if σ is not a clockable limit stage, then output 0. Otherwise, go on. (In our construction, a witness element α only enters B at successors of clockable limit stages $> \alpha$. These properties are indeed ORM-decidable.)
2. For each $e < \omega$, check whether $x_{e,\sigma} = \alpha$. If such an e exists, then go on to Step 3. Notice that if e exists, then it is unique, and we can always use it in subsequent steps by using this same process to search for it again. If no such e exists, output 0. (Only witness elements for $\mathcal{R}$-requirements ever enter B. We prove in Lemma 5 that we can compute $x_{e,\sigma}$ uniformly from e and σ.)
3. With the e from the previous step, we now simulate the operation of the program φ_e on input α with oracle A_σ for σ steps. Of course, we have no A_σ-oracle, so whenever our simulation asks whether some $\beta < \sigma$ lies in $\mathcal{A}_\sigma$, we simply use the entry algorithm to check (for all τ with $0 \leq \tau < \sigma$, starting with 0) whether β entered A at stage τ. Notice that this is allowed by our stack machine, since each such τ and β is strictly less than σ, even though the β might be $> \alpha$. (The construction ensured that $A_\sigma \subseteq \sigma$, so if the simulation asks whether some $\beta \geq \sigma$ lies in A_σ, we answer "no" immediately, without using the entry algorithm to check.) Thus we can determine whether $\varphi_{e,\sigma}^{A_\sigma}(\alpha)\downarrow = 0$. If not, output 0; if so, go on.
4. Now run the entry algorithm with α and with each stage $\tau < \sigma$, to see if $\alpha \in B_\sigma$. If so, output 0, since $\alpha \notin B_{\sigma+1} - B_\sigma$. If not, go on.
5. Now compute $x_{i,\sigma}$ for each $i < e$, and run the entry algorithm with each such $x_{i,\sigma}$ and with stage σ to determine whether any higher-priority requirement

than $\mathcal{R}_e$ enumerated any element into B at stage $\sigma+1$. For this we do not push σ onto the stage stack, because to do so would be illegal. Nor do we pop it off that stack at the end, because we want it to stay there until the end of the run of the entry algorithm on α and σ. However, we do push $(\omega^\sigma + x_{i,\sigma})$ onto the input stack, which is legal since $x_{i,\sigma} < x_{e,\sigma}$ for $i < e$, and take it off again when we move on to the next i. If we find any such $x_{i,\sigma}$ which entered B at stage $\sigma+1$, then output 0.

Also, we compute $y_{i,\sigma}$ for each $i < e$ and use the entry algorithm for A to determine whether any such $y_{i,\sigma}$ entered A at stage $\sigma+1$. Here we must be careful, because this run of the entry algorithm for B might have been called by the entry algorithm for A. So we check the top (i.e. smallest) element of the stage stack from the entry algorithm for A. If that element equals σ, then we leave it there, and just push $\omega^\sigma + y_{i,\sigma}$ onto the input stack, which is safe, because if the entry algorithm for A was already running and needed to know about $x_{e,\sigma}$ entering B, it must have been asking about an element $y_{j,\sigma}$ with $j \geq e > i$, so that $\omega^\sigma + y_{j,\sigma} > \omega^\sigma + y_{i,\sigma}$.

When the entry algorithm for A concludes, we find σ on top of its stage stack and consider the top two elements, say $\beta < \delta$, on its input stack. (If there is only one element β on the input stack, then we pop β off its stack and σ off its stack, leaving nothing on either stack.) Now β must equal $\omega^\sigma + y_{i,\sigma}$. Then we pop δ off the input stack and find the largest term of its Cantor normal form, say $\delta = \omega^\tau + \theta$ for some τ and θ. If $\tau = \sigma$, then we leave δ on top of the input stack and σ on top of the stage stack. If $\tau \neq \sigma$, then we leave δ on top of the input stack, but remove σ from the stage stack. Thus, even though we left no specific indicator, when calling the entry algorithm for A, of whether we had added a new σ on top of the stage stack or not, we have still determined at the conclusion of that algorithm whether or not a new σ had been added, and if it had, then we have now removed it again.

Having completed the entry algorithm for A, we now know whether any $x_{i,\sigma}$ with $i < e$ entered A at stage $\sigma+1$. If so, then output 0; if not, then output 1. This completes the entry algorithm for B.

(If any higher-priority requirement $\mathcal{R}_i$ or $\mathcal{S}_i$ enumerated an element into B at stage $\sigma+1$, then $\mathcal{R}_e$ would not enumerate an element of its own. If not, then all conditions are satisfied for $\alpha = y_{e,\sigma}$ to enter B at stage $\sigma+1$.)

We define an analogous entry algorithm to check whether an arbitrary α entered A at an arbitrary stage $\sigma+1$, of course. Indeed, these two algorithms call each other in certain cases, as detailed in Item 5. Because $\mathcal{R}_e$ has higher priority than $\mathcal{S}_e$, the entry algorithm for A must check in Item 5 whether any $x_{i,\sigma}$ with $i \leq e$ entered B at stage $\sigma+1$, rather than just checking for $i < e$. Otherwise, the entry algorithms are symmetric in A and B.

Both entry algorithms are computable by stack machines, and the proof that they give the correct answer is a fairly simple double induction, first on stages σ, and for each individual stage, on inputs $\alpha < \sigma$. The one twist to be noted is that we promised a separate algorithm to compute $x_{e,\sigma}$ for arbitrary e and σ;

and since this algorithm is used in the entry algorithm, it can only call the entry algorithm for smaller stages.

Lemma 5. *There are ORM's which compute $x_{e,\sigma}$ and $y_{e,\sigma}$, uniformly in e and σ, We refer to their programs as the* witness algorithms *for A and B. These ORM's use the entry algorithm from Lemma 4, but they only push stages $< \sigma$ onto the stage stack. (That is, they only ask whether elements entered A or B at stages $\tau + 1$ with $\tau < \sigma$.)*

Proof. The witness algorithm is allowed to call itself, but only with descending stages, just like the entry algorithms. To compute $y_{e,\sigma}$, we start with $e = y_{e,0}$ in the output register of a stack machine. Then go through all stages τ with $\tau+1 \leq \sigma$. At each such τ, we use the witness algorithm to compute $x_{0,\tau}, \ldots, x_{e,\tau}$ and then the entry algorithm to check whether any $\beta = x_{i,\tau}$ with $i \leq e$ entered B at stage $\tau + 1$. If so, then $y_{e,\tau+1}$ will have been redefined to equal $y_{e,\tau} + \beta + \tau + (e - i) + 1$, so we write this new value in the output register in place of $y_{e,\tau}$. Otherwise we leave the output register as it is. When $\tau + 1$ is finally $\geq \sigma$, the value in the output register will be $y_{e,\sigma}$.

Notice that the corresponding routine for computing $x_{e,\sigma}$ asks only about elements $y_{i,\tau}$ with $i < e$. This mirrors the priority ranking of the requirements, and is important for seeing that the inductive argument for correctness of this algorithm is well-founded. This proves Lemma 5, and also Lemma 4. □

With Lemma 4, it is clear that the sets A and B are ORM-enumerable. A, for instance, is the domain of the ORM-computable function which, on input α, starts with $\sigma = 0$, asks for each σ in turn whether α enters A at stage $\sigma + 1$, and halts, putting α in the domain, if it ever receives a positive answer.

ORM-enumerability immediately implies that $A \leq_{ORM} \emptyset^{\blacklozenge}$, but we promised that $A \leq_T \emptyset^{\lozenge}$. Given any ordinal x, we check first whether x is clockable. If not, then $x \notin A$. Otherwise, there is an ORM program q which writes x and then gives x as an input to the ORM-computable function whose domain is A. We can compute this q uniformly from x, and $x \in A$ iff $q \in \emptyset^{\lozenge}$. The same proof works for B, so we have a pair of ORM-incomparable semidecidable sets below $\emptyset^{\lozenge}$, as Theorem 2 claimed. □

3 Softer Proofs of the Result

In the preceding argument, in order to show that A and B are ORM-enumerable, we provided a detailed ORM procedure for deciding the entry algorithms for the sets A and B. We are pleased to have done so, and we believe that the procedure helps illustrate several useful techniques of ORM computation. Nevertheless, this part of the argument can be completely eliminated by making use of the main theorem of [6]. The structure of our previous argument was first to provide a set-theoretical definition of the sets A and B in terms of their approximations A_σ and B_σ, which ensured that A and B would be ORM-incomparable, and then to provide a detailed ORM algorithm to decide the entry problems $\alpha \in A_{\sigma+1} - A_\sigma$

and $\alpha \in B_{\sigma+1} - B_\sigma$, which ensured that A and B would be ORM-enumerable. The point we would like to make now is that once we have given the initial set-theoretic construction of the sets A_σ and B_σ, we can simply observe that this construction can be carried out inside Gödel's constructible universe L, in such a way that the construction is absolute to initial segments of L. That is, if some level of the constructibility hierarchy L_η satisfies that an ordinal α has entered A at stage σ, then this is truly the case. Therefore, in order to determine whether or not α enters A at stage σ, we need only search for an ordinal η such that L_η satisfies the set-theoretical assertion "α enters A at stage σ". But the question of whether L_η satisfies a given set-theoretical assertion $\varphi(\alpha, \sigma)$ is uniformly ORM-decidable from input $(\eta, \alpha, \sigma, \ulcorner\varphi\urcorner)$, by the main result of [6]. Consequently, both A and B are ORM-enumerable, as desired.

In addition, we would like to mention that an even softer argument can be made by appealing to the Sacks-Simpson [9] solution of Post's problem in α-recursion theory. It has been observed by Koepke, after the Bonn International Workshop on Ordinal Computability 2007, that for any admissible ordinal α, the subsets of α that are computable in α-recursion theory are exactly the sets that are ORM-computable in time less than α. Using the admissible ordinal γ, the supremum of the ORM-clockable ordinals, one may now transfer the solution of Post's problem from γ-recursion theory over to ordinal register machines. Koepke provides the details of this idea in [7] in the case of ordinal Turing machines, but explains how this argument also applies to ordinal register machines.

References

1. Friedberg, R.M.: Two recursively enumerable sets of incomparable degrees of unsolvability. In: Proc. Nat. Acad. Sci (USA) vol. 43, pp. 236–238 (1957)
2. Hamkins, J.D., Lewis, A.: Infinite-time Turing machines. Journal of Symbolic Logic 65(2), 567–604 (2000)
3. Hamkins, J.D., Lewis, A.: Post's problem for supertasks has both positive and negative solutions. Arch. Math. Logic 41, 507–523 (2002)
4. Harrington, L., Soare, R.I.: Post's Program and Incomplete Recursively Enumerable Sets. In: Proc. Nat. Acad. Sci (USA) vol. 88, pp. 10242–10246 (1991)
5. Koepke, P.: Turing computations on ordinals. Bulletin of Symbolic Logic 11(3), 377–397 (2005)
6. Koepke, P., Bissell-Siders, R.: Ordinal Register Machines (to appear)
7. Koepke, P.: α-Recursion theory and ordinal computability, electronic manuscript
8. Muchnik, A.A.: On the Unsolvability of the Problem of Reducibility in the Theory of Algorithms. Dokl. Akad. Nauk SSSR, N.S (Russian) 109, 194–197 (1956)
9. Sacks, G.E., Simpson, S.G.: The α-finite injury method. Annals of Mathematical Logic 4, 343–367 (1972)
10. Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, New York (1987)
11. Welch, P.: The lengths of infinite time Turing machine computations. Bulletin of the London Mathematical Society 32(2), 129–136 (2000)

Unique Existence and Computability in Constructive Reverse Mathematics

Hajime Ishihara

School of Information Science
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa 923-1292, Japan
ishihara@jaist.ac.jp

Abstract. We introduce, and show the equivalences among, relativized versions of Brouwer's fan theorem for detachable bars (FAN), weak König lemma with a uniqueness hypothesis (WKL!), and the longest path lemma with a uniqueness hypothesis (LPL!) in the spirit of constructive reverse mathematics. We prove that a computable version of minimum principle: if f is a real valued computable uniformly continuous function with at most one minimum on $\{0,1\}^{\mathbf{N}}$, then there exists a computable α in $\{0,1\}^{\mathbf{N}}$ such that $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$, is equivalent to some computably relativized version of FAN, WKL! and LPL!.

Keywords: unique existence, computability, Brouwer's fan theorem, weak König lemma, constructive mathematics, reverse mathematics.

1 Introduction

The purpose of constructive reverse mathematics [8] is to classify various theorems in intuitionistic, constructive recursive and classical mathematics by logical principles, function existence axioms and their combinations. Classifying mathematical theorems means finding logical principles and/or function existence axioms which are not only sufficient but also necessary to prove the theorems in a weak system. An informal approach [9] to constructive reverse mathematics, that is reverse mathematics in Bishop's constructive mathematics [3,4,6], seems to have started in Julian and Richman [11] proving that Brouwer's fan theorem for detachable bars is equivalent to a positivity property: every positively valued uniformly continuous function on $[0,1]$ has a positive infimum. (See Veldman [21] and others [16] for a similar program of intuitionistic reverse mathematics.)

Ishihara [7] showed in the context of constructive reverse mathematics that the statement

MIN: *every real valued uniformly continuous function f on a compact metric space X attains its infimum, that is, there exists an x in X such that $f(x) = \inf f(X)$,*

is equivalent to weak König's lemma

WKL: *every infinite binary tree has an infinite path.*

(The corresponding result in Friedman-Simpson program of (classical) reverse mathematics can be found in Simpson [19, IV.2].) Here a *binary tree* T is an inhabited subset of the set $\{0,1\}^*$ of finite binary sequences such that it is detachable in the sense that there exists a characteristic function χ_T of T (and hence $\forall a \in \{0,1\}^*(a \in T \vee a \notin T)$, which does not hold constructively in general), and downward closed with respect to the predecessor relation $\preceq$ defined, using the concatenation function $*$, by $a \preceq b := \exists c \in \{0,1\}^*(a * c = b)$. A binary tree T is *infinite* if for each n there is a in T with $|a| = n$, where $|a|$ denotes the length of a, and a binary sequence $\alpha \in \{0,1\}^{\mathbf{N}}$ is an *infinite path* of T if each finite initial segment $\overline{\alpha}n := (\alpha(0), \ldots, \alpha(n-1))$ of α is in T. (We will use a similar notation for a finite sequence $a = (a_0, \ldots, a_{m-1})$, that is, $\overline{a}n$ stands for $(a_0, \ldots, a_{n-1})$ if $n \leq m$, and a otherwise.)

A real valued function f on a metric space X *has at most one minimum* if

$$\forall x, y \in X[x \neq y \rightarrow \exists z \in X(f(z) < f(x) \vee f(z) < f(y)].$$

Note that if $m := \inf f(X)$ exists, the above condition is equivalent to the following condition in Berger et al. [1]:

$$\forall x, y \in X[x \neq y \rightarrow m < f(x) \vee m < f(y)],$$

and that the real valued function $f : y \mapsto d(x, y)$ on an inhabited subset S of a metric space (X, d) has at most one minimum if and only if x has at most one best approximation in S in the sense of Bridges [5] (see also [4, 7.2.11] and [1]). Berger et al. [1] showed that MIN with the uniqueness hypothesis

MIN!: *every real valued uniformly continuous function with at most one minimum on a compact metric space attains its infimum*

is equivalent to Brouwer's fan theorem for detachable bars

FAN: *every detachable bar is uniform.*

Here a *bar* B is a subset of $\{0,1\}^*$ such that for each α in $\{0,1\}^{\mathbf{N}}$ there exists n with $\overline{\alpha}n \in B$, and B is *uniform* if there exists n such that $\exists k \leq n(\overline{\alpha}k \in B)$ for all $\alpha \in \{0,1\}^{\mathbf{N}}$. As a corollary of the above results, we can see the implication from WKL to FAN, which was first proved indirectly in [7], and then directly in [10].

A binary tree T *has at most one path* if

$$\forall \alpha, \beta \in \{0,1\}^{\mathbf{N}}[\alpha \neq \beta \rightarrow \exists n(\overline{\alpha}n \notin T \vee \overline{\beta}n \notin T)].$$

Berger and Ishihara [2] showed indirectly that the equivalence between FAN and WKL with the uniqueness hypothesis

WKL!: *every infinite binary tree with at most one path has an infinite path.*

A nice direct proof of the equivalence between FAN and WKL! can be found in Schwichtenberg [18]. Schuster [17] dealt with unique solutions in the context of constructive reverse mathematics.

In this paper, we introduce relativized versions of FAN, WKL! and the longest path lemma with the uniqueness hypothesis

LPL!: every binary tree with at most one path has a longest path,

where a *longest path* of a binary tree T is an infinite binary sequence such that $\forall a \in \{0,1\}^*(a \in T \rightarrow \overline{\alpha}|a| \in T)$, and prove the equivalences among them in the spirit of constructive reverse mathematics. (Note that the equivalence between WKL and the longest path lemma

LPL: every binary tree has a longest path

was proved in [10].) Then we show that the following computable version of MIN! for the Cantor space $\{0,1\}^{\mathbf{N}}$:

> *if f is a real valued computable uniformly continuous function with at most one minimum on the Cantor space $\{0,1\}^{\mathbf{N}}$, then there exists a computable α in $\{0,1\}^{\mathbf{N}}$ such that $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$*

is equivalent to some (classically true) computably relativized versions of FAN, WKL!, and LPL!. Note that finding a zero of a real valued function g on $\{0,1\}^{\mathbf{N}}$ with $\inf |g(\{0,1\}^{\mathbf{N}})| = 0$ is reducible to MIN, and hence this result is related to the results in 6.3, especially Corollary 6.3.5, of [22].

2 Relativized Versions of FAN, WKL! and LPL!

Let $\mathcal{C}$ be a subset of $\mathbf{N}^{\mathbf{N}}$. Then $\alpha \in \mathbf{N}^{\mathbf{N}}$ is *computable in* $\mathcal{C}$ if there exist an index e and $\beta \in \mathcal{C}$ such that

$$\forall n \exists z[T(e, \beta, n, z) \wedge U(z) = \alpha(n)],$$

where T is Kleene's T-predicate and U is the result-extracting function in [20, 3.7.6], and α is *computable* if it is computable in $\{\lambda x.0\}$. We say that $\mathcal{C}$ is *computably closed* if every $\alpha \in \mathbf{N}^{\mathbf{N}}$ computable in $\mathcal{C}$ is in $\mathcal{C}$. Let *Rec* be the smallest computably closed inhabited subset of $\{0,1\}^{\mathbf{N}}$, that is, consisting of all computable $\alpha \in \{0,1\}^{\mathbf{N}}$.

A subset $\mathcal{B}$ of $\{0,1\}^{\mathbf{N}}$ is *full* if $\{0,1\}^* \subseteq \{\overline{\alpha}n \mid \alpha \in \mathcal{B} \wedge n \in \mathbf{N}\}$. Note that every computably closed inhabited subset of $\{0,1\}^{\mathbf{N}}$ is full.

Let $\mathcal{B}$ be a full subset of $\{0,1\}^{\mathbf{N}}$. Then a detachable subset B of $\{0,1\}^*$ is a *bar in* $\mathcal{B}$ if for each α in $\mathcal{B}$ there exists n with $\overline{\alpha}n \in B$. Similarly, we may say a uniform bar B in $\mathcal{B}$ if there exists n such that $\exists k \leq n(\overline{\alpha}k \in B)$ for all $\alpha \in \mathcal{B}$. But, if a bar B is uniform in $\mathcal{B}$, then, since $\mathcal{B}$ is full, we have

$$(*) \quad \exists n \forall a \in \{0,1\}^*[|a| = n \rightarrow \exists k \leq n(\overline{a}k \in B)].$$

Conversely, if $(*)$ holds, then B is uniform in any full subset of $\{0,1\}^{\mathbf{N}}$. Therefore, the notion of the uniformity on bars is independent of an underlying full set, and so we adopt $(*)$ as the definition of the uniformity.

Let $\mathcal{B}$ and $\mathcal{D}$ be subset of $\{0,1\}^{\mathbf{N}}$ such that $\mathcal{B}$ is full. Then we have the following relativized version of Brouwer's fan theorem for detachable bars.

$\mathrm{FAN}_{\mathcal{D}}(\mathcal{B})$: Every $\mathcal{D}$-bar in $\mathcal{B}$ is uniform,

where $\mathcal{D}$-bar is a detachable bar whose characteristic function is in $\mathcal{D}$ (here we assume a coding of finite binary sequences into natural numbers). Similarly, we will say $\mathcal{D}$-tree for a binary tree whose characteristic function is in $\mathcal{D}$. Note that Kleene [12,13] showed that $\mathrm{FAN}_{Rec}(Rec)$ is refutable (see also [20, 4.7.6]).

A binary tree T *has at most one path in* $\mathcal{B}$ if

$$\forall\alpha,\beta\in\mathcal{B}[\alpha\neq\beta\rightarrow\exists n(\overline{\alpha}n\notin T\vee\overline{\beta}n\notin T)].$$

Similar to FAN, we have the following relativized version of WKL! and LPL!.

$\mathrm{WKL!}_{\mathcal{D}}(\mathcal{B})$: *Every infinite $\mathcal{D}$-tree with at most one path in $\mathcal{B}$ has an infinite path in $\mathcal{B}$.*

$\mathrm{LPL!}_{\mathcal{D}}(\mathcal{B})$: *Every $\mathcal{D}$-tree with at most one path in $\mathcal{B}$ has a longest path in $\mathcal{B}$.*

Before discussing relations among $\mathrm{FAN}_{\mathcal{D}}(\mathcal{B})$, $\mathrm{WKL!}_{\mathcal{D}}(\mathcal{B})$ and $\mathrm{LPL!}_{\mathcal{D}}(\mathcal{B})$, we introduce a notion of uniformly having at most one path, which is similar to the notion of having uniformly at most one minimum introduced in [17] for non-negative functions; see also [14] and [15, 4.1]. A binary tree T *has uniformly at most one path* if

$$(**)\quad\forall k\exists n\geq k\forall a,b\in\{0,1\}^*[|a|=|b|=n\wedge\overline{a}k\neq\overline{b}k\rightarrow a\notin T\vee b\notin T].$$

Again, we may define this notion relative to a full subset $\mathcal{B}$ of $\{0,1\}^{\mathbf{N}}$: a binary tree T has uniformly at most one path in $\mathcal{B}$ if

$$\forall k\exists n\geq k\forall\alpha,\beta\in\mathcal{B}[\overline{\alpha}k\neq\overline{\beta}k\rightarrow\overline{\alpha}n\notin T\vee\overline{\beta}n\notin T].$$

But we can see that if a binary tree T has uniformly at most one path in some full set, then $(**)$ holds, and if $(**)$ holds, then T has uniformly at most one path in any full set. Hence this notion is independent of an underlying full set.

We show the following proposition in a similar way to [18].

Proposition 1. *Let $\mathcal{B}$ and $\mathcal{D}$ be computably closed inhabited subsets of $\{0,1\}^{\mathbf{N}}$, and assume $\mathrm{FAN}_{\mathcal{D}}(\mathcal{B})$. Then every $\mathcal{D}$-tree with at most one path in $\mathcal{B}$ has uniformly at most one path.*

Proof. Let T be a $\mathcal{D}$-tree with at most one path in $\mathcal{B}$. Then

$$\forall\alpha,\beta\in\mathcal{B}[\exists k(\overline{\alpha}k\neq\overline{\beta}k)\rightarrow\exists n(\overline{\alpha}n\notin T\vee\overline{\beta}n\notin T)],$$

and hence

$$\forall k\forall\alpha,\beta\in\mathcal{B}[\overline{\alpha}k\neq\overline{\beta}k\rightarrow\exists n(\overline{\alpha}n\notin T\vee\overline{\beta}n\notin T)].$$

Therefore, for given k, we have

$$\forall \alpha, \beta \in \mathcal{B} \exists n [\overline{\alpha}k \neq \overline{\beta}k \rightarrow \overline{\alpha}n \notin T \vee \overline{\beta}n \notin T],$$

and so, since T is downward closed, we have

$$\forall \alpha, \beta \in \mathcal{B} \exists n \geq k [\overline{\overline{\alpha}n}k \neq \overline{\overline{\beta}n}k \rightarrow \overline{\alpha}n \notin T \vee \overline{\beta}n \notin T].$$

For $\alpha \in \{0,1\}^{\mathbf{N}}$, let $E_\alpha(n) := \alpha(2n)$ and $O_\alpha(n) := \alpha(2n+1)$. Then, since $\mathcal{B}$ is computably closed, if $\alpha \in \mathcal{B}$, then $E_\alpha, O_\alpha \in \mathcal{B}$, and hence

$$\forall \alpha \in \mathcal{B} \exists n \geq k [\overline{\overline{E_\alpha}n}k \neq \overline{\overline{O_\alpha}n}k \rightarrow \overline{E_\alpha}n \notin T \vee \overline{O_\alpha}n \notin T].$$

For $a = (a_0, a_1, \ldots, a_{n-1}) \in \{0,1\}^*$, let $E(a) := (a_0, \ldots, a_{2m-2})$ and $O(a) := (a_1, \ldots, a_{2m-1})$, where $m = \lfloor n/2 \rfloor$. Note that $\overline{E_\alpha}n = E(\overline{\alpha}(2n))$ and $\overline{O_\alpha}n = O(\overline{\alpha}(2n))$. Then we have

$$\forall \alpha \in \mathcal{B} \exists n \geq k [\overline{E(\overline{\alpha}(2n))}k \neq \overline{O(\overline{\alpha}(2n))}k \rightarrow E(\overline{\alpha}(2n)) \notin T \vee O(\overline{\alpha}(2n)) \notin T].$$

Let

$$a \in B := 2k \leq |a| \wedge [\overline{E(a)}k \neq \overline{O(a)}k \rightarrow E(a) \notin T \vee O(a) \notin T].$$

Then, since $\mathcal{D}$ is computably closed, the characteristic function of B is in $\mathcal{D}$, and $\forall \alpha \in \mathcal{B} \exists n (\overline{\alpha}(2n) \in B)$, that is B is a bar in $\mathcal{B}$. By $\mathrm{FAN}_{\mathcal{D}}(\mathcal{B})$, there exists m such that

$$\forall a \in \{0,1\}^* [|a| = m \rightarrow \exists j \leq m (\overline{a}j \in B)].$$

Choose n such that $m \leq 2n$. Then, taking $0^m := (0, \ldots, 0)$ with $|0^m| = m$, there exists $j \leq m$ such that $\overline{0^m}j \in B$, and hence $2k \leq |\overline{0^m}j| = j \leq m \leq 2n$. Thus $k \leq n$. Let $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$ be in $\{0,1\}^*$ with $\overline{a}k \neq \overline{b}k$. Then, setting $c := (a_0, b_0, \ldots, a_{n-1}, b_{n-1})$, we have $m \leq 2n = |c|$, and there exists j with $j \leq m$ such that $\overline{\overline{c}m}j = \overline{c}j \in B$. Since $2k \leq j \leq m \leq 2n$, we have $\overline{E(\overline{c}j)}k = \overline{a}k \neq \overline{b}k = \overline{O(\overline{c}j)}k$, and hence either $E(\overline{c}j) \notin T$ or $O(\overline{c}j) \notin T$. In the former case, since $E(\overline{c}j) \preceq a$, we have $a \notin T$. Similarly, in the latter case, we have $b \notin T$. Thus T has uniformly at most one path.

The following proposition shows that a binary tree with uniformly at most one path has a longest path.

Proposition 2. *Let $\mathcal{D}$ be a computably closed inhabited subset of $\{0,1\}^{\mathbf{N}}$. Then every $\mathcal{D}$-tree with uniformly at most one path has a longest path in $\mathcal{D}$.*

Proof. Let T be a $\mathcal{D}$-tree with uniformly at most one path, and define a relation $\mathrm{big}(c, n)$ by

$$\mathrm{big}(c, n) := \forall d \in \{0,1\}^* [|d| = n \rightarrow c * d \notin T].$$

Note that the characteristic function of big is computable in $\mathcal{D}$. Then for given $c \in \{0,1\}^*$, since T has uniformly at most one path, there exists n such that

$$\forall a, b \in \{0,1\}^* [|a| = |b| = n \rightarrow c * (0) * a \notin T \vee c * (1) * b \notin T],$$

and hence $\neg\text{big}(c * (0), n) \rightarrow \text{big}(c * (1), n)$. Therefore for each $c \in \{0,1\}^*$ there exists n such that $\text{big}(c * (0), n) \vee \text{big}(c * (1), n)$, and so the function σ defined by

$$\sigma(c) := \min_n [\text{big}(c * (0), n) \vee \text{big}(c * (1), n)]$$

is computable in $\mathcal{D}$. Define functions δ and τ by

$$\delta(c) := \begin{cases} 0 \text{ if } \text{big}(c * (1), \sigma(c)), \\ 1 \text{ if } \neg\text{big}(c * (1), \sigma(c)), \end{cases}$$

and

$$\tau(0) := (),$$
$$\tau(n+1) := \tau(n) * \delta(\tau(n)).$$

Then, clearly, δ and τ are computable in $\mathcal{D}$. Let $\alpha(n) := \delta(\tau(n))$. Then $\alpha \in \{0,1\}^{\mathbf{N}}$ is computable in $\mathcal{D}$, and, since $\mathcal{D}$ is computably closed, α is in $\mathcal{D}$. Note that, by induction, $\overline{\alpha}n = \tau(n)$.

We prove that α is a longest path of T. Suppose that $\overline{\alpha}n = \tau(n) \notin T$. Then we show that $\forall d \in \{0,1\}^*(|d| = n \rightarrow d \notin T)$, or $\text{big}(\tau(0), n)$. To this end, we prove by induction that

$$\forall k \leq n[\text{big}(\tau(n-k), k)].$$

If $k = 0$, then, trivially, $\text{big}(\tau(n), 0)$. Assume that $\text{big}(\tau(n-k), k)$. Then $\text{big}(\tau(n-k-1) * \alpha(n-k-1), k)$, and hence $\sigma(\tau(n-k-1)) \leq k$. Either $\alpha(n-k-1) = 0$ or $\alpha(n-k-1) = 1$. In the former case, $\text{big}(\tau(n-k-1) * (0), k)$ and, since $\text{big}(\tau(n-k-1) * (1), \sigma(\tau(n-k-1)))$, we have $\text{big}(\tau(n-k-1) * (1), k)$. Hence $\text{big}(\tau(n-k-1), k+1)$. In the latter case, $\text{big}(\tau(n-k-1) * (1), k)$, and, since $\neg\text{big}(\tau(n-k-1) * (1), \sigma(\tau(n-k-1)))$, we have $\text{big}(\tau(n-k-1) * (0), \sigma(\tau(n-k-1)))$. Therefore $\text{big}(\tau(n-k-1) * (0), k)$, and so $\text{big}(\tau(n-k-1), k+1)$.

We omit proof of the following proposition which is an easy adaptation of the proof in [2] or [18].

Proposition 3. *Let $\mathcal{B}$ and $\mathcal{D}$ be computably closed inhabited subsets of $\{0,1\}^{\mathbf{N}}$. Then $\text{WKL!}_{\mathcal{D}}(\mathcal{B})$ implies $\text{FAN}_{\mathcal{D}}(\mathcal{B})$.*

The aforementioned propositions culminate in the following theorem.

Theorem 1. *Let $\mathcal{B}$ and $\mathcal{D}$ be computably closed inhabited subsets of $\{0,1\}^{\mathbf{N}}$ with $\mathcal{D} \subseteq \mathcal{B}$. Then the following statements are equivalent.*

1. $\text{FAN}_{\mathcal{D}}(\mathcal{B})$.
2. *Every $\mathcal{D}$-tree with at most one path in $\mathcal{B}$ has a longest path in $\mathcal{D}$.*
3. *Every infinite $\mathcal{D}$-tree with at most one path in $\mathcal{B}$ has an infinite path in $\mathcal{D}$.*
4. $\text{LPL!}_{\mathcal{D}}(\mathcal{B})$.
5. $\text{WKL!}_{\mathcal{D}}(\mathcal{B})$.

Proof. (1) $\rightarrow$ (2): Assume $\text{FAN}_{\mathcal{D}}(\mathcal{B})$, and let T be a $\mathcal{D}$-tree with at most one path in $\mathcal{B}$. Then, by Proposition 1, T has uniformly at most one path, and hence T has a longest path in $\mathcal{D}$, by Proposition 2. (2) $\rightarrow$ (3) and (4) $\rightarrow$ (5): Trivial. (2) $\rightarrow$ (4) and (3) $\rightarrow$ (5): By $\mathcal{D} \subseteq \mathcal{B}$. (5) $\rightarrow$ (1): By Proposition 3.

3 A Computable Version of MIN!

We assume that a real number x is given by a Cauchy sequence $(p_n)_n$ of rationals with a fixed modulus, that is, $\forall m, n(|p_m - p_n| < 2^{-m} + 2^{-n})$. For a real number $x := (p_n)_n$, we write $(x)_n$ for p_n. See [3,4,6,20] for more on constructive theory of the real numbers.

A uniformly continuous function f from the Cantor space $\{0,1\}^{\mathbf{N}}$ to $\mathbf{R}$ is *computable* if there exists an index e and a computable $M \in \mathbf{N}^{\mathbf{N}}$ such that

$$\forall \alpha \in \{0,1\}^{\mathbf{N}} \forall n \exists z [T(e, \alpha, n, z) \wedge U(z) = (f(\alpha))_n]$$

and

$$\forall k \forall \alpha, \beta \in \{0,1\}^{\mathbf{N}} [\overline{\alpha} M(k) = \overline{\beta} M(k) \rightarrow |f(\alpha) - f(\beta)| < 2^{-k}].$$

We show that the following computable version of MIN! for the Cantor space $\{0,1\}^{\mathbf{N}}$:

> *if f is a real valued computable uniformly continuous function with at most one minimum on the Cantor space $\{0,1\}^{\mathbf{N}}$, then there exists a computable α in $\{0,1\}^{\mathbf{N}}$ such that $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$*

is equivalent to the classically true relativized versions $\mathrm{WKL!}_{Rec}(\{0,1\}^{\mathbf{N}})$, $\mathrm{FAN}_{Rec}(\{0,1\}^{\mathbf{N}})$, and $\mathrm{LPL!}_{Rec}(\{0,1\}^{\mathbf{N}})$. We start with showing the following propositions.

Proposition 4. *Let T be an infinite Rec-tree with at most one path in $\{0,1\}^{\mathbf{N}}$. Then there exists a real valued computable uniformly continuous function f on the Cantor space $\{0,1\}^{\mathbf{N}}$ such that if $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$, then α is an infinite path of T.*

Proof. Define a real valued function f on the Cantor space $\{0,1\}^{\mathbf{N}}$ by

$$(f(\alpha))_n := \begin{cases} 2^{-n} & \text{if } \overline{\alpha} n \in T, \\ (f(\alpha))_{n-1} & \text{if } \overline{\alpha} n \notin T. \end{cases}$$

Then f is a computable uniformly continuous function with $\inf f(\{0,1\}^{\mathbf{N}}) = 0$. Let α, β be in $\{0,1\}^{\mathbf{N}}$ with $\alpha \neq \beta$. Then, since T has at most one path in $\{0,1\}^{\mathbf{N}}$, there exists n such that $\overline{\alpha} n \notin T$ or $\overline{\beta} n \notin T$, and hence either $0 < f(\alpha)$ or $0 < f(\beta)$. Thus f has at most one minimum. If $f(\alpha) = 0$, then $(f(\alpha))_n \leq 2^{-n}$ for all n, and hence $\overline{\alpha} n \in T$ for all n.

Proposition 5. *Let f be a real valued computable uniformly continuous function with at most one minimum on the Cantor space $\{0,1\}^{\mathbf{N}}$. Then there exists an infinite Rec-tree T with at most one path in $\{0,1\}^{\mathbf{N}}$ such that if α is an infinite path of T, then $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$.*

Proof. We may assume without loss of generality that $\inf f(\{0,1\}^{\mathbf{N}}) = 0$. Let $M \in \mathbf{N}^{\mathbf{N}}$ be a computable function such that

$$\forall k \forall \alpha, \beta \in \{0,1\}^{\mathbf{N}} [\overline{\alpha} M(k) = \overline{\beta} M(k) \rightarrow |f(\alpha) - f(\beta)| < 2^{-k}].$$

We may assume further that M is strictly increasing and $0 < M(0)$. For $a \in \{0,1\}^*$, we write $f(a)$ for $f(a * (\lambda x.0))$, where the concatenation is extended to concatenation of a finite sequence with an infinite sequence. Define a subset T of $\{0,1\}^*$ by

$$a \in T := \forall k[M(k) \le |a| \to (f(\overline{a}M(k)))_k < 2^{-|a|} + 2^{-k+1}].$$

Then T is a *Rec*-tree. For given n, choose $\alpha \in \{0,1\}^{\mathbf{N}}$ such that $f(\alpha) < 2^{-n}$, and set $a := \overline{\alpha}n$. Then, for each k with $M(k) \le |a| = n$, we have

$$\begin{aligned}(f(\overline{a}M(k)))_k &\le f(\overline{a}M(k)) + 2^{-k} = f(\overline{\alpha}M(k)) + 2^{-k} \\ &< f(\alpha) + 2^{-k+1} < 2^{-|a|} + 2^{-k+1},\end{aligned}$$

and hence $a \in T$. Therefore T is infinite. Let α, β be in $\{0,1\}^{\mathbf{N}}$ with $\alpha \ne \beta$. Since f has at most one minimum, there exists n such that either $5 \cdot 2^{-n} < f(\alpha)$ or $5 \cdot 2^{-n} < f(\beta)$. In the former case, since $n \le M(n)$, we have

$$\begin{aligned}(f(\overline{\alpha}M(n)))_n &\ge f(\overline{\alpha}M(n)) - 2^{-n} > f(\alpha) - 2^{-n+1} \\ &> 2^{-n} + 2^{-n+1} \ge 2^{-M(n)} + 2^{-n+1},\end{aligned}$$

and hence $\overline{\alpha}M(n) \notin T$. Similarly, in the latter case, we have $\overline{\beta}M(n) \notin T$. Therefore T has at most one path. If α is an infinite path of T, then, for each n, we have

$$\begin{aligned}f(\alpha) &< f(\overline{\alpha}M(n)) + 2^{-n} \le (f(\overline{\alpha}M(n)))_n + 2^{-n+1} \\ &< 2^{-M(n)} + 2^{-n+2} \le 5 \cdot 2^{-n},\end{aligned}$$

and hence $f(\alpha) = 0$.

The above propositions and Theorem 1 culminate in the following theorem.

Theorem 2. *The following statements are equivalent.*

1. *If f is a real valued computable uniformly continuous function with at most one minimum on the Cantor space $\{0,1\}^{\mathbf{N}}$, then there exists a computable α in $\{0,1\}^{\mathbf{N}}$ such that $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$.*
2. *If f is a real valued computable uniformly continuous function with at most one minimum on the Cantor space $\{0,1\}^{\mathbf{N}}$, then there exists α in $\{0,1\}^{\mathbf{N}}$ such that $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$.*
3. $\mathrm{WKL!}_{Rec}(\{0,1\}^{\mathbf{N}})$.
4. $\mathrm{LPL!}_{Rec}(\{0,1\}^{\mathbf{N}})$.
5. $\mathrm{FAN}_{Rec}(\{0,1\}^{\mathbf{N}})$.

Proof. Note that (1) $\to$ (2) is trivial. Then, by Theorem 1, it is enough to show that (2) $\to$ (3) and (3) $\to$ (1).

(2) $\to$ (3): By Proposition 4. (3) $\to$ (1): Let f be a real valued computable uniformly continuous function with at most one minimum on the Cantor space $\{0,1\}^{\mathbf{N}}$. Then, by Proposition 5, there exists an infinite *Rec*-tree T with at most one path in $\{0,1\}^{\mathbf{N}}$ such that if α is an infinite path of T, then $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$. By Theorem 1, there exists an infinite path α of T in *Rec*, that is computable, and hence $f(\alpha) = \inf f(\{0,1\}^{\mathbf{N}})$.

4 A Concluding Remark

Let $\mathcal{B}$ be a subset in between *Rec* and $\{0,1\}^{\mathbf{N}}$, say the set of characteristic functions of computably enumerable sets. Then, since $\mathrm{FAN}_{Rec}(Rec)$ is refutable as mentioned before, it is natural to ask whether $\mathrm{FAN}_{Rec}(\mathcal{B})$ is still refutable or is derivable from $\mathrm{FAN}_{Rec}(\{0,1\}^{\mathbf{N}})$. In the latter case, since it is trivial that $\mathrm{FAN}_{Rec}(\mathcal{B})$ implies $\mathrm{FAN}_{Rec}(\{0,1\}^{\mathbf{N}})$, we could see the equivalence between $\mathrm{FAN}_{Rec}(\mathcal{B})$ and $\mathrm{FAN}_{Rec}(\{0,1\}^{\mathbf{N}})$.

Acknowledgments. A clue for this research was obtained when Martin Ziegler was staying at Japan Advanced Institute of Science and Technology as a postdoctoral fellow of Japan Society for the Promotion of Science (JSPS). The author would like thank him for discussions, and JSPS for supporting his stay. Also this research was partly supported by a Grant-in-Aid for Scientific Research (C) No.15500005 of JSPS.

References

1. Berger, J., Bridges, D., Schuster, P.: The fan theorem and unique existence of maxima. J. Symbolic Logic 71, 713–720 (2006)
2. Berger, J., Ishihara, H.: Brouwer's fan theorem and unique existence in constructive analysis. MLQ Math. Log. Q. 51, 360–364 (2005)
3. Bishop, E.: Foundations of Constructive Analysis. McGraw-Hill, New York (1967)
4. Bishop, E., Bridges, D.: Constructive Analysis. Springer, Berlin (1985)
5. Bridges, D.: Recent progress in constructive approximation theory. In: Troelstra, A.S., van Dalen, D. (eds.) The L.E.J Brouwer Centenary Symposium, pp. 41–50. North-Holland, Amsterdam (1982)
6. Bridges, D., Richman, F.: Varieties of Constructive Mathematics. Cambridge University Press, Cambridge (1987)
7. Ishihara, H.: An omniscience principle, the König lemma and the Hahn-Banach theorem. Z. Math. Logik Grundlag. Math. 36, 237–240 (1990)
8. Ishihara, H.: Constructive reverse mathematics: compactness properties. In: Crosilla, L., Schuster, P. (eds.) From Sets and Types to Topology and Analysis, pp. 245–267. Oxford Univ. Press, Oxford (2005)
9. Ishihara, H.: Reverse mathematics in Bishop's constructive mathematics. Philosophia Scientiæ, Cahier spécial 6, 43–59 (2006)
10. Ishihara, H.: Weak König lemma implies Brouwer's fan theorem: a direct proof. Notre Dame J. Formal Logic 47, 249–252 (2006)
11. Julian, W., Richman, F.: A uniformly continuous function on $[0,1]$ that is everywhere different from its infimum. Pacific J. Math. 111, 333–340 (1984)
12. Kleene, S.C.: Recursive functions and intuitionistic mathematics. In: Graves, L.M., Hille, E., Smith, P.A., Zariski, O. (eds.) Proceedings of the International Congress of Mathematicians, Amer. Math Soc. Providence, pp. 679–685 (1952)
13. Kleene, S.C., Vesley, R.E.: The Foundations of Intuitionistic Mathematics, Especially in Relation to Recursive Functions. North-Holland, Amsterdam (1965)
14. Kohlenbach, U.: Effective moduli from ineffective uniqueness proofs: An unwinding of de La Vallée Poussin's proof for Chebycheff approximation. Ann. Pure Appl. Logic 64, 27–94 (1993)

15. Kohlenbach, U., Oliva, P.: Proof mining: a systematic way of analysing proofs in mathematics. In: Proc. Steklov Inst. Math. vol. 242, pp. 136–164 (2003)
16. Loeb, I.: Equivalents of the (weak) fan theorem. Ann. Pure Appl. Logic 132, 51–66 (2005)
17. Schuster, P.: Unique solutions. MLQ Math. Log. Q. 52, 534–539 (2006)
18. Schwichtenberg, H.: A direct proof of the equivalence between Brouwer's fan theorem and König lemma with a uniqueness hypothesis. J. UCS 11, 2086–2095 (2005)
19. Simpson, S.G.: Subsystems of second order arithmetic. Springer, Berlin (1999)
20. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics, Vol. I An Introduction. North-Holland, Amsterdam (1988)
21. Veldman, W.: Brouwer's fan theorem as an axiom and as a contrast to Kleene's alternative, preprint. Radboud University, Nijmegen (2005)
22. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)

Input-Dependence in Function-Learning

Sanjay Jain[1,⋆], Eric Martin[2,⋆⋆], and Frank Stephan[3,⋆⋆⋆]

[1] Department of Computer Science,
National University of Singapore,
Singapore 117543, Republic of Singapore
sanjay@comp.nus.edu.sg

[2] Department of Computer Science and Engineering,
The University of New South Wales,
Sydney 2052, NSW, Commonwealth of Australia
emartin@cse.unsw.edu.au

[3] Department of Mathematics,
National University of Singapore,
Singapore 117543, Republic of Singapore
fstephan@comp.nus.edu.sg

Abstract. In the standard literature on inductive inference, a learner sees as input the course of values of the function to be learned. In the present work, it is investigated how reasonable this choice is and how sensitive the model is with respect to variations like the overgraph or undergraph of the function. Several implications and separations are shown and for the basic notions, a complete picture is obtained. Furthermore, relations to oracles, additional information and teams are explored.

Keywords: Inductive inference, recursion theory, various forms of input presentation, team learning, learning with additional information.

1 Introduction

The central question of inductive inference is to investigate in clear mathematical terms, which types of classes of functions or sets can be learned in principle by an algorithm. This is formalized with notions from recursion theory and therefore the objects to be learned are – in most scenarios – either classes of recursive functions or classes of recursively enumerable sets. Gold [5] introduced the notion of learning sets in the limit from positive data: here the learner M is a recursive function which maps any finite string σ of data to a hypothesis $M(\sigma)$. This hypothesis is then an index from an acceptable numbering $W_0, W_1, \ldots$ of all recursively enumerable sets; this numbering is assumed to be based on an

⋆ Supported in part by NUS grant number R252-000-212-112.
⋆⋆ E. Martin is jointly appointed at the UNSW and National ICT Australia which is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence Program.
⋆⋆⋆ Supported in part by NUS grant number R252-000-212-112.

acceptable numbering $\varphi_0, \varphi_1, \ldots$ of the partial-recursive functions by letting W_e be the domain of φ_e. More precisely, the set W_e to be learned is described by a text T which is a sequence containing all members of W_e in arbitrary order and perhaps some pause-symbols denoted by #. Now M learns W_e from T iff there is an n such that $W_{M(T(0)T(1)\ldots T(m))} = W_e$ for all $m \geq n$. As one cannot check effectively whether $W_e = W_d$ for indices d, e, it is restrictive to postulate in addition that $M(T(0)T(1)\ldots T(n)) = M(T(0)T(1)\ldots T(m))$ for all $m \geq n$. Both notions turned out to be reasonable in inductive inference, the more general notion is called behaviourally correct learning and the more restrictive notion is called explanatory learning. In the present work, only explanatory learning is considered, but many questions could also be investigated in the more general behaviourally correct setting.

Of course, if a learner has to learn only one set W_e, it could just output e on all possible input data; therefore one considers the learnability of classes of sets instead of single sets. Some well-known examples are the following ones.

– The class of finite sets. Here the learner outputs at every stage a hypothesis which enumerates exactly the elements seen so far; the hypothesis is only revised if a new element is discovered in the input.
– The class of all sets $\mathbb{N} - \{x\}$ consisting of the natural numbers different from x. Here the learner conjectures, at every stage, a hypothesis for the set $\mathbb{N} - \{x\}$, where x is the least element not seen so far.
– Gold [5] showed that many natural classes are not learnable: the class of all cofinite sets; the class consisting of $\mathbb{N}$ and all its finite subsets; the class

$$\{E_f : f \in REC\} \text{ with } E_f = \{(x, y) : y = f(x)\}$$

of all graphs of recursive functions.

Here REC stands for the class containing all recursive functions from $\mathbb{N}$ into $\mathbb{N}$; note that properly partial-recursive functions are not in REC.

The learning of classes of functions in REC is a paradigm linked to the learning of sets as follows: A class $S \subseteq REC$ is learnable (from the viewpoint of function-learning) iff the corresponding class $\{E_f : f \in S\}$ is learnable from the viewpoint of learning sets from positive data. The new approach of this work compared to previous papers is that here using E_f to describe f is only seen as one possibility among several legitimate ones. Therefore it is investigated how the choice of the representation of the function on the input influences the learnability. Now, the following instantiations to replace the sets E_f are considered:

– $D_f = \{(x, y) : y \neq f(x)\}$, that is, D_f is the set of all (x, y) with y different from $f(x)$;
– $L_f = \{(x, y) : y < f(x)\}$, that is, L_f is the set of all (x, y) with y less than $f(x)$;
– $G_f = \{(x, y) : y > f(x)\}$, that is, G_f is the set of all (x, y) with y greater than $f(x)$.

Furthermore, the learning criteria E, D, L and G mean that one is learning functions f from texts for E_f, D_f, L_f and D_f, respectively.

Definition 1. A learner M is a recursive function which assigns to every initial segment of a text T a hypothesis e. Let I denote E, D, L or G. The learner M I-learns f iff M on any text for I_f outputs almost always the same index of the set E_f; in Section 5, other hypotheses spaces than E_f will be considered as well. Furthermore, the symbols D, E, G and L also stand for the collection of all classes $S \subseteq REC$ such that some learner D-identifies, E-identifies, G-identifies and L-identifies every $f \in S$ respectively.

Example 2. *A family $f_0, f_1, f_2, \ldots$ of functions is called recursively enumerable iff the two-ary function $n, x \mapsto f_n(x)$ is recursive. Every recursively enumerable family $f_0, f_1, f_2, \ldots$ is D-learnable as a learner can easily converge to an index for the first function f_n such that no pair (x, y) with $y = f_n(x)$ occurs in the data.*

The recursively enumerable class of those functions which take on almost all inputs the same value cannot be learned from data of type L or G.

Example 3. *A class S of functions is called self-describing iff $\varphi_{f(0)} = f$ for all functions $f \in S$; also other methods of self-description can be considered. The self-describing functions can be learned from all four types of data presentation as for each one of them one can find $f(0)$ in the limit.*

2 Inclusions

An inclusion like $D \subseteq E$ would mean that every D-learnable class is also E-learnable. One of the first topics of inductive inference is to determine which of such inclusions are true and which are false; after this is done, further questions are considered for the corresponding inference criteria.

Theorem 4. *Concerning L, G, D, E, the inclusions $L \subseteq D$, $D \subseteq E$, $L \subseteq E$ and $G \subseteq E$ hold, but no other ones.*

Proof (sketch). The inclusion $L \subseteq E$ is obtained by transitivity, thus only the other three inclusions are shown. These three inclusions of the form $I \subseteq J$ can be shown by translating the sets J_f to I_f according to the following formulas.

$$D_f = \{(x, y) : \exists z \neq y\, ((x, z) \in E_f)\};$$
$$L_f = \{(x, y) : \forall z \leq y\, ((x, z) \in D_f)\};$$
$$G_f = \{(x, y) : \exists z < y\, ((x, z) \in E_f)\}.$$

These transformations permit to translate texts for these sets as well. The indexes found by the corresponding learners need not be translated.

The noninclusions $D \not\subseteq L$, $D \not\subseteq G$, $E \not\subseteq L$, $E \not\subseteq G$ are all witnessed by the class of total functions that are constant almost everywhere, that is, the total functions f that satisfy $\exists y\, \forall x > y\, (f(x) = f(y))$. The proof of the nonlearnability parts is a direct translation of the proof that the class of cofinite sets is not learnable from positive data. The learning algorithm for this class is called "learning by enumeration", that is, given any uniformly recursive enumeration $f_0, f_1, \ldots$ of the class, the learner always conjectures the first f_e such that no data-item seen so far contradicts this hypothesis. This means that no data-item (x, y) with $y \neq f_e(x)$ has shown up in the case of E and that no data-item (x, y) with $y = f_e(x)$ has shown up in the case of D.

The noninclusion $G \not\subseteq D$ is witnessed by the family of all recursive functions satisfying one of the following two conditions:

– $f = \varphi_{f(0)}$ and for all x, $f(x) < f(x+1)$;
– there is a y such that $\forall x < y\ [f(x) = \varphi_{f(0)}(x) < f(x+1) = \varphi_{f(0)}(x+1)]$ and $\forall x \geq y\ [f(x) = f(y)]$.

This noninclusion translates also into $E \not\subseteq D$ as the f in the above class are also learnable from the sets E_f but not from the sets D_f.

The remaining noninclusions are witnessed by standard classes which correspond to Gold's example of an unlearnable family [5]. The noninclusion $L \not\subseteq G$ is given by the class of all functions of the form $1^n 0^\infty$ plus 1^∞. The noninclusion $G \not\subseteq L$ is witnessed by the class of all functions of the form $1^n 2^\infty$ plus 1^∞. □

3 Oracles

In the following, let A, A_1, A_2, A_3, A_4, B be sets of natural numbers which are used as oracles. Let K denote the halting problem. Criteria like E, L, ... are modified to $E[A]$, $L[A]$, ... where instead of recursive learners M, one considers A-recursive learners M^A. Adleman and Blum [1] established the following fundamental result.

Theorem 5 (Adleman and Blum [1]). *$REC \in E[A]$ iff A is high. That is, REC can be learned from data-type E and oracle A iff A is high.*

The corresponding variants of this theorem are that for D, high oracles are sufficient, but for the other modes of input presentation, REC cannot be learned using any oracle.

Theorem 6. *$REC \in D[A]$ iff A is high.*

Proof (sketch). If A is high, then there is an A-recursive procedure $Tot(e, t)$ such that for every e, $\lim_{t\to\infty} Tot(e, t) = 1$, if φ_e is total, and $\lim_{t\to\infty} Tot(e, t) = 0$, if φ_e is partial. Then the learner converges on any text T for D_f to the least e such that $\lim_{t\to\infty} Tot(e, t) = 1$ and no pair $(x, \varphi_e(x))$ shows up in the text T. It is easy to verify that this algorithm is correct.

Recall that $D \subseteq E$. The same holds for the relativized version $D[A] \subseteq E[A]$, that is, every $D[A]$-learner for REC can be translated into an $E[A]$-learner for REC. Thus such an A has to be high. Alternatively, one can use the second direction of Theorem 8 below for the necessity of A being high. □

Remark 7. The class of all cofinite sets cannot be learned from text with respect to any oracle [5]. Using this technique, one can show that

$$\{f \in REC : \forall x\,(f(x) \leq 1) \text{ and } \exists y\,\forall x \geq y\,(f(x) = 1)\}$$

is G-learnable but not $L[A]$-learnable for any oracle A. Similarly,

$$\{f \in REC : \forall x\,(f(x) \geq 1) \text{ and } \exists y\,\forall x \geq y\,(f(x) = 1)\}$$

is L-learnable but not $G[A]$-learnable for any oracle A. Thus one cannot overcome these separations with oracles.

Note that in the following, $W_{e,s}$ is the set of all elements which are enumerated into W_e within s steps. Without loss of generality, $\forall x \in W_{e,s}\,[x < s]$.

Theorem 8. *$G \subseteq D[A]$ iff A is high.*

Proof (sketch). The condition that A is high is certainly sufficient for this inclusion as REC can be $D[A]$-learned for all high oracles A.

Now consider the class $\{f_0, f_1, f_2, \ldots\}$ of functions where $f_e(x)$ is $e+s$ for the first stage s such that at least $\min\{x, |W_e|\}$ many elements are enumerated into $W_{e,s}$; thus $f_e(0) = e$ and gives away the index of the function in this enumeration.

This class is easily seen to be G-learnable. One can find the value e of the input function at 0 in the limit and knows then that f_e is the function to be learnt. Then the learner outputs a program which assigns to every input x the output $s+e$ for the first stage s with $|W_{e,s}| \geq x$. In the case that some pair (x, y) with $|W_{e,y}| < x$ is found, the learner knows that $f_e(z) = f_e(x)$ for all $z > x$ and can easily determine f_e in the limit.

For the converse direction, one first shows that the sets D_{f_e} are uniformly recursively enumerable. The reason is that a pair (x, y) is enumerated into D_{f_e} iff at least one of the following four conditions holds:

- $y < e$;
- $y \geq e$ and there is a stage s with $0 \leq s < y - e$ and $|W_{e,s}| \geq x$;
- $y \geq e$ and there is a stage s with $0 \leq s < y - e$ and $|W_{e,s}| = |W_{e,y-e}|$;
- $y \geq e$, $|W_{e,y-e}| < x$ and there is a stage s with $|W_{e,y-e}| < |W_{e,s}|$.

As these conditions are all Σ^0_1-conditions, a text T_e for D_{f_e} can be constructed uniformly from the index e. Given any e, one can simulate the behaviour of any $D[A]$-learner M on T_e in the limit and determine with oracle A' the final hypothesis e'. Note that $\varphi_{e'}$ is total. Now W_e is finite iff there are values x, s such that $e + s = \varphi_{e'(x)}$ and $|W_{e,s}| < x$. This condition is a Σ^0_1-condition and can also be checked with oracle A'. Hence $K' \leq_T A'$ and A is high. □

Theorem 9. *Given recursively enumerable sets A_1, A_2, A_3, A_4 there are classes $SE_{A_1}, SD_{A_2}, SL_{A_3}, SG_{A_4}$ such that*

$$\begin{aligned} SE_{A_1} \in E[B] &\ \ \textit{iff}\ \ A_1' \leq_T B'; \\ SD_{A_2} \in D[B] &\ \ \textit{iff}\ \ A_2' \leq_T B'; \\ SL_{A_3} \in L[B] &\ \ \textit{iff}\ \ A_3' \leq_T B'; \\ SG_{A_4} \in G[B] &\ \ \textit{iff}\ \ A_4' \leq_T B'. \end{aligned}$$

Furthermore, if the r.e. sets A_1, A_2, A_3, A_4 satisfy that $A_1 \leq_T A_2$, $A_2 \leq_T A_3$, $A_2 \leq_T A_4$, $K <_T A_1' <_T A_2'$, $A_3' \not\leq_T A_4'$ and $A_4' \not\leq_T A_3'$ then the class

$$S = SE_{A_1} \oplus SD_{A_2} \oplus SL_{A_3} \oplus SG_{A_4}$$

of the fourfold join of functions in SE_{A_1}, SD_{A_2}, SL_{A_3} and SG_{A_4} satisfies for every oracle B that

$$\begin{aligned} S \in E[B] &\text{ iff } A_1' \leq_T B'; \\ S \in D[B] &\text{ iff } A_2' \leq_T B'; \\ S \in L[B] &\text{ iff } A_3' \leq_T B'; \\ S \in G[B] &\text{ iff } A_4' \leq_T B'. \end{aligned}$$

So the best possible oracles for the four learning criteria are different in each case. The choice of such oracles A_1, A_2, A_3, A_4 is possible.

Proof (sketch). Kummer and Stephan [8] have shown that there is a class TE_{A_1} which is $E[B]$-learnable iff $A_1' \leq_T B'$. This class consists of $\{0,1\}$-valued functions. Now let

$$SE_{A_1} = \{g \in REC : \exists h \in TE_{A_1}\, \forall n \in \mathbb{N} \\ [g(2n) = h(n) \wedge g(2n+1) = 1 - h(n)]\,\}$$

and one can see that $SE_{A_1} \in E[B], D[B], L[B], G[B]$ iff $A_1' \leq_T B'$. The reason is that, from an approximation of $g(2n), g(2n+1)$ from below, one can exploit the equation $g(2n) + g(2n+1) = 1$ in order to get both values. Similarly for approximations from above. Hence, for these functions g, texts of E_g, D_g, L_g and G_g are equally useful for learning.

As A_2' is recursively enumerable relative to K, there is a total two-place approximation $\Psi(x,s)$ such that $x \in A_2'$ iff $\Psi(x,s) = 1$ for all but finitely many s. Now one defines, for every x, the function f_x as

$$f_x(y) = \begin{cases} x & \text{if } y = 0; \\ s & \text{for the first } s > f_x(y-1) \text{ with } \Psi(x,s) \neq 1 \\ & \text{if such an } s \text{ exists;} \\ f_x(y-1) & \text{otherwise.} \end{cases}$$

Note that the case-distinction is non-uniform in x. However, f_x is still recursive as either the second case applies for all $y > 0$ or f_x is an eventually constant function. Let $SD_{A_2} = \{f_0, f_1, \ldots\}$. Although one cannot effectively, from x, compute an index for f_x, one can compute an index $u(x)$ of the following variant of f_x:

$$\varphi_{u(x)}(y) = \begin{cases} x & \text{if } y = 0; \\ s & \text{for the first } s > \varphi_{u(x)}(y-1) \text{ with } \Psi(x,s) \neq 1 \\ & \text{if such an } s \text{ exists;} \\ \uparrow & \text{otherwise.} \end{cases}$$

Note that the sets D_{f_x}, L_{f_x} are uniformly r.e. in x as witnessed by the following two definitions of these sets. D_{f_x} is the set of all (a,b) satisfying one of the following conditions:

- $\exists c \leq a\ [\varphi_{u(x)}(c)\downarrow > b]$;
- $\exists c \geq a\ [\varphi_{u(x)}(c)\downarrow < b]$;
- $b \neq x \wedge \Psi(x,b) = 1$.

The set L_{f_x} is given by the first condition, that is, L_{f_x} is just the set of all (a,b) such that there is a $c \leq a$ with $\varphi_{u(x)}(c)\downarrow > b$.

Hence, if SD_{A_2} is $L[B]$-learnable or $D[B]$-learnable then $A_2' \leq_T B'$, as one can, for any given x, simulate the learner on L_{f_x} or D_{f_x}, respectively, find the final hypothesis e of the learner using B' and determine whether there is an y with $\varphi_e(y) = \varphi_e(y+1)$ using B' again; the latter has the answer YES if $x \in A_2'$ and the answer NO if $x \notin A_2'$.

The class SD_{A_2} is E-learnable and G-learnable, as one can figure out from the data, in the limit, the values $f_x(0)$ and x. Furthermore, it can be determined, in the limit, whether there is a y with $\varphi_{u(x)}(y)$ being defined and $(y+1, \varphi_{u(x)}(y)) \in E_{f_x}$ or $(y+1, \varphi_{u(x)}(y)+1) \in G_{f_x}$, respectively. If so, y is found in the limit and the learner knows that f_x equals the function which is $\varphi_{u(x)}$ below y and is constant from y onwards. If not, such a y will never be found and the learner will conjecture $\varphi_{u(x)}$ from the time it has determined x.

The class SD_{A_2} is $L[B]$-learnable and $D[B]$-learnable for all oracles B with $A_2' \leq_T B'$. The algorithm is similar to the one before. Again the learner observes from the data in the limit the values $f_x(0)$ and x. Furthermore, it can use the oracle B to determine in the limit whether $x \in A_2'$. If so, it finds the maximal y in the limit where $\varphi_{u(x)}(y)$ is defined and knows that f_x equals the function which is $\varphi_{u(x)}$ below y and is constant from y onwards. If $x \notin A_2'$ then $f_x = \varphi_{u(x)}$ and the learner converges to the hypothesis $u(x)$.

For the sets SL_{A_3} and SG_{A_4}, one takes functions f^{A_3}, f^{A_4} which are ${A_3}'$-recursive and ${A_4}'$-recursive, respectively, and which furthermore satisfy that every function dominating them computes ${A_3}'$ and ${A_4}'$, respectively. Now let

$$SL_{A_3} = \{(2n+1)^\infty : n \in \mathbb{N}\} \cup \{(2n+1)^m (2n)^\infty : n \in \mathbb{N}, m \leq f^{A_3}(n)\};$$
$$SG_{A_4} = \{(2n)^\infty : n \in \mathbb{N}\} \cup \{(2n)^m (2n+1)^\infty : n \in \mathbb{N}, m \leq f^{A_4}(n)\}.$$

The class SL_{A_3} is in $L[B]$ iff ${A_3}' \leq_T B'$: On one hand, if ${A_3}' \leq_T B'$ then the following algorithm works: One can find $g(0)$ and the parameter n in the limit. Then one can find an upper bound k for $f^{A_3}(n)$ relative to B in the limit. After that one can observe $g(0), g(1), \ldots, g(k)$ in the limit. Having these values, the function makes a step only before k and thus the learner can converge to a canonical hypothesis for $g(0)g(1)\ldots g(k)(g(k))^\infty$.

On the other hand, if one infers SL_{A_3} with oracle B, one can find for each function $(2n+1)^\infty$ in the limit a locking sequence [3] and let $h(n)$ be the largest first coordinate of a pair (x,y) occurring in this locking sequence. Then $h \leq_T B'$ and $h(n) \geq f^{A_3}(n)$ for all n; hence ${A_3}' \leq_T B'$.

The argumentation for $SG_{A_4} \in G[B]$ iff $A_4' \leq_T B'$ is symmetric. Furthermore, one can easily see that $SL_{A_3} \in G, D, E$ and $SG_{A_4} \in L, D, E$.

The second result on the combined class S is obtained by combining the four results above including the additional learnability properties mentioned above. Recall that $g \in SE_{A_1} \oplus SD_{A_2} \oplus SL_{A_3} \oplus SG_{A_4}$ iff there are $g_1 \in SE_{A_1}$,

$g_2 \in SD_{A_2}$, $g_3 \in SL_{A_3}$ and $g_4 \in SG_{A_4}$ such that for all $x \in \mathbb{N}$ and $y \in \{1,2,3,4\}$, $g(4x+y-1) = g_y(x)$. The hardness results translate immediately from the hardness of learning the components. So the more difficult part would be to verify the learnability.

Now let g, E_g, D_g, L_g and G_g be given. If the data is a text for E_g, then one can learn the component g_1 using oracle A_1 and the components g_2, g_3, g_4 without any oracle. If the data is a text for D_g, then the component g_1 can be learned using A_1, the component g_2 can be learned using A_2 and the components g_3 and g_4 do not require an oracle. If the data is a text for L_g, then the oracle A_3 is sufficient as it permits to reconstruct g_1, g_2 and g_3 in the limit; the component g_4 does not need an oracle. If the data is a text for G_g, then the oracle A_4 is sufficient as it permits to reconstruct g_1, g_2 and g_4 in the limit; the component g_3 does not need an oracle. It is easy to see how these things generalise if any oracle B is considered; it is necessary to use that $A_1' \leq_T A_2' \leq_T A_3'$ and $A_1' \leq_T A_2' \leq_T A_4'$ for obtaining this generalization. $\square$

4 Degrees of Inference

The question for which oracles A, B the inclusion $E[A] \subseteq E[B]$ holds has been investigated exhaustively [1,4,8]. Adleman and Blum [1] established that exactly the high oracles B satisfy that $E[A] \subseteq E[B]$ for all oracles A. Kummer and Stephan [8] showed that for non-high r.e. sets A, B, $E[A] \subseteq E[B]$ iff $A \leq_T B$. Besides the criterion E – which corresponds to explanatory function learning – a lot of other convergence criteria have been studied in the publications [4,8]. The approach taken in the present work differs from these by the fact that not the mode of convergence (explanatory or learning in the limit) is varied but the mode of presentation of the input is; therefore the standard criterion E coincides with explanatory learning while for L, G and D other results might be possible.

Remark 10. Assume that $E[A] \not\subseteq E[B]$ for some oracles A, B. Then there is a class R of $\{0,1\}$-valued functions witnessing this [8]. Now one defines

$$S = \{g : \exists h \in R \, \forall x \, (g(2x) = h(x) \wedge g(2x+1) = 1 - h(x))\}.$$

This set satisfies for every oracle C that

$$R \in E[C] \Leftrightarrow S \in E[C] \Leftrightarrow S \in L[C] \Leftrightarrow S \in G[C] \Leftrightarrow S \in D[C].$$

As a consequence, one has that $L[A] \not\subseteq L[B]$, $G[A] \not\subseteq G[B]$ and $D[A] \not\subseteq D[B]$.

The inference-degrees of D and E might be quite similar, as previous results already establish that the high oracles are also exactly those which are omniscient for D. Indeed it can be conjectured that they are the same. For L and G, things are a bit different.

Remark 11. Jain and Sharma [6] proved that there is no maximal inference-degree for learning recursively enumerable sets from positive data. Kummer and

Stephan [8] strengthened this result by showing that inclusion in these inference degrees implies Turing reducibility on the jump. Their witness classes for this fact consisted of recursive sets; thus one can translate them into witness-classes of functions for the input modes L and G, respectively. So one has that whenever $L[A] \subseteq L[B]$ or $G[A] \subseteq G[B]$ then $A' \leq_T B'$. In particular, for each oracle there is still a strictly more powerful oracle and so the structure of inference-degrees differs from that of E and D.

Conjecture 12. *The inference-degrees of E and D coincide. The inference-degrees of L and G both coincide with the inference-degrees of learning recursively enumerable sets from positive data.*

5 Additional Information and Teams

An oracle supplies information independent of the concrete learning task. In contrary to this, additional information and teachers are information linked to the concrete learning task, although they might not yield any nonrecursive information (like a solution for the halting problem). Indeed additional information is even finite, mostly an index or a bound of an index related to the learning task. The main result for function learning is that one can learn functions from any upper bound of any index given as additional information. This is even true independently of the chosen data type.

Proposition 13. *REC is L-learnable and G-learnable with an upper bound of an index of a function as additional information.*

The reason is that one can for each two indices i, j below the given bound of functions inconsistent at x find out in the limit whether $\varphi_i(x)$ or $\varphi_j(x)$ is incorrect. With this method, one can eventually eliminate all indices inconsistent with the data and converge to a hypothesis which amalgamates the set Z of the remaining programs in the sense that $\varphi_{f(Z)}(x) = y$ iff $\varphi_e(x) = y$ for some $e \in Z$.

As the traditional setting gives no new insight, the following more general setting is investigated, where each learning criterion has several aspects. A class S is $IJ(T)$-learnable with $I, J, T \in \{D, E, G, L\}$ iff there is a learner M doing the following.

- M receives data of type I as in the concept of I-learning in previous sections;
- M outputs hypotheses of type J, that is, the output of M converges to an index of a recursively enumerable set W which generates the set J_f linked to the function to be learnt;
- The additional information is an index of a recursively enumerable set V such that $V = T_f$; in the case of IJ-learning, no additional information is supplied.

Of course $REC \in JI(I)$ as the additional information already solves the learning task. Similarly, the inclusions from Section 2 carry over and $REC \in JL(D)$ for all $J \in \{D, E, G, L\}$: that is, an index for D_f is given as additional information and an index for L_f has to be learned; so one can simply translate the additional information into this form. Furthermore, data and additional information can complement each other; for example $EJ \subseteq LJ(G)$ as the additional information G plus the data L permit to generate the data E. The next theorem characterises many relations in this type of inference.

Theorem 14. *For $I, J \in \{D, E, G, L\}$, the following statements are equivalent.*

1. *Every J-index can be translated into an I-index;*
2. *$I \subseteq J$, that is, without any additional information and using hypotheses of type E, every class learnable from data of type I is also learnable from data of type J;*
3. *$EE(I) \subseteq EE(J)$, that is, for every class S, if S can be learned with data E, with hypothesis space E and with additional information I then S can also be learned with data E, hypothesis space E and additional information J;*
4. *$REC \in DI(J)$;*
5. *$REC \in EI(J)$;*
6. *$REC \in GI(J)$;*
7. *$REC \in LI(J)$.*

A further important topic is the question of team-learning where a class S is $[m, n]IJ$-learnable iff there is a team of n learners such that on every text for a set I_f for a function f in S at least m members of the team converge to an index for the language J_f. Here are some sample results which could be extended with well-known techniques.

Proposition 15. $[2, 3]IE = [1, 1]IE$ *for all $I \in \{D, E, G, L\}$. That is, for all modes of input presentation, as long as the hypotheses have to be of the type E any $[2, 3]IE$-learner can be replaced by a $[1, 1]IE$-learner which is the same as an I-learner.*

Proof (sketch). Let M_1, M_2, M_3 be a given $[2, 3]IE$-team learning S. Without loss of generality, the size of the hypotheses of the simulated machines increases at every mind change. Then a new learner N simulates M_1, M_2, M_3 and considers at every stage s the current outputs of the learning machines; in order of their size they are denoted as e_1, e_2, e_3. In the case that the $\varphi_{e_1}(x) = \varphi_{e_2}(x)$ for all $x \leq s$ where these two values are output within s stages of computation, the output of N does not depend on e_3 and is a value $f(e_1, e_2)$ such that $\varphi_{f(e_1,e_2)}(x)$ takes $\varphi_{e_1}(x)$ or $\varphi_{e_2}(x)$, whatever converges first. Otherwise the learner N outputs $g(e_1, e_2, e_3)$ such that $\varphi_{g(e_1,e_2,e_3)}(x)$ is y iff at least two of the values $\varphi_{e_1}(x)$, $\varphi_{e_2}(x)$ and $\varphi_{e_3}(x)$ are y. For the verification note that at least two machines converge and their final hypotheses e_1, e_2 are eventually below the hypothesis of the third machine. If φ_{e_1} and φ_{e_2} are consistent then $\varphi_{f(e_1,e_2)}$ equals the input function as one of these two functions must be the input function. If they are inconsistent, the third machine must converge as well to some final value e_3 and $\varphi_{g(e_1,e_2,e_3)}$ is equal to the input function. □

This proof technique can be generalised to obtain the following result which was for EE already obtained by Pitt and Smith [11].

Theorem 16. *For* $I \in \{D, E, G, L\}$, $[m, n]IE = [1, k]IE$ *iff* $\frac{1}{k+1} < \frac{m}{n} \leq \frac{1}{k}$.

In the case of ED-learning where the data is of type E and the hypotheses of type D one can assume that every inconsistent hypothesis, that is, every hypothesis e for a set D_e with $(x, f(x)) \in D_e$, is eventually updated and replaced by another one. Then $[2, 3]ED = [1, 1]ED$ by simply taking the union of the two oldest hypotheses in each stage. Again one can generalise the result.

Proposition 17. $[m, n]ED = [1, k]ED$ *iff* $\frac{1}{k+1} < \frac{m}{n} \leq \frac{1}{k}$.

The results can be very different if neither the data nor the hypotheses are of type E; then they have some similarity to the case of language learning.

Remark 18. Translating results of language learning obtained by Jain and Sharma [7] yields $[1, 1]GG \subset [2, 3]GG \subset [1, 2]GG = [3, 6]GG \subset [2, 4]GG$ and $[1, 1]LL \subset [2, 3]LL \subset [1, 2]LL = [3, 6]LL \subset [2, 4]LL$.

References

1. Adleman, L., Blum, M.: Inductive inference and unsolvability. Journal of Symbolic Logic 56, 891–900 (1991)
2. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45, 117–135 (1980)
3. Blum, L., Blum, M.: Towards a mathematical theory of inductive inference. Information and Control 28, 125–155 (1975)
4. Fortnow, L., Gasarch, W., Jain, S., Kinber, E., Kummer, M., Kurtz, S., Pleszkoch, M., Slaman, T., Solovay, R., Stephan, F.: Extremes in the degrees of inferability. Annals of Pure. and Applied Logic 66, 231–276 (1994)
5. Gold, M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
6. Jain, S., Sharma, A.: On the non-existence of maximal inference degrees for language identification. Information Processing Letters 47, 81–88 (1993)
7. Jain, S., Sharma, A.: Computational limits on team identification of languages. Information and Computation 130(1), 19–60 (1996)
8. Kummer, M., Stephan, F.: On the structure of degrees of inferability. Journal of Computer and System Sciences, Special Issue COLT (1993) 52, 214–238 (1996)
9. Odifreddi, P.: Classical Recursion Theory. North-Holland, Amsterdam (1989)
10. Osherson, D., Stob, M., Weinstein, S.: Systems That Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists. Bradford/The MIT Press, Cambridge, Massachusetts (1986)
11. Pitt, L., Smith, C.H.: Probability and plurality for aggregations of learning machines. Information and Computation 77, 77–92 (1988)
12. Soare, R.: Recursively Enumerable Sets and Degrees. A Study of Computable Functions and Computably Generated Sets. Springer, Heidelberg (1987)

Some Notes on Degree Spectra of the Structures

Iskander Kalimullin

Kazan State University, Kazan, 420008, Kremlevskaya str. 18, Russia
Iskander.Kalimullin@ksu.ru

Abstract. In the paper the problem of existence of an algebraic structure with the degree spectra $\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ is studied for arbitrary degree $\mathbf{b}$.

1 Restrictions on the Degree Spectra

A *representation* of a countable algebraic structure $\mathcal{A}$ is any isomorphic copy of $\mathcal{A}$ with the universe, which is a subset of ω (the set of natural numbers with zero). Under *degree spectrum* $\mathbf{Sp}\,(\mathcal{A})$ of a countable algebraic structure $\mathcal{A}$ we understand the collection of Turing degrees of atomic diagrams of all representations of $\mathcal{A}$.

The following well-known result presents the first restriction on possible degree spectra.

Theorem 1 (Knight [8]). *Let $\mathcal{A}$ be a countable structure in a finite language. Then precisely one of the following two statements holds:*

1. *For any two Turing degrees $\mathbf{c} \leq \mathbf{d}$, if $\mathbf{c} \in \mathbf{Sp}\,(\mathcal{A})$, then also $\mathbf{d} \in \mathbf{Sp}\,(\mathcal{A})$ (i.e., the degree spectrum is closed upwards).*
2. $\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{0}\}$. *(The structures with this property is called trivial).*

Each finite structure is an obvious example of a trivial structure, but there are also infinite trivial structures, such as the infinite complete graph.

In this paper we will consider only the nontrivial countable structures in finite languages. Theorem 1 shows that for nontrivial structures the degree spectrum is simply a collection of all degrees $\mathbf{x}$ such that the structure is $\mathbf{x}$-computable.

One of important and interesting area of studying non-computable structures is to describe which collections of degrees closed upward are realizable as a degree spectra of structures (or, of some special kind of structures such as linear orderings, Boolean algebras, groups, etc.). It is easy to check that the class of such collections is closed under intersection: for any structures $\mathcal{A}$ and $\mathcal{B}$ there a structure $\mathcal{C}$ such that $\mathbf{Sp}\,(\mathcal{A}) \cap \mathbf{Sp}\,(\mathcal{B}) = \mathbf{Sp}\,(\mathcal{C})$.

There are is a lot of various papers devoted to this direction (see e.g. [1], [3], [4], [9], [11], [13] etc.). In particular, by [11] for any degree $\mathbf{a}$ the collection $\{\mathbf{x} : \mathbf{x} \geq \mathbf{a}\}$ is realizable as spectrum of a structure.

There are also more surprising examples: Slaman [12] and, independently, Wehner [14], constructed structures with the degree spectrum $\{\mathbf{x} : \mathbf{x} > \mathbf{0}\}$. An easy relativization shows that for any degree $\mathbf{b}$ the collection $\{\mathbf{x} : \mathbf{x} > \mathbf{b}\}$ is also realizable as a spectrum. In the present paper we will try to obtain a more strong relativization: for a degree $\mathbf{b}$ to find a structure with the degree spectrum $\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$. We will see that for some degrees $\mathbf{b}$ this is not possible (Corollaries 4 and 5).

This problem is related to the following question posed by Miller [9]:

Question 1. (Miller). Does for any incomparable degrees $\mathbf{a}$ and $\mathbf{b}$ there exist a linear ordering $\mathcal{L}$ such that $\mathbf{a} \in \mathbf{Sp}\,(\mathcal{L})$ and $\mathbf{b} \notin \mathbf{Sp}\,(\mathcal{L})$?

As it follows from the next theorem the answer on this question is negative. Hence, there is a degree $\mathbf{b}$ such that, at least, the collection $\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ is not realizable as a spectrum of linear ordering.

Theorem 2 ([7])**.** *For each degree* $\mathbf{a} > \mathbf{0}$ *there is a degree* $\mathbf{b}$ *incomparable with* $\mathbf{a}$ *such that* $\mathbf{b}' \leq \mathbf{a}'$ *and for any linear ordering* $\mathcal{L}$

$$\mathbf{a} \in \mathbf{Sp}\,(\mathcal{L}) \Longrightarrow \mathbf{b} \in \mathbf{Sp}\,(\mathcal{L}).$$

Corollary 1. *There is a low degree* $\mathbf{b}$ *(i.e.,* $\mathbf{b}' = \mathbf{0}'$*), such that* $\mathbf{Sp}\,(\mathcal{L}) \neq \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ *for any linear ordering* $\mathcal{L}$.

Note that, in comparison with the linear orderings, in the general case (Theorem 7) for any low degree $\mathbf{b}$ we have some structure $\mathcal{A}_\mathbf{b}$ such that $\mathbf{Sp}\,(\mathcal{A}_\mathbf{b}) = \{\mathbf{x} \not\leq \mathbf{b}\}$.

The proof of Theorem 2 is just a more uniform version of the Richter's result, which is about only one ordering:

Theorem 3 (Richter [11])**.** *For every degree* $\mathbf{a} > \mathbf{0}$ *and any linear ordering* $\mathcal{L}$ *such that* $\mathbf{a} \in \mathbf{Sp}\,(\mathcal{L})$ *there is a degree* $\mathbf{b}$ *such that* $\mathbf{a} \not\leq \mathbf{b}$ *and* $\mathbf{b} \in \mathbf{Sp}\,(\mathcal{L})$. *(And hence, the collection* $\{\mathbf{x} : \mathbf{x} \geq \mathbf{a}\}$ *is realizable as a degree spectrum of a linear ordering if and only if* $\mathbf{a} = \mathbf{0}$*).*

Returning to algebraic structures in general, one can recall the following folklore result (see e.g. [13]).

Theorem 4. (Folklore). *Let* $\mathbf{A}$ *be a nonempty countable collection of degrees without least element and* $\mathcal{A}$ *be a structure such that* $\mathbf{A} \subseteq \mathbf{Sp}\,(\mathcal{A})$. *Then there is a degree* $\mathbf{b}$ *such that* $\mathbf{a} \not\leq \mathbf{b}$ *for all* $\mathbf{a} \in \mathbf{A}$, *and* $\mathbf{b} \in \mathbf{Sp}\,(\mathcal{A})$.

Corollary 2. (Folklore). *If* $\mathbf{A}$ *is a nonempty countable collection of degrees without least element, then the collection* $\cup_{\mathbf{a} \in \mathbf{A}}\{\mathbf{x} : \mathbf{x} \geq \mathbf{a}\}$ *is not realizable as a spectrum of an algebraic structure.*

Corollary 3. (Folklore). *If* $\mathbf{a}_0$ *and* $\mathbf{a}_1$ *are incomparable, then the collection* $\{\mathbf{x} : \mathbf{x} \geq \mathbf{a}_0\} \cup \{\mathbf{x} : \mathbf{x} \geq \mathbf{a}_1\}$ *is not realizable as a spectrum of an algebraic structure.*

For Theorem 4 the same uniformization is also possible:

Theorem 5 ([7]). *Let* $\mathbf{A}$ *be a nonempty countable collection of degrees without least element. Then there is a degree* $\mathbf{b}$ *such that* $\mathbf{a} \not\leq \mathbf{b}$ *for all* $\mathbf{a} \in \mathbf{A}$*, and for any algebraic structure* $\mathcal{A}$

$$\mathbf{A} \subseteq \mathbf{Sp}\,(\mathcal{A}) \Longrightarrow \mathbf{b} \in \mathbf{Sp}\,(\mathcal{A}).$$

Applying the last theorem with $\mathbf{A} = \{\mathbf{a}_0, \mathbf{a}_1\}$ for a pair of incomparable degrees $\mathbf{a}_0$ and $\mathbf{a}_1$, we get following

Corollary 4. *There is a degree* $\mathbf{b}$*, such that* $\mathbf{Sp}\,(\mathcal{A}) \neq \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ *for any algebraic structure* $\mathcal{A}$*.*

The construction of the degree $\mathbf{b}$ in the proof of the Theorem 5 essentially uses a list of all structures (up to isomorphism) which are computable relative to any element of $\mathbf{A}$. By this reason, it is very difficult to give an upper bound for the degree $\mathbf{b}$ from Corollary 4.

The following result is more weak than Theorem 5, but it has more constructive proof. This allows to bound the degree $\mathbf{b}$ by the double-jump.

Theorem 6 ([7]). *For any degree* $\mathbf{a}_0 > \mathbf{0}$ *there are degrees* $\mathbf{a}_1 \leq \mathbf{a}_0''$ *and* $\mathbf{b} \leq \mathbf{a}_0''$ *such that* $\mathbf{a}_0 \not\leq \mathbf{b}$*,* $\mathbf{a}_1 \not\leq \mathbf{b}$*, and for any algebraic structure* $\mathcal{A}$

$$\{\mathbf{a}_0, \mathbf{a}_1\} \subseteq \mathbf{Sp}\,(\mathcal{A}) \Longrightarrow \mathbf{b} \in \mathbf{Sp}\,(\mathcal{A}).$$

Corollary 5. *There is a degree* $\mathbf{b} \leq \mathbf{0}''$*, such that* $\mathbf{Sp}\,(\mathcal{A}) \neq \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ *for any algebraic structure* $\mathcal{A}$*.*

An essential idea of the proof of Theorem 6 is to use the following not difficult lemma:

Lemma. *For any set* A *there is a noncomputable set* $A_1 \leq_T A''$*, and there is a partially* A''*-computable function* θ *such that for any* $e \in \omega$

$$W_e^A \text{ is c.e. in } A_1 \iff \theta(e)\downarrow \iff W_e^A = W_{\theta(e)}.$$

Here W_e^A *is the standard numbering of all* A*-c.e. sets. In particular, the condition above is an effective version of* $\deg(A) \cap \deg(A_1) = \mathbf{0}$.

In fact, the degree $\mathbf{a}_1$ in Theorem 6 is the degree of the set A_1 from the lemma applied with $A \in \mathbf{a}_0$. Such set A_1 allows to bound existential types of structures $\mathcal{A}$ such that $\{\mathbf{a}_0, \mathbf{a}_1\} \subseteq \mathbf{Sp}\,(\mathcal{A})$: they must be c.e. and θ gives their c.e. indices.

2 The Structures with the Degree Spectra $\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$

In spite of Corollaries 4 and 5 there are a lot of nonzero degrees $\mathbf{b}$ such that the collection $\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ is a degree spectrum of a structure. Moreover, we can build such structures for any low degree $\mathbf{b}$.

Theorem 7 ([5]). *For any low degree* $\mathbf{b}$ *there is a structure* $\mathcal{A}$ *such that* $\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$.

The proof of this theorem is based on the same ideas as the proof of Wehner's result [14] on the structure with the degree spectrum $\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{x} : \mathbf{x} > \mathbf{0}\}$.

Namely, we first fix some effective coding $\mathcal{S} \mapsto \Gamma(\mathcal{S})$ of countable families $\mathcal{S}$ of subsets of ω into a irreflexive symmetric graphs $\Gamma(\mathcal{S})$ (see [3], [10]), such that for any degree $\mathbf{x}$

$$\mathbf{x} \in \mathbf{Sp}\,(\Gamma(\mathcal{S})) \iff \mathcal{S} \text{ is uniformly c.e. in } \mathbf{x}.$$

For example, we can define $\Gamma(\mathcal{S})$ as the graph with the vertices A, $\mathsf{B}_{i,j,X}$ (where $i, j \in \omega$, $X \in \mathcal{S}$), $\mathsf{C}_{i,j,X}$ (where $i \in \omega$, $j \in X \in \mathcal{S}$) and the edges $\{\mathsf{A}, \mathsf{B}_{i,0,X}\}$ (where $i \in \omega$, $X \in \mathcal{S}$), $\{\mathsf{B}_{i,j,X}, \mathsf{B}_{i,j+1,X}\}$ (where $i, j \in \omega$, $X \in \mathcal{S}$), $\{\mathsf{B}_{i,j,X}, \mathsf{C}_{i,j,X}\}$ (where $i \in \omega$, $j \in X \in \mathcal{S}$).

Then for a low degree $\mathbf{b}$ it is sufficient to find a countable family $\mathcal{S}$ such that for all degrees $\mathbf{x}$

$$\mathcal{S} \text{ is uniformly c.e. in } \mathbf{x} \iff \mathbf{x} \not\leq \mathbf{b}. \tag{1}$$

For the case $\mathbf{b} = \mathbf{0}$ Wehner [14], in fact, considered the family

$$\mathcal{F} = \{\{n\} \oplus F : n \in \omega \;\&\; F \subseteq \omega \;\&\; F \text{ is finite} \;\&\; F \neq W_n\},$$

where W_n is the standard numbering of all c.e. sets. By the Recursion Theorem, we immediately get that $\mathcal{F}$ is not uniformly c.e. (otherwise for every n we can effectively enumerate a set not equal to W_n). Note that, in the original proof Wehner used a direct diagonalization instead of using the Recursion Theorem. By this reason his definition is more complicate, but it can be equivalently reduced to the same form as the family $\mathcal{F}$.

Now to build a family $\mathcal{S} = \mathcal{F}_B$ satisfying the equivalence (1) with $\mathbf{b} = \deg(B)$, $\mathbf{b}' = \mathbf{0}'$, it is sufficient to consider the easy analogue of $\mathcal{F}$:

$$\mathcal{F}_B = \{\{n\} \oplus F : n \in \omega \;\&\; F \subseteq \omega \;\&\; F \text{ is finite} \;\&\; F \neq W_n^B\},$$

where W_n^B is the standard numbering of all B-c.e. sets. To prove that for all $\mathbf{x}$

$$\mathcal{F}_B \text{ is uniformly c.e. in } \mathbf{x} \iff \mathbf{x} \not\leq \deg(B).$$

it is necessary to use the fact that, if $B' \equiv_T \emptyset'$, then the predicate $K_0^B(m, n) \iff m \in W_n^B$ is a Δ_2^0-predicate.

By this reason, Theorem 7 can not be extended to non-low degrees $\mathbf{b}$ by the same way. For example, for $\mathbf{b} = \mathbf{0}'$ the predicate $K_0^{\emptyset'}$ is Σ_2^0-complete, although the theorem can be extended to such $\mathbf{b}$.

Theorem 8 ([6]). *For any c.e. degree* $\mathbf{b}$ *there is a structure* $\mathcal{A}$ *such that* $\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$.

For a c.e. set B the following family $\mathcal{S} = \mathcal{E}_B$ satisfies the equivalence (1) with $\mathbf{b} = \deg(B)$:

$$\mathcal{E}_B = \{\{n\} \oplus F : n \in \omega \;\&\; F \in \mathcal{P} \;\&\; F \neq W_n^B\},$$

where $\mathcal{P}$ is the family of all c.e. set, which are images of injective primitive recursive functions. Here such $\mathcal{P}$ is used because it is an example of sufficiently rich family which is uniformly c.e. and contains only infinite sets (in contrast with the infinite computable sets and the infinite c.e. sets).

We finish the section by the following remark. Theorems 7 and 8 give examples when a nontrivial union of two degree spectra is again a degree spectrum (by Corollary 3 this is not possible for such unions as $\{\mathbf{x} : \mathbf{x} \geq \mathbf{a}_0\} \cup \{\mathbf{x} : \mathbf{x} \geq \mathbf{a}_1\}$). Indeed, it is sufficient to take three different low (or c.e.) degrees $\mathbf{a}, \mathbf{b}, \mathbf{c}$ such that $\mathbf{a} \cap \mathbf{b} = \mathbf{c}$. Then, obviously,

$$\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{a}\} \cup \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\} = \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{c}\},$$

and each of these three collections is a degree spectrum. It follows from the next section (Corollary 7 and Theorem 11), that if $\mathbf{a}$ and $\mathbf{b}$ are low then the union $\{\mathbf{x} : \mathbf{x} \not\leq \mathbf{a}\} \cup \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}$ is a degree spectrum even though $\mathbf{a} \cap \mathbf{b}$ does not exist.

3 Some Other Degree Spectra Derived from the Families

The proof of Theorem 7 is based on the fact, that the predicate "$m \in W_n^B$" is Δ_2^0, if $B' \equiv \emptyset'$. This established the idea to change the numbering $\varepsilon_B(n) = W_n^B$ by an arbitrary numbering $\nu : \omega \to 2^\omega$ such that the predicate "$m \in \nu(n)$" is Δ_2^0, so called *a computable numbering of* Δ_2^0 *sets.* Let

$$\mathcal{F}(\nu) = \{\{n\} \oplus F : n \in \omega \ \& \ F \subseteq \omega \ \& \ F \text{ is finite} \ \& \ F \neq \nu(n)\},$$

and hence for the family $\mathcal{F}_B$ from the previous section we have $\mathcal{F}_B = \mathcal{F}(\varepsilon_B)$.

Note that the class of degree spectra of graphs $\Gamma(\mathcal{F}(\nu))$ is closed under intersection. Moreover, it is easy to check, that for any numberings $\nu, \eta : \omega \to 2^\omega$

$$\mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu))) \cap \mathbf{Sp}\,(\Gamma(\mathcal{F}(\eta))) = \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu + \eta))),$$

where $\nu + \eta$ is the standard sum of numberings: for all $n \in \omega$

$$(\nu + \eta)(2n) = \nu(n); \quad (\nu + \eta)(2n + 1) = \eta(n).$$

Theorem 9 describes the degree spectra of the graphs $\Gamma(\mathcal{F}(\nu))$, where the predicate "$m \in \nu(n)$" is Δ_2^0.

Theorem 9 ([5])**.** *Let ν be a computable numbering of Δ_2^0 sets. Then for a set $X \subseteq \omega$ the following conditions are equivalent:*

1. $\deg(X) \in \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu)))$;
2. *there is a computable function $f : \omega^2 \to \omega$ such that for all $m, n \in \omega$ we have $W_{f(n,m)}^X \neq \nu(n)$, $\{k \in \omega : k < m\} \subseteq W_{f(n,m)}^X$, and $W_{f(n,m)}^X$ is finite;*
3. *there is a computable function $f : \omega^2 \to \omega$ such that for all $m, n \in \omega$ we have $W_{f(n,m)}^X \neq \nu(n)$ and $\{k \in \omega : k < m\} \subseteq W_{f(n,m)}^X$.*

As a corollary, we get that for computable numberings ν of Δ^0_2 sets degree spectra of graphs $\Gamma(\mathcal{F}(\nu))$ have the same behavior on non-low degrees.

Corollary 6. *If $\mathbf{x}' > \mathbf{0}'$ and ν is a computable numbering of Δ^0_2 sets, then $\mathbf{x} \in \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu)))$.*

Indeed, if $\emptyset' <_T X'$ then a computable function f, such that

$$W^X_{f(n,m)} = X' \cup \{k \in \omega : k < m\},$$

satisfies the condition 3 of Theorem 9 (since $X' \notin \Delta^0_2$).

For some computable numberings ν of Δ^0_2 sets the description of $\mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu)))$ can be made more easy. Namely, we say that a numbering ν *is an LR-numbering*, if for some computable functions $L, R : \omega \to \omega$ we have

$$\nu(n) = \nu(L(n)) \oplus_1 \nu(R(n))$$

for each $n \in \omega$, where

$$X \oplus_1 Y = \{\langle 2x, y\rangle : \langle x, y\rangle \in X\} \cup \{\langle 2x+1, y\rangle : \langle x, y\rangle \in Y\}$$

is the bijection between $2^\omega \times 2^\omega$ and 2^ω. In fact, for the next theorem no matter which of $X \oplus_1 Y$ or the standard $X \oplus Y = \{2x : x \in X\} \cup \{2x+1 : x \in Y\}$ is used in the definition of LR-numberings, but we prefer to use $\oplus_1$ instead of $\oplus$ for the sake of Corollary 7.

Theorem 10 ([5]). *Let ν be a computable LR-numbering of Δ^0_2 sets. Then for a degree $\mathbf{x}$ the following conditions are equivalent:*

1. *$\mathbf{x} \in \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu)))$;*
2. *the family of all $\mathbf{x}$-c.e. sets is not a subset of the image of ν (i.e. there is an $\mathbf{x}$-c.e. set $Z \notin \{\nu(n) : n \in \omega\}$).*

Note that the numbering $\nu(n) = W^B_n$ is an LR-numbering for any $B \subseteq \omega$. Thus, Theorem 10 is a generalization of Theorem 7.

Let $X \oplus_2 Y$ be the another bijection between $2^\omega \times 2^\omega$ and 2^ω:

$$X \oplus_2 Y = \{\langle x, 2y\rangle : \langle x, y\rangle \in X\} \cup \{\langle x, 2y+1\rangle : \langle x, y\rangle \in Y\}.$$

For numberings ν and η define the numbering $\nu \times \eta$ as follows: for every $n, m \in \omega$

$$(\nu \times \eta)(\langle n, m\rangle) = \nu(n) \oplus_2 \nu(m).$$

By the obvious identity

$$(A \oplus_1 B) \oplus_2 (C \oplus_1 D) = (A \oplus_2 C) \oplus_1 (B \oplus_2 D)$$

it follows, that if ν and η are LR-numberings then $\nu \times \eta$ is also an LR-numbering. Now the next corollary follows immediately:

Corollary 7. *Let ν and η be computable LR-numberings of Δ^0_2 sets. Then $\nu+\eta$ and $\nu \times \eta$ are also computable LR-numberings of Δ^0_2 sets, and*

$$\mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu))) \cap \mathbf{Sp}\,(\Gamma(\mathcal{F}(\eta))) = \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu + \eta))),$$

$$\mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu))) \cup \mathbf{Sp}\,(\Gamma(\mathcal{F}(\eta))) = \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu \times \eta))).$$

Note that for the LR-numberings $\varepsilon_B = W^B_n$, $B' \equiv_T \emptyset'$, Corollary 7 can be strengthen:

Theorem 11 ([5]). *If $B' \equiv_T \emptyset'$, then for any computable numberings ν of Δ^0_2 sets*

$$\mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu))) \cup \mathbf{Sp}\,(\Gamma(\mathcal{F}(\varepsilon_B))) = \mathbf{Sp}\,(\Gamma(\mathcal{F}(\nu \times \varepsilon_B))).$$

4 Further Questions

The questions from this paragraph are closely related to the results from the previous three paragraphs. Namely, seeing Theorem 2 it is interesting to find two different degrees which compute the same (up to isomorphism) collection of linear orderings. By a result of Knight (see e.g. [1]) this two degrees must be incomparable.

Question 2. Are there two incomparable degrees $\mathbf{a}$ and $\mathbf{b}$ such that for any linear ordering $\mathcal{L}$

$$\mathbf{a} \in \mathbf{Sp}\,(\mathcal{L}) \iff \mathbf{b} \in \mathbf{Sp}\,(\mathcal{L})?$$

Also, it is not so clear how to find the degree $\mathbf{b}$ in Theorem 5 more effectively. In particular:

Question 3. Let $\mathbf{a}_0$ and $\mathbf{a}_1$ be incomparable arithmetical degrees. Is there an arithmetical degree $\mathbf{b}$ such that $\mathbf{a}_0 \not\leq \mathbf{b}$, $\mathbf{a}_1 \not\leq \mathbf{b}$, and for any algebraic structure $\mathcal{A}$

$$\{\mathbf{a}_0, \mathbf{a}_1\} \subseteq \mathbf{Sp}\,(\mathcal{A}) \Longrightarrow \mathbf{b} \in \mathbf{Sp}\,(\mathcal{A})?$$

The related questions are about possible extensions of Thorems 7 and 8:

Question 4. Does for any degree $\mathbf{b} \leq \mathbf{0}'$ there exist a structure $\mathcal{A}$ such that

$$\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{b}\}?$$

Question 5. Is there a structure $\mathcal{A}$ such that

$$\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{0}''\},$$

or, at least, $\mathbf{Sp}\,(\mathcal{A}) = \{\mathbf{x} : \mathbf{x} \not\leq \mathbf{0}^{(n)}\}$ for some $n \geq 2$?

Finally, Theorem 10 allows for any uniformly Δ^0_2 family $\mathcal{C}$, which is closed under left and right parts of $\oplus_1$ (i.e., if $X \oplus_1 Y \in \mathcal{C}$ then $X \in \mathcal{C}$ and $Y \in \mathcal{C}$), to create a structure with the degree spectra

$$\mathbf{S}(\mathcal{C}) = \{\mathbf{x} : (\exists Z \notin \mathcal{C})[Z \text{ is } \mathbf{x}\text{-c.e.}]\}.$$

For example, consider the family Δ^{-1}_ω of all ω-c.e. sets. Recall, that a set A is ω-c.e. if there are computable functions f and g such that for all $x \in \omega$

$$A(x) = \lim_s f(x,s) \text{ and card } \{s : f(x,s) \neq f(x,s+1)\} < g(x).$$

Then $\mathbf{S}(\Delta^{-1}_\omega)$ consists from the degrees of sets X such that the Turing jump X' is not ω-c.e. Note that the condition $X' \in \Delta^{-1}_\omega$ is not equivalent to a computability of X (for example, for the sets constructed for the Original Friedberg-Muchnik Theorem).

The family Δ^{-1}_ω is the first infinite level of Ershov Hierarchy [2]. The closest levels are Σ^{-1}_ω and Π^{-1}_ω. Namely, $A \in \Sigma^{-1}_\omega$, if there are a computable function f and a partially computable fucntion g such that for all $x \in \omega$ we have $A(x) = \lim_s f(x,s)$,

$$x \in A \Longrightarrow g(x) \text{ is defined, and}$$

$$g(x) \text{ is defined} \Longrightarrow \text{card } \{s : f(x,s) \neq f(x,s+1)\} < g(x).$$

The level Π^{-1}_ω consists from the complements of sets from Σ^{-1}_ω. The families Σ^{-1}_ω and Π^{-1}_ω are again uniformly Δ^0_2 families $\mathcal{C}$, closed under left and right parts of $\oplus_1$, but it is not clear, are the collections $\mathbf{S}(\Sigma^{-1}_\omega)$ and $\mathbf{S}(\Pi^{-1}_\omega)$ equal to $\mathbf{S}(\Delta^{-1}_\omega)$:

Question 6. Is there a set $X \subseteq \omega$ such that $X' \in \Sigma^{-1}_\omega - \Delta^{-1}_\omega$?

Question 7. Is there a set $X \subseteq \omega$ such that $X' \in \Pi^{-1}_\omega - \Delta^{-1}_\omega$?

Acknowledgement. The research is partially supported by RFBR grant 05-01-00605.

References

1. Downey, R.: On Presentations of Algebraic Structures. In: Sorbi, A. (ed.) Complexity, Logic, and Recursion Theory, pp. 157–205 Dekker, New York (1997)
2. Ershov, Y.L.: On a hierarchy of sets II. Algebra i Logika 7(4), 15–47 (1968)
3. Goncharov, S.S., Harizanov, V.S., Knight, J.F., McCoy, C., Miller, R., Solomon, R.: Enumerations in computable structure theory. Annals of Pure and Applied Logic 136, 219–246 (2005)
4. Hirschfeldt, D.R., Khoussainov, B., Shore, R.A., Slinko, A.M.: Degree spectra and computable dimensions in algebraic structures. Annals of Pure and Applied Logic 115, 71–113 (2002)
5. Kalimullin, I.Sh.: Degree spectra of some algebraic structures, Algebra and Logic (to appear)

6. Kalimullin, I.Sh.: A non-Δ_2^0 structure is computable in any non-Δ_2^0 degree (to appear)
7. Kalimullin, I.Sh.: Restrictions on degree spectra of structures (to appear)
8. Knight, J.F.: Degrees coded in jumps of orderings. Journal of Symbolic Logic 51, 1034–1042 (1986)
9. Miller, R.G.: The Δ_2^0-spectrum of a linear order. Journal of Symbolic Logic 66, 470–486 (2001)
10. Morozov, A.S., Puzarenko, V.G.: Σ-Subsets of Natural Numbers. Algebra and Logic 43(3), 162–178 (2004)
11. Richter, L.J.: Degrees of Structures. Journal of Symbolic Logic 46, 723–731 (1981)
12. Slaman, T.: Relative to any Nonrecursive Set. In: Proceedings of the American Mathematical Society vol. 126, pp. 2117–2122 (1998)
13. Soskov, I.N.: Degree spectra and co-spectra of structures. Ann. Univ. Sofia 96, 45–68 (2003)
14. Wehner, S.: Enumerations, Countable Structures, and Turing Degrees. In: Proceedings of the American Mathematical Society vol. 126, pp. 2131–2139 (1998)

Confluence of Cut-Elimination Procedures for the Intuitionistic Sequent Calculus

Kentaro Kikuchi

RIEC, Tohoku University
Katahira 2-1-1, Aoba-ku, Sendai 980-8577, Japan
kentaro@nue.riec.tohoku.ac.jp

Abstract. We prove confluence of two cut-elimination procedures for the implicational fragment of a standard intuitionistic sequent calculus. One of the cut-elimination procedures uses global proof transformations while the other consists of local ones. Both of them include permutation of cuts to simulate β-reduction in an isomorphic image of the λ-calculus. We establish the confluence properties through a conservativity result on the cut-elimination procedures.

Keywords: Sequent calculus, Cut-elimination, Confluence, λ-calculus, Explicit substitution.

1 Introduction

Gentzen's cut-elimination theorem [4] has long been a great influence on logic and theoretical computer science. Recent development of structural proof theory is revealing the computational aspect of cut-elimination procedures in the same sense that proof transformations in natural deduction play through the Curry-Howard correspondence [7]. In [8], the author identified a subset of proofs in a standard sequent calculus that correspond to simply typed λ-terms, and defined a reduction relation on those proofs that precisely corresponds to β-reduction of the simply typed λ-calculus. Since the reduction relation is simulated by a local-step cut-elimination procedure, the system of proof terms for the sequent calculus can be considered as a syntactical extension of the λ-calculus including reductions. It is worth noticing that the correspondence holds also for the type-free case, so the reduction system in [8] can simulate the type-free λ-calculus, which means that it is strong enough to represent all computations.

In this paper, we study confluence of a cut-elimination procedure based on the one introduced in [8]. Since the reduction system in [8] is not confluent, we modify one of the reduction rules to a more restricted form. The resulting system is still strong enough to simulate β-reduction in the isomorphic image of the λ-calculus. We also consider another cut-elimination procedure which includes global proof transformations in the style of [2]. The reduction system representing the cut-elimination procedure is similar to one considered in [3], which uses meta-operations like meta-substitution in the λ-calculus.

It is well-known that a local-step cut-elimination procedure has a similarity to explicit substitution calculi. Our proof method is essentially the one often used in the field of explicit substitutions (see, e.g. [1]), called the interpretation method [6]. This method projects reduction steps with explicit substitutions onto those using meta-substitution, and reduces the confluence problem of an explicit substitution calculus to that of the original λ-calculus. To apply this method to the case of a cut-elimination procedure, we need to find an appropriate reduction using meta-operations. Although meta-operations are used in the reduction system for the global cut-elimination procedure mentioned above, it turns out that the system is not appropriate for a target calculus of the method because proving confluence of it has a delicate matter that is not present in the case of the usual λ-calculus. So we define another reduction relation on a certain class of proof terms, and first prove its confluence by the method of parallel reduction [10]. Confluence of the two cut-elimination procedures is inferred from confluence of this reduction by the interpretation method.

Danos et al. [2] proved confluence of their cut-elimination procedures with global proof transformations, depending on confluence of proof nets [5]. In this paper, we give a direct proof of confluence of a similar cut-elimination procedure, using proof terms and meta-operations on them. Our method works also for cut-elimination procedures consisting of local proof transformations and for underlying untyped calculi allowing non-terminating computations.

The paper is organized as follows. In Section 2 we introduce sequent calculus and cut-elimination procedures. In Section 3 we study a subcalculus and meta-operations from the reduction systems. In Section 4 we define another reduction relation and prove its confluence. In Section 5 we prove confluence of the cut-elimination procedures. In Section 6 we conclude by suggestions for future work.

To save space we omit details of proofs, but a full version with all details is available at `http://www.nue.riec.tohoku.ac.jp/user/kentaro/`.

2 Sequent Calculus and Cut-Elimination Procedures

In this section we introduce a term notation for proofs in a standard sequent calculus for intuitionistic implicational logic, following [8]. Our cut-elimination procedures are represented as reduction rules for those terms.

First, the set of raw terms for sequent proofs is defined by the grammar: $t ::= x \mid \lambda x.t \mid \langle xt/x\rangle t \mid [t/x]t$ where x ranges over a denumerable set of variables. $\langle \_\_/\_\rangle\_$ and $[\_/\_]\_$ are function symbols like explicit substitutions and not meta-substitution ($[\_/\_]\_$ is called the cut-constructor). We use letters x, y, z, w for variables and t, s, r, u for terms. The notions of free and bound variables are defined as usual, with an additional clause that the variable x in $\langle ys/x\rangle t$ or $[s/x]t$ binds the free occurrences of x in t. The set of free variables of a term t is denoted by $FV(t)$. We often use the notation $\langle \underline{x}s/y\rangle t$ to denote $\langle xs/y\rangle t$ if $x \notin FV(s) \cup FV(t)$. The symbol $\equiv$ denotes syntactical equality modulo α-conversion; so for example, $\langle zr/x\rangle\langle \underline{x}s/y\rangle t \equiv \langle zr/w\rangle\langle \underline{w}s/y\rangle t$.

The term assignment for sequent proofs of intuitionistic implicational logic is given in Table 1. We define a context, ranged over by Γ, as a finite set of pairs $\{x_1 : A_1, \ldots, x_n : A_n\}$ where the variables are pairwise distinct. The context

Table 1. Sequent calculus and local cut-elimination

$$Ax\ \frac{}{\Gamma, x : A \vdash x : A} \qquad L\supset \frac{\Gamma \vdash s : A \quad \Gamma, y : B \vdash t : C}{\Gamma, x : A \supset B \vdash \langle xs/y\rangle t : C}\ y \notin \Gamma$$

$$R\supset \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\ x \notin \Gamma \qquad Cut\ \frac{\Gamma \vdash s : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash [s/x]t : B}\ x \notin \Gamma$$

$\langle \underline{x}s/y\rangle t$ is used for $\langle xs/y\rangle t$ when $x \notin FV(s) \cup FV(t)$. In that case we assume $x \notin \Gamma$ in the rule $L \supset$.

(1)	$[t/x]y \to y \quad (y \not\equiv x)$
(2)	$[t/x]x \to t$
(3)	$[s/x](\lambda y.t) \to \lambda y.[s/x]t$
(4)	$[r/z]\langle xs/y\rangle t \to \langle x([r/z]s)/y\rangle [r/z]t \quad (x \not\equiv z)$
(5)	$[r/x]\langle xs/y\rangle t \to [r/x]\langle \underline{x}([r/x]s)/y\rangle [r/x]t \quad$ if $x \in FV(s) \cup FV(t)$
(6)	$[z/x]\langle \underline{x}s/y\rangle t \to \langle zs/y\rangle t$
(7′)	$[\langle xs/y\rangle t/z]\langle \underline{z}s'/w\rangle t' \to \langle xs/y\rangle [t/z]\langle \underline{z}s'/w\rangle t'$
(*Beta*)	$[\lambda z.r/x]\langle \underline{x}s/y\rangle t \to [[s/z]r/y]t$
(*Perm*$_1$)	$[[r/x]\langle \underline{x}s/y\rangle t/z]\langle \underline{z}s'/w\rangle t' \to [r/x][\langle \underline{x}s/y\rangle t/z]\langle \underline{z}s'/w\rangle t'$
(*Perm*$_2$)	$[u/w][\lambda z.r/x]\langle \underline{x}s/y\rangle t \to [[u/w](\lambda z.r)/x][u/w]\langle \underline{x}s/y\rangle t$

$\Gamma, x : A$ denotes the union $\Gamma \cup \{x : A\}$, and $x \notin \Gamma$ means that x does not appear in Γ. For precise representation of proofs by terms, we should specify formulas on binders, but we will omit them for brevity. If $x \notin FV(s) \cup FV(t)$ in the term $\langle xs/y\rangle t$, we assume $x \notin \Gamma$ in the rule $L \supset$, which means the formula $A \supset B$ is introduced without implicit contraction.

The reduction rules in Table 1 define a cut-elimination procedure consisting of local proof transformations. The reduction relation $\to_{\mathrm{cut}}$ is defined by the contextual closures of these reduction rules. We use $\xrightarrow{+}_{\mathrm{cut}}$ for its transitive closure, and $\xrightarrow{*}_{\mathrm{cut}}$ for its reflexive transitive closure. These kinds of notations are also used for the notions of other reductions in this paper.

The reduction system without the rule (*Beta*) is denoted by x. This subcalculus plays an important role in this paper and is studied in detail in Section 3.

The reduction rules (1) through (5) correspond to cut-elimination steps that permute a cut upwards through its right subproof. The rules (6) and (7′) correspond to steps permuting a cut upwards through its left subproof. The rule (*Beta*) corresponds to the key-case which breaks a cut on an implication into two cuts on its subformulas. The rules (*Perm*$_1$) and (*Perm*$_2$) permute two cuts

Table 2. Global cut-elimination

$(Beta)$	$[\lambda z.r/x]\langle \underline{x}s/y\rangle t$	$\to$	$[[s/z]r/y]t$	
$(left)$	$[u/x]\langle \underline{x}s/y\rangle t$	$\to$	$\langle \pmb{\{} u \pmb{\}} s/y\rangle t$	if u is not of the form $\lambda z.r$
$(right)$	$[u/x]r$	$\to$	$\{u/x\}r$	if r is not of the form $\langle \underline{x}s/y\rangle t$

where $\{\_/\_\}\_$ and $\langle \pmb{\{}\_\pmb{\}}\_/\_\rangle\_$ are the meta-operations defined as follows:

$$\begin{aligned}
\{u/x\}y &=_{def} y \qquad (y \not\equiv x)\\
\{u/x\}x &=_{def} u\\
\{u/x\}(\lambda y.t) &=_{def} \lambda y.\{u/x\}t\\
\{u/x\}\langle zs/y\rangle t &=_{def} \langle z(\{u/x\}s)/y\rangle\{u/x\}t \qquad (z \not\equiv x)\\
\{u/x\}\langle xs/y\rangle t &=_{def} [u/x]\langle \underline{x}(\{u/x\}s)/y\rangle\{u/x\}t\\
\{u/x\}[s/y]t &=_{def} [\{u/x\}s/y]\{u/x\}t
\end{aligned}$$

$$\begin{aligned}
\langle \pmb{\{} z \pmb{\}} s/y\rangle t &=_{def} \langle zs/y\rangle t\\
\langle \pmb{\{} \lambda z.r \pmb{\}} s/y\rangle t &=_{def} [\lambda z.r/x]\langle \underline{x}s/y\rangle t\\
\langle \pmb{\{} \langle zs'/w\rangle r \pmb{\}} s/y\rangle t &=_{def} \langle zs'/w\rangle\langle \pmb{\{} r \pmb{\}} s/y\rangle t\\
\langle \pmb{\{} [s'/w]r \pmb{\}} s/y\rangle t &=_{def} [s'/w]\langle \pmb{\{} r \pmb{\}} s/y\rangle t
\end{aligned}$$

with some restrictions. In $(Perm_1)$, the left rule over the lower cut is another cut, and the right rules over both cuts must be $L\supset$ that introduces the cut-formula without implicit contraction. In $(Perm_2)$, the right rule over the lower cut is another cut, which must construct a proof corresponding to a redex of the rule $(Beta)$.

The original cut-elimination procedure in [8] uses the following rule (7) instead of (7′):

$$(7) \qquad [\langle xs/y\rangle t/z]r \to \langle xs/y\rangle [t/z]r$$

This rule makes the cut-elimination procedure non-confluent (e.g., the critical pair $w \leftarrow [\langle xs/y\rangle t/z]w \to \langle xs/y\rangle [t/z]w$ is not joinable). For a confluent cut-elimination procedure, it is therefore necessary to restrict reductions. The rule (7′) restricts the rule (7) so that the right rule over the cut must be $L\supset$ that introduces the cut-formula without implicit contraction. As shown in [8], this cut-elimination procedure is still strong enough to simulate β-reduction in the isomorphic image of the λ-calculus.

Table 2 presents another cut-elimination procedure which includes global proof transformations. The cut-elimination procedure is implemented by reduction rules that use meta-operations $\{\_/\_\}\_$ and $\langle \pmb{\{}\_\pmb{\}}\_/\_\rangle\_$, analogously to proof transformations in natural deduction. The operation $\langle \pmb{\{}\_\pmb{\}}\_/\_\rangle\_$ corresponds to the cut-elimination process where the right rule over the cut is $L\supset$ introducing the cut-formula without implicit contraction, and the cut is permuted upwards through its left subproof. Note that the conditions of $(left)$ and $(right)$ make the cut-elimination procedure first permute a cut upwards through its right subproof and then through its left subproof. The reduction relation generated by the rules (*Beta*), (*left*) and (*right*) is denoted by $\to_{\text{gcut}}$.

The following lemma is immediate from the definition of $\{\_/\_\}\_$.

Lemma 1. *If $x \notin FV(t)$ then $\{u/x\}t \equiv t$.*

Proof. By induction on the structure of t. □

3 The Subcalculus $\mathtt{x}$ and Meta-operations

In this section we study properties of the subcalculus $\mathtt{x}$ which is the reduction system in Table 1 without the rule (*Beta*). In the typed case, it corresponds to the cut-elimination steps except the key-case, i.e., the case where both left and right rules over the cut rule introduce the cut-formula. We show that the subcalculus $\mathtt{x}$ is strongly normalizing and confluent, and investigate its relation to the meta-operations in Table 2.

First we give a technical definition to prove strong normalization of the subcalculus $\mathtt{x}$.

Definition 1. *A term $[s/x]t$ is called an* application term *if t is one of the forms: $[u/w]\langle \underline{x}s'/y\rangle t'$, $\langle \underline{x}s'/y\rangle t'$ and $[\langle \underline{x}s'/y\rangle t'/z]\langle \underline{z}s''/w\rangle t''$, where x occurs only once in t.*

Lemma 2. *If $[s/x]t$ is an application term and $t \to_{\mathtt{x}} t'$, then $[s/x]t'$ is also an application term.*

Proof. It suffices to check each case. □

Proposition 1. *The subcalculus $\mathtt{x}$ is strongly normalizing.*

Proof. The proof is by interpretation. We define a function h as follows:

$$\begin{aligned}
h(x) &=_{def} 1\\
h(\lambda x.t) &=_{def} h(t)+1\\
h(\langle xs/y\rangle t) &=_{def} h(s)+h(t)+1\\
h([s/x]t) &=_{def} \begin{cases}(h(s)+1)^2 \times h(t) & \text{if } [s/x]t \text{ is an application term}\\ (h(s)+1)^{2\times h(t)} & \text{otherwise}\end{cases}
\end{aligned}$$

and observe that if $t \to_{\mathtt{x}} t'$ then $h(t) > h(t')$. If $t \equiv [s/x]r$ is an application term and $r \to_{\mathtt{x}} r'$, then we use Lemma 2. □

Proposition 2. *The subcalculus $\mathtt{x}$ is confluent.*

Proof. By Newman's Lemma, it suffices to check the local confluence. There are two critical pairs caused by the rules (7′) and (*Perm*$_1$), and by (*Perm*$_1$) and (*Perm*$_1$), both of which are joinable. □

As a result, we can define the unique $\mathtt{x}$-normal form of each term.

Definition 2. *The unique* $\mathtt{x}$*-normal form of a term* t *is denoted by* $\mathtt{x}(t)$*.*

A term in which every cut-constructor forms a redex of the rule ($Beta$) is called a *Beta-term*. The relation between *Beta*-terms and $\mathtt{x}$-normal forms is as follows.

Proposition 3. t *is a Beta-term if and only if* t *is in* $\mathtt{x}$*-normal form.*

Proof. The only if part is by induction on the structure of *Beta*-terms. We prove the if part by induction on the structure of t. Suppose that t is in $\mathtt{x}$-normal form. Then by the induction hypothesis, all subterms of t are *Beta*-terms. Now, if t is not a *Beta*-term then t is of the form $[u/x]r(\not\equiv [\lambda z.r'/x]\langle \underline{x}s/y\rangle t')$ where u, r are *Beta*-terms. In this case, t is an $\mathtt{x}$-redex, which is a contradiction. □

The next lemma shows that the subcalculus $\mathtt{x}$ correctly simulates the meta-operations on *Beta*-terms.

Lemma 3. *Let* u, t, s *be Beta-terms. Then*

1. $[u/x]t \xrightarrow{*}_{\mathtt{x}} \{u/x\}t$,
2. $[u/x]\langle \underline{x}s/y\rangle t \xrightarrow{*}_{\mathtt{x}} \langle \{\!\{u\}\!\}s/y\rangle t$. *Moreover,* $\langle \{\!\{u\}\!\}s/y\rangle t$ *is a Beta-term, hence* $\mathtt{x}([u/x]\langle \underline{x}s/y\rangle t) \equiv \langle \{\!\{u\}\!\}s/y\rangle t$.

Proof.

1. By induction on the structure of t.
2. By induction on the structure of u. □

Next we show that $\rightarrow_{\mathrm{gcut}}$ is sufficient to reach $\mathtt{x}$-normal forms.

Lemma 4. *Let* u, s, t *be Beta-terms. Then*

1. $[u/x]\langle \underline{x}s/y\rangle t \xrightarrow{*}_{\mathrm{gcut}} \mathtt{x}([u/x]\langle \underline{x}s/y\rangle t)$,
2. $\{u/x\}t \xrightarrow{*}_{\mathrm{gcut}} \mathtt{x}(\{u/x\}t)$.

Proof.

1. If $u \equiv \lambda z.r$ then $[u/x]\langle \underline{x}s/y\rangle t \equiv \mathtt{x}([u/x]\langle \underline{x}s/y\rangle t)$. If u is not of the form $\lambda z.r$, then $[u/x]\langle \underline{x}s/y\rangle t' \rightarrow_{left} \langle \{\!\{u\}\!\}s/y\rangle t' \equiv \mathtt{x}([u/x]\langle \underline{x}s/y\rangle t')$ by Lemma 3 (2).
2. By induction on the structure of t. □

Lemma 5. $t \xrightarrow{*}_{\mathrm{gcut}} \mathtt{x}(t)$*.*

Proof. By induction on the structure of t. □

The following lemmas are essential to the parallel reduction method in the next section. Note that $\{u/x\}\langle \{\!\{t\}\!\}s/y\rangle t' \equiv \langle \{\!\{\{u/x\}t\}\!\}s/y\rangle t'$ instead of Lemma 7 does not hold in general; for example, $\{z/x\}\langle \{\!\{x\}\!\}w/y\rangle w' \equiv [z/x]\langle xw/y\rangle w' \not\equiv \langle zw/y\rangle w' \equiv \langle \{\!\{\{z/x\}x\}\!\}w/y\rangle w'$. This makes it difficult to apply a direct parallel reduction method to $\rightarrow_{\mathrm{gcut}}$. So we consider the meta-operation $\{\_/\_\}\_$ followed by $\mathtt{x}$-reductions to $\mathtt{x}$-normal forms (i.e., $\mathtt{x}(\{\_/\_\}\_)$), and in the next section we define another reduction relation that matches such operation.

Lemma 6. $\langle \{\!\{\langle \{\!\{u\}\!\}s/y\rangle t\}\!\}s'/y'\rangle t' \equiv \langle \{\!\{u\}\!\}s/y\rangle \langle \{\!\{t\}\!\}s'/y'\rangle t'$*.*

Proof. By induction on the structure of u. □

Lemma 7. *Let u, t, s, t' be Beta-terms. Then*
$\mathtt{x}(\{u/x\}\langle\{\!\{t\}\!\}s/y\rangle t') \equiv \langle\{\!\{\mathtt{x}(\{u/x\}t)\}\!\}\mathtt{x}(\{u/x\}s)/y\rangle\mathtt{x}(\{u/x\}t')$.
In particular, if $x \notin FV(s) \cup FV(t')$ then
$\mathtt{x}(\{u/x\}\langle\{\!\{t\}\!\}s/y\rangle t') \equiv \langle\{\!\{\mathtt{x}(\{u/x\}t)\}\!\}s/y\rangle t'$.

Proof. By induction on the structure of t. □

Lemma 8. *Let u, s, t be Beta-terms with $y \notin FV(u)$. Then*
$\mathtt{x}(\{u/x\}\mathtt{x}(\{s/y\}t)) \equiv \mathtt{x}(\{\mathtt{x}(\{u/x\}s)/y\}\mathtt{x}(\{u/x\}t))$.

Proof. By induction on the structure of t. □

4 Confluence of β-Reduction

In this section we introduce another reduction relation on *Beta*-terms and show that it is confluent by the parallel reduction method [10]. Confluence of the two cut-elimination procedures is proved using projections onto this reduction.

The reduction relation $\to_\beta$ on *Beta*-terms is defined by the contextual closure of the rule:

$$(\beta) \quad [\lambda z.r/x]\langle \underline{x}s/y\rangle t \to \mathtt{x}(\{\mathtt{x}(\{s/z\}r)/y\}t)$$

This reduction relation is indeed an extension of β-reduction on pure terms (i.e., the isomorphic image of λ-terms) in [8].

Proposition 4. *Let t, t' be Beta-terms.*

1. *If $t \to_\beta t'$ then $t \xrightarrow{+}_{\mathrm{cut}} t'$.*
2. *If $t \to_\beta t'$ then $t \xrightarrow{+}_{\mathrm{gcut}} t'$.*

Proof. By induction on the reduction relation $\to_\beta$. We treat the case where the reduction is at the root. Then

$$\begin{array}{rll}
[\lambda z.r/x]\langle \underline{x}s/y\rangle t_0 & \to_{Beta} [[s/z]r/y]t_0 & \\
& \xrightarrow{*}_{\mathtt{x}} \mathtt{x}([\mathtt{x}([s/z]r)/y]t_0) & (*) \\
& \equiv \ \mathtt{x}([\mathtt{x}(\{s/z\}r)/y]t_0) & \text{(by Lemma 3 (1))} \\
& \equiv \ \mathtt{x}(\{\mathtt{x}(\{s/z\}r)/y\}t_0) & \text{(by Lemma 3 (1))}
\end{array}$$

where the step $(*)$ can also be established with $\xrightarrow{*}_{\mathrm{gcut}}$ by Lemma 5. □

The parallel reduction $\Rightarrow$ for $\to_\beta$ is defined by the rules in Table 3.

Lemma 9. *For every Beta-term t, $t \Rightarrow t$.*

Proof. By induction on the structure of t. □

Table 3. Parallel reduction

$$\frac{}{x \Rrightarrow x}\ (pr_1) \qquad \frac{t \Rightarrow t'}{\lambda x.t \Rightarrow \lambda x.t'}\ (pr_2) \qquad \frac{s \Rightarrow s' \quad t \Rightarrow t'}{\langle xs/y\rangle t \Rightarrow \langle xs'/y\rangle t'}\ (pr_3)$$

$$\frac{r \Rightarrow r' \quad s \Rightarrow s' \quad t \Rightarrow t'}{[\lambda z.r/x]\langle \underline{x}s/y\rangle t \Rightarrow [\lambda z.r'/x]\langle \underline{x}s'/y\rangle t'}\ (pr_4)$$

$$\frac{r \Rightarrow r' \quad s \Rightarrow s' \quad t \Rightarrow t'}{[\lambda z.r/x]\langle \underline{x}s/y\rangle t \Rightarrow \mathtt{x}(\{\mathtt{x}(\{s'/z\}r')/y\}t')}\ (pr_5)$$

Lemma 10

1. *If $t \to_\beta t'$ then $t \Rightarrow t'$.*
2. *If $t \Rightarrow t'$ then $t \xrightarrow{*}_\beta t'$.*
3. *If $u \Rightarrow u'$, $s \Rightarrow s'$ and $t \Rightarrow t'$ then $\langle \{\!\{u\}\!\} s/y\rangle t \Rightarrow \langle \{\!\{u'\}\!\} s'/y\rangle t'$.*
4. *If $u \Rightarrow u'$ and $t \Rightarrow t'$ then $\mathtt{x}(\{u/x\}t) \Rightarrow \mathtt{x}(\{u'/x\}t')$.*

Proof.

1. By induction on the reduction relation $\to_\beta$.
2. By induction on the definition of $t \Rightarrow t'$.
3. By induction on the definition of $u \Rightarrow u'$.
4. By induction on the definition of $t \Rightarrow t'$. □

Definition 3. *For each Beta-term t, the term $t^\star$ is defined inductively as follows:*

1. $x^\star =_{def} x$,
2. $(\lambda x.t)^\star =_{def} \lambda x.t^\star$,
3. $(\langle xs/y\rangle t)^\star =_{def} \langle xs^\star/y\rangle t^\star$,
4. $([\lambda z.r/x]\langle \underline{x}s/y\rangle t)^\star =_{def} \mathtt{x}(\{\mathtt{x}(\{s^\star/z\}r^\star)/y\}t^\star)$.

Lemma 11. *If $t \Rightarrow t'$ then $t' \Rightarrow t^\star$.*

Proof. By induction on the definition of $t \Rightarrow t'$. □

Lemma 12. *If $t \Rightarrow t_1$ and $t \Rightarrow t_2$ then there is t' such that $t_1 \Rightarrow t'$ and $t_2 \Rightarrow t'$.*

Proof. By Lemma 11. □

Theorem 1. *The reduction relation $\to_\beta$ is confluent.*

Proof. By Lemmas 10 and 12. □

Lemma 13. *Let u, t be Beta-terms.*

1. *If $u \to_\beta u'$ then $\mathtt{x}(\{u/x\}t) \xrightarrow{*}_\beta \mathtt{x}(\{u'/x\}t)$.*
2. *If $t \to_\beta t'$ then $\mathtt{x}(\{u/x\}t) \xrightarrow{*}_\beta \mathtt{x}(\{u/x\}t')$.*

Proof. These are derived from Lemmas 9 and 10. □

5 Confluence of Cut-Elimination Procedures

In this section we complete the proofs of confluence of the cut-elimination procedures. We also establish a conservativity result among the cut-elimination procedures and β-reduction on *Beta*-terms.

Lemma 14

1. $\mathtt{x}(\langle\{\!\{u\}\!\}s/y\rangle t) \equiv \langle\{\!\{\mathtt{x}(u)\}\!\}\mathtt{x}(s)/y\rangle\mathtt{x}(t)$,
2. $\mathtt{x}(\{u/x\}t) \equiv \mathtt{x}(\{\mathtt{x}(u)/x\}\mathtt{x}(t))$.

Proof.

1. By induction on the structure of u.
2. By induction on the structure of t. □

The next two lemmas show that the cut-elimination procedures project onto β-reduction on *Beta*-terms.

Lemma 15. *If* $t \to_{\text{gcut}} t'$ *then* $\mathtt{x}(t) \xrightarrow{*}_{\beta} \mathtt{x}(t')$.

Proof. By induction on the reduction relation $\to_{\text{gcut}}$. □

Lemma 16. *If* $t \to_{\text{cut}} t'$ *then* $\mathtt{x}(t) \xrightarrow{*}_{\beta} \mathtt{x}(t')$.

Proof. If $t \to_{\mathtt{x}} t'$ then $\mathtt{x}(t) \equiv \mathtt{x}(t')$. So it suffices to show that if $t \to_{Beta} t'$ then $\mathtt{x}(t) \xrightarrow{*}_{\beta} \mathtt{x}(t')$. This is proved in a similar way to Lemma 15. □

Now we have a conservativity result among the reductions on *Beta*-terms.

Theorem 2. *For any Beta-terms* t, t'*, the following are equivalent.*

1. $t \xrightarrow{*}_{\text{gcut}} t'$
2. $t \xrightarrow{*}_{\text{cut}} t'$
3. $t \xrightarrow{*}_{\beta} t'$

Proof. By Lemmas 15 and 16, and Proposition 4. □

We are now ready to show that the reduction relations $\to_{\text{gcut}}$ and $\to_{\text{cut}}$ are confluent, using confluence of $\to_{\beta}$ on *Beta*-terms (Theorem 1). The results also hold in the typed case, so that confluence of the cut-elimination procedures follows.

Theorem 3

1. *The reduction relation* $\to_{\text{gcut}}$ *is confluent.*
2. *The reduction relation* $\to_{\text{cut}}$ *is confluent.*

Proof.

1. Suppose that $t \xrightarrow{*}_{\text{gcut}} t_1$ and $t \xrightarrow{*}_{\text{gcut}} t_2$. Then by Lemma 15, $\mathtt{x}(t) \xrightarrow{*}_{\beta} \mathtt{x}(t_i)$ $(i = 1, 2)$, so by confluence of $\to_{\beta}$, there is a *Beta*-term t' such that $\mathtt{x}(t_i) \xrightarrow{*}_{\beta} t'$ $(i = 1, 2)$. Since $\mathtt{x}(t_i) \xrightarrow{*}_{\text{gcut}} t'$ by Theorem 2 and $t_i \xrightarrow{*}_{\text{gcut}} \mathtt{x}(t_i)$ by Lemma 5, we have $t_i \xrightarrow{*}_{\text{gcut}} t'$ $(i = 1, 2)$.
2. Similar, using Lemma 16 instead of Lemma 15. □

6 Conclusion

We have proved confluence of global and local cut-elimination procedures, using proof terms for a standard sequent calculus of intuitionistic logic. For the interpretation method to work, we have introduced β-reduction on *Beta*-terms, and proved its confluence by the method of parallel reduction. Then confluence of the two cut-elimination procedures has been obtained through projections onto the β-reduction. Additionally, we have established a conservativity result among the cut-elimination procedures and the β-reduction. Note that our proofs are also effective in the type-free case allowing non-terminating computations.

The problem on substitution lemmas (cf. the remark before Lemma 6) was also pointed out in [11, page 136] for the case of the classical sequent calculus. In future work, we will investigate the relation between their observations and ours, and develop proofs of confluence for some cut-elimination procedures in the classical sequent calculus.

Acknowledgements. This work was inspired by Stéphane Lengrand's conjectures on confluence of cut-elimination procedures that simulate β-reduction. The work was partially supported by the Japanese Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) 17700003.

References

1. Bloo, R., Geuvers, H.: Explicit substitution: On the edge of strong normalization. Theoretical Computer Science 211, 375–395 (1999)
2. Danos, V., Joinet, J.-B., Schellinx, H.: A new deconstructive logic: Linear logic. The Journal of Symbolic Logic 62, 755–807 (1997)
3. Espírito Santo, J.: Revisiting the correspondence between cut elimination and normalisation. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 600–611. Springer, Heidelberg (2000)
4. Gentzen, G.: Untersuchungen über das logische Schliessen. Mathematische Zeitschrift, 39: pp. 176–210, pp. 405–431, English translation in [9 pp. 68–131] (1935)
5. Girard, J.-Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)
6. Hardin, T.: Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs. Thèse de doctorat, Université de Paris VII (1987)
7. Howard, W.A.: The formulae-as-types notion of construction. In: Seldin, J.P., Hindley, J.R. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism, pp. 479–490. Academic Press, San Diego (1980)
8. Kikuchi, K.: On a local-step cut-elimination procedure for the intuitionistic sequent calculus. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 120–134. Springer, Heidelberg (2006)
9. Szabo, M.E. (ed.): The Collected Papers of Gerhard Gentzen. North-Holland (1969)
10. Takahashi, M.: Parallel reductions in λ-calculus. Information and Computation 118, 120–127 (1995)
11. Urban, C., Bierman, G.M.: Strong normalisation of cut-elimination in classical logic. Fundamenta Informaticae 45, 123–155 (2001)

The Polynomial and Linear Hierarchies in V^0

Leszek Aleksander Kołodziejczyk[1,⋆] and Neil Thapen[2,⋆⋆]

[1] Institute of Mathematics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland
lak@mimuw.edu.pl

[2] Mathematical Institute, Academy of Sciences of the Czech Republic, Žitná 25, CZ-115 67 Praha 1, Czech Republic
thapen@math.cas.cz

Abstract. We show that the bounded arithmetic theory V^0 does not prove that the polynomial time hierarchy collapses to the linear time hierarchy (without parameters). This result follows from a lower bound for bounded depth circuits computing prefix parity, where the circuits are allowed some auxiliary input.

This is a continuation of earlier work by the authors which showed that this collapse is not provable in PV under a cryptographic assumption.

Keywords: prefix parity, linear hierarchy, bounded arithmetic, bounded depth circuits.

1 Introduction

One approach to problems of structural complexity is to look at their behaviour in theories of bounded arithmetic. This allows us to consider how complexity classes behave in models not unreasonably different from the real world, and to study what logical resources are necessary to answer complexity-theoretic questions. The most important problem in this area is whether there is a model of full bounded arithmetic in which the polynomial hierarchy does not collapse to a finite level, see e.g. [6].

In the present paper we deal with the problem of the relation between the linear and polynomial time hierarchies, and look at it in the weak two-sorted theory V^0, which can be thought of as a subtheory of S^1_2. V^0 is strong enough to prove the basic properties of AC^0 circuits, but weak enough that we can use lower bounds on the strength of AC^0 circuits to obtain unconditional independence results. We show the existence of a model of V^0 in which a set in the second level Σ^p_2 of the polynomial time hierarchy is not contained in the parameter free linear time hierarchy. Our result uses a simple model-theoretic construction and the following circuit bound: a bounded depth, polynomial size circuit can compute the prefix parities of only an exponentially small fraction of n-bit inputs X, even

⋆ This work was carried out while the author was visiting the Mathematical Institute of the Academy of Sciences of the Czech Republic in Prague.

⋆⋆ Corresponding author. Supported in part by grant AV0Z10190503 and by the Eduard Čech Center grant LC505.

if it has access to some auxiliary input strings of length $n^{\frac{1}{4}}$ which may depend on X. Here the "prefix parity of X" is the string whose ith bit is the parity of bits $1, \ldots, i$ of X.

In the next section we explain how complexity classes are defined in nonstandard models of arithmetic and describe the theory V^0 and some of its simple properties; in Section 3 we prove our main result, assuming the lower bound; and in Section 4 we prove the lower bound, as a corollary of an old theorem of Ajtai about the fraction of its inputs for which a bounded depth, polynomial size circuit correctly computes the parity bit.

This paper continues work in [5] where we show, under a cryptographic assumption, that some important statements about structural complexity theory are not provable in the bounded arithmetic theories S^1_2 and PV. Our assumption is that there is no probabilistic polynomial time algorithm for factoring. This guarantees the existence of a model of S^1_2 in which the injective weak pigeonhole principle fails for a polynomial time function f (that is, f is an injection from n^2 to n for some n) but in which the surjective weak pigeonhole principle holds for all polynomial time functions (that is, for any n and any poly-time g, g is not a surjection from n to n^2). We use this to construct a model of PV in which the polynomial time hierarchy does not collapse to the linear time hierarchy; a model of S^1_2 in which an NP set is not in the second level of the linear time hierarchy; and a model of S^1_2 in which the polynomial hierarchy does collapse to the linear hierarchy, but does not collapse to any finite level Σ^p_i. Parameters are allowed in the definition of these hierarchies.

Here we use the same basic technique as [5], which is to take a model of some theory and close an initial segment of it (and possibly a few elements more) under a certain set of functions (there PV, here FAC^0). This gives a new model in which formulas whose quantifiers only range over the common initial segment keep their truth values unchanged, but in which some formulas with bigger quantifiers will change their truth values. The present work has the advantage that it does not use any assumptions. It has two main disadvantages, besides the obvious one that V^0 is a weaker theory than PV.

The first is that our result holds for the parameter-free versions of the hierarchies, while it seems that the natural definition of the hierarchies in nonstandard models would allow parameters – we say more about this below. We deal with parameters in [5] essentially by iterating our basic step and using a union-of-chains construction. A similar approach does not appear to be possible here, at least using our circuit lower bound.

The second disadvantage is related to properties of the set which is not provably in the linear time hierarchy. In the present case, this set is rather artificial — for example, it is empty in the standard model, and even in nonstandard models of stronger bounded arithmetic theories. Additionally, it is from the second level Σ^p_2 of the polynomial time hierarchy, whereas the set in [5] is in NP. The definition of that NP set depends on having a function f in a model of PV which defines an injection from the set of numbers of length n^2 (in binary notation) into the set of numbers of length n. By the non-provability of the relativized pigeonhole principle in $I\Delta_0$ we know that there are models of V^0 in which there is a definable injection from some $n+1$ to n, giving an injection from strings of length $n+1$ to strings of length n. But this is too small a difference between domain and range, and in the absence of the ability to iterate functions polynomially many times (available in PV, but not in V^0) there seems to be no way of amplifying it. Of course the existence of a model of V^0 with a definable injection from n^2 to n is equivalent to an old open problem, about the provability of the relativized weak pigeonhole principle in $I\Delta_0$. The existence of a definable injection between strings of these lengths, instead of just the numbers, is an interesting question in its own right, and may be an easier version of this open problem.

2 Definitions

Most of the definitions below are based on [2].

We work in a language $\mathcal{L}_A^2$ of two-sorted arithmetic. We will write variables of the number or "first-order" sort as $i, j, k, \ldots$ and variables of the finite set or "second-order" sort (which we will think of as strings) as $X, Y, Z, \ldots$. The language consists of the function and predicate symbols $\{0, 1, +, \cdot, |\ |, \in, \leq, =\}$. Here $+, \cdot, \leq$ only apply to the number sort. $|X|$ is the least upper bound of the set X, or 0 if X is empty; by abuse of notation we will also use it to mean the length of X when we think of X as a string.

A Σ_0^B formula is a formula in this language in which the only quantifiers are bounded number quantifiers, that is, quantifiers of the form $\forall i < t$ or $\exists i < t$ where t is a number term (not containing i). Here a number term is one taking a value of the number sort; it is allowed to contain subterms of the form $|X|$.

A polynomially bounded string quantifier is of the form $\forall X\, (|X| < t \to \ldots)$ or $\exists X\, (|X| < t \wedge \ldots)$ where t is a number term (not containing the variable X). We will write these as $\forall X < t$ and $\exists X < t$. A linearly bounded string quantifier is defined in the same way, with the important difference that the bounding term t is not allowed to contain multiplication.

For $i \in \mathbb{N}$, the Σ_i^B formulas consist of i alternations of blocks of polynomially bounded string quantifiers, beginning with an existential quantifier, followed by a Σ_0^B formula. The Σ_i^{LIN} or "linear" formulas are defined similarly, but with linearly bounded string quantifiers. Π_i^B and Π_i^{LIN} are defined dually. Σ_∞^B and Σ_∞^{LIN} are the unions of the respective sets of formulas over all $i \in \mathbb{N}$.

It is straightforward to see that for $i \geq 1$ the sets of strings definable in the standard model by Σ_i^B formulas are exactly the sets from the ith level Σ_i^p of the polynomial hierarchy. Similarly the sets of strings definable by Σ_∞^{LIN} formulas are exactly the sets from the linear hierarchy [7, 4] – the linear hierarchy is not defined so robustly, so we do not seem to have the level-by-level correspondence. Hence in a nonstandard model of an arithmetical theory in this language it is natural to identify the polynomial hierarchy with the Σ_∞^B definable sets of strings and the linear hierarchy with the Σ_∞^{LIN} sets of strings.

It seems natural to allow parameters to be used in defining these sets, firstly because this captures the idea of limiting the time bounds of our Turing machines to the standard polynomials, while letting the input and the code of the machine range over the whole model; secondly because if there is a model in which the polynomial hierarchy is contained in the linear hierarchy without parameters, then we must have this containment already in the standard model. However we are not currently able to prove our result for V^0 in the version with parameters.

V^0 is a theory of bounded arithmetic in our two-sorted language $\mathcal{L}^2_A$, based on a theory of Zambella [8]. For a complete introduction to the version we use here see [2]. V^0 consists of a set 2-BASIC of axioms fixing the basic properties of its language and the following comprehension axiom for each Σ^B_0 formula ϕ, possibly with parameters:

$$\exists Z < j\, \forall i < j\, (i \in Z \leftrightarrow \phi(i)).$$

Notice that together with the properties of the $|\ |$ function, this gives induction for Σ^B_0 formulas. In fact, V^0 is conservative over $I\Delta_0$.

Let $\phi(i, \bar{X})$ be any Σ^B_0 formula, with a free number variable i, some free string variables $\bar{X}$, and no other free variables. Let $t(\bar{X})$ be any number-valued term. Then ϕ and t naturally give rise to a function $F_{\phi,t}$: the output of $F_{\phi,t}$ on input $\bar{X}$ is the string of length $t(\bar{X})$ whose bits are given by the values of $\phi(i, \bar{X})$ for $i = 1$ to $t(\bar{X})$. We call the functions defined in this way the uniform FAC^0 functions. They correspond to the string functions defined by uniform families of polynomial size, bounded depth circuits.

Now let $\mathbf{M} = (N, M)$ be a model of V^0, where N is the set of number elements and M the set of string elements. For any $S \subseteq M$, let $T \subseteq M$ be the closure in $\mathbf{M}$ of S under all uniform FAC^0 functions and let U be the set of lengths of strings from T. Then (U, T) is a model of V^0; closure under the uniform FAC^0 functions is exactly what is needed to guarantee that comprehension holds.

3 Main Theorem

We first state our lemma about small bounded depth circuits. The proof is postponed until the next section.

Lemma 1. *Let $k \in \mathbb{N}$. Let (C_n) be a family of polynomial-size, bounded depth circuits where each circuit has as input one string X of length n^2 and k many auxiliary input strings, each of length $\sqrt{n}$, and each circuit has as output a string Y of length n^2.*

Then for all sufficiently large n, for all but a fraction of at most $2^{-\sqrt{n}}$ input strings X, C_n fails to output the prefix parity of X for any choice of auxiliary strings.

Let $\phi(A)$ be the formula "for some X with $|X| = |A|^4$, there is no prefix parity Y of X". This is Σ^B_2, since we can express "Y is the prefix parity of X" in a Σ^B_0 way as

$$|Y| = |X| \wedge Y(1) \equiv X(1) \wedge \forall i < |X|\, (Y(i+1) \equiv Y(i) \oplus X(i+1)),$$

where we use $X(i)$ to mean the ith bit of the string X.

Theorem 2. *There is a model of* V^0 *in which* $\phi(A)$ *is not equivalent to any formula* $\psi(A)$ *in* Σ^{LIN}_∞ *without parameters.*

Proof. It is enough to show that the theory

$$\mathrm{V}^0 + \{\exists A\, \neg(\phi(A) \leftrightarrow \psi(A)) : \psi \in \Sigma^{LIN}_\infty\}$$

is finitely satisfiable. So suppose for a contradiction that we have finitely many linear formulas $\psi_1, \ldots, \psi_m$ and that

$$\mathrm{V}^0 \vdash \bigvee_i \forall A,\, \phi(A) \leftrightarrow \psi_i(A).$$

We define a theory Γ with new constant symbols $U^1, \ldots, U^{m+1}$ and $n_1, \ldots, n_{m+1}$. For each $i = 1, \ldots, m+1$, each $k \in \mathbb{N}$ and each FAC^0 function F, Γ contains the sentence "$|U^i| = n_i^2$ and for all auxiliary strings $Z_1, \ldots, Z_k$, each of length $\sqrt{n_i}$, the output of $F(U^i, \bar{Z})$ is not the prefix parity of U^i". Γ also contains "$n_{i+1} > n_i^4$" for each $i = 1, \ldots, m$.

To see that Γ is finitely satisfiable in $\mathbb{N}$, consider any $k \in \mathbb{N}$ and any finite number of FAC^0 functions. By the bound on any single FAC^0 function given by the lemma, for arbitrarily large n_1 there is a string U^1 of length n_1^2 such that none of our finitely many functions can calculate the prefix parity of U^1, for any choice of auxiliary input. Pick such an n_1 and U^1, then find $n_2 > n_1^4$ and a string U^2 with the same property, and so on.

Let $\mathbf{M}$ be a model of Γ together with the theory of true arithmetic in our language $\mathcal{L}^2_A$. For each i, let $\mathbf{M}_i$ be the model of V^0 given by taking the closure in $\mathbf{M}$ of the set $\{\text{strings of length} \le \sqrt{n_i}\} \cup \{U^i\}$ under all FAC^0 functions in $\mathbf{M}$, as in the previous section.

For each i, by our assumption ϕ must be equivalent in $\mathbf{M}_i$ to some ψ_t. Since we have more models than we have linear formulas ψ, by the pigeonhole principle there must be two models $\mathbf{M}_i$ and $\mathbf{M}_j$, with $i < j$, in both of which ϕ is equivalent to the same ψ_t.

Now let A be the string consisting of $\sqrt{n_i}$ many 1s. $\psi_t(A)$ must have the same truth value in $\mathbf{M}_i$ as in $\mathbf{M}_j$, since it only talks about strings whose lengths are linear in $|A|$ and about numbers polynomial in $|A|$, and these are the same in $\mathbf{M}_i$ and $\mathbf{M}_j$ (since, for $r \in \mathbb{N}$, numbers less than $|A|^r$ can be thought of as r-tuples of numbers less than $|A|$).

However $\phi(A)$ is false in $\mathbf{M}_j$, since $n_j \ge n_i^4$ so $\mathbf{M}_j$ is the same as $\mathbf{M}$ for all strings of length $|A|^4$ and thus contains a prefix parity for every such string. But $\phi(A)$ is true in $\mathbf{M}_i$, since U^i is in $\mathbf{M}_i$ but, by construction, the unique prefix parity (in $\mathbf{M}$) of U^i is not in $\mathbf{M}_i$.

Hence $\phi(A)$ cannot be equivalent to $\psi_t(A)$ in both $\mathbf{M}_i$ and $\mathbf{M}_j$, which gives a contradiction. □

We note that minor changes to the argument show that V^0 does not prove that the polynomial hierarchy collapses to the quadratic time hierarchy, or to any time hierarchy given by polynomials of fixed degree. Also, a similar argument shows that there exists a model in which our formula ϕ is not equivalent to any parameter-free Σ^B_1 formula.

4 The Circuit Lower Bound

It remains to give the proof of Lemma [1], which relies on a result of Ajtai. In the calculations of probabilities below we use the vertical lines $|\delta|$ to mean the absolute value of a real number δ.

Theorem 3 (Ajtai [1]). *Let (C_n) be a polynomial size family of bounded depth circuits, where each C_n has n input bits. Let P_n be the fraction of input strings X of length n for which the output bit of C_n is the parity of X. Then for any $\epsilon > 0$, for all sufficiently large n,*

$$|P_n - \tfrac{1}{2}| < 2^{-n^{1-\epsilon}}.$$

Note that by the nonuniformity of this result the bound n can be chosen so as to depend only on ϵ and the depth d and size exponent r of the circuit family. Otherwise for arbitrarily large n there would exist some circuit D_n of depth d and size n^r with distance from $\frac{1}{2}$ greater than $2^{-n^{1-\epsilon}}$. These circuits would thus define a family (D_n) violating the theorem.

It is also worth noting that our argument appears to need Ajtai's strong bound on the advantage away from $\frac{1}{2}$ here. The bounds that can be obtained from Håstad's method of switching lemmas do not seem to be strong enough. See Chapter 8 of [3].

Now consider a polynomial size family (C_n) of bounded depth circuits, where the nth circuit takes n^2 input bits and has n output bits. We think of the input as a $n \times n$ binary matrix $\bar{X}$ with rows $X_1, \ldots, X_n$. We imagine the circuit as attempting to output a "parity vector", the ith entry of which is the parity of the vector X_i. We will write C_n^i for the subcircuit which, on input $\bar{X}$, calculates bit i of the circuit's output.

We will show that the circuit outputs the correct parity vector with an appropriately small probability.

Lemma 4. *Take any $\epsilon > 0$. Fix n sufficiently large. We omit the subscript n in what follows.*

For $k \leq n$ let P^k be the probability, over input matrices $\bar{X}$, that $C^i(\bar{X}) =$ parity(X_i) for every $i \leq k$. Then

$$|P^k - \tfrac{1}{2^k}| < 2 \cdot 2^{-n^{1-\epsilon}}.$$

Proof. Let d be the depth and r the size exponent of the circuit family (C_n). We take the n given by Theorem 3 with parameters ϵ, $d+4$ and $r+1$; all the circuits in the proof will be of this size or smaller. Let $\delta = 2^{-n^{1-\epsilon}}$.

The proof is by induction. The base case, $k = 1$, follows from Theorem 3 and an averaging argument. Suppose that, for a random matrix $\bar{X}$, $C^1(\bar{X}) =$ parity(X_1) with probability more than $\frac{1}{2} + \delta$. Then there must be some fixed vectors $Z_2, \ldots, Z_n$ such that if we take a random n-bit vector X_1 and give C^1 the matrix with rows $X_1, Z_2, \ldots, Z_n$ as input, then C^1 outputs the correct parity of X_1 with probability more than $\frac{1}{2} + \delta$, which is impossible. The same argument works if the probability over $\bar{X}$ is less than $\frac{1}{2} - \delta$.

Suppose the lemma is true for k. Say that $P^k = \frac{1}{2^k} + \alpha$, where $|\alpha| < 2\delta$. We will calculate P^{k+1}.

First let $\Pr(C^{k+1}(\bar{X}) = \text{parity}(X_{k+1})) = \frac{1}{2} + \beta$. By averaging, $|\beta| < \delta$.

Now consider the following function f, which takes as input a matrix $\bar{X}$ and tries to output the parity of X_{k+1}. If $C^1(\bar{X}), \dots, C^k(\bar{X})$ correctly output the parities of $X_1, \dots, X_k$, then f outputs $C^{k+1}(\bar{X})$. Otherwise f outputs $\neg C^{k+1}(\bar{X})$. Let $\Pr(f(\bar{X}) = \text{parity}(X_{k+1})) = \frac{1}{2} + \gamma$.

We claim that $|\gamma| < \delta$. Otherwise, again by our averaging argument, there are some fixed values of $Z_1, \dots, Z_k, Z_{k+2}, \dots, Z_n$ for the rows other than $k+1$ over which, for a random row X_{k+1}, f correctly calculates parity(X_{k+1}) with too high (or too low) a probability. This allows us to violate Theorem 3 by defining a bounded depth circuit for parity(X_{k+1}) as follows. Take $C^1, \dots, C^n$ and hardwire in $Z_1, \dots, Z_k, Z_{k+2}, \dots, Z_n$ as the appropriate rows of the input. At the bottom of the circuit, check whether $C^1, \dots, C^k$ compute the parities of $Z_1, \dots, Z_k$ correctly (since these strings are fixed, we can hardwire in their parities); if so, output the output of C^{k+1}; otherwise output the inverse of C^{k+1}. This construction adds no more than four levels to depth of the circuit C_n and no more than $2n$ nodes to the size. This completes the proof of our claim, since we chose n large enough to work for circuits of this depth and size.

Now for a random matrix $\bar{X}$, f is correct in precisely two cases:

1. $C^1, \dots, C^k$ are all correct and C^{k+1} is correct. The probability of this is P^{k+1}.
2. $C^1, \dots, C^k$ are not all correct and C^{k+1} is not correct. The probability of this is

$$\begin{aligned} &1 - \Pr(C^1, \dots, C^k \text{ all correct}) - \Pr(C^{k+1} \text{ correct}) + P^{k+1} \\ &\qquad = 1 - (\tfrac{1}{2^k} + \alpha) - (\tfrac{1}{2} + \beta) + P^{k+1}. \end{aligned}$$

Now we can equate our two expressions for $\Pr(f \text{ is correct})$ to get

$$\tfrac{1}{2} + \gamma = P^{k+1} + 1 - (\tfrac{1}{2^k} + \alpha) - (\tfrac{1}{2} + \beta) + P^{k+1}$$

and hence

$$P^{k+1} = \tfrac{1}{2}(\tfrac{1}{2^k} + \alpha + \beta + \gamma).$$

But $|\alpha| < 2\delta$ and $|\beta|, |\gamma| < \delta$. So the advantage of P^{k+1} away from $\frac{1}{2^{k+1}}$ is smaller than 2δ, as required. □

Proof of Lemma 1. First observe that from a small bounded depth circuit computing prefix parities of strings of length n^2, we can easily produce a small bounded depth circuit computing parity vectors of $n \times n$ matrices. So, using Lemma 4, for large n any circuit C_n with fixed auxiliary inputs will successfully calculate the prefix parity for at most a fraction $2^{-n^{\frac{2}{3}}}$ of inputs X; but there are only $2^{k\sqrt{n}}$ possible auxiliary strings. Hence there are no more than a fraction $2^{-n^{\frac{2}{3}}} \cdot 2^{k\sqrt{n}} \leq 2^{-\sqrt{n}}$ of inputs X for which there is at least one auxiliary string which helps to compute the prefix parity of X. □

References

[1] Ajtai, M.: Σ^1_1 formulae on finite structures. Annals of Pure. and Applied Logic 24, 1–48 (1983)
[2] Cook, S., Nguyen, P.: Foundations of proof complexity: Bounded arithmetic and propositional translations, (2006), book in preparation, available online at http://www.cs.toronto.edu/~sacook/.
[3] Håstad, J.T.: Computational limitations for small depth circuits. MIT Press, Cambridge (1987)
[4] Immerman, N.: Languages that capture complexity classes. SIAM Journal on Computing 16, 760–778 (1987)
[5] Kołodziejczyk, L.A., Thapen, N.: The polynomial and linear hierarchies in models where the weak pigeonhole principle fails, preprint (2006)
[6] Krajíček, J.: Bounded arithmetic, propositional logic, and complexity theory. Cambridge University Press, Cambridge (1995)
[7] Lynch, J.F.: Complexity classes and theories of finite models. Mathematical Systems Theory 15(2), 127–144 (1982)
[8] Zambella, D.: Notes on polynomially bounded arithmetic. Journal of Symbolic Logic 61, 942–966 (1996)

The Uniformity Principle for Σ-Definability with Applications to Computable Analysis*

Margarita Korovina[1] and Oleg Kudinov[2]

[1] Institute of Informatics Systems,
pr. Lavrenteva 6, 630090, Novosibirsk, Russia
korovina@brics.dk
http://www.brics.dk/~korovina
[2] Sobolev Institute of Mathematics,
pr. Koptuga 4, 630090, Novosibirsk, Russia
kud@math.nsc.ru

Abstract. In this paper we prove the Uniformity Principle for Σ–definability over the real numbers extended by open predicates. Using this principle we show that if we have a Σ_K-formula, i.e. a formula with quantifier alternations where universal quantifiers are bounded by computable compact sets, then we can eliminate all universal quantifiers obtaining a Σ-formula equivalent to the initial one. We also illustrate how the Uniformity Principle can be employed for reasoning about computability over continuous data in an elegant way.

1 Introduction

This work is the next step in a series of papers [7,8,5,9] using arguments from definability theory to logically characterise computable continuous data. In order to do this we have proposed the notion of majorant-computability and developed logical approach to computability over continuous data. This approach is based on representations of continuous data by suitable structures without the equality test and Σ-definability in extensions of the structures by hereditarily finite sets. One of the main features of the notion of majorant-computability is that on the one side it is independent from concrete representations of the elements of structures on the other side it is flexible, i.e. we can change the language of Σ-formulas to express appropriate computability properties.

In this paper we introduce and study the language of Σ_K-formulas which is an extension of the language of Σ-formulas. This language simplifies reasoning about computability of higher type continuous data, and admits elimination of universal quantifiers bounded by computable compact sets. In order to show these properties we prove the Uniformity Principle for Σ-definability over the real numbers extended by open sets. We illustrate how the language of Σ_K-formulas and the Uniformity Principle can be employed for reasoning about computability over continuous data.

* This research was partially supported by Grant Scientific School-4413.2006.1, RFBR-DFG Project GZ: 436 RUS 113/850/01:06-01-04002 and RFBR 05-01-00819a.

The structure of this paper is as follows. In Section 2 we recall basic notions and introduce the language of Σ_K-formulas. In Section 3 we prove the Uniformity Principle for Σ-definability over the real numbers extended by open sets and show that the language of Σ_K-formulas admits elimination of existential and universal quantifiers bounded by computable compact sets. Section 4 illustrates how the language of Σ_K-formulas can be used to prove computability of continuous data.

2 Basic Definitions and Notions

In this paper we consider the ordered structure of the real numbers in *finite predicate languages without equality*, $\langle \mathbb{R}, \sigma_P, < \rangle = \langle \mathbb{R}, \sigma_0 \rangle$, where σ_P satisfies the following assumption.

Assumption 1. *The set σ_P is a finite set of open predicates i.e. interpreted over the reals as open sets.*

We extend the real numbers by the set of hereditarily finite sets $\mathrm{HF}(\mathbb{R})$ which is rich enough for information to be coded and stored. We construct the set of hereditarily finite sets, $\mathrm{HF}(\mathbb{R})$ over the reals, as follows:

1. $\mathrm{HF}_0(\mathbb{R}) \rightleftharpoons \mathbb{R}$,
2. $\mathrm{HF}_{\mathrm{n}+1}(\mathbb{R}) \rightleftharpoons \mathcal{P}_\omega(\mathrm{HF}_{\mathrm{n}}(\mathbb{R})) \cup \mathrm{HF}_{\mathrm{n}}(\mathbb{R})$, where $n \in \omega$ and for every set B, $\mathcal{P}_\omega(B)$ is the set of all finite subsets of B.
3. $\mathrm{HF}(\mathbb{R}) = \bigcup_{m \in \omega} \mathrm{HF}_m(\mathbb{R})$.

We define $\mathbf{HF}(\mathbb{R})$ as the following model: $\mathbf{HF}(\mathbb{R}) \rightleftharpoons \langle \mathrm{HF}(\mathbb{R}), R, \sigma_0, \emptyset, \in \rangle \rightleftharpoons \langle \mathrm{HF}(\mathbb{R}), \sigma \rangle$, where the constant $\emptyset$ stands for the empty set and the binary predicate symbol $\in$ has the set-theoretic interpretation. We also add a predicate symbol R for elements of $\mathbb{R}$. For our convenience, we use variables subject to the following conventions:
$r, x, y, z, a, b, c, \ldots\ldots$ range over $\mathbb{R}^n$, $n \in \mathbb{N}$,
$\mathbf{r}, \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots\ldots$ range over $\mathrm{HF}(\mathbb{R})$,
$\bar{r}, \bar{x}, \bar{y}, \bar{\mathbf{z}}, \bar{\mathbf{a}}, \bar{\mathbf{b}}, \ldots\ldots$ denote sequences.
We use the same letters as for variables to denote elements from the corresponding structures.

The set of *Δ_0-formulas* is the closure of the set of atomic formulas under $\wedge, \vee, \neg$, bounded quantifiers $(\exists \mathbf{x} \in \mathbf{y})$ and $(\forall \mathbf{x} \in \mathbf{y})$, where $(\exists \mathbf{x} \in \mathbf{y})\ \Psi$ denotes $\exists \mathbf{x}(\mathbf{x} \in \mathbf{y} \wedge \Psi)$, $(\forall \mathbf{x} \in \mathbf{y})\ \Psi$ denotes $\forall \mathbf{x}(\mathbf{x} \in \mathbf{y} \to \Psi)$ and $\mathbf{y}$ ranges over sets.

The set of *Σ-formulas* is the closure of the set of Δ_0-formulas under $\wedge,\vee$, $(\exists \mathbf{x} \in \mathbf{y})$, $(\forall \mathbf{x} \in \mathbf{y})$ and $\exists$, where $\mathbf{y}$ ranges over sets.

The set of *Σ_K-formulas* is the closure of the set of Σ-formulas under $\wedge,\vee$, $\exists$, $\exists x \in K$, and $\forall x \in K$, where K is a computable compact subset of $\mathbb{R}^n$.

Assumption 2. *We assume that the predicate $<$ and all predicates from σ_P occur positively in all Σ- and Σ_K-formulas.*

We define *Π-formulas* as negations of Σ-formulas.

Definition 1. *1. A relation $B \subseteq \mathrm{HF}(\mathbb{R})^n$ is* **Σ-definable**, *if there exists a Σ-formula Φ such that $\mathbf{x} \in B \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \Phi(\mathbf{x})$.*

In a similar way, we define the notions of *Σ_K-definable and Π-definable sets.* The following theorem reveals algorithmic properties of Σ-formulas over $\mathbf{HF}(\mathbb{R})$.

Theorem 1 (Semantic Characterisation of Σ-definability)
A set $B \subseteq \mathbb{R}^n$ is Σ-definable if and only if there exists an effective sequence of existential formulas in the language σ_0, $\{\Phi_s(x)\}_{s\in\omega}$, such that

$$x \in B \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \bigvee_{s\in\omega} \Phi_s(x).$$

The proof of this theorem is based on Gandy's theorem for abstract structures without equality [6] and the technique developed in [5]. It is worth noting that both of the directions of this characterisation are important. The right direction gives us an effective procedure which generates existential formulas approximating Σ-relations. The converse direction provides tools for descriptions of the results of effective infinite approximating processes by finite formulas.

3 Uniformity Principle for Σ-Definability

Now we assume $\sigma_0 = \{\mathcal{M}^*_E, \mathcal{M}^*_H, \mathcal{P}^+_E, \mathcal{P}^+_H, <\}$, $\sigma = \sigma_0 \cup \{\emptyset, \in\}$, where $\mathcal{M}^*_E, \mathcal{M}^*_H$ are interpreted as an open epigraph and an open hypograph of multiplication respectively, and $\mathcal{P}^+_E, \mathcal{P}^+_H$ are interpreted as an open epigraph and an open hypograph of addition respectively. In sequel we will use the following notations: $x \cdot y < z$ for $\mathcal{M}^*_E(x,y,z)$, $x \cdot y > z$ for $\mathcal{M}^*_H(x,y,z)$, $x + y < z$ for $\mathcal{P}^+_E(x,y,z)$, and $x + y > z$ for $\mathcal{P}^+_H(x,y,z)$. It is worth noting that in Σ-formulas we can also use the expressions $x > 0$, $y < 1$ (and similar) as notations of the formulas $\exists y\, (x > y \cdot y)$, $\exists z > 0\, (x \cdot z < z)$ respectively. In sequel we assume that $|| \cdot ||$ is the standard norm on $\mathbb{R}^n$, $[a,b]$ denotes a closed interval, and $\bar{B}(x,\epsilon)$ denotes a closed ball with a center x and a radius ϵ. The following property of Σ-definable sets over the reals in the language σ_0 is a straightforward corollary of Theorem 1.

Corollary 1. *A set $B \subseteq \mathbb{R}^n$ is Σ-definable if and only if B is c.e. open.*

One of the main goals of this section is to prove that the language of Σ_K-formulas admits elimination of universal quantifiers bounded by computable compact sets. In other words, we are going to show that if we have a formula with quantifier alternations where universal quantifiers are bounded by computable compact sets then we can eliminate all universal quantifiers obtaining a Σ-formula equivalent to the initial one. In order to do that, first we prove the Uniformity Principle for Σ-definability. We extend the given language σ by new predicate symbols P and P'_λ with the following meaning

- P defines an open subset of $\mathbb{R}^n$;
- $P'_\lambda(a,b,x_2,\dots,x_n) \leftrightarrow \forall x_1 \in [a,b] P(x_{1\lambda},\dots,x_{n\lambda})$, where $\lambda : \{1,\dots,n\} \to \{1,\dots,n\}$.

The following lemma shows that both languages $\sigma \cup \{P\}$ and $\sigma \cup \{P'_\lambda | \lambda : \{1, \ldots, n\} \to \{1, \ldots, n\}\}$ are subject to Assumption 1.

Lemma 1. *If P defines an open subset of $\mathbb{R}^n$ then P'_λ defines an open subset of $\mathbb{R}^m$, where m depends on λ.*

Proof. We give the main idea of the proof for $\lambda = id_{\{1,\ldots,n\}}$. It is sufficient to show that for every closed interval $[c, d]$ the set

$$B_{c,d} = \{(a, b, x_2, \ldots, x_n) \in [c, d]^{n+1} | [a, b] = \emptyset \vee [a, b] \subset (c, d) \wedge P'_\lambda(a, b, x_2, \ldots, x_n)\}$$

is open. Let us consider $A_{c,d}$, the complement of $B_{c,d}$ in $[c, d]$, which is defined as follows.

$$A_{c,d} = \{(a, b, x_2, \ldots, x_n) \in [c, d]^{n+1} | [a, b] \subseteq [c, d] \wedge \\ (a = c \vee b = d \vee \exists x_1 \in [a, b] \neg P(x_1, \ldots, x_n))\}.$$

Since $A_{c,d}$ is a projection of the compact set

$$\{(a, b, x_2, \ldots, x_n) \in [c, d]^{n+1} | [a, b] \subseteq [c, d] \wedge (a = c \vee b = d \vee \neg P(x_1, \ldots, x_n))\},$$

$A_{c,d}$ is compact. So P'_λ is open as the union of all open sets $B_{c,d}$ where $c \in \mathbb{R}$ and $d \in \mathbb{R}$.

Let us consider particularly interesting corollaries of Lemma 1. If $f \in C(\mathbb{R})$, then the sets $P^-_f = \{(x, c) | f(x) > c\}$ and $P^+_f = \{(x, c) | f(x) < c\}$ are open. Choosing λ to be identical on $\{1, 2\}$ we get the following corollary.

Corollary 2. *For every $f \in C(\mathbb{R})$, the sets*

$$E_f(x_1, x_2, z) \rightleftharpoons f|_{[x_1,x_2]} < z \text{ and } H_f(x_1, x_2, z) \rightleftharpoons f|_{[x_1,x_2]} > z \text{ are open .}$$

If $f \in C([0, 1])$, then applying Corollary 1 to the function

$$g(x) = \begin{cases} f(0), \text{ if } x < 0 \\ f(x), \text{ if } x \in [0, 1] \\ f(1), \text{ if } x > 1 \end{cases}$$

we get straightforwardly the following.

Corollary 3. *For $f \in C([0, 1])$, the sets $E_f(x_1, x_2, z) \rightleftharpoons f|_{[x_1,x_2]\cap[0,1]} < z$ and $H_f(x_1, x_2, z) \rightleftharpoons f|_{[x_1,x_2]\cap[0,1]} > z$ are open.*

Theorem 2 (Uniformity principle). *For every Σ-formula φ in the language $\sigma \cup \{P\}$ there exists Σ-formula ψ in the language $\sigma \cup \{P'_\lambda | \lambda : \{1, \ldots, n\} \to \{1, \ldots, n\}\}$ such that*

$$\mathbf{HF}(\mathbb{R}) \models \forall x \in [a, b] \varphi(x, x_2, \ldots, x_n) \text{ iff } \mathbf{HF}(\mathbb{R}) \models \psi(a, b, x_2, \ldots, x_n),$$

where free variables range over $\mathbb{R}$.

Proof. First we consider the case of $\exists$-formulas in the language $\sigma_0 \cup \{P\}$. Using induction on the structure of a $\exists$-formula φ, we show how to obtain a required formula ψ. Then, based on Theorem 1 we construct a required formula ψ for an arbitrary Σ-formula in the language $\sigma \cup \{P\}$.

Atomic case.
1. If $\varphi(x_1,\dots,x_n) \rightleftharpoons P(x_{1\lambda},\dots,x_{n\lambda})$, then $\psi \rightleftharpoons P'_\lambda$.
2. If φ does not contain the predicate symbol P, we have a finite number of subcases. We consider nontrivial ones.
2.1 If $\varphi(x,z) \rightleftharpoons x\cdot x > z$ then

$$\psi(a,b,z) \rightleftharpoons z<0 \vee a>b \vee (a>0\wedge b>0 \wedge a\cdot a>z) \vee (a<0 \wedge b<0 \wedge b\cdot b>z)\,.$$

2.2 If $\varphi(x,z) \rightleftharpoons x\cdot x<z$ then $\psi(a,b,z) \rightleftharpoons a>b \vee (a\cdot a<z \wedge b\cdot b<z)\,.$
2.3 If $\varphi(x,y) \rightleftharpoons x\cdot y > x$ then

$$\psi(a,b,z) \rightleftharpoons a>b \vee (a>0 \wedge b>0 \wedge y>1) \vee (a<0 \wedge b<0 \wedge y<1)\,.$$

2.4 If $\varphi(x) \rightleftharpoons x\cdot x > x$ then $\psi(a,b) \rightleftharpoons a>b \vee (a>1 \wedge b>1) \vee (a<0 \wedge b<0)\,.$
2.5 If $y\cdot z<x$ then $\psi(a,b,y,z) \rightleftharpoons y\cdot z<a \vee b<a$. Other atomic subcases can be considered by analogy.

Conjunction.
If $\varphi \rightleftharpoons \varphi_1 \wedge \varphi_2$ and ψ_1, ψ_2 are already constructed for $\varphi_1,\ \varphi_2$ then $\psi \rightleftharpoons \psi_1 \wedge \psi_1$.

Disjunction.
Suppose $\varphi \rightleftharpoons \varphi_1 \vee \varphi_2$ and ψ_1, ψ_2 are already constructed. Since $[a,b]$ is compact, validity of the formula $\forall x \in [a,b]\,(\varphi_1 \vee \varphi_2)$ is equivalent to existence of a finite family of open intervals $\{(\alpha_i,\beta_i)\}_{i=1,\dots,r+s}$ such that $[a,b] \subseteq \bigcup_{i=1}^r (\alpha_i,\beta_i)$, for $i=1,\dots,r$ $\mathbb{R}\models\varphi_1$ and for $i=r+1,\dots,s$ $\mathbb{R}\models\varphi_2$. Since φ_1 and φ_2 define open sets, this is equivalent to existence of a finite family of closed intervals $\{[\alpha'_i,\beta'_i]\}_{i=1,\dots,r+s}$ such that $[a,b] \subseteq \bigcup_{i=1}^r [\alpha'_i,\beta'_i]$, for $i=1,\dots,r$ $\mathbb{R}\models\varphi_1$ and for $i=r+1,\dots,s$ $\mathbb{R}\models\varphi_2$. It is represented by the following formula.

$$\bigvee_{r\in\omega}\bigvee_{r\in\omega} \exists\alpha'_1\dots\exists\alpha'_{s+1}\exists\beta'_1\dots\exists\beta'_{s+1}\left(\bigwedge_{i=1}^{r}\forall x\in[\alpha'_i,\beta'_i]\varphi_1 \wedge \bigwedge_{j=r+1}^{s}\forall x\in[\alpha'_j,\beta'_j]\varphi_2\right).$$

By induction hypothesis and Theorem 1, this formula is equivalent to a Σ-formula ψ in the language $\sigma \cup \{P'_\lambda | \lambda : \{1,\dots,n\} \to \{1,\dots,n\}\}$.

Existential case.
Suppose $\varphi \rightleftharpoons \exists z\varphi_1(z,x_1,\dots,x_n)$. As $[a,b]$ is compact and

$$\{\{x_1 | \mathbb{R}\models\varphi_1(z,x_1,\dots,x_n)\}\}_{z\in\mathbb{R}} = \{V_z\}_{z\in\mathbb{R}}$$

is its cover, there exists a finite set $J=\{z_1,\dots,z_s\}\subset\mathbb{R}$ such that $[a,b]\subseteq \bigcup_{z\in J} V_z$. So, validity of the formula $\forall x_1\in[a,b]\exists z\varphi_1(z,x_1,\dots,x_n)$ is equivalent to existence of the finite set $J=\{z_1,\dots,z_s\}$ such that

$$\mathbb{R}\models\forall x_1\in[a,b]\exists z\varphi_1(z,x_1,\dots,x_n) \leftrightarrow \mathbb{R}\models\forall x_1\in[a,b]\varphi^s(z_1,\dots,z_s,x_1,\dots,x_n),$$

where $\varphi^s(z_1,\dots,z_s,x_1,\dots,x_n) \rightleftharpoons \varphi_1(z_1,x_1,\dots,x_n) \vee \dots \vee \varphi_1(z_s,x_1,\dots,x_n)$. By induction hypotheses, for every $J=\{z_1,\dots,z_s\}$ there exists a Σ-formula $\psi^s(z_1,\dots,z_s,a,b,x_2,\dots,x_n)$ in the language $\sigma \cup \{P'_\lambda | \lambda : \{1,\dots,n\} \to \{1,\dots,n\}\}$ which is equivalent to $\forall x_1 \in [a,b]\varphi^s(z_1,\dots,z_s,x_1,\dots,x_n)$. Finally,

$$\begin{aligned}&\mathbb{R} \models \forall x_1 \in [a,b] \exists z \varphi_1(z,x_1,\dots,x_n) \leftrightarrow\\ &\mathbf{HF}(\mathbb{R}) \models \bigvee\nolimits_{s\in\omega} \exists z_1 \dots \exists z_s \left(\psi^s(z_1,\dots,z_s,a,b,x_2,\dots,x_n)\right).\end{aligned}$$

A required Σ-formula ψ can be constructed using Theorem 1.

Now we are ready to construct a required formula ψ for a Σ-formula. Suppose φ is a Σ-formula. By Lemma 1 and Theorem 1, there exists an effective sequence of existential formulas $\{\varphi_i\}_{i\in\omega}$ such that $\mathbf{HF}(\mathbb{R}) \models \varphi \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \bigvee_{i\in\omega} \varphi_i$. As $[a,b]$ is compact and $\{\{x_1 | \mathbb{R} \models \varphi_i(x_1,\dots,x_n)\}\}_{i\in\omega} = \{U_i\}_{i\in\omega}$ is its cover, there exist $k \in \omega$ and a finite family $\{U_i\}_{i\le k}$ such that $[a,b] \subseteq \bigcup_{i\le k} U_i$. So,

$$\begin{aligned}&\mathbb{R} \models \forall x_1 \in [a,b]\varphi(x_1,\dots,x_n) \leftrightarrow\\ &\mathbf{HF}(\mathbb{R}) \models \bigvee\nolimits_{k\in\omega} \forall x_1 \in [a,b] \bigvee\nolimits_{i\le k} \varphi_i(x_1,\dots,x_n)\end{aligned}$$

By induction hypotheses, for every $k \in \omega$ there exits $\psi_k(a,b,x_2,\dots,x_n)$ in the language $\sigma \cup \{P'_\lambda | \lambda : \{1,\dots,n\} \to \{1,\dots,n\}\}$ which is equivalent to $\forall x_1 \in [a,b] \bigvee_{i\le k} \varphi_i(x_1,\dots,x_n)$. A required Σ-formula ψ can be constructed using Theorem 1.

It is worth noting that the Uniformity Principle holds for any finite extension of σ by open predicates.

Corollary 4. *For every Σ-formula φ in the language σ there exists a Σ-formula ψ in the language σ such that*

$$\mathbf{HF}(\mathbb{R}) \models \forall x \in [a,b]\varphi(x,\bar{y}) \textit{ iff } \mathbf{HF}(\mathbb{R}) \models \psi(a,b,\bar{y}),$$

where free variables range over $\mathbb{R}$.

The following corollary shows that we can extend the given language in a certain way without enlarging the set of Σ-definable sets.

Corollary 5. *For every Σ-formula $\varphi(\bar{y})$ in the language σ and polynomials $p_1(\bar{x}),\dots,p_n(\bar{x})$ with rational coefficients there exists a Σ-formula ψ in the language σ such that $\mathbf{HF}(\mathbb{R}) \models \varphi(p_1(\bar{x}),\dots,p_n(\bar{x}),\bar{z})$ iff $\mathbf{HF}(\mathbb{R}) \models \psi(\bar{x},\bar{z})$.*

Proof. Let $\varphi(\bar{y})$ be a Σ-formula and $p_1(\bar{x}),\dots,p_n(\bar{x})$ be polynomials with rational coefficients. It is easy to note that

$$\begin{aligned}&\mathbf{HF}(\mathbb{R}) \models \varphi(p_1(\bar{x}),\dots,p_n(\bar{x}),\bar{z}) \text{ iff } \mathbf{HF}(\mathbb{R}) \models \exists a_1 \dots \exists a_n \exists b_1 \dots \exists b_n\\ &\forall y_1 \in [a_1,b_1] \dots \forall y_n \in [a_n,b_n] \bigwedge\nolimits_{1\le i\le n} (p_i(\bar{x}) < b_i \wedge p_i(\bar{x}) > a_i) \wedge \varphi(\bar{y},\bar{z}).\end{aligned}$$

Since the formulas $p_i(\bar{x}) < z$ and $p_j(\bar{x}) > z$ are equivalent to Σ-formulas with positive occurrences of the basic predicates $\mathcal{M}_E^*$, $\mathcal{M}_H^*$, $\mathcal{P}_E^+$ and $\mathcal{P}_H^+$, we can construct a required formula ψ using Corollary 4.

Corollary 6. *Suppose B is Π-definable and $B \subseteq [-q,q]^n$ for some rational q. For every Σ-formula φ in the language σ there exists a Σ-formula ψ in the language σ such that* $\mathbf{HF}(\mathbb{R}) \models \forall x \in B\varphi(x,\bar{y})$ *iff* $\mathbf{HF}(\mathbb{R}) \models \psi(\bar{y})$, *where free variables range over* $\mathbb{R}$.

Proof. Suppose $B \subseteq [-q,q]^n$ is definable by a Π-formula η. It is easy to see that $\forall x \in B\varphi(x,\bar{y})$ is equivalent to the formula

$$\forall x \in [-q,q]^n \left(\neg\eta(x) \vee \varphi(x,\bar{y})\right). \tag{1}$$

By Corollary 4 and Corollary 5, the formula (1) is equivalent to a Σ-formula.

Corollary 7. *Suppose K is a co-semicomputable compact set. For every Σ-formula φ in the language σ there exists a Σ-formula ψ in the language σ such that* $\mathbf{HF}(\mathbb{R}) \models \forall x \in K\varphi(x,\bar{y})$ *iff* $\mathbf{HF}(\mathbb{R}) \models \psi(\bar{y})$, *where free variables range over* $\mathbb{R}$.

Proof. It is easy to see that $\forall x \in K\varphi(x,\bar{y})$ is equivalent to the formula

$$\forall x \in [-q,q]^n \left(x \notin K \vee \varphi(x,\bar{y})\right) \tag{2}$$

for some rational q which can be find effectively by K. By properties of co-semicomputable closed sets, the distance function d_K is lower semicomputable [1], and, as a corollary, $\{x | x \notin K\} = \{x | d_K(x) > 0\}$ is Σ-definable. By Corollary 6, the formula (2) is equivalent to a Σ-formula.

Corollary 8. *Suppose K is a semicomputable compact set. For every Σ-formula φ in the language σ there exists a Σ-formula ψ in the language σ such that*

$$\mathbf{HF}(\mathbb{R}) \models \exists x \in K\varphi(x,\bar{y}) \textit{ iff } \mathbf{HF}(\mathbb{R}) \models \psi(\bar{y}),$$

where free variables range over $\mathbb{R}$.

Proof. Let us note that $\exists x \in K\varphi(x,\bar{y})$ is equivalent to the formula

$$\exists x' \exists \epsilon > 0 \left(\varphi(x',\bar{y}) \wedge d_K(x') < \epsilon \wedge \forall z \in \bar{B}(x',\epsilon)\varphi(z,\bar{y})\right). \tag{3}$$

By properties of semicomputable closed sets, the distance function d_K is upper semicomputable [1], and, as a corollary, the set $\{(x',\epsilon) | d_K(x') < \epsilon\}$ is Σ-definable. By the Uniformity Principle, the formula (3) is equivalent to a Σ-formula.

Theorem 3. *For every Σ_K-formula $\varphi(x)$ in the language σ there exists Σ-formula $\psi(x)$ such that* $\mathbf{HF}(\mathbb{R}) \models \varphi(x) \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \psi(x)$.

4 The Uniformity Principle and Computable Analysis

In this section we illustrate how we can prove computability of continuous data using the language of Σ_K-formulas and the Uniformity Principle for Σ-definability.

Let $f \in C[0,1]$. We extend the language σ by two predicates $Q(x_1,x_2,z) \rightleftharpoons f|_{[x_1,x_2]} < z$ and $P(x_1,x_2,z) \rightleftharpoons f|_{[x_1,x_2]} > z$.

Proposition 1. *For every $\lambda : \{1,2,3\} \to \{1,2,3\}$ there exist Σ-formulas ψ^- and ψ^+ in the language $\sigma \cup \{P,Q\}$ which do not depend on the choice of f and*

$$\mathbf{HF}(\mathbb{R}) \models P'_\lambda(a,b,x_2,x_3) \leftrightarrow \psi^-(P,a,b,x_2,x_3) \text{ and}$$
$$\mathbf{HF}(\mathbb{R}) \models Q'_\lambda(a,b,x_2,x_3) \leftrightarrow \psi^+(Q,a,b,x_2,x_3).$$

Proof. We show how to construct the required formulas ψ^- for some λ. If $\lambda = id_{\{1,2,3\}}$ then $\psi^-(a,b,x_2,x_3) \rightleftharpoons b < a \vee P(a,y,z)$. If $\lambda = \{<1,1>,<2,1>,<3,3>\}$ then $\psi^-(a,b,x_2,x_3) \rightleftharpoons P(a,b,z)$. In the nontrivial case, where $\lambda = \{<1,1>,<2,2>,<3,1>\}$, we have

$$\begin{aligned}&\mathbf{HF}(\mathbb{R}) \models \forall x_1 \in [a,b] P(x_1,x_2,x_1) \leftrightarrow \\ &\mathbf{HF}(\mathbb{R}) \models x_2 < a \vee b < a \vee (P(a,x_2,a) \wedge \theta(a,b,x_2)),\end{aligned}$$

where

$$\begin{aligned}\theta(a,b,x_2) \rightleftharpoons &\Big(x_2 < b \wedge \Big(P(a,x_2,x_2) \vee \textstyle\bigvee_{m\in\omega} \exists t_1 \ldots \exists t_m \,(a = t_0 < \ldots \\ &\cdots < t_m < t_{m+1} = x_2 \wedge \textstyle\bigwedge_{i\le m} P(t_i,t_{i+1},t_{i+1})\big)\Big)\Big) \vee \\ &\Big(P(b,x_2,b) \wedge \Big(P(a,b,b) \vee \textstyle\bigvee_{m\in\omega} \exists t_1 \ldots \exists t_m \,(a = t_0 < \ldots \\ &\cdots < t_m < t_{m+1} = b \wedge \textstyle\bigwedge_{i\le m} P(t_i,t_{i+1},t_{i+1})\big)\Big)\Big).\end{aligned}$$

Using this equivalence and Theorem 1 we can effectively construct ψ^-.

In [9] we have shown that a functional $F : C[0,1]^n \to \mathbb{R}$ is majorant-computable iff it is computable in the sense of computable analysis [10]. Now we are going to generalise this result to functionals $F : C[0,1]^n \times \mathbb{R}^m \to \mathbb{R}$. For the definition and properties of majorant-computability we refer to [9].

Theorem 4. *For every functional $F : C[0,1]^n \times \mathbb{R}^m \to \mathbb{R}$ the following assertions are equivalent*:

1. *The functional F is majorant-computable.*
2. *The functional F is computable.*

Proof. Without loss of generality let us consider the case $n = m = 1$. For simplicity of notation, we will give the construction only for that case, since the main ideas are already contained here. Let $F : C[0,1] \times \mathbb{R} \to \mathbb{R}$ be a majorant-computable functional. For $f \in C[0,1]$ we denote $E_f(x_1,x_2,z) \rightleftharpoons f|_{[x_1,x_2]\cap[0,1]} < z$ and $H_f(x_1,x_2,z) \rightleftharpoons f|_{[x_1,x_2]\cap[0,1]} > z$. Let us define $G : C[0,1]^2 \to \mathbb{R}$ by the rule $G(f,g) = F(f,g(\frac{1}{2}))$. We show that G is also majorant-computable. It is easy to see that

$$\begin{aligned}&G(f,g) < y \leftrightarrow \\ &\mathbf{HF}(\mathbb{R}) \models \exists x_1 \exists x_2 \left(x_1 < x_2 \wedge \forall x \in [x_1,x_2] F(f,x) < y \wedge x_1 < g(\tfrac{1}{2}) < x_2\right), \\ &G(f,g) > y \leftrightarrow \\ &\mathbf{HF}(\mathbb{R}) \models \exists x_1 \exists x_2 \left(x_1 < x_2 \wedge \forall x \in [x_1,x_2] F(f,x) > y \wedge x_1 < g(\tfrac{1}{2}) < x_2\right).\end{aligned}$$

By the the Uniformity Principle and Theorem 1 [9], the formulas
$\forall x \in [x_1,x_2] F(f,x) < y$, $\forall x \in [x_1,x_2] F(f,x) > y$ are equivalent to $\forall x \in$

$[x_1, x_2]\varphi^-(E_f, H_f, x, y)$ and $\forall x \in [x_1, x_2]\varphi^+(E_f, H_f, x, y)$ for some Σ-formulas φ^-, φ^+, and, by Proposition 1, to Σ-formulas ψ^-, ψ^+.

Since

$$x_1 < g(\tfrac{1}{2}) < x_2 \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \exists u \exists v \left(u < \tfrac{1}{2} < v \wedge E_g(u, v, x_1) \wedge H_g(u, v, x_1)\right),$$

by Corollary 5, the formula $x_1 < g(\frac{1}{2}) < x_2$ is equivalent to a Σ-formula in the language $\sigma \cup \{E_g, H_g\}$. As we can see the relations $G(f, g) < y$, $G(f, g) > y$ can be represented by Σ-formulas in the language $\sigma \cup \{E_f, H_f, E_g, H_g\}$. So, G is majorant-computable. In [9] we have shown that a functional $H : C[0, 1]^n \to \mathbb{R}$ is computable iff it is majorant-computable. Hence, G is computable. Since $F(f, x) = G(f, \lambda z.x)$, F is computable as composition of computable functions.

If F is computable, then G is also computable. By Theorem 3 [9], there exist two Σ-formulas φ^- and φ^+ such that

$$\begin{aligned} G(f, g) < y &\leftrightarrow \mathbf{HF}(\mathbb{R}) \models \varphi^+(E_f, H_f, E_g, H_g, y),\\ G(f, g) > y &\leftrightarrow \mathbf{HF}(\mathbb{R}) \models \varphi^-(E_f, H_f, E_g, H_g, y) \end{aligned}$$

If we substitute E_g by $U \rightleftharpoons [0, 1] \times (x, +\infty)$ and H_g by $U \rightleftharpoons [0, 1] \times (-\infty, x)$ then we get

$$\begin{aligned} F(f, x) < y &\leftrightarrow \mathbf{HF}(\mathbb{R}) \models \varphi^+(E_f, H_f, U, V, y),\\ F(f, x) > y &\leftrightarrow \mathbf{HF}(\mathbb{R}) \models \varphi^-(E_f, H_f, U, V, y) \end{aligned}$$

By Theorem 1 [9], F is majorant-computable.

Now we show how the Uniformity Principle can be used to investigate computability of the borders of regular compact sets. In [4] it has been proven that the border operator defined on the closed sets over the reals is Σ^0_2-complete in Borel hierarchy. In contrast the following theorem shows that in special cases it is possible to prove computability of borders.

Theorem 5. *Suppose $K \subset \mathbb{R}^n$ is a computable regular compact set, D is its interior, Γ is its border, and $d_K : \mathbb{R}^n \to \mathbb{R}$ is its distance function. Then the following assertions are equivalent:*

1. *Γ is $\Pi-$definable.*
2. *D is $\Sigma-$definable.*
3. *Γ is computable.*

Proof. We give only main ideas of the proof.
$1 \to 2$. Suppose that Γ is Π-definable. It is clear that the formula $d_\Gamma(x) > d_K(x)$ defines D. By properties of co-semicomputable closed sets, D is Σ-definable.
$2 \to 3$. In order to show that Γ is computable it is sufficient to prove that the epigraph and the hypograph of its distance function d_Γ are Σ-definable. Indeed,

$$\begin{aligned} &d_\Gamma(x) < \epsilon \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \exists y \in D \exists z \notin K \left(||x - y|| < \epsilon \wedge ||x - z|| < \epsilon\right) \text{ and}\\ &d_\Gamma(x) > \epsilon \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \bar{B}(x, \epsilon) \subset D \vee \bar{B}(x, \epsilon) \subset \mathbb{R}^n \setminus K. \end{aligned}$$

So, Γ is computable.
$3 \to 1$ Since Γ is computable, the distance function d_Γ is upper and lower semicomputable. So, $\Gamma = \{x | d_\Gamma(x) = 0\}$ is Π-definable.

Theorem 6. *Suppose $K \subset \mathbb{R}^n$ is a computable regular compact set, Γ is its border, and every component of Γ is a smooth variety of codimension 1. Then Γ is computable.*

Proof. It is sufficient to show that Γ is Π-definable. Since Γ is smooth variety of codimension 1, for any $z \in \Gamma$ the following statement holds:

$$\exists y \in D \exists x \notin K (x - z = z - y \wedge B(y, ||y - z||) \subset D \wedge B(x, ||x - z||) \subset \mathbb{R}^n \setminus K).$$

So $\psi(z) \rightleftharpoons z \in K \wedge \exists y \in K \forall t > 0\,(t \leq 1 \vee d_K(z - t(y - z)) \geq t||y - z||)$ defines Γ. Since K is co-semicomputable, K is Π-definable. So, Γ is Π-definable by Theorem 3.

References

1. Brattka, V., Weihrauch, K.: Computability on subsets of euclidean space I: Closed and compact sets. TCS 219, 65–93 (1999)
2. Barwise, J.: Admissible sets and Structures. Springer, Berlin (1975)
3. Ershov, Y.L.: Definability and computability. Plenum, New-York (1996)
4. Gherardi, G.: Some results in computable analysis and effective Borel measurability. PhD thesis, Siena (2006)
5. Korovina, M.V.: Computational aspects of sigma-definability over the real numbers without the equality test. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 330–344. Springer, Heidelberg (2003)
6. Korovina, M.V.: Gandy's theorem for abstract structures without the equality test. In: Vardi, M.Y., Voronkov, A. (eds.) LPAR 2003. LNCS, vol. 2850, pp. 290–301. Springer, Heidelberg (2003)
7. Korovina, M.V., Kudinov, O.V.: Characteristic properties of majorant-computability over the reals. In: Gottlob, G., Grandjean, E., Seyr, K. (eds.) CSL 1998. LNCS, vol. 1584, pp. 188–203. Springer, Heidelberg (1999)
8. Korovina, M.V., Kudinov, O.V.: Semantic characterisations of second-order computability over the real numbers. In: Fribourg, L. (ed.) CSL 2001 and EACSL 2001. LNCS, vol. 2142, pp. 160–172. Springer, Heidelberg (2001)
9. Korovina, M.V., Kudinov, O.V.: Towards computability of higher type continuous data. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 235–241. Springer, Heidelberg (2005)
10. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)

Circuit Complexity of Regular Languages

Michal Koucký

Mathematical Institute of the Academy of Sciences of Czech Republic
Žitná 25, CZ-115 67 Praha 1, Czech Republic
koucky@math.cas.cz

Abstract. We survey our current knowledge of circuit complexity of regular languages. We show that regular languages are of interest as languages providing understanding of different circuit classes. We also prove that regular languages that are in AC^0 and ACC^0 are all computable by almost linear size circuits, extending the result of Chandra et al. [5].

Keywords: regular languages, circuit complexity.

1 Introduction

Regular languages and associated finite state automata occupy a prominent position in computer science. They come up in a broad range of applications from text processing to automatic verification. In theoretical computer science they play an important role in understanding computation. The celebrated result of Furst, Saxe and Sipser [6] separates circuit classes by showing that the regular language PARITY is not in AC^0, the class of languages that are computable by bounded-depth polynomial-size circuits consisting of unbounded fan-in And, Or gates and unary Not gates. The result of Barrington [1] shows that there are regular languages that are complete for the class NC^1, the class of languages computable by logarithmic-depth circuits consisting of fan-in two And, Or gates and unary Not gates. Recently in [8], regular languages were shown to separate classes of languages computable by ACC^0 circuits using linear number of gates and using linear number of wires. The ACC^0circuits are similar to AC^0 circuits but in addition they may contain unbounded fan-in Mod-q gates.

There is a rich classification of regular languages based on properties of their *syntactic monoids* (see e.g. [17,16]). (The syntactic monoid of a regular language is essentially the monoid of transformations of states of the minimal finite state automata for the language. See the next section for precise definitions.) It turns out that there is a close connection between algebraic properties of these monoids and computational complexity of the associated regular languages. In this article we survey our current knowledge of this relationship from the perspective of circuit complexity and we point out the still unresolved open questions. Beside that we provide a proof that all regular languages that are in AC^0 and ACC^0 are recognizable by AC^0 and ACC^0 circuits, resp., of almost linear size.

2 Preliminaries on Monoids

In order to understand regular languages one needs to understand their finite state automata. It turns out that the proper framework for such a study are associated transformation monoids. We recall few elementary concepts regarding monoids. A *monoid* M is a set together with an associative binary operation that contains a distinguished identity element 1_M. We will denote this operation multiplicatively, e.g., for all $m \in M$, $m1_M = 1_M m = m$. For a finite alphabet Σ, an example of a monoid is the (*free*) monoid Σ^* with the operation concatenation and the identity element the empty word. Except for the free monoid Σ^*, all the monoids that we consider in this paper will be finite.

An element $m \in M$ is called an *idempotent* if $m = m^2$. Since M is finite, there exists the smallest integer $\omega \geq 1$, the *exponent of* M, such that for every $m \in M$, m^ω is an idempotent. A monoid G where for every element $a \in G$ there is an *inverse* $b \in G$ such that $ab = ba = 1_G$ is a *group.* A monoid M is called *group-free* if every group $G \subseteq M$ is of size 1. (A group G in a monoid M does not have to be a subgroup of M, i.e., 1_M may differ from 1_G).

For a monoid M the *product over* M is the function $f : M^* \to M$ such that $f(m_1 m_2 \cdots m_n) = m_1 m_2 \cdots m_n$. The *prefix-product over* M is the function $\Pi_p : M^* \to M^*$ defined as $\Pi_p(m_1 m_2 \ldots m_n) = p_1 p_2 \cdots p_n$, where for $i = 1, \ldots, n$, $p_i = m_1 m_2 \cdots m_i$. Similarly we can define the *suffix-product over* M as a function $\Pi_s : M^* \to M^*$ defined by $\Pi_s(m_1 m_2 \ldots m_n) = s_1 s_2 \cdots s_n$, where $s_i = m_i m_{i+1} \cdots m_n$. For $a \in M$, the *a-word problem over* M is the language of words from M^* that multiply out to a. When we are not concerned about a particular choice of a we will refer to such problems as *word problems over* M.

For monoids M, N, a function $\phi : N \to M$ is a *morphism* if for all $u, v \in N$, $\phi(uv) = \phi(u)\phi(v)$. We say that $L \subseteq \Sigma^*$ can be *recognized by* M if there exist a morphism $\phi : \Sigma^* \to M$ and a subset $F \subseteq M$ so that $L = \phi^{-1}(F)$. A trivial variant of Kleene's theorem states that a language L is *regular* iff it can be recognized by some finite monoid. For every such L there is a minimal monoid $M(L)$ that recognizes L, which we call the *syntactic monoid of* L, and the associated morphism $\nu_L : \Sigma^* \to M(L)$ we call the *syntactic morphism of* L. The syntactic monoid $M(L)$ of L happens to be the monoid of state transformations generated by the minimum state finite automaton recognizing L, i.e. every element of $M(L)$ can be thought of as a map of states of the automaton to itself.

2.1 Boolean Circuits

Boolean circuits are classified by their size, depth and type of gates they use. For us the following classes of circuits will be relevant. NC^1 circuits are circuits of logarithmic depth consisting of fan-in two AND and OR gates, and unary NOT gates. Because of the bound on the depth and fan-in, NC^1 circuits are of polynomial size. AC^0, $\mathrm{AC}^0[q]$, ACC^0, TC^0 circuits are all of constant depth and polynomial size. AC^0 circuits consist of unbounded fan-in AND and OR gates, and unary NOT gates whereas $\mathrm{AC}^0[q]$ circuits contain in addition unbounded fan-in MOD-q gates. (A MOD-q gate is a gate that evaluates to one iff the number of ones that are feed into it is divisible by q.) ACC^0 circuits are union of $\mathrm{AC}^0[q]$ circuits over all $q \geq 1$. Finally, TC^0 circuits are circuits consisting of unbounded fan-in AND, OR and MAJ gates, and unary NOT gates. (A MAJ gate is a gate that evaluates to one iff the majority of its inputs is set to one.)

There are two possible measures of the circuit size—the number of gates and the number of wires. As these two measures usually differ by at most a square the difference in these measures is usually not important. As we will see for us it will make a difference. Unless we say otherwise we will mean by the size of a circuit the number of its gates.

Beside languages over a binary alphabet we consider also languages over an arbitrary alphabet Σ. In such cases we assume that there is some fixed encoding of symbols from Σ into binary strings of fixed length, and inputs from Σ^* to circuits are encoded symbol by symbol using such encoding. We use similar convention for circuits outputting non-Boolean values.

There is a close relationship between a circuit complexity of a regular language L and the circuit complexity of a word problem over its syntactic monoid $M(L)$. One can easily establish the following relationship.

Proposition 1

1. *If a regular language L is computable by a circuit family of size $s(n)$ and depth $d(n)$ and for some $k \geq 0$, $\nu_L(L^{=k}) = M(L)$ then the product over its syntactic monoid $M(L)$ is computable by a circuit family of size $O(s(O(n))+n)$ and depth $d(O(n)) + O(1)$.*
2. *If the product over a monoid M is computable by a circuit family of size $s(n)$ and depth $d(n)$ then any regular language with the syntactic monoid M is computable by a circuit family of size $s(n) + O(n)$ and depth $d(n) + O(1)$.*

The somewhat technical condition that for some k, $\nu_L(L^{=k}) = M(L)$ is unfortunately necessary as the language $LENGTH(2)$ of strings of even length does not satisfy the conclusion of the first part of the claim in the case of AC^0 circuits. However, the first part of the proposition applies in particular to regular languages that contain a *neutral letter*, a symbol that can be arbitrarily added into any word without affecting its membership/non-membership in the language. For $L \subseteq \Sigma^*$, $L^{=k}$ means $L \cap \Sigma^k$.

3 Mapping the Landscape

It is folklore that all regular languages are computable by linear size NC^1 circuits. Indeed by Proposition 1 it suffices to show that there are NC^1 circuits of linear size for the product of n elements over a fixed monoid M: recursively reduce computation of a product of n elements over M to a product of $n/2$ elements over M by computing the product of adjacent pairs of elements in parallel. Turning such a strategy into a circuit provides a circuit of logarithmic depth and linear size. Thus we can state:

Theorem 1. *Every regular language is computable by* NC^1 *circuits of linear size.*

Can all regular languages be put into even smaller circuit class? A surprising result of Barrington [1] indicates that this is unlikely: if a monoid M contains a non-solvable group then the word problem over M is hard for NC^1 under projections. Here, a *projection* is a simple reduction that takes a word w from a language L and maps it to a word w' from a language L' so that each symbol of w' depends on at most one symbol of w and the length of w' depends only on the length of w. Thus, unless NC^1 collapses to a smaller class such as TC^0, NC^1 circuits are optimal for some regular languages. The theorem of Barrington was further extended by Barrington et al. [2] to obtain the following statement.

Theorem 2 ([1,2]). *Any regular language whose syntactic monoid contains a non-solvable group is hard for* NC^1 *under projections.*

An example of a monoid with a non-solvable group is the group S_5 of permutations on five elements. Thus for example the word problem over the group S_5 is hard for NC^1 under projections.

Chandra, Fortune and Lipton [5] identified a large class of languages that are computable by AC^0 circuits.

Theorem 3 ([5]). *If a language L has a group-free syntactic monoid $M(L)$ then L is in* AC^0*.*

The regular languages with group-free syntactic monoids have several alternative characterizations. They are precisely the *star-free languages*, the languages that can be described by a regular expression using only union, concatenation and complement operations but not the operation star where the atomic expressions are languages $\{a\}$ for every $a \in \Sigma$. They are also the *non-counting languages*, the languages L that satisfy: there is an integer $n \geq 0$ so that for all words x, y, z and any integer $m \geq n$, $xy^m z \in L$ iff $xy^{m+1}z \in L$.

The proof of Chandra et al. uses the characterization of counter-free regular languages by flip-flop automata of McNaughton and Papert [9]. Using this characterization one only needs to prove that the prefix-product over *carry semigroup* is computable by AC^0 circuits. The carry semigroup is a monoid with three elements P, R, S which multiply as follows: $xP = x$, $xR = R$, $xS = S$ for any $x \in \{P, S, R\}$. The carry semigroup is especially interesting because of its relation to the problem of computing the addition of two binary numbers.

Chandra et al. also prove a partial converse of their claim.

Theorem 4 ([5]). *If a monoid M contains a group then the product over M is not in* AC^0*.*

Their proof shows how a product over a monoid with a group can be used to count the number of ones in an input from $\{0,1\}^*$ modulo some constant $k \geq 2$. However, by the result of Furst, Saxe and Sipser [6] that cannot be done in AC^0 so the product over monoids containing groups cannot be in AC^0.

There is still an apparent gap between Theorems 3 and 4. Namely, the language $LENGTH(2)$ of words of even length is in AC^0 although its syntactic monoid contains a group. This gap was closed by Barrington et al. [2].

Theorem 5 ([2]). *A regular language is in* AC^0 *iff for every* $k \geq 0$*, the image of* $L^{=k}$ *under the syntactic morphism* $\nu_L(L^{=k})$ *does not contain a group.*

Surprisingly, there is a beautiful characterization of these languages using regular expressions provided by [2]. L is in AC^0 iff it can be described by a regular expression using operations union, concatenation and complement with the atoms $\{a\}$ for every $a \in \Sigma$ and $LENGTH(q)$ for every $q \geq 1$. $LENGTH(q)$ is the language of all words whose length is divisible by q.

The remaining gap between regular languages with group-free monoids and monoids that contain non-solvable groups was essentially closed by Barrington:

Theorem 6 ([1]). *If a syntactic monoid of a language contains only solvable groups then the language is computable by* ACC^0 *circuits.*

An example of such a language is the language PARITY of words from $\{0,1\}^*$ that contain an even number of ones. There is a very nice characterization of also these languages by regular expressions of certain type. For this characterization we need to introduce one special regular operation on languages. For a language $L \subseteq \Sigma^*$ and $w \in \Sigma^*$, let L/w denote the number of initial segments of w which are in L. For integers $m > 1$ and $0 \leq r < m$ we define $\langle L, r, m\rangle = \{w \in \Sigma^*;\ L/w \equiv r \bmod m\}$. Straubing [14] shows that the syntactic monoid of a language contains only solvable groups iff the language can be described by a regular expression built from atoms $\{a\}$, for $a \in \Sigma$, using operations union, concatenation, complement and $\langle La, r, m\rangle$, for any $a \in \Sigma$, $m > 1$ and $0 \leq r < m$.

The above results essentially completely classify all regular languages with respect to their circuit complexity—they are complete for NC^1, they are computable in AC^0 or otherwise they are in ACC^0. It is interesting to note that the class TC^0 does not get assigned any languages unless it is equal either to NC^1 or ACC^0. Proving that a regular language with its syntactic monoid containing non-solvable group is in TC^0 would collapse NC^1 to TC^0. Currently not much is known about the relationship of classes ACC^0, TC^0, and NC^1 except for the trivial inclusions ACC^0$\subseteq$TC^0$\subseteq$NC^1.

4 Circuit Size of Regular Languages

In the previous section we have shown that all regular languages are computable by linear size NC^1 circuits. Can anything similar be said about regular languages in AC^0 or ACC^0? The answer may be somewhat surprising in the light of the following example. Let Th_2 be the language over the alphabet $\{0,1\}$ of words that contain at least two ones. This is clearly a regular language and it is in AC^0: check for all pairs of input positions whether anyone of them contains two ones. However this gives an AC^0 circuit of quadratic size and it is not at all obvious whether one can do anything better. Below we show a general procedure that produces more efficient circuits. We note here that the language Th_2 as well as all the *threshold languages* Th_k for up-to even poly-logarithmic k are in fact computable by linear size AC^0 circuits [12]. The construction of Ragde and

Wigderson [12] is based on perfect hashing and it is not known if it could be applied to other regular languages.

Despite that we can reduce the size of constant depth circuits computing regular languages as follows. Assume that a regular language L and the product over its syntactic monoid is computable by $O(n^k)$-size constant-depth circuits. We construct $O(n^{(k+1)/2})$-size constant-depth circuits for product over $M(L)$: divide an input $x \in M(L)^n$ into consecutive blocks of size $\sqrt{n}$ and compute the product of each block in parallel; then compute the product of the $\sqrt{n}$ block products. This construction can be iterated constantly many times to obtain:

Proposition 2. *Let L be a regular language computable by a polynomial-size constant-depth circuits over arbitrary gates. If the product over its syntactic monoid $M(L)$ is computable by circuits of the same size then for every $\epsilon > 0$, there is a constant-depth circuit family of size $O(n^{1+\epsilon})$ that computes L.*

A substantial improvement comes in the work of Chandra et al. [5] who prove:

Theorem 7 ([5]). *Let $g_0(n) = n^{1/4}$ and further for each $d = 0, 1, 2, \ldots$, $g_{d+1}(n) = g_d^*(n)$. Every regular languages L with a group-free syntactic monoid is computable by AC^0 circuits of depth $O(d)$ and size $O(n \cdot g_d^2(n))$, for any $d \geq 0$.*

Here $g^*(n) = \min\{i;\ g^{(i)}(n) \leq 1\}$, where $g^{(i)}(\cdot)$ denotes $g(\cdot)$ iterated i-times. Hence, Chandra et al. prove that almost all languages that are in AC^0 are computable by circuit families of almost linear size. Clearly the same is true for the product over group-free monoids. We generalize this to all regular languages computable in AC^0.

Theorem 8. *Let $g_d(n)$ be as in Theorem 7. Every regular languages L in AC^0 is computable by AC^0 circuits of depth $O(d)$ and size $O(n \cdot g_d^2(n))$, for any $d \geq 0$.*

The proof is a simple extension of the result of Chandra et al. We need to establish the following proposition that holds for all languages in AC^0.

Proposition 3. *Let for every $n \geq 0$, the image of $L^{=n}$ under the syntactic morphism ν_L does not contain any group. Then there is a $k \geq 1$ and a group-free monoid $M \subseteq M(L)$ such that for all $w \in \Sigma^*$, if k divides $|w|$ then $\nu_L(w) \in M$.*

Due to space limitations we omit proofs of Proposition 3 and Theorem 8 from this version of the paper. They can be found in the full version of the paper.

Chandra et al. prove actually the even stronger statement that the prefix-problem of these regular languages is computable in that size and using that many wires. We use the technique of Chandra et al. [5] together with the regular expression characterization of languages to show a similar statement for the regular languages in ACC^0. (Alternatively, we could use Thérien's characterization of regular languages in ACC^0 [17].)

Theorem 9. *Let $g_i(n)$ be as in Theorem 7. Every regular language L that is computable by ACC^0 circuits is computable by ACC^0 circuits of size $O(n \cdot g_i^2(n))$.*

The following general procedure that allows to build more efficient circuits for the prefix-product over a monoid M from circuits for the product over monoid M and less efficient circuits for the prefix-product over M is essentially the procedure of Chandra et al. Together with the inductive characterization of regular languages by regular expressions it provides the necessary tools to prove the above theorem. Let $g : \mathcal{N} \to \mathcal{N}$ be a non-decreasing function such that for all $n > 0$, $g(n) < n$, and M be a monoid with the product and prefix-product computable by constant-depth circuits.

CFL procedure:

Step 0. We split the input $x \in M^n$ iteratively into sub-words. We start with x as the only sub-word of length n and we divide it into $n/g(n)$ sub-words of size $g(n)$. We iterate and further divide each sub-word of length $l > 1$ into $l/g(l)$ sub-words of length $g(l)$. Hence, for $i = 0, \dots, g^*(n)$ we obtain $n/g^{(i)}(n)$ sub-words of length $g^{(i)}(n)$.

Step 1. For every sub-word obtained in Step 0 we compute its product over M.

Step 2. Using results from Step 1 and existing circuits for prefix-product, for each sub-word of length $l > 1$ from Step 0 we compute the prefix-product of the products of its $l/g(l)$ sub-words.

Step 3. For each $i = 1, \dots, n$, we compute the product of $x_1 \cdots x_i$ by computing the product of $g^*(n)$ values of the appropriate prefixes obtained in Step 2.

Let us analyze the circuit obtained from the above procedure. Assume that we have existing circuits of size $s(n)$ and constant depth d_s for computing product over M and of size $p(n)$ and constant depth d_p for computing prefix-product over M. Then the above procedure gives a circuits of depth $2d_s + d_p$ and size

$$\sum_{i=0}^{g^*(n)} \frac{n}{g^{(i)}(n)} \cdot s(g^{(i)}(n)) + \sum_{i=0}^{g^*(n)-1} \frac{n}{g^{(i)}(n)} \cdot p\left(\frac{g^{(i)}(n)}{g^{(i+1)}(n)}\right) + n \cdot s(g^*(n)).$$

We demonstrate the use of the above procedure. Let M be a monoid such that the product over M is computable by polynomial size constant-depth circuits. Let $\epsilon > 0$. Proposition 2 gives us circuits of size $s(n) = O(n^{1+(\epsilon/2)})$ for computing the product over M. By choosing $g(n) = n/2$ we obtain the following proposition.

Proposition 4. *Let L be a regular language computable by a polynomial-size constant-depth circuits over arbitrary gates. If the product over its syntactic monoid $M(L)$ is computable by similar circuits then for every $\epsilon > 0$, there is a constant-depth circuit family of size $O(n^{1+\epsilon})$ that computes the prefix-product over the monoid $M(L)$.*

Proposition 4 states clearly something non-trivial as a naïve construction of a prefix-product circuit would provide at least quadratic size circuits. By setting $g(n) = g_i^2(n)$ we obtain the following key lemma from the CFL procedure.

Lemma 1. *Let $g_i(n)$ be as in Theorem 7. Let M be a monoid. If there is a size $O(n \cdot g_{i+1}(n))$ depth d_s circuit family for computing product over M and* *a size $O(n \cdot g_i^2(n))$ depth d_p circuit family for computing prefix-product over M then there is a size $O(n \cdot g_{i+1}^2(n))$ depth $2d_s + d_p$ circuit family for computing prefix-product over M.*

It is trivial that if we can compute the prefix-product over some monoid M by $O(n \cdot g_i^2(n))$ circuits then we can also compute the product by the same size circuits. The above lemma provides essentially the other direction, i.e., building efficient circuits for the prefix-product from circuits for the product.

These two claims are sufficient to prove Theorem 9. We omit the proof due to space limitations. The proof can be found in the full version of this paper.

5 Wires vs. Gates

It is a natural question whether all languages that are in AC^0 and ACC^0 could be computed by AC^0 and ACC^0 circuits, resp., of linear size. This is not known, yet:

Problem 1. Is every regular language in AC^0 or ACC^0 computable by linear-size AC^0 or ACC^0 circuits?

One would be tempted to conjecture that this must be the case as $O(n \cdot g_d(n))$ may not look like a very natural bound. However, as we shall see further such an intuition fails when considering the number of wires in a circuit. As we mentioned earlier, Chandra et al. in fact proved Theorem 3 in terms of wires instead of gates. A close inspection of our arguments in the previous section reveals that our Theorems 8 and 9 also hold in terms of wires. Hence we obtain:

Theorem 10. *Let L be a regular language, $d > 0$ be an integer and functions g_d be as in Theorem 7.*

- *If L is in AC^0 then it is computable by AC^0 circuits with $O(ng_d^2(n))$ wires.*
- *If L is in ACC^0 then it is computable by ACC^0 circuits with $O(ng_d^2(n))$ wires.*

Interestingly enough the wire variant of Problem 1 was answered negatively:

Theorem 11 ([8]). *There is a regular language in AC^0 that requires AC^0 and ACC^0 circuits of depth $O(d)$ to have $\Omega(n \cdot g_d(n))$ wires.*

The language from the theorem is the simple language $U = c^*(ac^*bc^*)^*$. Although we have described it by a regular expression using the star-operation it is indeed in AC^0. What is really interesting about this language is that it is computable by ACC^0 circuits using a linear number of gates.

Theorem 12 ([8]). *The class of regular languages computable by ACC^0 circuits using linear number of wires is a proper subclass of the languages computable by ACC^0 circuits using linear number of gates.*

It is not known however whether the same is true for AC^0.

Problem 2. Are the classes of regular languages computable by AC^0 circuits using linear number of gates and liner number of wires different?

[8] provides a precise characterization of regular languages with neutral letter that are computable by AC^0 and ACC^0 circuits using linear number of wires.

Theorem 13 ([8]). *Let L be a regular language with a neutral letter.*

- *L is computable by AC^0 circuits with linear number of wires iff the syntactic monoid $M(L)$ satisfies the identity $(xyz)^\omega y(xyz)^\omega = (xyz)^\omega$, for every $x, y, z \in M(L)$.*
- *L is computable by ACC^0 circuits with linear number of wires iff the syntactic monoid $M(L)$ contains only commutative groups and $(xy)^\omega (yx)^\omega (xy)^\omega = (xy)^\omega$, for every $x, y, z \in M(L)$.*

Due to space limitations we omit here several other beautiful characterizations of the language classes from the previous theorem [8,15,16].

6 Conclusions

We have demonstrated that regular languages are very low on the ladder of complexity—they are computable by almost linear size circuits of different types. Still they provide important examples of explicit languages that separate different complexity classes. It is not much of an exaggeration to say that the current state of the art circuit separations are based on regular languages. Regular languages could still provide enough ammunition to separate say ACC^0 from NC^1. Such a separation is currently a major open problem.

Several other questions that may be more tractable remain also open. We already mentioned the one whether all languages that are in AC^0 and ACC^0 are computable by linear size constant-depth circuits. The language U defined in the previous section is particularly interesting as it is the essentially simplest regular language not known to be computable by linear size AC^0 circuits. It is also closely related to Integer Addition: if two binary represented numbers can be summed up in AC^0 using linear size circuits then U is computable by linear size circuits as well. We can state the following open problem of wide interest:

Problem 3. What is the size of AC^0 and ACC^0 circuits computing Integer Addition?

(Previously an unsupported claim appeared in literature that Integer Addition can be computed by linear size AC^0 circuits [12,7].) If U indeed is computable by linear size AC^0 circuits then it presents an explicit language that separates the classes of languages computable in AC^0 using linear number of gates and using linear number of wires. Such an explicit language is already known [8] however that language is not very natural and was constructed explicitly to provide this separation. If U is not computable by AC^0 circuits of linear size then neither is Integer Addition. We conclude with yet another very interesting problem that reaches somewhat outside of the realm of regular languages.

Problem 4. What is the number of wires in AC^0 and ACC^0 circuits computing Th_k, for $k \in \omega(n)$?

Acknowledgements

Theorem 9 was obtained jointly with Denis Thérien. I would like to thank him for allowing me to publish the proof here and for sharing with me his insights on regular languages. I am grateful to Pascal Tesson and anonymous referees for useful comments that helped to improve this paper. Partially supported by grant GA ČR 201/07/P276 and 201/05/0124.

References

1. Barrington, D.A.: Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. Journal of Computer and System Sciences 38(1), 150–164 (1989)
2. Barrington, D.A.M., Compton, K.J., Straubing, H., Thérien, D.: Regular languages in NC^1. Journal of Computer and System Sciences 44(3), 478–499 (1992)
3. Barrington, D.A.M., Thérien, D.: Finite Monoids and the Fine Structure of NC^1. Journal of ACM 35(4), 941–952 (1988)
4. Chandra, A.K., Fortune, S., Lipton, R.J.: Lower bounds for constant depth circuits for prefix problems. In: Proc. of the 10th ICALP, pp. 109–117 (1983)
5. Chandra, A.K., Fortune, S., Lipton, R.J.: Unbounded fan-in circuits and associative functions. Journal of Computer and System Sciences 30, 222–234 (1985)
6. Furst, M., Saxe, J., Sipser, M.: Parity, circuits and the polynomial time hierarchy. Mathematical Systems Theory 17, 13–27 (1984)
7. Chaudhuri, S., Radhakrishnan, J.: Deterministic restrictions in circuit complexity. In: Proc. of the 28th STOC, pp. 30–36 (1996)
8. Koucký, M., Pudlák, P., Thérien, D.: Bounded-depth circuits: Separating wires from gates. In: Proc. of the 37th STOC, pp. 257–265 (2005)
9. McNaughton, R., Papert, S.A.: Counter-Free Automata. The MIT Press, Cambridge (1971)
10. Pin, J.-E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Chap. 10 in Handbook of language theory, vol. I, pp. 679–746. Springer, Heidelberg (1997)
11. Pudlák, P.: Communication in bounded depth circuits. Combinatorica 14(2), 203–216 (1994)
12. Ragde, P., Wigderson, A.: Linear-size constant-depth polylog-threshold circuits. Information Processing Letters 39, 143–146 (1991)
13. Schwentick, T., Thérien, D., Vollmer, H.: Partially ordered two-way automata: a new characterization of DA. In: Proc. of DLT, pp. 242–253 (2001)
14. Straubing, H.: Families of recognizable sets corresponding to certain varieties of finite monoids. Journal of Pure. and Applied Algebra 15(3), 305–318 (1979)
15. Tesson, P., Thérien, D.: Restricted Two-Variable Sentences, Circuits and Communication Complexity. In: Proc. of ICALP, pp. 526–538 (2005)
16. Tesson, P., Thérien, D.: Bridges between algebraic automata theory and complexity theory. The Complexity Column, Bull. EATCS 88, 37–64 (2006)
17. Thérien, D.: Classification of Finite Monoids: the Language Approach. Theoretical Computer Science 14, 195–208 (1981)
18. Vollmer, H.: Introduction to Circuit Complexity - A Uniform Approach. In: Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (1999)

Definability in the Homomorphic Quasiorder of Finite Labeled Forests

Oleg V. Kudinov[1,⋆] and Victor L. Selivanov[2,⋆⋆]

[1] S.L. Sobolev Institute of Mathematics
Siberian Division Russian Academy of Sciences
kud@math.nsc.ru
[2] A.P. Ershov Institute of Informatics Systems
Siberian Division Russian Academy of Sciences
and
Theoretische Informatik, Universität Würzburg
selivanov@informatik.uni-wuerzburg.de

Abstract. We prove that for any $k \geq 3$ each element of the homomorphic quasiorder of finite k-labeled forests is definable, provided that the minimal non-smallest elements are allowed as parameters. As corollaries, we show that the structure is atomic and characterize the automorphism group of the structure. Similar results hold true for two other relevant structures: the homomorphic quasiorder of finite k-labeled trees, and of finite k-labeled trees with a fixed label of the root element.

Keywords: Labeled tree, forest, homomorphic quasiorder, definability, atomic structure, automorphism.

1 Introduction

In [Se04, Se06, KS06] the structure $(\mathcal{F}_k; \leq)$, $k \geq 2$, of finite k-labeled forests with the homomorphic quasiorder was studied. The structure is interesting in its own right since the homomorphic quasiorder is one in a series of relations on words, trees and forests relevant to computer science. The original interest to this structure [Se04] was motivated by its close relationship to the Boolean hierarchy of k-partitions. See [H96, Ko00, KW00, Se04, Se06, KS06, Ku06] for more motivation and background. Throughout this paper, k denotes an arbitrary integer, $k \geq 2$, which is identified with the set $\{0, \ldots, k-1\}$. The cardinality of a set F is denoted by $|F|$.

We use some standard notation and terminology on posets which may be found in any book on the subject, see e.g. [DP94]. We will not be very cautious when applying notions about posets also to quasiorders (known also as preorders); in such cases we mean the corresponding quotient-poset of the quasiorder.

⋆ Supported by RFBR Grant 05-01-00819a and by DFG-RFBR Grant 06-01-04002.

⋆⋆ Supported by RFBR Grant 4413.2006.1, by DFG Mercator program and by DFG-RFBR Grant 06-01-04002.

A poset $(P; \leq)$ will be often shorter denoted just by P (this applies also to structures of other signatures in place of $\{\leq\}$). Any subset of P may be considered as a poset with the induced partial ordering. In particular, this applies to the "cones" $\check{x} = \{y \in P | x \leq y\}$ and $\hat{x} = \{y \in P | y \leq x\}$ defined by any $x \in P$.

By *a forest* we mean a finite poset in which every lower cone $\hat{x}$ is a chain. *A tree* is a forest having a unique minimal element (called *the root* of the tree). Note that any forest is uniquely representable as a disjoint union of trees, the roots of the trees being the minimal elements of the forest. *A proper forest* is a forest which is not a tree. Notice that our trees and forests "grow bottom up", like the natural ones while trees in [Se04, Se06] grow in the opposite direction.

A *k-labeled poset* (or just a k-poset) is an object $(P; \leq, c)$ consisting of a poset $(P; \leq)$ and a labeling $c : P \to k$. Sometimes we simplify notation of a k-poset to (P, c) or even to P. A *homomorphism* $f : (P; \leq, c) \to (P'; \leq', c')$ between k-posets is a monotone function $f : (P; \leq) \to (P'; \leq')$ respecting the labelings, i.e. satisfying $c = c' \circ f$.

Let $\mathcal{F}_k$ and $\mathcal{T}_k$ be the classes of all finite k-forests and finite k-trees, respectively. Define [Ko00, KW00, Se04] a quasiorder $\leq$ on $\mathcal{F}_k$ as follows: $(P, c) \leq (P', c')$, if there is a homomorphism from (P, c) to (P', c'). By $\equiv$ we denote the equivalence relation on $\mathcal{F}_k$ induced by $\leq$. For technical reasons we consider also the empty k-forest $\emptyset$ (which is not assumed to be a tree) assuming that $\emptyset \leq P$ for each $P \in \mathcal{F}_k$. Note that in this paper (contrary to notation in [Se04]) we assume that $\emptyset \in \mathcal{F}_k$.

For arbitrary finite k-trees $T_0, \ldots, T_n$, let $F = T_0 \sqcup \cdots \sqcup T_n$ be their join, i.e. the disjoint union. Then F is a k-forest whose trees are exactly $T_0, \ldots, T_n$. Of course, every k-forest is (equivalent to) the join of its trees. Note that the join operation applies also to k-forests, and the join of any two k-forests is clearly their supremum under $\leq$. Hence, $(\mathcal{F}_k; \leq)$ is an upper semilattice.

A natural subset of k-trees is formed by k-chains, i.e. by words over the alphabet $k = \{0, \ldots, k-1\}$. We will denote such words in the usual way, as strings of symbols. E.g., 01221 and 011022 denote some words over the alphabet $\{0, 1, 2\}$. Note that any word is equivalent modulo $\equiv$ to a unique repetition-free word. E.g., the words above are equivalent to 0121 and 0102, respectively.

For every finite k-forest F and every $i < k$, let $p_i(F)$ be the k-tree obtained from F by joining a new smallest element and assigning the label i to this element. In particular, $p_i(\emptyset)$ will be the singleton tree carrying the label i. In [Se06] some properties of the operations $p_0, \ldots, p_{k-1}$ were established. It is clear that any k-forest is equivalent to a term of signature $\{\sqcup, p_0, \ldots, p_{k-1}, \emptyset\}$ without variables. E.g., the words from the preceding paragraph are equivalent to $p_0(p_1(p_2(p_1(\emptyset))))$ and $p_0(p_1(p_0(p_2(\emptyset))))$, respectively. Below we omit parenthesis whenever they are clear from the context, e.g. we could write the last term as $p_0 p_1 p_0 p_2(\emptyset)$.

For each $i < k$, let $\mathcal{T}_k^i$ be the set of finite k-trees the roots of which carry the label i. Our interest to the sets $\mathcal{F}_k$, $\mathcal{T}_k$ and $\mathcal{T}_k^i$ is explained by the above-mentioned relation to the Boolean hierarchy of k-partitions. Namely, the sets $\mathcal{T}_k^i$

and $\mathcal{F}_k \setminus \mathcal{T}_k$ generalize respectively the non-self-dual and self-dual levels levels of the Boolean hierarchy of sets (i.e., of 2-partitions).

The main result of this paper is now formulated as follows.

Theorem 1. *For any $k \geq 3$, each element of the quotient structure of $(\mathcal{F}_k; \leq, 0, \ldots, k-1)$ is definable. The same is true for the quotient structures of $(\mathcal{T}_k^0; \leq, 01, \ldots, 0(k-1))$ and $(\mathcal{T}_k; \leq, 0, \ldots, k-1)$.*

Recall that a relation in a structure is *definable* if there is a first-order formula of signature of the structure true exactly on the tuples that satisfy the relation. An element is definable if the corresponding singleton set is definable.

Because of space constraints, we give in this conference paper the proof of the main theorem only for the first structure; the omitted proofs for the other two are rather similar to the proof we present below, if one takes into account the corresponding results and techniques from [Se04, Se06, KS06].

Then we deduce some corollaries. The first main corollary states that the quotient structures of $(\mathcal{F}_k; \leq)$, $(\mathcal{T}_k^i; \leq)$ and $(\mathcal{T}_k; \leq)$ are atomic for all $k \geq 2$ and $i < k$. The second main corollary completely characterizes the automorphism groups of the three structures.

In Section 2 we formulate some necessary auxiliary facts, in Section 3 we prove the main theorem, in Section 4 we present some corollaries of the main theorem and in Section 5 we conclude with some additional remarks and open questions.

2 Auxiliary Facts

In this section we formulate without proofs several necessary facts established in [Se04, Se06]. Most of the corresponding proofs are short and straightforward and hence maybe hopefully easily recovered by the reader. Otherwise, please consult the source papers.

Recall that a quasiorder is called *a well quasiorder (wqo)* if it has neither infinite descending chains nor infinite antichains. Any wqo P has *a rank* $\text{rk}(P)$ which is the greatest ordinal isomorphically embeddable into P. For any $x \in P$, $\text{rk}(x) = \text{rk}(\hat{x})$ is the rank of x. With any quasiorder we associate also its *width* $w(P)$ defined as follows: if P has antichains with any finite number of elements, then $w(P) = \omega$, otherwise $w(P)$ is the greatest natural number n for which P has an antichain with n elements.

The first result cites some facts established in [Se04]. It implies in particular that any element x of $\mathcal{F}_k$ is uniquely representable as a finite union of join-irreducible elements; we call this the *canonical lattice representation* of x.

Proposition 1

(i) For any $k \geq 2$, $(\mathcal{F}_k; \leq)$ is a wqo with $rk(\mathcal{F}_k) = \omega$.
(ii) $w(\mathcal{F}_2) = 2$ and $w(\mathcal{F}_k) = \omega$ for $k > 2$.
(iii) For any $k \geq 2$, the quotient structure of $(\mathcal{F}_k; \leq)$ is a distributive lattice.
(iv) The finite k-trees define exactly the non-empty join-irreducible elements of the lattice $(\mathcal{F}_k; \leq)$.

The next easy fact was observed in [Se06].

Proposition 2

(i) $(\mathcal{T}_k^0, \ldots, \mathcal{T}_k^{k-1})$ *is a partition of* $\mathcal{T}_k$ *modulo* $\equiv$.

(ii) For any bijection $f : k \to k$, *the map* $(F, c) \mapsto (F, f \circ c)$ *defines an automorphism of* $(\mathcal{F}_k; \leq)$.

(iii) For all $i, j < k$, $(\mathcal{T}_k^i; \leq)$ *is isomorphic to* $(\mathcal{T}_k^j; \leq)$. *Moreover, there is an automorphism of* $(\mathcal{F}_k; \leq)$ *sending* $\mathcal{T}_k^i$ *onto* $\mathcal{T}_k^j$.

Next we formulate an interesting property of the lattice $\mathcal{F}_k$ enriched by the unary operations p_i from Introduction. For this we recall the following notion introduced in [Se82].

Definition 1. *By a semilattice with discrete closures of rank k (a dc-semilattice for short) we mean a structure* $(S; \cup, p_0, \ldots, p_{k-1})$ *satisfying the following axioms:*

1) $(S; \cup)$ *is an upper semilattice, i.e. it satisfies* $(x \cup y) \cup z = x \cup (y \cup z)$, $x \cup y = y \cup x$ *and* $x \cup x = x$; *as usual, by* $\leq$ *we denote the induced partial order on* S *defined by* $x \leq y$ *iff* $x \cup y = y$.

2) Every p_i, $i < k$, *is a closure operation on* $(S; \leq)$, *i.e. it satisfies* $x \leq p_i(x)$, $x \leq y \to p_i(x) \leq p_i(y)$ *and* $p_i(p_i(x)) \leq p_i(x)$.

3) The operations p_i *have the following discreteness property: for all distinct* $i, j < k$, $p_i(x) \leq p_j(y) \to p_i(x) \leq y$.

4) Every $p_i(x)$ *is join-irreducible, i.e.* $p_i(x) \leq y \cup z \to (p_i(x) \leq y \vee p_i(x) \leq z)$.

The next fact was observed in [Se06].

Proposition 3. *The quotient structure of* $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ *is a dc-semilattice, and* $p_i(\mathcal{F}_k) = \mathcal{T}_k^i$ *for any* $i < k$.

We recall also a result on the minimal forests, i.e. k-forests not equivalent (under $\equiv$) to k-forests of lesser cardinality. For a finite poset P, $h(P)$ will denote *the height of* P, i.e. the number of elements of the longest chain in P. For any $i, 1 \leq i \leq h(P)$, let $P(i) = \{x \in P | h(\hat{x}) = i\}$. Then $P(1), \ldots, P(h(P))$ is a partition of P on "levels"; note that $P(1)$ is the set of minimal elements of P.

The next assertion is Theorem 1.4 from [Se06] giving a kind of inductive description (by induction on the cardinality) of the minimal k-forests.

Proposition 4

(i) Any trivial (i.e., empty or singleton) k-forest is minimal.

(ii) A nontrivial k-tree (T, c) *is minimal iff* $\forall x \in T(1) \forall y \in T(2)(c(x) \neq c(y))$ *and the k-forest* $(T \setminus T(1), c)$ *is minimal.*

(iii) A proper k-forest is minimal iff all its k-trees are minimal and pairwise incomparable under $\leq$.

This provides the *canonical term representations* of the elements of $\mathcal{F}_k$ by variable-free terms of signature $\{\sqcup, p_0, \ldots, p_{k-1}, \emptyset\}$. The canonical terms are obtained from minimal trees by the usual procedure relating terms and trees.

3 Definability

Here we present a proof of the first assertion of Theorem 1. We start with a series of lemmas. For any $x \in \mathcal{T}_k$, let $x' = \bigsqcup\{y \in \mathcal{T}_k : y < x\}$. Since x is join-irreducible, $x' < x$. We need some properties of the introduced function $x \mapsto x'$ from $\mathcal{T}_k$ to $\mathcal{F}_k$. We call a partial function $f : \mathcal{F}_k \rightharpoonup \mathcal{F}_k$ definable if its graph is definable (note that this implies that both domain and range of f are definable). The first lemma is obvious.

Lemma 1

(i) For any $x \in \mathcal{T}_k$, x' is the biggest element y of $\mathcal{F}_k$ with $y < x$.

(ii) The function $x \mapsto x'$ is definable.

For $x \in \mathcal{F}_k$, let $\mathrm{pred}(x)$ be the set of maximal elements in $\{y \in \mathcal{F}_k : y < x\}$.

Lemma 2

(i) For any $x \in \mathcal{T}_k$, $\mathrm{pred}(x) = \{x'\}$.

(ii) For all $n > 0$ and pairwise incomparable $x_0, \ldots, x_n \in \mathcal{T}_k$, $\mathrm{pred}(x_0 \sqcup \cdots \sqcup x_n) = \{y_0, \ldots, y_n\}$, where $y_j = x'_j \sqcup (\bigsqcup_{l \neq j} x_l)$.

(iii) Let $n > 0$, $i < k$ and $x_0, \ldots, x_n$ be pairwise incomparable elements in $\mathcal{T}_k \setminus \mathcal{T}_k^i$. Then $p_i(y_0), \ldots, p_i(y_n)$ (where y_i are as in (ii)) are pairwise incomparable.

(iv) For all $x, \tilde{x} \in \mathcal{F}_k \setminus \mathcal{T}_k$, $\mathrm{pred}(x) = \mathrm{pred}(\tilde{x})$ implies $x = \tilde{x}$.

Proof.

(i) is obvious.

(ii) Let $x = x_0 \sqcup \cdots \sqcup x_n$, then obviously $y_j \leq x$ for any $j \leq n$. Since $x_j \not\leq y_j$, $y_j < x$. Elements $y_0, \ldots, y_n$ are pairwise incomparable because $x_l \leq y_j$ and $x_l \not\leq y_l$ for all $j \neq l$. It remains to show that if $z < x$ then $z \leq y_j$ for some $j \leq n$. By distributivity (see Proposition 1), $z = \tilde{x}_0 \sqcup \cdots \sqcup \tilde{x}_n$ for some $\tilde{x}_0 \leq x_0, \ldots, \tilde{x}_n \leq x_n$. Since $z < x$, $\tilde{x}_j < x_j$, hence $\tilde{x}_j \leq x'_j$ for some $j \leq n$. Therefore, $z \leq y_j$.

(iii) Follows from the fact that $x_l \leq p_i(y_j)$ and $x_l \not\leq p_i(y_l)$ for all $l \neq j$.

(iv) Let $x = x_0 \sqcup \cdots \sqcup x_n$ and $\tilde{x} = \tilde{x}_0 \sqcup \cdots \sqcup \tilde{x}_m$ be canonical lattice representations, so $n, m > 0$. From $\mathrm{pred}(x) = \mathrm{pred}(\tilde{x})$ and (ii) we get $n = m$ and w.l.o.g. $y_0 = \tilde{y}_0, \ldots, y_n = \tilde{y}_n$. For any $j \leq n$ we have $x_j < x$, hence $x_j \leq \tilde{y}_l$ for some $l \leq n$. Therefore, $x_j \leq \tilde{x}$. Since j was arbitrary, $x \leq \tilde{x}$. By symmetry, $\tilde{x} \leq x$. This completes the proof of the lemma.

The next lemma implies that the function $x \mapsto x'$ may be computed by induction on the canonical term representation from Section 2.

Lemma 3

(i) For any $i < k$, $(p_i(\emptyset))' = \emptyset$.

(ii) For all $n > 0$, $i < k$, and pairwise incomparable elements $x_0, \ldots, x_n \in \mathcal{T}_k \setminus \mathcal{T}_k^i$, $(p_i(x_0 \sqcup \cdots \sqcup x_n))' = p_i(y_0) \sqcup \cdots \sqcup p_i(y_n)$, where $y_j = x'_j \sqcup (\bigsqcup_{l \neq j} x_l)$, and the elements $p_i(y_0), \ldots, p_i(y_n)$ are pairwise incomparable.

(iii) For all $x \in \mathcal{F}_k$ and distinct $i, j < k$, $(p_i p_j(x))' = p_j(x) \sqcup p_i((p_j(x))')$ and the elements $p_j(x), p_i((p_j(x))')$ are incomparable.

Proof.
(i) is obvious.
(ii) It suffices to check that if $z \in \mathcal{T}_k$ and $z < p_i(x_0 \sqcup \cdots \sqcup x_n)$ then $z \leq p_i(y_0) \sqcup \cdots \sqcup p_i(y_n)$, because other properties follow from Lemma 2. If $z \notin \mathcal{T}_k^i$ then $z \leq x_0 \sqcup \cdots \sqcup x_n$, hence $z \leq x_j$ for some $j \leq n$. Therefore, $z \leq p_i(y_0) \sqcup \cdots \sqcup p_i(y_n)$. If $z \in \mathcal{T}_k^i$ then $z = p_i(z_0)$ for some canonical term $p_i(z_0)$. Then $z_0 < x_0 \sqcup \cdots \sqcup x_n$, hence $z_0 \leq y_j$ for some $j \leq n$. Therefore, $z \leq p_i(y_0) \sqcup \cdots \sqcup p_i(y_n)$.
(iii) First we check incomparability. The relation $p_j(x) \not\leq p_i((p_j(x))')$ is clear. Suppose that $p_i((p_j(x))') \leq p_j(x)$. Since $i \neq j$, $p_i((p_j(x))') < p_j(x)$ and therefore $p_i((p_j(x))') \leq p_j(x)'$. Since $p_j(x)' \notin \mathcal{T}_k$ by induction, this is a contradiction.

Clearly, $p_j(x) \sqcup p_i((p_j(x)') < p_i p_j(x)$. It remains to show that if $y \in \mathcal{T}_k$ and $y < p_i p_j(x)$ then $y \leq p_j(x) \sqcup p_i((p_j(x)')$. If $y \notin \mathcal{T}_k^i$ then $y \leq p_j(x)$ and we are done. Otherwise, $y \equiv p_i(z)$ for some z. Assuming w.l.o.g. that terms in the expression $p_i(z) < p_i p_j(x)$ are canonical (see end of Section 2), by definition of $\leq$ we get $z < p_j(x)$. Therefore, $z \leq (p_j(x))'$ and hence $y \leq p_i((p_j(x)')$. This completes the proof.

For any $x \in \mathcal{F}_k$, let $l(x) = \{i < k : i \leq x\}$, i.e. $l(x)$ is the set of labels assigned to nodes of some (or, equivalently, of any) forest representing x. The next assertion is an immediate corollary of the preceding one.

Lemma 4. *For any $x \in \mathcal{T}_k \setminus k$, $l(x) = l(x')$.*

The function $x \mapsto x'$ is in general not injective. E.g., if $x, y \in \mathcal{T}_k$ are of the same rank, $l(x) = l(y)$ and $|l(x)| < 3$ then $x' = y'$ is the unique element of rank $rk(x) - 1$ below x. The next assertion implies that the function $x \mapsto x'$ is injective in all other cases. Let $\mathcal{G}_k = \{x \in \mathcal{T}_k : |l(x)| \geq 3\}$.

Lemma 5. *If at least one of elements $x, y \in \mathcal{T}_k$ is in $\mathcal{G}_k$ then $x' = y'$ implies $x = y$.*

Proof. In case when exactly one of x, y is in $\mathcal{G}_k$ the assertion follows from Lemma 4. So assume that $x, y \in \mathcal{G}_k$, then $x', y' \in \mathcal{G}_k$ by Lemma 4. Let $x = p_i(t)$ and $y = p_j(s)$ be the canonical term representations.

First we check that $i = j$. Suppose not, and consider several cases. If $t \notin \mathcal{T}_k$ and $s \in \mathcal{T}_k$ then, by Lemma 3, all components in the canonical lattice representation of x' are in $\mathcal{T}_k^i$ while one of the components of y' is in $\mathcal{T}_k^j$, a contradiction with $y' = x'$. The case when $t \in \mathcal{T}_k$ and $s \notin \mathcal{T}_k$ follows by symmetry.

Now let $s, t \in \mathcal{T}_k$. By Lemma 3(iii), $t = p_j(s')$ and $s = p_i(t')$. Since $s' \notin \mathcal{T}_k$ by Lemma 3, $s' < t$. By Lemma 1, $s' \leq t'$. A symmetric argument shows that $t' \leq s'$, hence $s' = t'$. If $t' \in \mathcal{G}_k$ then, by induction on $(rk(x), rk(y))$, $s = t$ and therefore $p_j(s') = p_i(t')$, a contradiction. Finally, assume that $t' \notin \mathcal{G}_k$, i.e. $|l(t')| \leq 2$. By Lemma 4, $l(t') = l(t)$ and $l(s') = l(s)$. Since $t = p_j(s')$ and $s = p_i(t')$, $i \in l(s)$ and $j \in l(t)$. Therefore, $l(t) = l(s) = \{i, j\}$ and consequently $l(x) = l(y) = \{i, j\}$, a contradiction.

We have checked that $i = j$, so $x = p_i(t)$ and $y = p_i(s)$. The case when exactly one of s, t is in $\mathcal{T}_k$ is impossible by an above-used argument. If $s, t \notin \mathcal{T}_k$ then, by Lemma 3, $\{p_i(z) : z \in \text{pred}(t)\} = x' = y' = \{p_i(z) : z \in \text{pred}(s)\}$. From

Lemma 2 it follows that $\mathrm{pred}(t) = \mathrm{pred}(s)$ and hence $t = s$. Therefore, $x = y$. If $s, t \in \mathcal{T}_k$ then, by Lemma 3, $t \sqcup p_i(t') = s \sqcup p_i(s')$ and the components are incomparable. Hence, $s = t$ and $x = y$. This completes the proof of the lemma.

Now we start to define some elements in our structures.

Lemma 6. *Elements* $01, 10, 02, 20, 12, 21$ *are definable in* $(\mathcal{F}_3; \leq, 0, 1, 2)$.

Proof. By symmetry, it suffices to show that 01 and 10 are definable. It is easy to see that $x \in \{01, 10\}$ iff x is a minimal join-irreducible element above $0, 1$ with $2 \not\leq x$. The following formula $\psi(x)$ is true on 01 and false on 10:

$$\exists z \exists y (ir(z) \wedge \mathrm{ir}(y) \wedge \forall t (t < z \leftrightarrow t \leq x \sqcup y) \wedge \forall t (t < y \leftrightarrow t \leq 0 \sqcup 2))$$

where ir is a formula of signature $\{\leq\}$ that defines in every lattice exactly the non-zero join-irreducible elements. (Such a formula is written easily in signature $\{0, \leq, \cup\}$, namely $x \neq 0 \wedge \forall y \forall z (x \leq y \cup z \to (x \leq y \vee x \leq z))$. Since 0 and $\cup$ are first-order definable in signature $\{\leq\}$ the last formula may be rewritten as an equivalent formula of $\{\leq\}$.)

Indeed, for the case $x = 01$ we put $z = p_0(1 \sqcup 2)$, $y = 02$ and immediately obtain $\psi(01)$. Now let $x = 10$, and assume, towards a contradiction, that $\psi(10)$ is true and let z, y be some satisfying values. Then, by the second equivalence, $y \in \{02, 20\}$, let e.g. $y = 02$. By the first equivalence and Proposition 3, z is above at least one of $p_0(10 \sqcup 02)$, $p_1(10 \sqcup 02)$, $p_2(10 \sqcup 02)$. Taking in the first case $t = 010$, in the second case $t = 12$ and in the third case $t = 210$ we obtain a contradiction with the first equivalence. This completes the proof of the lemma.

Lemma 7. *The set* $\mathcal{T}_2^0$ *is definable in* $(\mathcal{F}_3; \leq, 0, 1, 2)$.

Proof. Let $\mathcal{L}$ be the set of 3-trees which have labels 0 and 2 and carry label 2 only on the leaves (i.e., maximal elements). Let $\mathcal{T}_{2,0}$ be the set of trees in $\mathcal{T}_2$ having a label 0. Let $\mathrm{del} : \mathcal{L} \to \mathcal{T}_{2,0}$ be the function that deletes all 2-labeled leaves. It is easy to see that the function del respects the homomorphic quasiorder. Then we have:

$$\begin{aligned} x \in \mathcal{T}_2 &\leftrightarrow \mathrm{ir}(x) \wedge 2 \not\leq x, \\ x \in \mathcal{T}_{2,0} &\leftrightarrow \mathrm{ir}(x) \wedge 0 \leq x \wedge 2 \not\leq x, \\ x \in \mathcal{L} &\leftrightarrow \mathrm{ir}(x) \wedge 0 \leq x \wedge 2 \leq x \wedge 20 \not\leq x \wedge 21 \not\leq x, \end{aligned}$$

and $\mathrm{del}(x) = \bigsqcup\{z \in \mathcal{T}_2 : z \leq x\}$. It is easy to check that for all $y \in \mathcal{F}_3$ there holds

$$y \in \mathcal{T}_2^0 \leftrightarrow y \in \mathcal{T}_{2,0} \wedge \forall x \in \mathcal{L}(y = \mathrm{del}(x) \to 02 \leq x).$$

By Lemma 6 and definitions above, $\mathcal{T}_2^0$ is definable. This completes the proof of the lemma.

Now we can prove a strengthening of Lemma 6.

Lemma 8. *For any* $k \geq 3$*, each element* $a \in \mathcal{T}_k \setminus \mathcal{G}_k$ *is definable in* $(\mathcal{F}_k, \leq, 0, \ldots, k-1)$.

Proof. If $|l(a)| \leq 1$ i.e. $a \in \{\emptyset, 0, \ldots, k-1\}$ then the assertion is obvious. Now let $l(a) = \{i, j\}$ for some distinct $i, j < k$. By Proposition 2(ii), there is an automorphism of $(\mathcal{F}_k; \leq)$ sending i to 0 and j to 1, hence we can w.l.o.g. assume that $i = 0$ and $j = 1$, i.e. a is (represented by) a repetition-free binary word w.

We use induction on the length $|w|$ of w. Suppose $|w| > 1$ and w starts with 0 (the last assumption may be made by symmetry), so $w = 0v$ for the unique repetition-free word of length $|w|-1$ starting with 1. By induction, v is definable by a formula $\phi_v(x)$. Then w is the least (under $\leq$) element y of $\mathcal{T}_2^0$ such that $\exists x(\phi_v(x) \wedge x \leq y)$. Since $\mathcal{T}_2^0$ is definable by the previous lemma, the last informal definition of w may be rewritten as a formula of signature $\{\leq, 0, \ldots, k-1\}$. This completes the proof.

Proof of Theorem 1 for $\mathcal{F}_k$. Let $k \geq 3$ and $a \in \mathcal{F}_k$. We check by induction on $\mathrm{rk}(a)$ that a is definable in $(\mathcal{F}_k, \leq, 0, \ldots, k-1)$. The case $\mathrm{rk}(a) \leq 1$, i.e. $a \in \{\emptyset, 0, \ldots, k-1\}$, is obvious. Let $a \notin \mathcal{T}_k \cup \{\emptyset\}$ and $a = a_0 \sqcup \cdots \sqcup a_n$, $n > 0$, be the canonical lattice representation of a. By induction, a_i is definable by a formula $\phi_{a_i}(x)$ for all $i \leq n$. Then the formula

$$\phi_a(x) = \exists x_0 \cdots \exists x_n((\bigwedge_{i \leq n} \phi_{a_i}(x_i)) \wedge x = x_0 \sqcup \cdots \sqcup x_n)$$

defines a. Now let $a \in \mathcal{T}_k$ and $rk(a) > 1$. If $a \in \mathcal{T}_k \setminus \mathcal{G}_k$ the assertion holds by the previous lemma. Finally, let $a \in \mathcal{G}_k$. By induction, there is a formula $\phi_{a'}(x)$ that defines a'. By Lemmas 5 and 1, the formula $\phi_a(x) = \mathrm{ir}(x) \wedge \exists y(\phi_{a'}(y) \wedge y = x'))$ defines a. This completes the proof.

4 Atomicity and Automorphisms

In this section we establish some corollaries of the main theorem. First we show that all three our structures are atomic, i.e. realize only the principal types (for definition of these well-known notions from model theory see any standard text on this subject, e.g. [CK73]).

Since each element of the structure $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ is represented by a term without variables, this structure is obviously atomic for every $k \geq 2$. Informally this means that the structure is the smallest in a model-theoretic sense. In [Se06] it was shown that this structure is also the smallest in an algebraic sense.

From the main theorem it follows that the structure $(\mathcal{T}_k; \leq, 0, \ldots, k-1)$ is atomic for every $k \geq 2$. The next result is a slight strengthening of this fact.

Theorem 2. *For all $k \geq 2$ and $i < k$, the quotient structures of $(\mathcal{F}_k; \leq)$, $(\mathcal{T}_k^i; \leq)$ and $(\mathcal{T}_k; \leq)$ are atomic.*

Proof. We give a proof only for the first structure but it works for the other two structures as well. As is well-known, atomicity is equivalent to definability of orbits of all tuples of elements. So we have to show that for any tuple $\bar{a} =$

$(a_0, \dots, a_n)$, $n \geq 0$, of elements of $\mathcal{F}_k$ its orbit $\mathrm{Orb}(\bar{a}) = \{f(\bar{a}) : f \in Aut(\mathcal{F}_k; \leq)\}$ is definable, where $f(\bar{a}) = (f(a_0), \dots, f(a_n))$. By the main theorem, for any $i \leq n$ there is a formula $\phi_{a_i}(x)$ of signature $\{\leq, 0, \dots, k-1\}$ that defines a_i. Then the formula $\phi_{\bar{a}}(\bar{x}) = \phi_{a_0}(x_0) \wedge \cdots \wedge \phi_{a_n}(x_n)$ of signature $\{\leq, 0, \dots, k-1\}$ defines $\bar{a}$.

Since any automorphism of $(\mathcal{F}_k; \leq)$ preserves ranks of elements and $(\mathcal{F}_k; \leq)$ is a wqo, the orbit $\mathrm{Orb}(\bar{a})$ is finite, i.e. $\mathrm{Orb}(\bar{a}) = \{\bar{b}_0, \dots, \bar{b}_m\}$ for some $m < \omega$ and tuples $\bar{b}_0, \dots, \bar{b}_m$. Then the formula $\phi(\bar{x}) = \phi_{b_0}(\bar{x}) \vee \cdots \vee \phi_{b_m}(\bar{x}))$ of signature $\{\leq, 0, \dots, k-1\}$ defines the orbit $\mathrm{Orb}(\bar{a})$ in $(\mathcal{F}_k; \leq, 0, \dots, k-1)$. The set $\{0, \dots, k-1\}$ is definable in $(\mathcal{F}_k; \leq)$ by a formula $\mu(x)$ stating that x is minimal among all non-smallest elements. Let $\psi(\bar{x})$ be the formula

$$\exists u_0 \cdots \exists u_{k-1}((\bigwedge_{i \neq i} u_i \neq u_j) \wedge (\bigwedge_{i<k} \mu(u_i)) \wedge \phi'(\bar{x}))$$

where $\phi'(\bar{x})$ is obtained from $\phi(\bar{x})$ by substituting variables $u_0, \dots, u_{k-1}$ in place of constant symbols $0, \dots, k-1$, respectively. Then $\psi(\bar{x})$ defines $\mathrm{Orb}(\bar{a})$ in $(\mathcal{F}_k; \leq)$, completing the proof of the theorem.

Since each element of the structure $(\mathcal{F}_k; \sqcup, p_0, \dots, p_{k-1})$ is represented by a term without variables, this structure is rigid, i.e. it has only the trivial identity automorphism. From the main theorem it follows that the structure $(\mathcal{F}_k; \leq, 0, \dots, k-1)$, as well as the other two structures with the constants, is also rigid. It is easy to obtain also a complete description of the automorphism groups of the structures without the constants. Let $\mathbf{S}_k$ be the symmetric group on k elements, i.e. the group of permutations of elements $0, \dots, k-1$. Let $\mathrm{Aut}(A)$ denote the automorphism group of a structure A. By $\simeq$ we denote the isomorphism relation.

Theorem 3

(i) For any $k \geq 2$, $\mathrm{Aut}(\mathcal{F}_k; \leq) \simeq \mathrm{Aut}(\mathcal{T}_k; \leq)$.
(ii) $\mathrm{Aut}(\mathcal{T}_2; \leq) \simeq \mathbf{S}_2^\omega$.
(iii) For any $k \geq 3$, $\mathrm{Aut}(\mathcal{F}_k; \leq) \simeq \mathbf{S}_k$.
(iv) For all $k \geq 2$ and $i < k$, $\mathrm{Aut}(\mathcal{T}_k^i; \leq) \simeq \mathbf{S}_{k-1}$.

Proof.
(i) For any $f \in \mathrm{Aut}(\mathcal{F}_k; \leq)$, let f^* be the restriction of f to $\mathcal{T}_k$. Since $\mathcal{T}_k$ is definable in $(\mathcal{F}_k; \leq)$, $f^* \in \mathrm{Aut}(\mathcal{T}_k; \leq)$. By Proposition 1, the map $f \mapsto f^*$ is a surjective group homomorphism with the trivial kernel. Hence, $\mathrm{Aut}(\mathcal{F}_k; \leq) \simeq \mathrm{Aut}(\mathcal{T}_k; \leq)$.
(ii) It is well known and obvious that $(\mathcal{T}_2; \leq)$ is isomorphic to the partial order obtained from $(\omega; \leq)$ by substituting a pair of incomparable points (a_n^0, a_n^1) in place of any $n \in \omega$. The group $\mathbf{S}_2^\omega$ is isomorphic to the ω-cartesian power of the cyclic group $\{0, 1\}$ with two elements, hence it consists of functions $h : \omega \to \{0, 1\}$. Relate to any such h the function $h^* : \mathcal{T}_2 \to \mathcal{T}_2$ defined by: for any $n < \omega$, if $h(n) = 0$ then $h^*(a_n^0) = a_n^0$ and $h^*(a_n^1) = a_n^1$; otherwise, $h^*(a_n^0) = a_n^1$ and $h^*(a_n^1) = a_n^0$. Then $h \mapsto h^*$ is a desired automorphism.
(iii) For any $f \in \mathrm{Aut}(\mathcal{F}_k; \leq)$, let f^* be the restriction of f to $\{0, \dots, k-1\}$. By Proposition 2(ii), the map $f \mapsto f^*$ is a group homomorphism from $\mathrm{Aut}(\mathcal{F}_k; \leq)$

onto $\mathbf{S}_k$. Since $(\mathcal{F}_k; \leq, 0, \ldots, k-1)$ is rigid, the kernel of this homomorphism is trivial. Hence, $\mathrm{Aut}(\mathcal{F}_k; \leq) \simeq \mathbf{S}_k$.

The assertion (iv) is proved similarly to (iii). This completes the proof.

5 Conclusion

In this paper definability in our structures was used to establish important algebraic and model-theoretic properties of the structures. In the journal version [KS0?] of [KS06] the definability was used to show that for every $k \geq 3$ the elementary theory of any of the three structures is computably isomorphic to the elementary theory of the structure $(\omega; +, \cdot)$. A small addition to that proof shows that actually the structure $(\omega; +, \cdot)$ is definable in $(\mathcal{F}_k; \leq)$ (as well as in the other two structures) without parameters. At the same time, some natural definability questions remain open. E.g., we do not currently know whether the set $\mathcal{C}_k$ of finite k-chains or the relation "$y = f(x)$ for some $f \in \mathrm{Aut}(\mathcal{F}_k; \leq)$" are definable in $(\mathcal{F}_k; \leq)$.

References

[CK73] Chang, C.C., Keisler, H.J.: Model Theory. North Holland, Amsterdam (1973)

[DP94] Davey, B.A., Pristley, H.A.: Introduction to Lattices and Order. Cambridge (1994)

[H96] Hertling, P.: Unstetigkeitsgrade von Funktionen in der effectiven Analysis. PhD thesis, FernUniversität Hagen, Informatik-Berichte pp. 208–211 (1996)

[Ko00] Kosub, S.: On NP-partitions over posets with an application to reducing the set of solutions of NP problems. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 467–476. Springer, Heidelberg (2000)

[KS06] Kudinov, O.V., Selivanov, V.L.: Undecidability in the homomorphic quasiorder of finite labeled forests. In: Beckmann, A. (ed.) Conf. Computability in Europe-2006. LNCS, vol. 3988, pp. 289–296. Springer, Berlin (2006)

[KS0?] Kudinov, O.V., Selivanov, V.L.: Undecidability in the homomorphic quasiorder of finite labeled forests. Accepted by Journal of Logic and Computation

[Ku06] Kuske, D.: Theories of orders on the set of words. Theoretical Informatics and Applications 40, 53–74 (2006)

[KW00] Kosub, S., Wagner, K.: The Boolean hierarchy of NP-partitions. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 157–168. Springer, Heidelberg (2000)

[Se82] Selivanov, V.L.: On the structure of degrees of generalized index sets. Algebra and Logic, Russian, there is an English translation 21(4), 472–491 (1982)

[Se04] Selivanov, V.L.: Boolean hierarchy of partitions over reducible bases. Algebra and Logic, 43, N 1 (2004), pp. 77–109 (Russian, there is an English translation), Technical Report 276, Institut für Informatik, Universität Würzburg (2001), see also http://www.informatik.uni-wuerzburg.de

[Se06] Selivanov, V.L.: The algebra of labeled forests modulo homomorphic equivalence. Conf. Computability in Europe- 2006. Beckmann, A., et.al. (eds.) University of Swansea Report Series #CSR 7-2006, pp. 241–250 (full version to appear in Algebra and Logic) (2006)

Physics and Computation: The Status of Landauer's Principle (Extended Abstract)

James Ladyman

Department of Philosophy, University of Bristol

1 Introduction

Realism about computation is the view that whether or not a particular physical system is performing a particular computation is at least sometimes a mind-independent feature of reality. The caveat 'at least sometimes' is necessary here because a realist about computation need not believe that all instances of computation should be realistically construed. The computational theory of mind presupposes realism about computation. If whether or not the human nervous system implements particular computations is not a natural fact about the world that is independent of whether we represent it as doing so, then the computational theory of mind fails to naturalise the mind. Realism about computation is also presupposed by attempts to use computational principles such as Landauer's Principle to dispel Maxwell's Demon. Realism about computation has been challenged by Hilary Putnam and John Searle among others. Various arguments have been put forward purporting to show that any physical system of sufficient complexity trivially implements all computations. Ladyman et al. (2007) offer a precisification and general proof Landauer's Principle. In order to do this they present an analysis of what it is for a physical process to implement a logical transformation. In this paper, their analysis is explained and its implications for realism about computation and the use of Landauer's Principle in foundational debates is assessed.

When we are concerned with the logical form of a computation and its formal properties then it can be theoretically described in terms of functions and relations between abstract entities. However, actual computation is realised by some physical process, and the latter is of course subject to physical laws and the laws of thermodynamics in particular. It is therefore important to consider whether or not there are any systematic connections between the logical properties of computations consider abstractly and the thermodynamical properties of their realizations. Rolf Landauer (1961) proposed such a general connection, known as Landauer's Principle, namely that the erasure of information in any computational device is necessarily accompanied by an appropriate increase in the thermodynamic entropy of the device and/or its environment. This result is often generalised as follows: (a) any logically irreversible process must result in an entropy increase in the non-information bearing degrees of freedom of the information-processing system or its environment; (b) any logically reversible process can be implemented thermodynamically reversibly (see for example Charles Bennett 2003). Landauer's Principle is the subject of much debate. In particular, John Norton (2005) and Owen Maroney (2005) both argue that Landauer's Principle has not been shown to hold in general.

In order to clarify the status of Landauer's Principleis it is necessary to precisely define a computation, and what it means to say that a computation is physically realized. In particular, this paper offers precise definitions of logical irreversibility and thermodynamic irreversibility, and a detailed analysis of what it means for a physical system to implement a logical transformation. The result of this analysis is the notion of an *L-machine*. This is a hybrid physical-logical entity that combines a physical device, a specification of which physical states of that device correspond to various logical states, and an evolution of that device which corresponds to the logical transformation L. Landauer's Principle can be restated and generalized to the claim that the logical irreversibility of L implies thermodynamic irreversibility of every corresponding L-machine.

Everyone agrees that there are both logically reversible and irreversible transformations, and that every logically reversible transformation is implementable in a thermodynamically reversible way, and that any such transformation can also be implemented in a thermodynamically irreversible way. Everyone also agrees that a logically irreversible transformation can be implemented in a thermodynamically irreversible way. So the issue is whether there are any logically irreversible transformations that can be implemented in a thermodynamically reversible way (as illustrated in table 1).

Table 1. A table representing the different possibilities for logical and thermodynamic reversibility. This paper addresses the issue of whether any logically irreversible transformation can be implemented thermodynamically reversibly.

Possibilities	Thermodynamically reversible	Thermodynamically irreversible
Logically reversible	✓	✓
Logically irreversible	?	✓

It is important to make a clear distinction between the logical and physical domains, and to avoid talk of logical 'processes' and refer instead to logical transformations and their implementation by *families* of physical processes. The term 'process' always refers to a physical process in which a system starts in some particular state and is guaranteed to end in some particular state[1]. Landauer's

[1] In general, the particular end state may be a probabilistic mixture of thermodynamic states, but usually the final state is not such a mixture. Although, in the former case, the system may be supposed to actually be in some specific component of the mixture, it is not guaranteed to end up in that component, and so this component state cannot be considered as the final state of the process.

Principle is only considered in the general and precise form introduced above: If L is logically irreversible, then every L-machine is thermodynamically irreversible[2].

2 Logical Irreversibility

A logical transformation is a *mathematical* operation, consisting of a single-valued map L from a finite set X of input states, into a finite set Y of output states (i.e. each input state is mapped by L to a unique output state). For example, consider the case of binary-valued logic, in which the input and output states are bit-strings (with 0 and 1 usually representing 'false' and 'true' respectively); the mapping L can be represented by a truth table, or as a digital circuit constructed from some set of universal gates (e.g. NAND and COPY). A logical transformation is *logically reversible* if and only if $L : X \rightarrow Y$ is a one-to-one (injective) mapping[3]. Hence with a reversible logical transformation, it is possible to uniquely reconstruct the input state from the output state. If L is not a one-to-one mapping, then it is *logically irreversible.*

It is crucial that there is a distinction between a logical transformation, which is a map from a *set* of logical states to a *set* of logical states, and a physical process, which is a change in a physical system whereby it goes from a *particular* physical state to a *particular* physical state. It follows that it makes no sense to talk of the implementation of a logical transformation by a physical process, rather in so far as logical transformations are implemented using physical systems, they are implemented by a family of processes. For the physical system to implement the logical transformation reliably, the family of processes must take each of the physical states that represent the logical input states to the appropriate physical state, that is the one that represents the right logical output state (The point here is clear in the case of a truth table, where each member of the family of processes corresponds to a single row). The notion of implementation of a logical transformation by a physical device is discussed in section 4 below.

3 Thermodynamic Irreversibility

Thermodynamic irreversibility is a feature of *physical* processes, expressed by the second law of thermodynamics. There is much controversy about how the latter can be justified on the basis of statistical mechanics. Without assuming anything about the relationship between phenomenological thermodynamics and statistical mechanics, its is assumed that the second law stated in terms of *thermodynamic entropy* is valid.

In thermodynamics various operational assumptions are made that allow the definition of the thermodynamic entropy of individual macroscopic states (up to

[2] An L-machine is just the most general way of capturing the idea of physically implementing a logical transformation L.

[3] Note that whether or not L is surjective is irrelevant for the present paper. This is because if there are output states that do not get arrived at by the implementation of the transformation these are irrelevant to thermodynamic considerations.

a constant)[4]. This is almost universally accepted, however, there is controversy about the assignment of entropy to probabilistic mixtures of macrostates (for example, see Norton (2005)). For example, consider the mixture of macrostates M_i, with probabilities q_i. Assuming that the assignment of entropy to such a state is legitimate, it might be supposed that it is simply the average of the individual entropies $S(M_i)$; explicitly, $\sum_i q_i S(M_i)$. However, it is common to also include a term to represent the contribution to the entropy of the probability distribution itself; explicitly:

$$S_{mixture} = \sum_i q_i S(M_i) - k \sum_i q_i \ln q_i \tag{1}$$

The latter term is an information theoretic entropy and its inclusion in thermodynamic calculations currently lacks rigorous foundational justification[5]. Ladyman et al (2007) offers a proof of Landauer's Principle that depends on the use of the information theoretic entropy and a proof that is independent of it.

Consider a system in a heat reservoir at temperature T undergoing some thermodynamic process p. If $\varDelta S_{sys}(p)$ is the change in the entropy of the system during the process p, and $\varDelta Q(p)$ is the heat flow from the system into the reservoir during the same process, then the second law requires that

$$\forall\, p, \quad \varDelta S_{sys}(p) + \frac{\varDelta Q(p)}{T} \geq 0 \tag{2}$$

Identifying $\varDelta S_{res}(p) = \varDelta Q(p)/T$ as the entropy change of the heat reservoir, define

$$\varDelta S_{tot}(p) = \varDelta S_{sys}(p) + \varDelta S_{res}(p) \tag{3}$$

as the total entropy change of the system and reservoir together. The second law can then be restated in the familiar form

$$\forall p, \quad \varDelta S_{tot}(p) \geq 0 \tag{4}$$

i.e. total entropy is non-decreasing over time.

A process p is *thermodynamically reversible* if and only if $\varDelta S_{tot}(p) = 0$.

If $\varDelta S_{tot}(p) > 0$, the physical process p cannot be run in reverse, as the reverse process p' would have $\varDelta S_{tot}(p') < 0$, and hence violate the second law. Therefore any process p for which $\varDelta S_{tot}(p) > 0$ is *thermodynamically irreversible.* As is well known, there are a number of formulations of the second law that are provably equivalent to this, modulo certain assumptions.

A family of physical processes is thermodynamically irreversible if and only if at least one of its members is. This is important for the definition of irreversibility for L-machines in the next section.

[4] See, for example, Fermi (1936), Chapter IV.

[5] However, such a justification is the subject of work in progress by Ladyman, Presnell and Short.

4 Implementing a Logical Transformation with a Physical Device

In order to analyze the connection between *logical* transformations, and *physical* thermodynamic processes, it is necessary to consider what it means for a physical system to implement a logical transformation. As stated above, a physical system can only implement a logical transformation through a family of processes. To physically implement a logical transformation, there must be: A physical device, a specification of which physical states of that device correspond to the possible logical states (call the former *representative states*), and a time evolution operator of that device. This combined system is an *L-machine.* Note that L names a particular logical transformation, so there are L_{AND}-machines, and so on.

The time evolution operator must generate the relevant family of processes, and the reliability of the implementation consists in the time evolution operator being such as to ensure that *whichever* of the representative physical states the device is prepared in, it ends up in the appropriate representative state. This insistence on generality is an important difference between the present approach and that of Maroney (2005) who considers only individual processes.

Furthermore, it is important to note that the time evolution operator must encode everything about the behaviour of the device, and so the possibility of an external agent intervening during its operation is ruled out. In particular this prohibits any such external agent affecting the time evolution of the system by making use of information about its state while it is running. In other words, intelligent agents (such as demons) may be introduced only if their knowledge and actions affecting the operation of the device are included in the specification of the L-machine and its time evolution. Heuristically, suppose that the interaction between the L-machine and the rest of the world is limited to the setting of the input state and the pressing of the 'go' button.

Formally, an L-machine is an ordered set

$$\{D, \{D_{in}(x)|x \in X\}, \{D_{out}(y)|y \in Y\}, \Lambda_L\} \tag{5}$$

consisting of

– A physical *device* D, situated in a heat bath at temperature T.
– A set $\{D_{in}(x)|x \in X\}$ of macroscopic input states of the device, which are distinct thermodynamic states of the system (i.e. no microstate is a component of more than one thermodynamic state). $D_{in}(x)$ is the representative physical state of the logical input state x.
– A set $\{D_{out}(y)|y \in Y\}$ of distinct thermodynamic output states of the device. $D_{out}(y)$ is the representative physical state of the logical output state y. Note that the set of input states and output states may overlap.
– A time-evolution operator Λ_L for the device, such that

$$\forall\, x \in X, \Lambda_L(D_{in}(x)) = D_{out}(L(x)). \tag{6}$$

An L-machine $\{D, \{D_{in}(x)|x \in X\}, \{D_{out}(y)|y \in Y\}, \Lambda_L\}$ physically implements L in the following sense. If D is prepared in the input state $D_{in}(x)$ corresponding to the logical input state $x \in X$, and is then evolved using Λ_L, it will be left in the output state $D_{out}(y)$ corresponding to the logical output state $y = L(x) \in Y$. This physical process is denoted by p_x.

$$\begin{tikzcd}
x \arrow[r, "L"] \arrow[d, equal] & y \arrow[d, equal] \\
D_{in}(x) \arrow[r, "\Lambda_L"'] & D_{out}(y)
\end{tikzcd}$$

Fig. 1. An illustration of the relationship between the logical states x and y and their representative physical states $D_{in}(x)$ and $D_{out}(y)$, showing the logical transformation L and the physical time evolution operator Λ_L

Note that the labelling of the states is essential to the identity of a L-machine. For example, exactly the same device and time-evolution operator could be used as part of both an L_{AND}-machine, and an L_{OR}-machine by the appropriate relabelling of the physical input and output states.

Consider the thermodynamics of the process p_x. If the entropy of the system in the state $D_{in}(x)$ is $S_{in}(x)$, the entropy of the system in state $D_{out}(L(x))$ is $S_{out}(L(x))$, and the heat flow from the system into the reservoir during the process is $\Delta Q(p_x) = T\Delta S_{res}(p_x)$, the total entropy change $\Delta S_{tot}(p_x)$ for the process will be given by

$$\Delta S_{tot}(p_x) = S_{out}(L(x)) - S_{in}(x) + \frac{\Delta Q(p_x)}{T} \geq 0. \tag{7}$$

This particular process will be thermodynamically reversible if $\Delta S_{tot}(p_x) = 0$. Note that in the commonly considered case in which the initial and final entropies of the system are the same, ΔS_{tot} is proportional to the heat flow from the system into the reservoir. Furthermore if the initial and final energies of the system are the same as well, then from the first law of thermodynamics, this heat flow is equal to the work done on the system.

An L-machine is *thermodynamically reversible* if and only if for all $x \in X$, $\Delta S_{tot}(p_x) = 0$ (i.e. if all of the processes p_x are thermodynamically reversible). An L-machine is therefore *thermodynamically irreversible* if there exists an $x \in X$ for which $\Delta S_{tot}(p_x) > 0$.

Note implementing L by implementing some other 'stronger' L' from which the outputs of L can be deduced is ruled out; for example, the logical transformation L' corresponding to the combination of L and keeping a copy of the input. Formally, a logical transformation L' is *stronger* than a logical transformation L just in case, for every input x, $L(x)$ can be recovered from $L'(x)$, but for at least one x, $L'(x)$ cannot be recovered from $L(x)$. It follows that if L' is stronger than L, then, for every x, $L(x) = L^*(L'(x))$, where L^* is a logically irreversible transformation[6]. In general an implementation of a logically stronger L' is not an implementation of L, and is *unfaithful* in the following sense: it allows that, for some x, more can be learnt about the value of x from the output $L'(x)$ than from $L(x)$ itself. Allowing that L can be implemented by the implementation of a logically stronger transformation L' must also be ruled out because it begs the question at issue here by implicitly assuming that Landauer's Principle is false: it would always be possible to implement a logically irreversible process by implementing a stronger logically reversible process, and all sides agree that this could be done in a thermodynamically reversible way.

Note also that in the above definition a unique representative state is assigned to each logical state as this makes for a clear and simple analysis. However, in general it could be allowed that more than one physical state represents the same logical state, in which case, for each x, $D_{in}(x)$ would be replaced by a set $\{D_{in}^{(1)}(x), D_{in}^{(2)}(x) \ldots\}$ of distinct physical states (and similarly for each y). Call such a generalisation a 'multi-L-machine'. The condition (6) on the time-evolution operator of the device would then generalise in an obvious way to

$$\forall\, x \in X, \forall\, D_{in}^{(i)}(x), \exists\, D_{out}^{(j)}(L(x)) \ : \ \Lambda_L(D_{in}^{(i)}(x)) = D_{out}^{(j)}(L(x)). \tag{8}$$

By definition, each representative state is a physically distinguishable macrostate, so assume that the device can be prepared in a specific $D_{in}^{(i)}(x)$, and it can be determined which of the $D_{out}^{(j)}(y)$ it ends up in. Hence, a *refinement* of any multi-L-machine, is the multi-L-machine obtained by choosing a particular representative state for each logical input state, and their corresponding output states, and keeping the device and time evolution operator the same.

For many multi-L-machines, every refinement is an L-machine and in such cases nothing is gained by considering the generalisation. However, in every other case there exists a refinement which under relabelling of its output states is an L'-machine, for some L' that is logically stronger than L. This is unfaithful in the sense defined above, and hence is ruled out. Furthermore, without ruling out these cases then, for any logically irreversible L, a machine that implements L merely in virtue of the fact that it is stipulated that for every logical input state x, the same physical state represents x and $L(x)$, where the time evolution operator is the identity operator could be considered. This clearly trivialises the notion of implementing a logical transformation. It is ruled out by the prescription above since it could be used to implement the logically stronger identity operation.

On the basis of the above definitions it is possible to prove Landauer's Principle from the Kelvin statement of the Second Law of Thermodynamics using a thermodynamic cycle.

[6] Note that $L'(x)$ can itself be logically irreversible, such as the logical transformation L' corresponding to the combination of L_{AND} and keeping a copy of the second input bit. L' is stronger than L_{AND} but is still logically irreversible.

Acknowledgements. This abstract is based on Ladyman, J., Presnell, S., Short, A. and Groisman, B. (2007), 'The connection between logical and thermodynamic irreversibility', in Studies in History and Philosophy of Modern Physics.

References

Bennett, C.H.: The logical reversibility of computation. IBM Journal of Research and Development 17, 525–532 (1973)

Bennett, C.H.: The Thermodynamics of Computation?A Review International Journal of Theoretical Physics 21, 905–940 (1982) (Reprinted in Leff and Rex (1990), 213–248)

Bennett, C.H.: Demons, Engines and the Second Law. Scientific American, vol. 257, pp. 108–116

Bennett, C.H.: Notes on Landauer's principle, reversible computation, and Maxwell's demon. Studies in the History and Philosophy of Modern Physics 34, 501–510 (2003)

Brillouin, L.: Maxwell's demon cannot operate: Information and entropy. I. Journal of Applied Physics 22, 338–343 (1951)

Bub, J.: Maxwell's Demon and the thermodynamics of computation. Studies in the History and Philosophy of Modern Physics 32, 569–579 (2001)

Earman, J., Norton, J.D.: Exorcist XIV: The wrath of Maxwell's demon. Part I: From Maxwell to Szilard. Studies in the History and Philosophy of Modern Physics 29, 435–471 (1998)

Earman, J., Norton, J.D.: Exorcist XIV: The wrath of Maxwell's demon. Part II: From Szilard to Landauer and beyond. Studies in the History and Philosophy of Modern Physics 30, 1–40 (1999)

Fermi, E.: Thermodynamics. Dover, New York (1936)

Feynman, R.P.: Feynman Lectures on Computation. In: Hey, J.G., Allen, W. (eds.) Reading, MA, Addison-Wesley, London (1996)

Jones, D.S.: Elementary Information Theory. Clarendon Press, Oxford (1979)

Ladyman, J., Presnell, S., Short, A., Groisman, B.: The connection between logical and thermodynamic irreversibility. Studies in History and Philosophy of Modern Physics (2007)

Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development (Reprinted in Leff and Rex (1990)) 5, 183–191 (1961)

Landauer, R.: Dissipation and heat generation in the computing process. IBM Journal of Research and Development 5, 183–191 (1961)

Leff, H.S., Rex, A.F. (eds.): Maxwell's demon: Entropy, information, computing, Bristol: Adam Hilger (1990)

Leff, H.S., Rex, A.F. (eds.): Maxwell's demon 2: Entropy, classical and quantum information, computing. Bristol: Institute of Physics (2003)

Maroney, O.J.E.: The (absence of a) relationship between thermodynamic and logical reversibility. Studies in History and Philosophy of Modern Physics 36, 355–374 (2005)

Norton, J.D.: Eaters of the lotus: Landauer's principle and the return of Maxwell's demon. Studies in the History and Philosophy of Modern Physics 36, 375–411 (2005)

Piechocinska, B.: Information erasure. Physical Review A, 61, 062314, 1–9 (2000)

Shizume, K.: Heat generation required by information erasure. Physical Review E 52, 3495–3499 (1995)

Szilard, L.: On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings. Zeitschrift für Physik 53, 840–856 (1929) (Reprinted in Leff and Rex (1990), 124–133)

Uffink, J.: Bluff Your Way in the Second Law of Thermodynamics. Studies In History and Philosophy of Modern Physics 32, 305–394 (2001)

Strict Self-assembly of Discrete Sierpinski Triangles

James I. Lathrop, Jack H. Lutz[⋆], and Scott M. Summers

Department of Computer Science, Iowa State University, Ames, IA 50014, USA
jil@cs.iastate.edu
lutz@cs.iastate.edu
summers@cs.iastate.edu

Abstract. Winfree (1998) showed that discrete Sierpinski triangles can self-assemble in the Tile Assembly Model. A striking molecular realization of this self-assembly, using DNA tiles a few nanometers long and verifying the results by atomic-force microscopy, was achieved by Rothemund, Papadakis, and Winfree (2004).

Precisely speaking, the above self-assemblies tile completely filled-in, two-dimensional regions of the plane, with labeled subsets of these tiles representing discrete Sierpinski triangles. This paper addresses the more challenging problem of the *strict self-assembly* of discrete Sierpinski triangles, i.e., the task of tiling a discrete Sierpinski triangle and nothing else.

We first prove that the standard discrete Sierpinski triangle *cannot* strictly self-assemble in the Tile Assembly Model. We then define the *fibered Sierpinski triangle*, a discrete Sierpinski triangle with the same fractal dimension as the standard one but with thin fibers that can carry data, and show that the fibered Sierpinski triangle strictly self-assembles in the Tile Assembly Model. In contrast with the simple XOR algorithm of the earlier, non-strict self-assemblies, our strict self-assembly algorithm makes extensive, recursive use of Winfree counters, coupled with measured delay and corner-turning operations. We verify our strict self-assembly using the local determinism method of Soloveichik and Winfree (2005).

Keywords: fractals, molecular computing, self-assembly, Sierpinski triangles.

1 Introduction

Structures that self-assemble in naturally occurring biological systems are often fractals of low dimension, by which we mean that they are usefully modeled as fractals and that their fractal dimensions are less than the dimension of the space or surface that they occupy. The advantages of such fractal geometries

[⋆] This author's research was supported in part by National Science Foundation Grant 0344187 and in part by Spanish Government MEC Project TIN 2005-08832-C03-02.

for materials transport, heat exchange, information processing, and robustness imply that structures engineered by nanoscale self-assembly in the near future will also often be fractals of low dimension.

The simplest mathematical model of nanoscale self-assembly is the Tile Assembly Model (TAM), an extension of Wang tiling [11,12] that was introduced by Winfree [14] and refined by Rothemund and Winfree [6,5]. (See also [1,4,9].) This elegant model, which is described in the technical appendix, uses tiles with various types and strengths of "glue" on their edges as abstractions of molecules adsorbing to a growing structure. (The tiles are squares in the two-dimensional TAM, which is most widely used, cubes in the three-dimensional TAM, etc.) Despite the model's deliberate oversimplification of molecular geometry and binding, Winfree [14] proved that the TAM is computationally universal in two or more dimensions. Self-assembly in the TAM can thus be directed algorithmically.

This paper concerns the self-assembly of fractal structures in the Tile Assembly Model. The typical test bed for a new research topic involving fractals is the Sierpinski triangle, and this is certainly the case for fractal self-assembly. Specifically, Winfree [14] showed that the *standard discrete Sierpinski triangle* **S**, which is illustrated in Figure 1, self-assembles from a set of seven tile types in the Tile Assembly Model. Formally, **S** is a set of points in the discrete Euclidean plane $\mathbb{Z}^2$. The obvious and well-known resemblance between **S** and the Sierpinski triangle in $\mathbb{R}^2$ that is studied in fractal geometry [3] is a special case of a general correspondence between "discrete fractals" and "continuous fractals" [13]. Continuous fractals are typically bounded (in fact, compact) and have intricate structure at arbitrarily small scales, while discrete fractals like **S** are unbounded and have intricate structure at arbitrarily large scales.

A striking molecular realization of Winfree's self-assembly of **S** was reported in 2004. Using DNA double-crossover molecules (which were first synthesized in pioneering work of Seeman and his co-workers [8]) to construct tiles only a few nanometers long, Rothemund, Papadakis and Winfree [7] implemented the molecular self-assembly of **S** with low enough error rates to achieve correct placement of 100 to 200 tiles, confirmed by atomic force microscopy (AFM). This gives strong evidence that self-assembly can be algorithmically directed at the nanoscale.

The abstract and laboratory self-assemblies of **S** described above are impressive, but they are not (nor were they intended or claimed to be) true fractal self-assemblies. Winfree's abstract self-assembly of **S** actually tiles an *entire quadrant* of the plane in such a way that five of the seven tile types occupy positions corresponding to points in **S**. Similarly, the laboratory self-assemblies tile completely filled-in, two-dimensional regions, with DNA tiles at positions corresponding to points of **S** marked by inserting hairpin sequences for AFM contrast. To put the matter figuratively, what self-assembles in these assemblies is not the fractal **S** but rather a two-dimensional canvas on which **S** has been painted.

In order to achieve the advantages of fractal geometries mentioned in the first paragraph of this paper, we need self-assemblies that construct fractal shapes *and nothing more*. Accordingly, we say that a set $F \subseteq \mathbb{Z}^2$ *strictly self-assembles* in the Tile Assembly Model if there is a (finite) tile system that eventually places a tile on each point of F and never places a tile on any point of the complement, $\mathbb{Z}^2 - F$.

The specific topic of this paper is the strict self-assembly of discrete Sierpinski triangles in the Tile Assembly Model. We present two main results on this topic, one negative and one positive.

Our negative result is that the standard discrete Sierpinski triangle $\mathbf{S}$ *cannot* strictly self-assemble in the Tile Assembly Model. That is, there is no tile system that places tiles on all the points of $\mathbf{S}$ and on none of the points of $\mathbb{Z}^2 - \mathbf{S}$. This theorem appears in section 3. The key to its proof is an extension of the theorem of Adleman, Cheng, Goel, Huang, Kempe, Moisset de Espanés, and Rothemund [2] on the number of tile types required for a finite tree to self-assemble from a single seed tile at its root.

Our positive result is that a slight modification of $\mathbf{S}$, the *fibered Sierpinski triangle* $\mathbf{T}$ illustrated in Figure 2, strictly self-assembles in the Tile Assembly Model. Intuitively, the fibered Sierpinski triangle $\mathbf{T}$ (defined precisely in section 4) is constructed by following the recursive construction of $\mathbf{S}$ but also adding a thin *fiber* to the left and bottom edges of each stage in the construction. These fibers, which carry data in an algorithmically directed self-assembly of $\mathbf{T}$, have thicknesses that are logarithmic in the sizes of the corresponding stages of $\mathbf{T}$. Intuitively, this means that $\mathbf{T}$ is visually indistinguishable from $\mathbf{S}$ at sufficiently large scales. Mathematically, it implies that $\mathbf{T}$ has the same fractal dimension as $\mathbf{S}$.

Since our strict self-assembly must tile the set $\mathbf{T}$ "from within," the algorithm that directs it is perforce more involved than the simple XOR algorithm that directs Winfree's seven-tile-type, non-strict self-assembly of $\mathbf{S}$. Our algorithm, which is described in section 5, makes extensive, recursive use of Winfree counters [14], coupled with measured delay and corner-turning operations. It uses 69 tile types, but these are naturally positioned into small functional groups, so that we can use Soloveichik and Winfree's local determinism method [10] to prove that $\mathbf{T}$ strictly self assembles in our tile system.

2 Preliminaries

We work in the n-dimensional discrete Euclidean space $\mathbb{Z}^n$, where n is a positive integer. (In fact, we are primarily concerned with the discrete Euclidean plane $\mathbb{Z}^2$.) We write U_n for the set of all *unit vectors*, i.e., vectors of length 1, in $\mathbb{Z}^n$. We regard the $2n$ elements of U_n as (names of the cardinal) *directions* in $\mathbb{Z}^n$.

We write $[X]^2$ for the set of all 2-element subsets of a set X. All *graphs* here are undirected graphs, i.e., ordered pairs $G = (V, E)$, where V is the set of *vertices* and $E \subseteq [V]^2$ is the set of *edges*.

An n-dimensional *grid graph* is a graph $G = (V, E)$ in which $V \subseteq \mathbb{Z}^n$ and every edge $\{\boldsymbol{a}, \boldsymbol{b}\} \in E$ has the property that $\boldsymbol{a} - \boldsymbol{b} \in U_n$. The *full grid graph* on a set $V \subseteq \mathbb{Z}^n$ is the graph $G^{\#}_V = (V, E)$ in which E contains *every* $\{\boldsymbol{a}, \boldsymbol{b}\} \in [V]^2$ such that $\boldsymbol{a} - \boldsymbol{b} \in U_n$.

We now give a brief and intuitive sketch of the Tile Assembly Model that is adequate for reading this paper. More formal details and discussion may be found in [14,6,5].

Intuitively, a tile type t is a unit square that can be translated, but not rotated, so it has a well-defined "side $\boldsymbol{u}$" for each $\boldsymbol{u} \in U_2$. Each side $\boldsymbol{u}$ is covered with a "glue" of "color" $col_t(\boldsymbol{u})$ and "strength" $str_t(\boldsymbol{u})$ specified by its type t. If two tiles are placed with their centers at adjacent points $\boldsymbol{a}, \boldsymbol{a}-\boldsymbol{u} \in \mathbb{Z}^2$, where $\boldsymbol{u} \in U_2$, and if their abutting sides have glues that match in both color and strength, then they form a *bond* with this common strength. If the glue does not so match, then no bond is formed between these tiles. In this paper, all glues have strength 0, 1, or 2. When drawing a tile as a square, each side's glue strength is indicated by whether the side is a dotted line (0), a solid line (1), or a double line (2). Each side's "color" is indicated by an alphanumeric label.

Given a set T of tile types, an *assembly* is a partial function $\alpha : T \dashrightarrow \mathbb{Z}^2$ that is *stable* in the sense that it cannot be "broken" into smaller assemblies without breaking bonds of total strength at least τ, where $\tau = 2$ is this paper.

Self-assembly begins with a *seed assembly* σ and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves stability at all times. A *tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, σ is a seed assembly with finite domain, and $\tau = 2$ is the temperature. A *generalized tile assembly system* (*GTAS*) is defined similarly, but without the finiteness requirements. An assembly α is *terminal*, and we write $\alpha \in \mathcal{A}_{\Box}[\mathcal{A}][\mathcal{T}]$, if no tile can be stably added to it. A GTAS $\mathcal{T}$ is *definitive*, or *produces a unique assembly*, if it has exactly one terminal assembly.

A set $X \subseteq \mathbb{Z}^2$ *weakly self-assembles* if there exist a TAS $\mathcal{T} = (T, \sigma, \tau)$ and a set $B \subseteq T$ such that $\alpha^{-1}(B) = X$ holds for every terminal assembly α. The set X *strictly self-assembles* if there is a TAS $\mathcal{T}$ for which every terminal assembly has domain X.

An *assembly sequence* in a TAS $\mathcal{T} = (T, \sigma, \tau)$ is an infinite sequence $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \ldots)$ of assemblies in which $\alpha_0 = \sigma$ and each α_{i+1} is obtained from α_i by the addition of a single tile. In general, even a definitive TAS may have a very large (perhaps uncountably infinite) number of different assembly sequences leading to a terminal assembly. This seems to make it very difficult to prove that a TAS is definitive. Fortunately, Soloveichick and Winfree [10] have recently defined a property, *local determinism*, of assembly sequences and proven the remarkable fact that, if a TAS $\mathcal{T}$ has *any* assembly sequence that is locally deterministic, then $\mathcal{T}$ is definitive.

We briefly review the standard discrete Sierpinski triangle and the calculation of its zeta-dimension.

Let $V = \{(1,0), (0,1)\}$. Define the sets $S_0, S_1, S_2, \cdots \subseteq \mathbb{Z}^2$ by the recursion

$$\begin{aligned} S_0 &= \{(0,0)\}\,, \\ S_{i+1} &= S_i \cup \left(S_i + 2^i V\right)\,, \end{aligned} \tag{2.1}$$

where $A+cB = \{\boldsymbol{a}+c\boldsymbol{b} | \boldsymbol{a} \in A \text{ and } \boldsymbol{b} \in B\}$. Then the *standard discrete Sierpinski triangle* is the set

$$\mathbf{S} = \bigcup_{i=0}^{\infty} S_i,$$

which is illustrated in Figure 1. It is well known that $\mathbf{S}$ is the set of all $(k, l) \in \mathbb{N}^2$ such that the binomial coefficient $\binom{k+1}{k}$ is odd. For this reason, the set $\mathbf{S}$ is also called *Pascal's triangle modulo 2*.

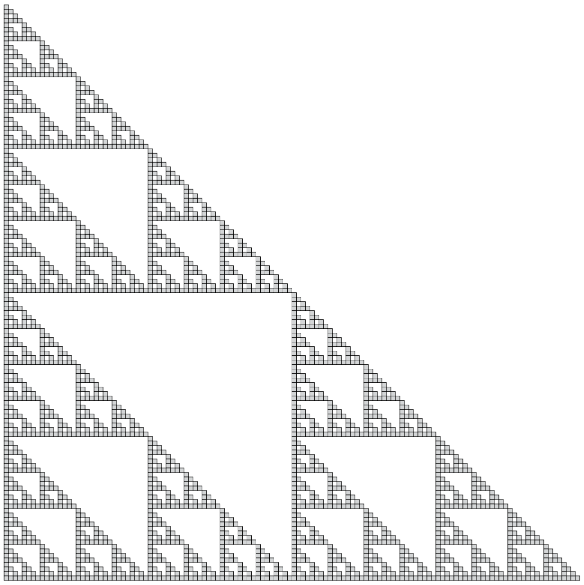


Fig. 1. Standard Discrete Sierpinski Triangle **S**

3 Impossibility of Strict Self-assembly of S

This section presents our first main theorem, which says that the standard discrete Sierpinski triangle $\mathbf{S}$ does not strictly self-assemble in the Tile Assembly Model. In order to prove this theorem, we first develop a lower bound on the number of tile types required for the self-assembly of a set X in terms of the depths of finite trees that occur in a certain way as subtrees of the full grid graph $G^{\#}_{X}$ of X.

Intuitively, given a set D of vertices of $G^{\#}_{X}$, we define a D-subtree of $G^{\#}_{X}$ to be any rooted tree in $G^{\#}_{X}$ that consists of *all* vertices of $G^{\#}_{X}$ that lie at or on the far side of the root from D. For simplicity, we state the definition in an arbitrary graph G.

Definition 1. *Let $G = (V, E)$ be a graph, and let $D \subseteq V$.*

1. *For each $r \in V$, the D-r-rooted subgraph of G is the graph*
$$G_{D,r} = \big(V_{D,r}, E_{D,r}\big),$$
where
$$V_{D,r} = \left\{ v \in V \,\middle|\, \begin{array}{l} \textit{every path from } v \textit{ to (any vertex in)} \\ D \textit{ in } G \textit{ goes through } r \end{array} \right\}$$
and
$$E_{D,r} = E \cap [V_{D,r}]^2 .$$
(Note that $r \in V_{D,r}$ in any case.)
2. *A D-subtree of G is a rooted tree B with root $r \in V$ such that $B = G_{D,r}$.*
3. *A branch of a D-subtree B of G is a simple path $\pi = (v_0, v_1, \ldots)$ that starts at the root of B and either ends at a leaf of B or is infinitely long.*

We use the following quantity in our lower bound theorem.

Definition 2. *Let $G = (V, E)$ be a graph and let $D \subseteq V$. The finite-tree depth of G relative to D is*
$$\textit{ft-depth}_D(G) = \sup \{\textit{depth}(B) | B \textit{ is a finite } D\textit{-subtree of } G\} .$$

We emphasize that the above supremum is only taken over *finite* D-subtrees. It is easy to construct an example in which G has a D-subtree of infinite depth, but $\text{ft-depth}_D(G) < \infty$.

Our lower bound result is the following.

Theorem 1. *Let $X \subseteq \mathbb{Z}^2$. If X strictly self-assembles in an n-GTAS $\mathcal{T} = (T, \sigma, \tau)$, then $|T| \geq \textit{ft-depth}_{\text{dom } \sigma}(G^{\#}_X)$.*

We note that Adleman, Cheng, Goel, Huang, Kempe, Moisset de Espanés, and Rothemund [2] proved the special case of Theorem 1 in which $G^{\#}_X$ is itself a finite tree and $\text{dom } \sigma = \{\boldsymbol{r}\}$, where $\boldsymbol{r}$ is the root of $G^{\#}_X$.

We now show that the standard discrete Sierpinski triangle $\mathbf{S}$ has infinite finite-tree depth.

Lemma 1. *For every finite set $D \subseteq S$, $\textit{ft-depth}_D(G^{\#}_{\boldsymbol{S}}) = \infty$.*

We now have the machinery to prove our first main theorem.

Theorem 2. $\mathbf{S}$ *does not strictly self-assemble in the Tile Assembly Model.*

4 The Fibered Sierpinski Triangle T

We now define the fibered Sierpinski triangle and show that it has the same fractal-dimension as the standard discrete Sierpinski triangle.

As in Section 2, let $V = \{(1, 0), (0, 1)\}$. Our objective is to define sets $T_0, T_1, T_2, \cdots \subseteq \mathbb{Z}^2$, sets $F_0, F_1, F_2, \cdots \subseteq \mathbb{Z}^2$, and functions $l, f, t : \mathbb{N} \to \mathbb{N}$ with the following intuitive meanings.

1. T_i is the i^{th} stage of our construction of the fibered Sierpinski triangle.
2. F_i is the *fiber* associated with T_i. It is the smallest set whose union with T_i has a vertical left edge and a horizontal bottom edge, together with one additional layer added to these two now-straight edges.
3. $l(i)$ is the length (number of tiles in) the left (or bottom) edge of $T_i \cup F_i$.
4. $f(i) = |F_i|$.
5. $t(i) = |T_i|$.

These five entities are defined recursively by the equations

$$\begin{aligned}
&T_0 = S_2 \text{ (stage 2 in the construction of } S\text{)},\\
&F_0 = (\{-1\} \times \{-1,0,1,2,3\}) \cup (\{-1,0,1,2,3\} \times \{-1\}),\\
&l(0) = 5,\\
&f(0) = 9 = t(0),\\
&T_{i+1} = T_i \cup ((T_i \cup F_i) + l(i)V), \qquad\qquad (4.1)\\
&F_{i+1} = F_i \cup (\{-i-1\} \times \{-i-1, -i, \cdots, l(i+1)-i-2\})\\
&\cup (\{-i-1, -i, \cdots, l(i+1)-i-2\} \times \{-i-1\}),\\
&l(i+1) = 2l(i)+1,\\
&f(i+1) = f(i) + 2l(i+1) - 1,\\
&t(i+1) = 3t(i) + 2f(i).
\end{aligned}$$

Comparing the recursions (2.1) and (4.1) shows that the sets $T_0, T_1, T_2, \cdots$ are constructed exactly like the sets $S_0, S_1, S_2, \cdots$, except that the fibers F_i are inserted into the construction of the sets T_i. A routine induction verifies that this recursion achieves conditions 2, 3, 4, and 5 above. The *fibered Sierpinski triangle* is the set

$$\mathbf{T} = \bigcup_{i=0}^{\infty} T_i,$$

which is illustrated in Figure 2. The resemblance between $\mathbf{S}$ and $\mathbf{T}$ is clear from the illustrations. We now verify that $\mathbf{S}$ and $\mathbf{T}$ have the same fractal-dimension.

Lemma 2. *$Dim_\zeta(\mathbf{T}) = Dim_\zeta(\mathbf{S})$.*

Proof. Solving the recurrences for l, f, and t, in that order, gives the formulas

$$\begin{aligned}
l(i) &= 3 \cdot 2^{i+1} - 1,\\
f(i) &= 3(2^{i+3} - i - 5),\\
t(i) &= \frac{3}{2}\left(3^{i+3} - 2^{i+5} + 2i + 11\right),
\end{aligned}$$

which can be routinely verified by induction. It follows readily that

$$\mathrm{Dim}_\zeta(\mathbf{T}) = \limsup_{n\to\infty} \frac{\log t(n)}{\log l(n)} = \log 3 = \mathrm{Dim}_\zeta(\mathbf{S}).$$

We note that the thickness $i+1$ of a fiber F_i is $O(\log l(i))$, i.e., logarithmic in the side length of T_i. Hence the difference between S_i and T_i is asymptotically negligible as $i \to \infty$. Nevertheless, we show in the next section that $\mathbf{T}$, unlike $\mathbf{S}$, strictly self-assembles in the Tile Assembly Model.

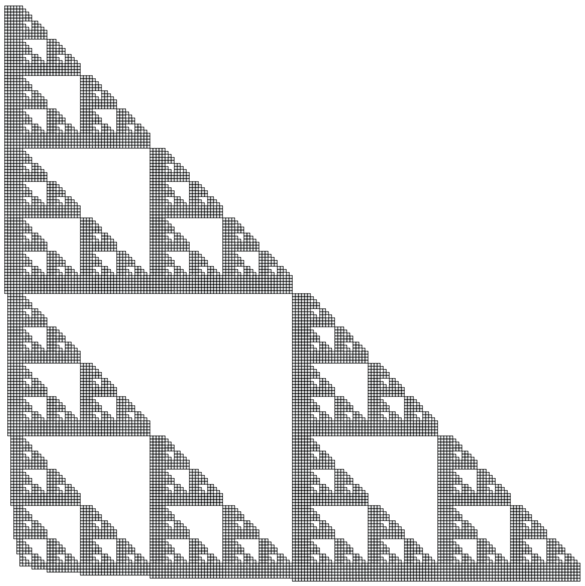


Fig. 2. Fibered Sierpinski Triangle $\mathbf{T}$

5 Strict Self-assembly of T

In this section, we present our second main result, which is the following.

Theorem 3. $\mathbf{T}$ *strictly self-assembles in the Tile Assembly Model.*

To show this, we use four modified versions of Winfree's binary counter that produce output ports (points at which structures may attach to the modified counter) which are used to grow additional modified counters at right angles. Recursively applying this procedure to the new counters yields the fibered Sierpinski triangle, as shown in Figure 2. Our singly-seeded tile set that produces the fibered Sierpinski triangle in this manner contains 69 tile types, and can be shown to be definitive using the method of local determinism [10].

We first construct a vertical log-width modified counter having the property that every natural number other than 0 is counted once and then is essentially repeated as many times as there are zero bits to the right of its right most one bit. We use shifters embedded in the counter to insert such *spacing rows.* Although log-width counters are infinite, the width of any row is fixed, and is in fact logarithmic in the number that it represents. It is easy to verify that the number of rows in a particular stage of width w of a log-width counter is $l(w-2)$, where the function l was defined in Section 4. As such, we use log-width counters to self-assemble the two axes of $\mathbf{T}$.

Self-assembly of the "internal structure" of $\mathbf{T}$ is accomplished via fixed-width modified counters, which are constructed directly from log-width modified counters. A fixed-width counter of width w will attach to a contiguous sequence of w spacing rows of an oppositely oriented modified counter. Counting starts at $2^w + 1$ and stops when the count overflows to 0. One can verify that the number of rows in a fixed-width modified counter of width w is $l(w-2) - w$.

The following figure illustrates a snapshot of a natural, infinite assembly sequence for $\mathbf{T}$ using our modified counters.

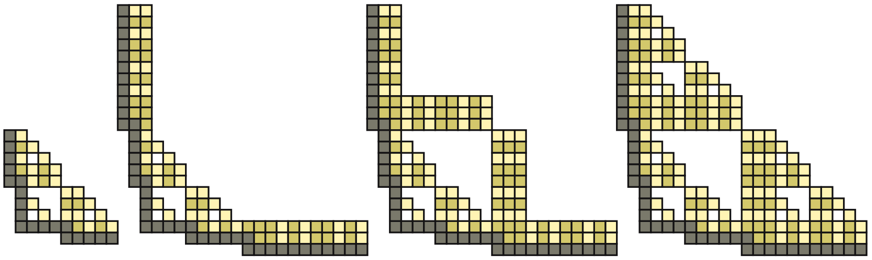


Fig. 3. Assembly sequence for $\mathbf{T}$

The darkest squares in the above figure represent tile types that are exclusive to log-width modified counters. It is these tile types that allow log-width counters to never stop counting. The lightest squares represent rows in which an increment operation occurs. All other squares represent the spacing rows, which not only delay the count by the appropriate amount but also provide the locations to which oppositely oriented modified counters ultimately attach.

Acknowledgment. We thank Dave Doty, Xiaoyang Gu, John Mayfield, Satya Nandakumar, and Kirk Sykora for useful discussions.

References

1. Adleman, L.: Towards a mathematical theory of self-assembly, Tech. report (1999)
2. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.-D. A., Kempe, D., de Espanés, P. M., Rothemund, P. W. K.: Combinatorial optimization problems in self-assembly, pp. 23–32 (2002)
3. Falconer, K.: Fractal geometry: Mathematical foundations and applications, 2nd edn. Wiley, Chichester (2003)
4. Reif, J.H.: Molecular assembly and computation: From theory to experimental demonstrations, International Colloquium on Automata, Languages and Programming, pp. 1–21 (2002)

5. Rothemund, P.W.K.: Theory and experiments in algorithmic self-assembly, Ph.D. thesis, University of Southern California (December 2001)
6. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). STOC, pp. 459–468 (2000)
7. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles, PLoS Biology 2(12) (2004)
8. Seeman, N.C.: Nucleic-acid junctions and lattices. Journal of Theoretical Biology 99, 237–247 (1982)
9. Soloveichik, D., Winfree, E.: Complexity of compact proofreading for self-assembled patterns, The Eleventh International Meeting on DNA Computing (2005)
10. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA Computing. LNCS, vol. 3384, Springer, Heidelberg (2005)
11. Wang, H.: Proving theorems by pattern recognition – II. The Bell System Technical Journal XL(1), 1–41 (1961)
12. Wang, H.: Dominoes and the AEA case of the decision problem. In: Proceedings of the Symposium on Mathematical Theory of Automata (New York, 1962), Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn, N.Y., pp. 23–55 (1963)
13. Willson, S.J.: Growth rates and fractional dimensions in cellular automata. Physica D 10, 69–74 (1984)
14. Winfree, E.: Algorithmic self-assembly of DNA, Ph.D. thesis, California Institute of Technology, June (1998)

Binary Trees and (Maximal) Order Types

Gyesik Lee

LIX - INRIA Futurs, Ecole Polytechnique, 91128 Palaiseau Cedex, France
leegy@lix.polytechnique.fr

Abstract. Concerning the set of rooted binary trees, one shows that Higman's Lemma and Dershowitz's recursive path ordering can be used for the decision of its maximal order type according to the homeomorphic embedding relation as well as of the order type according to its canonical linearization, well-known in proof theory as the Feferman-Schütte notation system without terms for addition. This will be done by showing that the ordinal ω_{n+1} can be found as the (maximal) order type of a set in a cumulative hierarchy of sets of rooted binary trees.

1 Introduction

Well-partial-ordering: A *quasi-ordering* is a pair $(X, \preceq)$, where X is a set and $\preceq$ is a transitive, reflexive binary relation on X. If $Y \subseteq X$ we write $(Y, \preceq)$ instead of $(Y, \preceq \restriction Y \times Y)$. A quasi-ordering $(X, \preceq)$ is called a *partial ordering* if $\preceq$ is antisymmetric, too.

For any partial ordering $(X, \preceq)$ and any $x, y \in X$ we write $x \prec y$ for $x \preceq y$ and $y \not\preceq x$. A *linear ordering* is a partial ordering $(X, \preceq)$ in which any two elements are $\preceq$-comparable.

A *well-quasi-ordering* (*wqo*) is a quasi-ordering $(X, \preceq)$ such that there is no infinite sequence $\langle x_i \rangle_{i \in \omega}$ of elements of X satisfying: $x_i \not\preceq x_j$ for all $i < j$. A *well-partial-ordering* (*wpo*) is a partial ordering which is well-quasi-ordered. $(X, \prec)$ is called *well-ordering* if $(X, \preceq)$ is a linear *wpo*. The following condition is necessary and sufficient for a partial ordering $(X, \preceq)$ to be a *wpo*:

Every extension of $\preceq$ to a linear ordering on X is a well-ordering.

In the following, we assume a basic knowledge about ordinals up to ε_0 and their arithmetic. Here are some notations:

$$\omega_0(\alpha) := \alpha \qquad \omega_{n+1}(\alpha) := \omega^{\omega_n(\alpha)} \qquad \omega_n := \omega_n(1)$$

The *order type* of a well-ordering $(X, \prec)$, *otyp*$(\prec)$, is the least ordinal for which there is an order-preserving function $f : X \to \alpha$:

$$otyp(\prec) := \min\{\alpha \colon \text{there is an order-preserving function } f : X \to \alpha\}$$

Given a *wpo* $(X, \preceq)$ consider an extension $(X, \prec^+)$ which is a well-ordering. How big is the order type of the well-ordering? Is there any *non-trivial* upper bound for it? Here 'non-trivial' means that the bound is lower than the obvious upper bound obtained by considering the cardinality of X. To this question, de Jongh and Parikh [1] gives a clear answer.

Definition 1. *Given a wpo* $(X, \preceq)$ *its* maximal order type *is defined as follows:*

$$o(X, \preceq) := \sup\{otyp(\prec^+) \colon \prec^+ \text{ is a well-ordering on } X \text{ extending } \preceq\}.$$

We simply write $o(X)$ for $o(X, \preceq)$ if it causes no confusion.

Theorem 2 (de Jongh and Parikh [1]). *If* $(X, \preceq)$ *is a wpo, then there is a well-ordering* $\prec^+$ *on* X *extending* $\preceq$ *such that* $o(X) = otyp(\prec^+)$*.*

We refer to Schmidt [2] for more extensive study concerning maximal order type.

Higman embedding: Given a set A, let A^* be the set of finite sequences of elements from A. Let $(A, \preceq)$ be a partial ordering. The *Higman embedding* $\preceq_{\mathrm{H}}$ is the partial ordering on A^* defined as follows:

$$a_1, \ldots, a_m \preceq_{\mathrm{H}} b_1, \ldots, b_n$$

if there is a strictly increasing function $g : [1, m] \to [1, n]$ such that $a_i \preceq b_{g(i)}$ for all $i \in [1, m]$.

Theorem 3

1. (Higman's Lemma) *If* $(A, \preceq)$ *is a wpo(resp. wqo), then so is* $(A^*, \preceq_{\mathrm{H}})$*.*
2. (de Jongh and Parikh) *If* $(A, \preceq)$ *is a wpo with* $o(A, \preceq) = \alpha > 0$*, then we have*:

$$o(A^*, \preceq_{\mathrm{H}}) = \begin{cases} \omega^{\omega^{\alpha-1}} & \text{if } \alpha \in \omega \setminus \{0\}. \\ \omega^{\omega^{\alpha}} & \text{if } \alpha = \beta + m, \text{ where } \beta \geq \omega,\ \beta \neq \omega^{\beta}, \text{ and } m \in \omega. \\ \omega^{\omega^{\alpha+1}} & \text{otherwise}. \end{cases}$$

Proof. See e.g. [3,1,2,4]. □

Binary trees: A *rooted binary tree* T is a set of nodes such that, if it is not empty, there is one distinguished node called the root of T and the remaining nodes are partitioned into two rooted binary trees. Here is a formal definition:

Assume a constant o and a binary function symbol φ are given. The set of rooted binary trees $\mathcal{B}$ is the least set of terms defined as follows:

- $o \in \mathcal{B}$;
- if α, $\beta \in \mathcal{B}$, then $\varphi(\alpha, \beta) \in \mathcal{B}$.

We will write $\varphi\alpha\beta$ instead of $\varphi(\alpha, \beta)$ if it causes no confusion. The *homeomorphic embeddability* relation $\trianglelefteq$ on $\mathcal{B}$ is the least subset of $\mathcal{B} \times \mathcal{B}$ defined as follows:

- $o \trianglelefteq \beta$ for all $\beta \in \mathcal{B}$;
- if $\alpha = \varphi\alpha_1\alpha_2$, $\beta = \varphi\beta_1\beta_2$, then $\alpha \trianglelefteq \beta$ if one of the following cases holds:
  (*i*) $\alpha \trianglelefteq \beta_1$ or $\alpha \trianglelefteq \beta_2$;
  (*ii*) $\alpha_1 \trianglelefteq \beta_1$ and $\alpha_2 \trianglelefteq \beta_2$.

Higman [3] showed that $(\mathcal{B}, \trianglelefteq)$ is a *wpo*, and in an unpublished paper, de Jongh showed that $o(\mathcal{B}, \trianglelefteq) = \varepsilon_0$. Furthermore, one easily finds a well-ordering $<$ extending $\trianglelefteq$ such that $otyp(<) = \varepsilon_0$: $\alpha < \beta$ is true if

- $\alpha = o$ and $\beta \neq o$; or
- $\alpha = \varphi\alpha_1\alpha_2$, $\beta = \varphi\beta_1\beta_2$ and one of the following cases holds:
  (*i*) $\alpha_1 < \beta_1$ and $\alpha_2 < \beta$; or
  (*ii*) $\alpha_1 = \beta_1$ and $\alpha_2 < \beta_2$; or
  (*iii*) $\alpha_1 > \beta_1$ and $\alpha \leq \beta_2$.

One can easily see that $\leq$ extends $\trianglelefteq$, and it is a folklore in proof theory that $<$ is a well-ordering on $\mathcal{B}$ with $otyp(<) = \varepsilon_0$. In fact, the system $(\mathcal{B}, <)$ is the system which is obtained from the Feferman-Schütte notation system for Γ_0 by omitting the addition terms. See e.g. [5,6,7,8] for more details.

In this paper, we will give a new proof that $o(\mathcal{B}, \trianglelefteq) = otyp(\mathcal{B}, <) = \varepsilon_0$. Furthermore, this will be done by characterizing the subsets of $\mathcal{B}$ which have ω_n as their maximal order types according to the homeomorphic embedding relation.

2 Cumulative Hierarchies $(\mathcal{B}^d)_d$ and $(\mathcal{B}^{d,k})_k$

In Weiermann [9], a cumulative hierarchy of $\mathcal{B}^d$ sucht that $\bigcup_d \mathcal{B}_d = \mathcal{B}$ is presented. Here we give cumulative hierchies $(\mathcal{B}^{d,k})_k$ such that $\bigcup_k \mathcal{B}^{d,k} = \mathcal{B}^d$ for any $d > 0$.[1]

Given a natural number d we define $\mathcal{B}^d$ recursively as follows:

- $o \in \mathcal{B}^d$;
- if $d > 0$, $\alpha \in \mathcal{B}^{d-1}$, and $\beta \in \mathcal{B}^d$, then $\varphi\alpha\beta \in \mathcal{B}^d$.

And define $\rho^d(\alpha)$ for $\alpha \in \mathcal{B}$ as follows:

$$\rho^0(\alpha) = \alpha \quad \text{and} \quad \rho^{d+1}(\alpha) = \varphi\rho^d(\alpha)0$$

Lemma 4. *Let d be a natural number.*

1. $\mathcal{B} = \bigcup\{\mathcal{B}^d \colon d \in \omega\}$.
2. *If* $\alpha \in \mathcal{B}^d$, *then* $\alpha < \rho^{d+1}(o)$ *and* $\rho^k(\alpha) \in \mathcal{B}^{d+k}$.
3. $\rho^{d+1}(o) \in \mathcal{B}^{d+1} \setminus \mathcal{B}^d$.
4. *If* $\alpha < \beta$, *then* $\rho^d(\alpha) < \rho^d(\beta)$.
5. *If* $\alpha \trianglelefteq \beta$, *then* $\rho^d(\alpha) \trianglelefteq \rho^d(\beta)$.
6. *If* $\alpha \in \mathcal{B}^{d+1} \setminus \mathcal{B}^d$ *and* $\beta \in \mathcal{B}^d$, *then* $\alpha \ntrianglelefteq \beta$ *and* $\beta < \alpha$.

[1] These cumulative hierarchies are essential for the proofs of phase transition of some combinatorial properties with respect to PA or IΣ_n respectively since they allow one to a structural approach to the sets from below. See Weiermann [9] and Lee [10] for more about phase transition concerning binary trees.

Proof. The first five claims are obvious. We show the last assertion by induction on α and β. If $\beta = 0$ there is nothing to show. Let $\alpha = \varphi\alpha_1\alpha_2$ and $\beta = \varphi\beta_1\beta_2$. If $\alpha_1 \in \mathcal{B}^{d-1}$, then $\alpha_2 \in \mathcal{B}^{d+1} \setminus \mathcal{B}^d$. Hence $\beta < \alpha_2 < \alpha$ by I.H. Now assume $\alpha_1 \in \mathcal{B}^d \setminus \mathcal{B}^{d-1}$. Then $\beta_1 < \alpha_1$ and $\beta_2 < \alpha$ by I.H., so $\beta < \alpha$ and $\alpha \ntrianglelefteq \beta$. □

Note that ω and B^1 can be identified by the isomorphism f defined as follows: $f(0) := o$ and $f(n+1) := \varphi(o, f(n))$. Hence we may talk about occurrences of natural numbers in $\alpha \in \mathcal{B}^d$, $d \geq 1$.

For $k \geq 1$ define

- $\mathcal{B}^{1,k} := \{0, 1, \ldots, k-1\}$.
- $\mathcal{B}^{d+1,k} := \{\alpha\colon \alpha = 0 \text{ or } \alpha = \varphi\beta\gamma, \text{ where } \beta \in \mathcal{B}^{d,k} \text{ and } \gamma \in \mathcal{B}^{d+1,k}\}$.

Lemma 5. *Let d, k be natural numbers.*

1. $\mathcal{B}^d = \bigcup_{k>0} \mathcal{B}^{d,k}$.
2. *If* $\alpha \in \mathcal{B}^{d+1,k}$, *then* $\alpha < \rho^d(k)$.
3. *If* $\alpha \in \mathcal{B}^{d,k+1} \setminus \mathcal{B}^{d,k}$ *and* $\beta \in \mathcal{B}^{d,k}$, *then* $\beta < \alpha$ *and* $\alpha \ntrianglelefteq \beta$.

Proof. Every claim can be shown by an simple induction on k. □

Given a positive natural number n define $\mathcal{B}_n$ by

$$\mathcal{B}_n := \begin{cases} \mathcal{B}^{d+1} & \text{if } n = 2d \\ \mathcal{B}^{d+1,2} & \text{if } n = 2d-1\,. \end{cases}$$

We claim

$$o(\mathcal{B}_n, \trianglelefteq \restriction \mathcal{B}_n) = otyp(< \restriction \mathcal{B}_n) = \omega_{n+1}\,.$$

3 Maximal Order Types

In general it is not a simple task to decide the maximal order type of a *wpo*. Some interesting methods are introduced in [2,11,4]. However, there is a problem that in most cases they can be carried out in a long-winded way only. Fortunately, there is a much more simple way for our case. We are going to take a well-known *wpo* and compare it with $(\mathcal{B}_n, \trianglelefteq)$.

Note first that the two sets $\mathcal{B}^{d+1}$ and $(\mathcal{B}^d)^*$ are similarly constructed. In fact, every $\alpha \in \mathcal{B}^{d+1}$ is of the form $\alpha = \varphi\alpha_1\varphi\alpha_2\cdots\varphi\alpha_m o$, where $\alpha_i \in \mathcal{B}^d$. If $\beta = \varphi\beta_1\varphi\beta_2\cdots\varphi\beta_n o \in \mathcal{B}^{d+1}$ and $\alpha_1\cdots\alpha_m \trianglelefteq_{\mathrm{H}} \beta_1\cdots\beta_n$ then $\alpha \trianglelefteq \beta$. And, though this relationship is not isomorphic, we can in fact show that $o(\mathcal{B}^{d+1}, \trianglelefteq) = o((\mathcal{B}^d)^*, \trianglelefteq_{\mathrm{H}})$.

We need the following obvious fact.

Lemma 6. *Let $(A, \preceq_1)$ and $(B, \preceq_2)$ be wpo's and $f\colon A \to B$ an injective function such that*

$$a \preceq_1 b \iff f(a) \preceq_2 f(b)$$

for all $a, b \in A$. Then it holds that $o(A) \leq o(B)$.

Theorem 7. *For any $d > 0$, $o(\mathcal{B}^{d+1}, \trianglelefteq) = o(\mathcal{B}^{d+1} \setminus \mathcal{B}^d, \trianglelefteq) = o((\mathcal{B}^d)^*, \trianglelefteq_{\mathrm{H}})$.*

Proof. Define $f\colon \mathcal{B}^{d+1} \to (\mathcal{B}^d)^*$ and $g\colon (\mathcal{B}^d)^* \to \mathcal{B}^{d+1} \setminus \mathcal{B}^d$ defined as follows:

$$f(\alpha) := \begin{cases} \epsilon & \text{if } \alpha = o \\ \alpha & \text{if } \alpha = \varphi\alpha_1\alpha_2 \in \mathcal{B}^d \\ \alpha_1, f(\alpha_2) & \text{if } \alpha = \varphi\alpha_1\alpha_2 \notin \mathcal{B}^d \end{cases}$$

and

$$g(\alpha_1, \dots, \alpha_m) := \varphi\alpha_1\varphi\alpha_2 \cdots \varphi\alpha_m \rho^{d+1}(o)$$

where ϵ denotes the empty sequence. It is then very easy to show that f and g satisfy the conditions in Lemma 6. So we have the desired equalities. □

Corollary 8. *For any $d > 0$, $(\mathcal{B}^d, \trianglelefteq)$ is a wpo and $o(\mathcal{B}^d, \trianglelefteq) = \omega_{2d-1}$.*

Proof. By induction on $d > 0$. If $d = 1$, then $\mathcal{B}^1 = \{o, \varphi oo, \varphi o(\varphi oo), \dots\}$ is linearly ordered by $\trianglelefteq$ and so $o(\mathcal{B}^1, \trianglelefteq) = \omega$. If $d > 1$, use I.H., Theorem 7, and Theorem 3. □

Corollary 9. *$(\mathcal{B}, \trianglelefteq)$ is a well-ordering and $o(\mathcal{B}) = \varepsilon_0$.*

Lemma 10. *Let d, k be positive natural numbers. Then*

$$o(\mathcal{B}^{d,k}, \trianglelefteq \restriction \mathcal{B}^{d,k}) = \begin{cases} k & \text{if } d = 1 \\ \omega_{2(d-1)}(k-1) & \text{otherwise}. \end{cases}$$

Proof. Similar to Corollary 8. □

Theorem 11. *$o(\mathcal{B}_n, \trianglelefteq \restriction \mathcal{B}_n) = \omega_{n+1}$ for any positive natural number n.*

4 Order Types

We are now going to compute the order types of $(\mathcal{B}_n, < \restriction \mathcal{B}_n)$. It is not so obvious as it might seem. $\mathcal{B}$ will be considered as ordinal notation systems based on the recursive path ordering on strings.

Definition 12 (Recursive path ordering). *Let $(\mathcal{A}, \prec)$ be a well-ordering. The* recursive path ordering *$\prec_{rpo}$ on $\mathcal{A}^*$ is defined as follows: Let ϵ be the empty list.*

- *If $\epsilon \prec_{rpo} u$ for $u \neq \epsilon$.*
- *If $u = au_1$ and $v = bv_1$, then $u \prec_{rpo} v$ if one of the following holds:*
  - *(i) $a \prec b$ and $u_1 \prec_{rpo} v$;*
  - *(ii) $a = b$ and $u_1 \prec_{rpo} v_1$;*
  - *(iii) $b \prec a$ and $u \preceq_{rpo} v_1$.*

Dershowitz [12] shows that the recursive path ordering preserves the well-orderedness.

Theorem 13 (Dershowitz). *If $(\mathcal{A}, \prec)$ is a well-ordering, so is $(\mathcal{A}^*, \prec_{rpo})$.*

Let $\xi < \varepsilon_0$ be the order type of $\prec$ on $\mathcal{A}$ and $\eta \mapsto a_\eta$, $\eta < \xi$, the enumeration function of $\mathcal{A}$. Using the idea elaborated by Touzet [13] we are going to characterize the order type of $\prec_{rpo}$ on $\mathcal{A}^*$.

Lemma 14. *For each limit ordinal $\alpha < \omega^{\omega^\xi}$ there are unique γ, β, and $\eta < \xi$ such that*

(i) $\alpha = \gamma + \omega^{\omega^\eta} \cdot \beta$,
(ii) $0 < \beta < \omega^{\omega^{\eta+1}}$, *and*
(iii) there are no $\mu \in \omega^{\omega^{\eta+1}} \setminus \{0\}$ and $\delta \in \omega^{\omega^\xi}$ such that $\gamma = \delta + \mu$.

Proof. Let $\alpha =_{NF} \omega^{\alpha_0} + \cdots + \omega^{\alpha_n}$. Let $\eta < \xi$ and j be such that

$$\omega^\eta \leq \alpha_n < \omega^{\eta+1} \quad \text{and} \quad j := \min\{k \colon \omega^\eta \leq \alpha_k < \omega^{\eta+1}\}\,.$$

There are δ_k, $j \leq k \leq n$, such that $\alpha_j = \omega^\eta + \delta_j, \ldots, \alpha_n = \omega^\eta + \delta_n$. Hence $\alpha = \gamma + \omega^{\omega^\eta} \cdot \beta$, where $\gamma =_{NF} \omega^{\alpha_0} + \cdots + \omega^{\alpha_{j-1}}$ and $\beta =_{NF} \omega^{\delta_j} + \cdots + \omega^{\delta_n}$, and η, β, γ satisfy the conditions (ii) and (iii).

We now prove the uniqueness of the decomposition. Let η', β', γ' also satisfy (i) $\sim$ (iii). If $\beta =_{NF} \omega^{\beta_0} + \cdots + \omega^{\beta_m}$ and $\beta' =_{NF} \omega^{\beta'_0} + \cdots + \omega^{\beta'_\ell}$ and if γ is in Cantor normal form too, then conditions (ii) and (iii) guarantee that

$$\alpha =_{NF} \gamma + \omega^{\omega^\eta + \beta_1} + \cdots + \omega^{\omega^\eta + \beta_m} =_{NF} \gamma' + \omega^{\omega^{\eta'} + \beta'_1} + \cdots + \omega^{\omega^{\eta'} + \beta'_\ell}$$

and hence $\eta = \eta'$. Suppose for instance $\gamma < \gamma'$. Then $\gamma' = \gamma + \omega^{\omega^\eta + \beta_1} + \cdots + \omega^{\omega^\eta + \beta_p}$ for some $p \leq m$. This contradicts (iii). So $\gamma = \gamma'$ and hence $m = \ell$, $\beta_k = \beta'_k$, $1 \leq k \leq m$. □

In the sequel, $\gamma + \omega^{\omega^\eta} \cdot \beta$ means always in the sense of Lemma 14. For ordinals $\beta > 0$, $-1 + \beta$ denotes $\beta - 1$ if $\beta < \omega$ and β otherwise.

Definition 15. *Let $(\mathcal{A}, \prec)$ be a well-ordering and $otyp(\prec) = \xi \in \varepsilon_0 \setminus \{0\}$. The function $\mathcal{O} \colon \omega^{\omega^{-1+\xi}} \to \mathcal{A}^*$ is defined by:*

$$\mathcal{O}(\alpha) := \begin{cases} \epsilon & \text{if } \alpha = 0 \\ a_0 \mathcal{O}(\beta) & \text{if } \alpha = \beta + 1 \\ a_{1+\eta} \mathcal{O}(-1+\beta)\mathcal{O}(\gamma) & \text{if } \alpha = \gamma + \omega^{\omega^\eta} \cdot \beta\,. \end{cases}$$

Now we are going to show that the definition of $((B^d)^*, <_{rpo})$ is just another way to see $(B^{d+1}, <)$.

Theorem 16. *Let $(\mathcal{A}, \prec)$ be a well-ordering. If $otyp(\prec) = \xi \in \varepsilon_0 \setminus \{0\}$ on $\mathcal{A}$, then we have on $\mathcal{A}^*$*

$$otyp(\prec_{rpo}) = \omega^{\omega^{-1+\xi}} = \begin{cases} \omega^{\omega^{\xi-1}} & \text{if } \xi \in \omega \setminus \{0\} \\ \omega^{\omega^\xi} & \text{otherwise}\,. \end{cases}$$

Proof. We show that the function $\mathcal{O}\colon (\omega^{\omega^{-1+\xi}}, <) \to (\mathcal{A}^*, \prec_{rpo})$ is an isomorphism.

1. $\mathcal{O}$ is order-preserving, i.e. $\mathcal{O}(\alpha) \prec_{rpo} \mathcal{O}(\beta)$ if $\alpha < \beta$. Note that the ordering $<$ on ordinals is the transitive closure of the schemes $\forall n \in \omega(\alpha_n < \alpha)$, where $(\alpha_n)_n$ builds a fundamental sequence for α. (The definition of the fundamental sequence will be directly given below in the proof.) So it suffices to show that $\forall n \in \omega(\mathcal{O}(\alpha_n) \prec_{rpo} \mathcal{O}(\alpha))$ for any $\alpha < \xi$.
   (a) $\alpha = \beta + 1$: Then $\alpha_n = \beta$ and $\mathcal{O}(\alpha_n) = \mathcal{O}(\beta) \prec_{rpo} a_0\mathcal{O}(\beta) = \mathcal{O}(\alpha)$.
   (b) $\alpha = \gamma + \omega^{\omega^\eta} \cdot (\beta + 1)$:
      – $\eta = 0$, i.e. $\alpha_n = \gamma + \omega^{\omega^0} \cdot \beta + n + 1$: Then
$$\mathcal{O}(\alpha_n) = \begin{cases} a_0^{n+1}\mathcal{O}(\gamma) & \text{if } \beta = 0 \\ a_0^{n+1}a_1\mathcal{O}(-1+\beta)\mathcal{O}(\gamma) & \text{otherwise} \end{cases}$$
$$\prec_{rpo}$$
$$\mathcal{O}(\alpha) = \begin{cases} a_1\mathcal{O}(\gamma) & \text{if } \beta = 0 \\ a_1\mathcal{O}(-1+\beta+1)\mathcal{O}(\gamma) & \text{otherwise}. \end{cases}$$
      – $\eta = \eta_0 + 1$, i.e. $\alpha_n = \gamma + \omega^{\omega^\eta} \cdot \beta + \omega^{\omega^{\eta_0}} \cdot \omega^{\omega^{\eta_0} \cdot n}$: Then
$$\mathcal{O}(\alpha_n) = \begin{cases} a_{1+\eta_0}^{n+1}\mathcal{O}(\gamma) & \text{if } \beta = 0 \\ a_{1+\eta_0}^{n+1}a_{1+\eta}\mathcal{O}(-1+\beta)\mathcal{O}(\gamma) & \text{otherwise} \end{cases}$$
$$\prec_{rpo}$$
$$\mathcal{O}(\alpha) = \begin{cases} a_{1+\eta}\mathcal{O}(\gamma) & \text{if } \beta = 0 \\ a_{1+\eta}\mathcal{O}(-1+\beta+1)\mathcal{O}(\gamma) & \text{otherwise}. \end{cases}$$
      – η is a limit ordinal, i.e.$\alpha_n = \gamma + \omega^{\omega^\eta} \cdot \beta + \omega^{\omega^{\eta_n}}$: Then
$$\mathcal{O}(\alpha_n) = \begin{cases} a_{1+\eta_n}\mathcal{O}(\gamma) & \text{if } \beta = 0 \\ a_{1+\eta_n}a_\eta\mathcal{O}(-1+\beta)\mathcal{O}(\gamma) & \text{otherwise} \end{cases}$$
$$\prec_{rpo}$$
$$\mathcal{O}(\alpha) = \begin{cases} a_\eta\mathcal{O}(\gamma) & \text{if } \beta = 0 \\ a_\eta\mathcal{O}(-1+\beta+1)\mathcal{O}(\gamma) & \text{otherwise}. \end{cases}$$
   (c) $\alpha = \gamma + \omega^{\omega^\eta} \cdot \lambda$, where λ is a limit ordinal: Then $\alpha_n = \gamma + \omega^{\omega^\eta} \cdot \lambda_n$ and $\mathcal{O}(\alpha_n) = a_{1+\eta}\mathcal{O}(-1+\lambda_n)\mathcal{O}(\gamma) \prec_{rpo} a_{1+\eta}\mathcal{O}(-1+\lambda)\mathcal{O}(\gamma) = \mathcal{O}(\alpha)$.
   We have shown that $\mathcal{O}$ is order-preserving, so it is injective.
2. Let $u \in \mathcal{A}^*$. By induction on the length of u we show that there is an $\alpha < \omega^{\omega^\xi}$ such that $\mathcal{O}(\alpha) = u$.
   (a) $u = \epsilon$: $\mathcal{O}(0) = \epsilon$.
   (b) $u = a_0 v$: Then $\mathcal{O}(\beta + 1) = a_0 v$, where $\mathcal{O}(\beta) = v$.
   (c) $u = a_\eta v$, $\eta > 0$: Then let $\eta' = \eta$ if $\eta \geq \omega$ and $\eta' = \eta + 1$ otherwise.

- $v \in \{a_0, \ldots, a_\eta\}^*$: Let $\mathcal{O}(-1+\beta) = v$. Then $-1+\beta < \omega^{\omega^{\eta'}}$ and $\mathcal{O}(\omega^{\omega^{-1+\eta}} \cdot \beta) = a_\eta \mathcal{O}(-1+\beta) = a_\eta v = u$.
  Note that this case implies, in particular, that $\mathcal{O}\colon \omega^{\omega^{\xi-1}} \to \mathcal{A}^*$ is an isomorphism if $\xi \in \omega \setminus \{0\}$. Indeed, if $\mathcal{A} = \{a_0, \ldots, a_\eta\}$ and $\xi = \eta + 1$ then we have just shown that $\alpha < \omega^{\omega^\eta}$ for α such that $\mathcal{O}(\alpha) = u$.
- $v \notin \{a_0, \ldots, a_\eta\}^*$: Let $b \in \mathcal{A} \setminus \{a_0, \ldots, a_\eta\}$, $v_1 \in \{a_0, \ldots, a_\eta\}^*$, and $v_2 \in \mathcal{A}^*$ such that $v = v_1 b v_2$. Let $\mathcal{O}(-1+\beta) = v_1$ and $\mathcal{O}(\gamma) = bv_2$. Then $\mathcal{O}(\gamma + \omega^{\omega^{-1+\eta}} \cdot \beta) = a_\eta \mathcal{O}(-1+\beta)\mathcal{O}(\gamma) = a_\eta v_1 b v_2 = u$. □

Corollary 17. *For any $d > 0$, $(\mathcal{B}^d, <)$ is a well-ordering and otyp$(< \restriction \mathcal{B}^d) = \omega_{2d-1}$.*

Proof. Note just that $(\mathcal{B}^{d+1}, <)$ is isomorphic to $((\mathcal{B}^d)^*, <_{rpo})$. □

Corollary 18. *$(\mathcal{B}, <)$ is a well-ordering and otyp$(<) = \varepsilon_0$.*

Lemma 19. *Let d, k be positive natural numbers. Then*

$$otyp(< \restriction \mathcal{B}^{d,k}) = \begin{cases} k & \text{if } d = 1 \\ \omega_{2(d-1)}(k-1) & \text{otherwise}. \end{cases}$$

Proof. Similar to Corollary 17. □

Theorem 20. *otyp$(\mathcal{B}_n, < \restriction \mathcal{B}_n) = \omega_{n+1}$ for any positive natural number n.*

Finally, Theorem 11 and Theorem 20 imply the main claim.

Theorem 21. *$o(\mathcal{B}_n, \trianglelefteq \restriction \mathcal{B}_n) = otyp(< \restriction \mathcal{B}_n) = \omega_{n+1}$ for any positive natural number n.*

References

1. de Jongh, D.H.J., Parikh, R.: Well-partial orderings and hierarchies. Nederl. Akad. Wetensch. Proc. Ser. A 80=Indag. Math 39(3), 195–207 (1977)
2. Schmidt, D.: Well-Partial Orderings and Their Maximal order Types. Habilitationsschrift, Heidelberg (1979)
3. Higman, G.: Ordering by divisibility in abstract algebras. In: Proc. London Math. Soc 2(3), 326–336 (1952)
4. Hasegawa, R.: Well-ordering of algebras and Kruskal's theorem. In: Sato, M., Hagiya, M., Jones, N.D. (eds.) Logic, Language and Computation. LNCS, vol. 792, pp. 133–172. Springer, Heidelberg (1994)
5. Feferman, S.: Systems of predicative analysis. J. Symbolic Logic 29, 1–30 (1964)
6. Feferman, S.: Systems of predicative analysis. II. Representations of ordinals. J. Symbolic Logic 33, 193–220 (1968)
7. Schütte, K.: Predicative well-orderings. In: Formal Systems and Recursive Functions. In: Proc. Eighth Logic Colloq, Oxford, 1963. North-Holland pp. 280–303 (1965)

8. Schütte, K.: Proof theory. In: Translated from the revised German edition by Crossley, J. N. Grundlehren der Mathematischen Wissenschaften, Band, Springer, Heidelberg (1977)
9. Weiermann, A.: Phase transition thresholds for some Friedman-style independence results (to appear)
10. Lee, G.: Slowly-well-orderedness of binary trees in Peano arithmetic and its fragments. (Preprint)
11. Rathjen, M., Weiermann, A.: Proof-theoretic investigations on Kruskal's theorem. Ann. Pure Appl. Logic 60(1), 49–88 (1993)
12. Dershowitz, N.: Orderings for term-rewriting systems. Theoret. Comput. Sci. 17(3), 279–301 (1982)
13. Touzet, H.: A characterisation of multiply recursive functions with Higman's lemma. Inform. and Comput, RTA '99 (Trento) 178(2), 534–544 (2002)

A Weakly 2-Random Set That Is Not Generalized Low*

Andrew Lewis[1], Antonio Montalbán[2], and André Nies[3]

[1] University of Siena, Italy
[2] University of Wellington, New Zealand
[3] University of Auckland, New Zealand

Abstract. A guiding question in the study of weak 2-randomness is whether weak 2-randomness is closer to 1-randomness, or closer to 2-randomness. Recent research indicates that the first alternative holds. We add further evidence in this direction by showing that, in contrast to the case for 2-randomness, a weakly 2-random set can fail to be generalized low.

1 Introduction

Martin-Löf randomness, also called 1-randomness, is a central algorithmic randomness notion for subsets of $\mathbb{N}$ (see for instance [6]). We say that a set is 2-random if it is 1-random relative to $\emptyset'$. 2-randomness was first studied by Kurtz [5], and more recently in [8], where a characterization was given using the plain Kolmogorov complexity of the initial segments. Kurtz also considered weak 2-randomness, an interesting notion lying strictly in between 1-randomness and 2-randomness: a set is weakly 2-random if it is not in any Π^0_2 null class. Part of the attractiveness of this notion stems from its conceptually very simple definition. It is also the weakest randomness notion that eliminates some of the intuitively 'non-random' features that 1-random sets can have, such as being left-c.e. On the other hand, contrary to the case for 1- and 2-randomness, there is no universal test for weak 2-randomness (i.e., there is no largest Π^0_2 null class).

The separation from 1-randomness and the separation from 2-randomness both hold up to Turing degree ([4]). For the first separation, note that a 1-random set can be Δ^0_2 (even left-c.e.) while a weakly 2-random set forms a minimal pair with $\emptyset'$ [2]. For the second separation, if a set has hyperimmune free degree then it is 1-random iff it is weakly 2-random ([8], by an observation of Yu), and such sets exist by the hyperimmune free basis theorem. On the other

* The first author was supported by Marie-Curie Fellowship MEIF-CT-2005-023657 and partially supported by the NSFC Grand International Joint Project no. 60310213, New Directions in the Theory and Applications of Models of Computation. The second author was partially supported by NSF Grant DMS-0600824 and by the the Marsden Foundation of New Zealand, via a postdoctoral fellowship. The third author was partially supported by the Marsden Fund of New Zealand, grant no. 03-UOA-130.

hand, each 2-random set is of hyperimmune degree (Kurtz [5], or see the more recent, shorter proof in [8]).

A guiding question in the study of weak 2-randomness is whether weak 2-randomness is closer to 1-randomness, or closer to 2-randomness. Recent research indicates that it is closer to 1-randomness:

(a) a set is low for weak 2-randomness iff it is low for 1-randomness ([2,6], also unpublished work of J. Miller);
(b) by a result of Hirschfeldt and Miller, see the last Section of [7], a 1-random set Z is already weakly 2-random unless there is a noncomputable set $A \leq_T \emptyset', Z$. (If so, the set A can actually be chosen to be c.e., and if Z is not Turing above $\emptyset'$, then such an A is K-trivial, by [3].) In a sense, then, there are few 1-random sets that are not weakly 2-random.

By [8], each 2-random set is generalized low (and even low for Ω, a stronger property). In our main result, we add another piece of evidence that weak 2-randomness is closer to 1-randomness, by showing that *a weakly 2-random set Z can fail to be generalized low.* In fact, one can choose Z of hyperimmune free degree, whence it suffices to make Z 1-random to obtain weak 2-randomness.

We thank one of the anonymous referees for supplying a proof that weak 2-randomness can also be separated from 2-randomness within the sets of hyperimmune degree. In other words, there is a weakly 2-random set D of hyperimmune degree such that no set of the same Turing degree is 2-random. To see this, take any weakly 2-random set A that is not 2-random (for instance, choose a 1-random set A of hyperimmune-free degree). A forms a minimal pair with $\emptyset'$ by the comments above, so the class $\mathcal{C} = \{B : A \oplus B \text{ forms a minimal pair with } \emptyset'\}$ has measure 1. Thus there exists a set B in $\mathcal{C}$ that is 2-random relative to A, and hence of hyperimmune degree. By van Lambalgen's Theorem (see for instance [6]), $D = A \oplus B$ is 1-random. Since $D \in \mathcal{C}$, D is weakly 2-random. Finally, no set Turing equivalent to D is 2-random, otherwise A would be 2-random, being 1-random and Turing below a 2-random set [8].

2 The Main Result

We consider the Cantor space 2^ω and denote the standard measure on 2^ω by μ. $\Lambda \subseteq 2^{<\omega}$ is said to be downward closed if, whenever $\tau \in \Lambda$, all initial segments of τ are in this set. Given any $\Lambda \subseteq 2^{<\omega}$ we denote by $[\Lambda]$ the set of infinite paths through Λ i.e. those sets A such that there exist an infinite number of initial segments of A in Λ. $\mathcal{P} \subseteq 2^\omega$ is a Π^0_1 class if there exists a downward closed computable $\Lambda \subseteq 2^{<\omega}$ such that $\mathcal{P} = [\Lambda]$. Given $\tau \in 2^{<\omega}$ we let $[\tau]$ denote the set of infinite sequences extending τ. A Martin-Löf test $\{U_e\}_{e\in\omega}$ is a uniformly c.e. sequence of sets of strings such that for all $e \in \mathbb{N}$, $\mu(U_e) < 2^{-(e+1)}$ and $U_e \supseteq U_{e+1}$. A set A is called 1-random if for every Martin-Löf test $\{U_e\}_{e\in\omega}$ we have $A \notin \cap_e U_e$. We say that A is hyperimmune free if every function computable in A is majorized by a computable function. We say A is generalized low if $A' \equiv_T A \oplus \emptyset'$. We let $\{\Psi_e\}_{e\in\omega}$ be an effective listing of the Turing functionals.

Theorem 1. *There exists Z which is 1-random, hyperimmune free (and therefore weakly 2-random) and which is not generalized low.*

Proof. In order to construct Z we shall use forcing with Π^0_1 classes of positive measure. Initially we suppose given $\mathcal{P}_0 = [\Lambda_0]$ of positive measure and which contains only 1-random sets. At stages $2n$, given $\mathcal{P}_{2n} = [\Lambda_{2n}]$, we define $\mathcal{P}_{2n+1} \subseteq \mathcal{P}_{2n}$ of positive measure such that if $Z \in \mathcal{P}_{2n+1}$ then:

$\mathcal{H}_n$: if Ψ^Z_n is total then it is majorized by some computable f.

At stages $2n+1$ we define $\mathcal{P}_{2n+2} \subseteq \mathcal{P}_{2n+1}$ of positive measure such that if $Z \in \mathcal{P}_{2n+2}$ then:

$\mathcal{J}_n$: $\Psi^{Z \oplus \emptyset'}_n \neq Z'$.

Since the Cantor space is compact $\bigcap_n \mathcal{P}_n$ will be non-empty. During the course of the construction we will make use of the following lemma which was originally proved by Kučera and which is frequently very useful in dealing with Π^0_1 classes of positive measure. For a very simple proof we refer the reader to [1].

Lemma 1 (Kučera). *Given any Π^0_1 class $\mathcal{P}$ of positive measure there exists a Π^0_1 class of positive measure $\mathcal{K}(\mathcal{P}) \subseteq \mathcal{P}$ such that the intersection of $\mathcal{K}(\mathcal{P})$ with any Π^0_1 class is either empty or of positive measure.*

2.1 Stage $2n$

First we define $\mathcal{Q} = \mathcal{K}(\mathcal{P}_{2n})$ and let Λ be such that $\mathcal{Q} = [\Lambda]$. Then we ask, does there exist $Z \in \mathcal{Q}$ and $m \in \omega$ such that $\Psi^Z_n(m)\uparrow$? In the case that this question receives a positive answer we can just define $\mathcal{P}_{2n+1}$ to be the set of all $Z \in \mathcal{Q}$ such that $\Phi^Z_n(m)\uparrow$. Since $\mathcal{P}_{2n+1}$ has non-empty intersection with $\mathcal{Q}$ it is of positive measure. If this question receives a negative response then we can define $\mathcal{P}_{2n+1} = \mathcal{Q}$. A standard argument suffices to show that for any $Z \in \mathcal{Q}$ there exists a computable function f which majorizes Φ^Z_n. In order to define $f(m)$ for any $m \in \omega$ find s such that $\Phi^\tau_n(m)\downarrow$ for all $\tau \in \Lambda$ of length s and then define $f(m)$ to be greater than all such values.

2.2 Stage $2n+1$

The activity at stage $2n+1$ is divided into several steps.

Step (1). We define $\mathcal{Q}_0 = \mathcal{K}(\mathcal{P}_{2n+1})$ and let $\mathcal{Q}_0 = [\Lambda]$.

Step (2). We define $\mathcal{Q}_1 \subseteq \mathcal{Q}_0$ which is of positive measure and such that for each τ which is an initial segment of some $Z \in \mathcal{Q}_1$ there exists $\tau' \supseteq \tau$ with $[\tau'] \cap \mathcal{Q}_1 = \emptyset$ and $[\tau'] \cap \mathcal{Q}_0$ is of positive measure.

It is clear that such $\mathcal{Q}_1$ exists, but one simple way of defining such a class is to proceed as follows. Let the finite binary strings be ordered according to length and then from left to right. Let i be such that $\Sigma_{j \geq i} 2^{-j} < \mu(\mathcal{Q}_0)$. For each string $\tau \in \Lambda$ let j be such that τ is the j^{th} string according to the ordering specified above. Begin by removing from $\mathcal{Q}_0$ all strings extending the first string in Λ extending τ of length $i+j$, τ' say, if there exists such. If it subsequently turns out that $[\tau'] \cap \mathcal{Q}_0 = \emptyset$ then proceed to remove from $\mathcal{Q}_0$ all strings extending the second string in Λ extending τ of length $i+j$ if there exists such, and so on.

Step (3). Let e be such that, for all Z, m:

$$\Psi_e^Z(m) \downarrow = 0 \text{ if } Z \notin \mathcal{Q}_1 \text{ and } \Psi_e^Z(m) \uparrow \text{ otherwise.}$$

We ask whether there exists τ an initial segment of some $Z \in \mathcal{Q}_1$ and $\sigma \subset \emptyset'$ such that $\Psi_n^{\tau \oplus \sigma}(e) \downarrow = 0$. If not then we can define $\mathcal{P}_{2n+2} = \mathcal{Q}_1$. For all $Z \in \mathcal{P}_{2n+2}$, $Z'(e) = 0 \neq \Psi_n^{Z \oplus \emptyset'}(e)$, so requirement $\mathcal{J}_n$ is satisfied. Otherwise there exists $\tau' \supseteq \tau$ such that $[\tau'] \cap \mathcal{Q}_1 = \emptyset$ and $[\tau'] \cap \mathcal{Q}_0$ is of positive measure. In this case we can define $\mathcal{P}_{2n+2} = [\tau'] \cap \mathcal{Q}_0$. For all $Z \in \mathcal{P}_{2n+2}$, $Z'(e) = 1 \neq \Psi_n^{Z \oplus \emptyset'}(e) = 0$, so requirement $\mathcal{J}_n$ is satisfied.

References

1. Downey, R., Miller, J.: A basis theorem for Π_1^0 classes of positive measure and jump inversion for random reals. In: Proc. Amer. Math. Soc. vol. 134, pp. 283–288 (2006)
2. Downey, R., Nies, A., Weber, R., Yu, L.: Lowness and Π_2^0 nullsets. J. Symbolic Logic 71, 1044–1052 (2006)
3. Hirschfeldt, D., Nies, A., Stephan, F.: Using random sets as oracles. (to appear)
4. Kautz, S.: Degrees of random sets. Ph.D. Dissertation, Cornell University (1991)
5. Kurtz, S.: Randomness and genericity in the degrees of unsolvability. In: Ph.D. Dissertation, University of Illinois, Urbana (1981)
6. Nies, A.: Computability and Randomness. Oxford University Press. To appear in the series Oxford Logic Guides
7. Nies, A.: Eliminating concepts. In: Proceedings of the IMS workshop on computational prospects of infinity (to appear)
8. Nies, A., Stephan, F., Terwijn, S.A.: Randomness, relativization and Turing degrees. J. Symbolic Logic 70(2), 515–535 (2005)

Speed-Up Theorems in Type-2 Computation

Chung-Chih Li

School of Information Technology
Illinois State University, Normal, IL 61790-5150, USA
cli2@ilstu.edu

Abstract. A classic result known as the speed-up theorem in machine-independent complexity theory shows that there exist some computable functions that do not have best programs for them [2,3]. In this paper we lift this result into type-2 computation under the notion of our type-2 complexity theory depicted in [15,13,14]. While the speed-up phenomenon is essentially inherited from type-1 computation, we cannot directly apply the original proof to our type-2 speed-up theorem because the oracle queries can interfere the speed of the programs and hence the cancellation strategy used in the original proof is no longer correct at type-2. We also argue that a type-2 analog of the operator speed-up theorem [16] does not hold, which suggests that this curious phenomenon disappears in higher-typed computation beyond type-2.

1 Introduction

Speed-up phenomena have been extensively studied by mathematicians for more than a half century, which was first remarked by Gödel [8] in the context of theorem proving.[1] In [2,3] Blum re-discovered the speed-up theorem in terms of computable functions and his complexity measures. The theorem asserts that the best program does not always exist for some computable functions. In order to state the theorem precisely, we first fix some notations and conventions. By computable we mean Turing machine computable. A function is said to be recursive if it is total and computable. Let φ_e denote the function computed by the e^{th} Turing machine and Φ_e denote the cost function associated to the e^{th} Turing machine. More precisely, let $\langle\varphi_i\rangle_{i\in\mathbf{N}}$ be an *acceptable programming system* [17] and $\langle\Phi_i\rangle_{i\in\mathbf{N}}$ be a *complexity measure* [2] associated to $\langle\varphi_i\rangle_{i\in\mathbf{N}}$, where $\mathbf{N}$ is the set of natural numbers. The standard asymptotic notion, $\overset{\infty}{\forall}$, is read as *for all but finitely many*[2]. We state the original speed-up theorem as follows:

Theorem 1 (The Speed-up Theorem [2,3]). *For any recursive function r, there exists a recursive function f such that*

$$(\forall\, i : \varphi_i = f)\ (\exists j : \varphi_j = f)\ (\overset{\infty}{\forall}\, x)\ \left[r(\Phi_j(x)) \le \Phi_i(x)\right].$$

[1] The original remarks were translated in [7], pages 82-83. More discussion about the relation between the computational speed-up phenomena and Gödel's speed-up results in logic can be found in [21].

[2] The negation of "*for all but finitely many*" is "*exist infinitely many*" denoted by $\overset{\infty}{\exists}$.

We say that function f in the theorem above is *r-speedable*. We revise the original proof of this theorem so that it can be easily modified for our type-2 speed-up theorem (see [12] for details). More original proofs can be found in [2,3,21,6,22,4,19]. Many variations of the speed-up theorem have been proven since Blum's [2,3]. We are interested in Meyer and Fischer's operator speed-up theorem [16] where the speed-up factor r, a recursive function, is strengthened to an effective operator Θ as follows:

Theorem 2 (The Operator Speed-up Theorem [16]). *For any total effective operator Θ, there is a recursive function f that can be uniformly constructed such that*

$$\forall\, i : \varphi_i = f \;\exists j : \varphi_j = f \;\overset{\infty}{\forall}\, x[\Theta(\Phi_j)(x) \leq \Phi_i(x)].$$

Our goal in the present paper is to lift these two speed-up theorems to type-2 computation. We obtain a type-2 analog of Theorem 1. However, Theorem 2 fails to hold in the context of type-2 computation, which suggests that there always exist best programs in higher-typed computation beyond type-2.

In the next section, we briefly introduce the current status of type-2 complexity theory and describe some necessary preliminaries. These paragraphs are perforce brief and superficial due to space constraints. Since the speed-up theorem is for the most part independent from the other parts of the theory, our coverage will be limited to the topics pertinent to our results. More details can be found in [15,13,14].

2 Conventions and Type-2 Complexity Theory

We consider natural numbers as type-0 objects and functions over natural numbers as type-1 objects. Type-2 objects are called *functionals* that take as inputs and produce as outputs type-0 or type-1 objects. By convention, we consider objects of lower type as special cases of higher type, and thus, type-0 $\subset$ type-1 $\subset$ type-2. Without loss of generality we restrict type-2 functionals to our standard type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, where $\mathcal{T}$ is the set of total functions and $\rightharpoonup$ means possibly partial. Note that $f \in \mathcal{T}$ may not be computable. For $n \in \mathbf{N}$, $|n|$ denotes the length of the binary bit string representing n. For type-2 computation we use the Oracle Turing Machine (OTM) as our standard computing formalism. An OTM is a Turing machine equipped with a function oracle. Before an OTM begins to run, the type-1 argument should be presented to the OTM as an oracle. In addition to the standard I/O tape for type-0 input/output and intermediate working space, an OTM has two extra tapes – one is for oracle queries and the other one is for the answers to the queries. During the course of the computation, the OTM may enter a special state called query-state. In this state the oracle will read the query left on the query-tape and prepare its answer on the the answer-tape for the OTM to read. All this will be done at no cost to the OTM. However, the OTM has to prepare the queries and read their answers at its own computational cost. We also fix a programming system $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$ associated with some complexity measure $\langle \widehat{\Phi}_i \rangle_{i \in \mathbf{N}}$ for OTM. By convention, we take the number of steps as our time complexity measure, i.e., the number of times an OTM moves its read/write heads. Also, we use $\widehat{M}_e$ to denote the OTM with index e and $\widehat{\varphi}_e$ is the functional computed by $\widehat{M}_e$. Following these conventions, Seth [20] adapted Hartmanis and Stearns's notion [9] to define type-2 complexity classes. He proposed two alternatives:

1. Given recursive $t : \mathbf{N} \to \mathbf{N}$, let $DTIME(t)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME(t)$, F is total and there is an OTM $\widehat{M}_e$ that computes F and, on *every* $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{M}_e$ halts within $t(m)$ steps, where $m = |\mathsf{max}(\{x\} \cup Q)|$ and Q is the set of all answers returned from the oracle during the course of the computation.
2. Given computable functional $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, let $DTIME(H)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME(H)$, F is total and there is an OTM $\widehat{M}_e$ that computes F and, on *every* $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{M}_e$ halts within $H(f, x)$ steps.

The key idea behind Seth's complexity classes is directly lifted from [9]. The same machine characterization idea can also be found in other works such as Kapron and Cook's [10] and Royer's [18]. In Seth's first definition stated above, the resource bound is determined by the sizes of oracle answers; but the set Q in the definition of $DTIME(t)$ in general is not computable and hence can't be available before the computation halts, if ever. Alternatively, we may update the bound dynamically upon each answer returned from the oracle during the course of the computation. But if we do so, there is no guarantee that a clocked OTM must be total. For example, Cook's POTM [5] is an OTM bounded by a polynomial in this manner but a POTM may run forever. Kapron and Cook's proposed their remedies in the context of feasible functionals and gave a very neat characterizations of type-2 Basic Feasible Functionals (BFF) in [10], where the so-called second-ordered polynomial is used as the bound. In [13,14] we adapted all these ideas and extended the second-ordered polynomial to a general type-2 computable functional to have the following complexity class:

$$DTIME(H) = \{F \bigm| \exists e[\widehat{\varphi}_e = F \text{ and } \widehat{\Phi}_e \leq^*_{\mathbf{2}} H]\}. \tag{1}$$

The relation, $\leq^*_{\mathbf{2}}$, used above will be defined in Definition 3, which is crucial to our works. Along the lines of the classical complexity theory initiated by a series of seminal papers [9,2,3], our previous results in [15,13,14] show that the complexity theory at type-2 does not parallel to its type-1 counterpart. To begin with, we defined $\leq^*_{\mathbf{2}}$ with a workable and reasonable type-2 analog of asymptotic notion. We equated our notion of *finitely many* at type-2 to the *compact sets* in some Baire-like topology [1] that was relatively defined by the concerned functionals. As there is no type-2 equivalent of Church-Turing thesis, the compactness in our definition is the key to computability of our construction. In [14] we examined some alternative clocking schemes for OTM and defined a class of limit functionals determined by some computable functions to serve as type-2 time bounds. With these type-2 time bounds, we were able to define an explicit type-2 complexity class similar to ([1]) for a general type-2 complexity theory. Unlike many other complexity theorems, the speed-up theorems do not need a precisely defined complexity classes. We thus skip details regarding our explicit type-2 complexity classes. However, the asymptotic notion is still indispensable in the present paper. We formalize the notion as follows. Let $\mathcal{F}$ denote the set of *finite* domain functions over natural numbers, i.e., $\sigma \in \mathcal{F}$ iff $\mathsf{dom}(\sigma)$ is finite. Given $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, let $F(f,x) \downarrow= y$ denote the case that F is defined at (f,x) and its value is y. For $\sigma \in \mathcal{F}$ and $f \in \mathcal{F} \cup \mathcal{T}$, let $\sigma \subset f$ denote the case that f is an extension of σ.

Definition 1. *Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$. We say that (σ, x) is a locking fragment of F if and only if*

$$\exists y \in \mathbf{N}\ \forall f \in \mathcal{T}\left[\sigma \subset f \Rightarrow F(f,x) \downarrow= y\right].$$

Also, we say that (σ, x) is a *minimal locking fragment* of F if (σ, x) is a locking fragment of F and, for every $\tau \in \mathcal{F}$ with $\tau \subset \sigma$, (τ, x) is not a locking fragment of F. Clearly, if F is total and computable, then for every $(f,x) \in \mathcal{T} \times \mathbf{N}$, there must exist a unique $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that (σ, x) is a minimal locking fragment of F. It is also clear that, in general, whether of not (σ, x) is a minimal locking fragment of F cannot be effectively decided. For any $\sigma \in \mathcal{F}$, let $(\!(\sigma)\!)$ be the set of total extensions of σ, i.e., $(\!(\sigma)\!) = \{f \in \mathcal{T} \mid \sigma \subset f\}$. Also, if $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, let $(\!(\sigma, x)\!) = \{(f,x) \mid f \in (\!(\sigma)\!)\}$. We observe that, $(\!(\sigma_1)\!) \cap (\!(\sigma_2)\!) = (\!(\sigma_1 \cup \sigma_2)\!)$ if σ_1 and σ_2 are consistent; otherwise, $(\!(\sigma_1)\!) \cap (\!(\sigma_2)\!) = \emptyset$. The union operation $(\!(\sigma_1)\!) \cup (\!(\sigma_2)\!)$ is conventional. Given any $f, g \in \mathcal{T}$, it is clear that, if $f \neq g$, then there exist $\sigma \subset f, \tau \subset g$, and $k \in \mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau)$ such that $\sigma(k) \neq \tau(k)$. In stead of taking every $(\!(\sigma, x)\!)$ with $\sigma \in \mathcal{F}$ as the basic open set[3], we consider only those that are related to the concerned functionals as follows.

Definition 2. *Given any continuous functionals, F_1 and F_2, let $\mathbb{T}(F_1, F_2)$ denote the topology induced from $\mathbb{T} \times \mathbb{N}$ by F_1 and F_2, where the basic open sets are defined as follows: $(\!(\sigma, a)\!)$ is a basic open set of $\mathbb{T}(F_1, F_2)$ if and only if, for some $(f, a) \in \mathcal{T} \times \mathbf{N}$, (σ_1, a) and (σ_2, a) are the minimal locking fragments of F_1 and F_2, respectively, and $(\!(\sigma, a)\!) = (\!(\sigma_1, a)\!) \cap (\!(\sigma_2, a)\!)$.*

Note that, in the definition above, since $(\!(\sigma, a)\!) = (\!(\sigma_1, a)\!) \cap (\!(\sigma_2, a)\!) = (\!(\sigma_1 \cup \sigma_2, a)\!)$ we have that if $(\!(\sigma, a)\!)$ is a basic open set of $\mathbb{T}(F_1, F_2)$, then (σ, a) must be a locking fragment of both F_1 and F_2. Let $X_{[F_1 \leq F_2]} \subseteq \mathcal{T} \times \mathbf{N}$ denote the set $\{(f,a) \mid F_1(f,a) \leq F_2(f,a)\}$. $X_{[F_1 > F_2]}$ is simply the complement of $X_{[F_1 \leq F_2]}$ called the *exception set* of $F_1 \leq F_2$. Now, we are in a position to define our type-2 almost-everywhere relation.

Definition 3. *Let $F_1, F_2 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. Define*

$$F_1 \leq_{\mathbf{2}}^{*} F_2 \quad \textit{if and only if} \quad X_{[F_1 \leq F_2]} \quad \textit{is co-compact in} \quad \mathbb{T}(F_1, F_2).$$

[3] This will form the product topology $\mathbb{T} \times \mathbb{N}$, where $\mathbb{T}$ is the Baire topology and $\mathbb{N}$ the discrete topology on $\mathbf{N}$.

Using the same idea of compactness in Definition 3, two modified quantifiers, for all but finitely many and exist infinitely many, can be understood in type-2 context as follows: For continuous functionals $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, we have $\overset{\infty}{\forall}_2\, (f,x)[F(f,x) \leq G(f,x)]$ if and only if $\{(f,x) \mid F(f,x) \leq G(f,x)\}$ is compact in $\mathbb{T}(F,G)$. Similarly, we say that $\overset{\infty}{\exists}_2\, (f,x)[F(f,x) \leq G(f,x)]$ if and only if $\{(f,x) \mid F(f,x) \leq G(f,x)\}$ is not compact in $\mathbb{T}(F,G)$. One can verify that

$$F \leq_2^* G \Longleftrightarrow \overset{\infty}{\forall}_2\, (f,x)[F(f,x) \leq G(f,x)] \Longleftrightarrow \neg \overset{\infty}{\exists}_2\, (f,x)[F(f,x) > G(f,x)].$$

When the concerned functionals F and G are clear from the context, we simply read $\overset{\infty}{\forall}_2\, (f,x)$ as *"for all (f,x) except those in a compact set such that"*, and $\overset{\infty}{\exists}_2\, (f,x)$ as *"there exists a noncompact set such that, for all (f,x) in the set"*, where *compact* is understood as $\mathbb{T}(F,G)$-*compact.*

3 Lifting Speed-Up Theorems to Type-2

Since type-1 computations are just a special case of type-2 computations, the speedable function constructed for the original speed-up theorem can be seen as a type-2 functional that just does not make any oracle queries. In other words, as long as the concerned complexity measure satisfies Blum's two axioms, the proof of the original speed-up theorem should remain valid at type-2. Clearly, our standard complexity measure $\langle \widehat{\Phi}_i \rangle$, the number of steps the OTM performs, does satisfy Blum's two axioms. However, we observe that oracle queries in type-2 computation have introduced some difficulties when we attempt a direct translation of the original proof. Recall that the original construction of the speedable function is based on the cancellation on some programs when their run times fall into certain ranges. When we directly lift the construction to type-2, we note that there are cases in which the oracle queries may be used to slow down or speed up the computation in such a way the programs can escape from being canceled. Note that the proofs of the Union Theorem and Gap Theorem do not involve the cancellation but directly construct time bounds and let the definition of the complexity class take care of the rest. Unfortunately, one can easily show that there are functionals that always make unnecessary oracle queries. Consider functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ defined by,

$$F(f,x) = \begin{cases} f(0)+1 & \text{if } \varphi_x(x)\downarrow \text{ in } f(0) \text{ steps;} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Clearly, F is computable and total. Fix any a such that, $\varphi_a(a)\uparrow$. Then, on input (f,a), the value of $f(0)$ only affects the speed of computing $F(f,a)$. Thus, $F(f,a) = 0$ for any $f \in \mathcal{T}$, and hence $(\emptyset, a)$ is the minimal locking fragment of F on (f,a). That means any queries made during the computation of F on (f,a) are unnecessary. Thus, if there were an OTM that would not make any unnecessary queries for F, one could modify such OTM to solve the halting problem, which is impossible. However, the answer to the query does affect the speed of the machine to halt. The smaller the value of $f(0)$ is, the sooner the computation halts. In fact, it is easy to construct computable functionals that make unnecessary queries on all inputs, and moreover, the number of unnecessary queries can be arbitrarily large. Such kind of unnecessary but speed-affecting queries is the problem for us to get around in lifting the speed-up theorems into type-2.

It is clear that our $\widehat{\varphi}$-programming system for OTM can be used to code the entire class of type-1 computable functions. Thus, the speedable function constructed in the original speed-up theorem can be coded in our $\widehat{\varphi}$-programming system. To that speedable function, any queries made during the course of computation are unnecessary. However, as we have seen, unnecessary queries may affect the computational time. Therefore, we cannot simply cancel those $\widehat{\varphi}$-programs that make oracle queries. Moreover, if we intuitively enumerate all possible queries in our construction, we face another difficulty in trying to make our speedable functional total, because we cannot decide whether a query is necessary or not; thus our construction will tend to be fooled by infinitely many unnecessary queries and fail to converge. Fortunately, we will see that our notion of $\leq_{\mathbf{2}}^{*}$ defined by Definition 3 based on the compactness of the relative topologies (Definition 2) resolves this problem automatically and easily.

4 Type-2 Speed-Up Theorems

Type-2 speed-up theorems vary with the nature of the speed-up factors that can be either type-1 or type-2. For type-3 speed-up factors, the theorem becomes a type-2 analog of the operator speed-up theorem, and we will argue that there is no such theorem. From Theorem 2 (the operator speed-up theorem) we immediately have the following corollary, in which we replace the operator $\Theta : \mathcal{T} \to \mathcal{T}$ by a functional $R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

Corollary 1. *For any computable functional $R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, there exists a recursive function f such that,*

$$\forall\, i : \varphi_i = f \;\exists j : \varphi_j = f \;\; \overset{\infty}{\forall}\, x[R(\Phi_j, x) \leq \Phi_i(x)].$$

However, this corollary is of no interest. Our goal is to construct a type-2 speedable functional using our programming system $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$ for OTM. We are interested in the following two theorems:

Theorem 3. *For any recursive function $r : \mathbf{N} \to \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that,*

$$\forall\, i : \widehat{\varphi}_i = F_r \;\exists j : \widehat{\varphi}_j = F_r \;\; [r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i].$$

Theorem 4 (Type-2 Speed-up Theorem). *For any computable functional $R : \mathcal{T} \times \mathbf{N} \times \mathbf{N} \to \mathbf{N}$, there exists a computable functional $F_R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that,*

$$\forall\, i : \widehat{\varphi}_i = F_R \;\exists j : \widehat{\varphi}_j = F_R \;\; [\lambda f, x.R(f, x, \widehat{\Phi}_j(f, x)) \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i].$$

Theorem 3 and Theorem 4 are obtained by lifting Theorem 1 and Theorem 2, respectively, into type-2 compuation. Note that since Theorem 3 is a special case of Theorem 4, we rather consider Theorem 4 as our type-2 speed-up theorem. Instead of proving Theorem 4 directly, we prove a simpler result of Theorem 3. The idea can be applied to prove Theorem 4.

Consider Theorem 3. We observe that $F_r = \widehat{\varphi}_i = \widehat{\varphi}_j$. By Definition 3, the relative topology for the type-2 relation, $r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^* \widehat{\Phi}_i$, is

$$\mathbb{T}(r \circ \widehat{\Phi}_j, \widehat{\Phi}_i) = \mathbb{T}(r \circ \widehat{\varphi}_j, \widehat{\varphi}_j) = \mathbb{T}(\widehat{\varphi}_j) = \mathbb{T}(\widehat{\varphi}_i) = \mathbb{T}(F_r).$$

Thus, if we construct F_r with $(\emptyset, x)$ as its minimal locking fragment for every $x \in \mathbf{N}$, then the relative topology for $\leq_{\mathbf{2}}^*$ in the theorem is the coarsest one, i.e., the topology with basic open sets: $(\!(\emptyset, 0)\!), (\!(\emptyset, 1)\!), \ldots$. Our idea is that: given any $S \subset \mathcal{T} \times \mathbf{N}$ with S being noncompact in the topology $\mathbb{T}(F_r)$, we then must have that the type-0 component of the elements of S has infinitely many different values. If a $\widehat{\varphi}$-program i needs to be canceled, we thus have infinitely many chances to do so on some type-0 inputs. We can therefore ignore the effects of the type-1 input in the computation. In other words, it is not necessary to introduce another parameter for the type-1 argument when defining the cancellation sets.

This wishful thinking, however, is problematic in the corresponding type-2 pseudo-speed-up theorem. Because, for every $\widehat{\varphi}$-program i for F_r, its *pseudo* sped-up version, $\widehat{\varphi}$-program j, does not exactly compute $\widehat{\varphi}_i$ on some finitely many type-0 inputs, and hence $\widehat{\varphi}_i$ and $\widehat{\varphi}_j$ may define two different topologies. Thus, if we ignore the effect of the type-1 argument, the almost everywhere relation $r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^* \widehat{\Phi}_i$ may fail in topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$. To fix this problem, we introduce a weaker type-2 pseudo-speed-up theorem, in which the compactness is not considered. The theorem in weaker in a sense that we do not use the type-2 almost everywhere relation. Nevertheless, this weaker type-2 pseudo-speed-up theorem will be sufficient for our proof of Theorem 3.

Theorem 5 (Type-2 Pseudo-Speed-up Theorem). *For any recursive function function $r : \mathbf{N} \to \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, for every $\widehat{\varphi}$-program i for F_r, there is another $\widehat{\varphi}$-program j such that,*

$$\overset{\infty}{\forall}\, x \in \mathbf{N}\ \forall\, f \in \mathcal{T}[(\widehat{\varphi}_j(f,x) = F_r(f,x)) \ \wedge\ (\widehat{\Phi}_i(f,x) > r \circ \widehat{\Phi}_j(f,x))].$$

We shall omit detailed proof of this pseudo-speed-up theorem due to space constraints and refer readers to [12] for details.

Proof of Theorem 3: According to the construction of $\widehat{\varphi}_e$ in Theorem 5, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(0, f, x) = \widehat{\varphi}_e(0, f_0, x)$, where $f_0 = \lambda x.0$. It follows that $(\emptyset, x)$ is the minimal locking fragment of $\widehat{\varphi}_{s(e,0)}$ on every $(f, x) \in \mathcal{T} \times \mathbf{N}$. Let $\widehat{\varphi}_i = \widehat{\varphi}_{s(e,0)}$ and $j = s(e, i+1)$. Note that $\widehat{\varphi}_i =_{\mathbf{2}}^* \widehat{\varphi}_j$ and $r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^* \widehat{\Phi}_i$ does not hold in general because $(\!(\emptyset, x)\!)$ may not be a basic open set for some x. Consider the following exception set

$$E = \left\{(f, x) \;\middle|\; \widehat{\varphi}_i(f, x) \neq \widehat{\varphi}_j(f, x)\right\}.$$

Although E may not be compact in topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$, $\{x|(f,x) \in E\}$ must be finite. Thus, we can have a patched $\widehat{\varphi}$-program j' such that the program will search a look-up table if the type-0 argument is in $\{x|(f,x) \in E\}$. In such a way, the type-1 input will not affect the result, and hence the minimal locking fragment becomes $(\emptyset, x)$. On the other hand, if type-0 argument $x \notin \{x|(f,x) \in E\}$, then $\widehat{\varphi}$-program j' starts running $\widehat{\varphi}$-program j. Similarly, consider

$$E' = \left\{(f,x) \mid r \circ \widehat{\Phi}_j(f,x) > \widehat{\Phi}_i(f,x)\right\}.$$

Set $\{x|(f,x) \in E'\}$ is finite. Also, consider the patched $\widehat{\varphi}$-program, j'. We have

$$E'' = \left\{(f,x) \mid r \circ \widehat{\Phi}_{j'}(f,x) > \widehat{\Phi}_i(f,x)\right\},$$

and $\{x|(f,x) \in E''\}$ is finite. Therefore, E'' is compact in $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_{j'})$, because $\widehat{\varphi}_i = \widehat{\varphi}_{j'}$ and, for every $x \in \mathbf{N}$, $(\emptyset, x)$ is the only basic open set in $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_{j'})$.

Finally, we shall discuss the case that there may exist some best $\widehat{\varphi}$-program for $\widehat{\varphi}_{s(e,0)}$ using some unnecessary queries to escape from being canceled. This is possible because we replace the actual type-1 input by f_0 for every $\widehat{\varphi}$-program, and hence we do not know the program's behavior on actual $f \in \mathcal{T}$. Clearly, by Claim 6 in the proof of Theorem 5, this problem can be ignored, because any program that will make any query on some inputs does not compute our speedable functional. This completes the proof of Theorem 3. $\square$

5 Type-2 Operator Anti-Speed-Up Theorem

In the previous section we established two speed-up theorems. The speed-up factor in Theorem 3 is a type-1 function and the proof is directly modified from a proof for the original speed-up theorem. In Theorem 4 we consider type-2 speed-up factors and it is lifted from the original operator speed-up theorem. In this section we consider a type-2 analog of the operator speed-up theorem, namely, we will try to explore a speed-up phenomenon when the speed-up factor is type-3. Clearly, a proof fo such theorem needs a general type-2 s-m-n and a type-2 recursion theorem, which we don't have. Instead, we argue that the type-2 analog of the operator speed-up theorem does not exist.

By "an effective type-2 operator" we mean a computable type-3 functional [11] of type $(\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ with inputs restricted to computable total type-2 functionals. Thus, we can think up that an effective type-2 operator is computed by a $\widehat{\varphi}$-program that takes a total $\widehat{\varphi}$-program as its input and outputs another total $\widehat{\varphi}$-program. Our next theorem asserts that there is an effective type-2 operator $\widehat{\Theta}$ such that, for *every* total $\widehat{\varphi}$-program e, there is no $\widehat{\Theta}$-sped up version for e. In other words, the $\widehat{\Theta}$-best programs always exist. Our theorem is stronger than a direct negation of the operator speed-up theorem in the sense that we claim that every $\widehat{\varphi}$-program is a $\widehat{\Theta}$-best $\widehat{\varphi}$-program.

Theorem 6 (Type-2 Operator Anti-Speed-up Theorem). *There is a type-2 effective operator* $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ *such that, for every computable functional,* $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, *we have*

$$\forall i : \widehat{\varphi}_i = F \; \forall j : \widehat{\varphi}_j = F \;\; \overset{\infty}{\exists}_{\mathbf{2}} \, (f, x)[\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)].$$

Proof: Define $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ by

$$\widehat{\Theta}(F)(f, x) = f(2^{F(f,x)+1}).$$

Clearly, such $\widehat{\Theta}$ is a type-2 effective operator. Fix any computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Also, fix a $\widehat{\varphi}$-program i for F. By contradiction, suppose that j is a $\widehat{\Theta}$-sped-up version of i, i.e., $\widehat{\Theta}(\widehat{\Phi}_j) \leq^*_{\mathbf{2}} \widehat{\Phi}_i$. If so, for all but finitely many $x \in \mathbf{N}$ such that, for every $f \in \mathcal{T}$, we have

$$\Theta(\widehat{\Phi}_j)(f, x) \leq \widehat{\Phi}_i(f, x).$$

Fix such x and f. By the definition of $\widehat{\Theta}$ and our assumption, we have

$$f(2^{\widehat{\Phi}_j(f,x)+1}) \leq \widehat{\Phi}_i(f, x).$$

Since there is no such query $f(2^{\widehat{\Phi}_j(f,x)+1}) =?$ during the course of the computation of $\widehat{\Phi}_j(f, x)$, it follows that the value of f at $2^{\widehat{\Phi}_j(f,x)+1}$ has no effect on the value of $\widehat{\Phi}_j(f, x)$. Therefore, if $f(2^{\widehat{\Phi}_j(f,x)+1})$ is sufficiently large, then $\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)$. This contradicts our assumption. □

Corollary 2. *There is a type-2 effective operator* $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ *such that, for all computable* $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, *we have*

$$\exists i : \widehat{\varphi}_i = F \; \forall j : \widehat{\varphi}_j = F \;\; \overset{\infty}{\exists}_{\mathbf{2}} \, (f, x)[\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)].$$

It is clear that Corollary 2 follows Theorem 6 immediately. Note that the corollary is the direct negation of the operator speed-up theorem.

6 Conclusion

In spite of the fact that oracle queries might interfere with the speed of an OTM, our investigation shows that the speed-up phenomena indeed exist in type-2 computation as long as the complexity measure satisfies Blum's two axioms. However, the phenomena disappear in higher-typed computation after type-2. We therefore have a strong belief that our investigation has completed the study of speed-up phenomena along the classical formulation of computational complexity, i.e., Blum's complexity measure. On the other hand, Blum's complexity measure may not be appropriate at type-2. For example, the query-complexity apparently fails to meet Blum's two axioms but it is such a commonly concerned resource in type-2 computation. Thus, a new approach is needed in understanding the concept of query-optimum programs. With a clear notion of query-optimum programs, we then can further examine the speed-up phenomena with respect to the notion of query-optimum programs. It would be interesting to continue research along this direction.

References

1. Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.): Handbook of Logic in Computer Science. In: Background: Mathematical Structures, Oxford University Press, Oxford (1992)
2. Blum, M.: A machine-independent theory of the complexity of recursive functions. Journal of the ACM 14(2), 322–336 (1967)
3. Blum, M.: On effective procedures for speeding up algorithms. Journal of the ACM 18(2), 290–305 (1971)
4. Calude, C.: Theories of Computational Complexity. Number 35 in Annals of Discrete Mathematics. North-Holland, Elsevier Science Publisher, B.V (1988)
5. Cook, S.A.: Computability and complexity of higher type functions. In: Logic from Computer Science, pp. 51–72. Springer, Heidelberg (1991)
6. Cutland, N.: Computability: An introduction to recursive function theory. Cambridge, New York (1980)
7. Davis, M.: The Undecidable. Raven Press, New York (1965)
8. Gödel, K.: Über die länge der beweise. Ergebnisse eines Math. Kolloquiums 7, pp. 23–24, Translation in [7], pp. 82–83, On the length of proofs (1936)
9. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Transitions of the American Mathematics Society, pp. 285–306, May (1965)
10. Kapron, B.M., Cook, S.A.: A new characterization of type 2 feasibility. SIAM Journal on Computing 25, 117–132 (1996)
11. Kleene, S.C.: Turing-machine computable functionals of finite types II. In: Proceedings of London Mathematical Society 12, 245–258 (1962)
12. Li, C.-C.: Speed-up theorems in type-2 computation (full version). http://www.itk.ilstu.edu/faculty/chungli/mypapers/SpeedUpFullVersion.pdf
13. Li, C.-C.: Asymptotic behaviors of type-2 algorithms and induced baire topologies. In: Proceedings of the Third International Conference on Theoretical Computer Science, pp. 471–484, Toulouse, France, August (2004)
14. Li, C.-C.: Clocking type-2 computation in the unit cost model. In: Proceedings of Computability in Europe: Logical Approach to Computational Barriers, pp. 182–192, Swansea, UK, Report# CSR 7-2006 (2006)
15. Li, C.-C., Royer, J.S.: On type-2 complexity classes: Preliminary report. pp. 123–138, May (2001)
16. Meyer, A.R., Fischer, P.C.: Computational speed-up by effective operators. The. Journal of Symbolic Logic 37, 55–68 (1972)
17. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. First paperback edition published by MIT Press in 1987, McGraw-Hill, (1967)
18. Royer, J.S.: Semantics vs. syntax vs. computations: Machine models of type-2 polynomial-time bounded functionals. Journal of Computer and System Science 54, 424–436 (1997)
19. Seiferas, J.I.: Machine-independent complexity theory. In: van Leeuwen, J (ed.), Handbook of Theoretical Computer Science, volume A, pp. 163–186. North-Holland, Elsevier Science Publisher, B.V, MIT press for paperback edition (1990)
20. Seth, A.: Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay (1994)
21. Van Emde Boas, P.: Ten years of speed-up. In: Proceedings of the Fourth Symposium Mathematical Foundations of Computer Science, pp. 13–29, 1975. Lecture Notes in Computer Science (1975)
22. Wagner, K., Wechsung, G.: Computational Complexity. In: Mathematics and its applications, D. Reidel Publishing Company, Dordrecht (1985)

The Complexity of Quickly ORM-Decidable Sets

Joel David Hamkins[1], David Linetsky[2], and Russell Miller[3]

[1] The College of Staten Island of CUNY and
The CUNY Graduate Center
jdh@hamkins.org
http://jdh.hamkins.org
[2] The CUNY Graduate Center
365 Fifth Avenue, New York NY 10016
dlinetsky@gc.cuny.edu
https://wfs.gc.cuny.edu/DLinetsky/www/
[3] Queens College of CUNY and
The CUNY Graduate Center
Russell.Miller@qc.cuny.edu
http://qcpages.qc.cuny.edu/math/faculty/miller.htm

Abstract. The Ordinal Register Machine (ORM) is one of several different machine models for infinitary computability. We classify, by complexity, the sets that can be decided quickly by ORMs. In particular, we show that the arithmetical sets are exactly those sets that can be decided by ORMs in times uniformly less than ω^ω. Further, we show that the hyperarithmetical sets are exactly those sets that can be decided by an ORM in time uniformly less than ω_1^{CK}.

Keywords: Ordinal, ordinal computation, infinite time computation, computability, register machine, arithmetical hierarchy, hyperarithmetical hierarchy, complexity.

1 Introduction

The Ordinal Register Machine (ORM) is one of several different machine models for infinitary computability that can be found in the literature. They are a direct generalization of classical register register machines that differ from their classical counterparts in that they are permitted to contain arbitrary ordinal values in their registers and to run for ordinal time. At limit times, the program state of an ORM is determined by taking a limit inferior (liminf) of the previous states and the content of a register is defined to be the liminf of the values that appeared in it at all previous times. ORMs have been shown to be extremely powerful; the sets of ordinals that they can compute from finitely many ordinal parameters have been characterized by Koepke ([5]) as precisely the constructible sets of ordinals, that is, those found in the Gödel constructible universe, L.

We noticed a curious difference between ORMs and infinite time Turing machines, however, much lower down in the hierarchy, in terms of the lengths of time that these machines take to decide arithmetic truth. Specifically, the natural recursive algorithm for deciding arithmetic truth with ORMs takes time unbounded in ω^ω, while with infinite time Turing machines this can be done before ω^2 (see [2]). For infinite time Turing machines, each limit allows one to decide two additional quantifiers, with Σ^0_{2n} truth being decidable in time ωn, and one pushes already into the hyperarithmetical hierarchy at time ω^2. But with ORMs, although the first limit allows one essentially to decide three quantifiers, one needs infinitely many additional limits (time ω^2) to go even just a little bit further and decide the Δ^0_4 sets. In general with ORMs, we prove that in time ω^{n+1}, one decides exactly the Δ^0_{3n+1} sets. It follows that ORMs really do need this extra time to decide arithmetic truth in comparison with infinite time Turing machines. We conclude the paper by proving that for hyperarithmetical truth, however, ORMs require time up to ω_1^{ck}, just as for infinite time Turing machines.

2 Arithmetical Sets

Given their demonstrated power, it is certainly clear that ORMs can decide any arithmetical set. However, it is not immediately clear how long such computations actually require. To begin with, we take the case of some relatively simple arithmetic sets and provide a general description of the algorithm used to decide membership in them.

Lemma 1. *Every Σ^0_3 and Π^0_3 subset of ω can be decided by an ORM in $\omega + 2$ steps.*

Proof. Let $A = \{t \mid \exists x \forall y \exists z R(t, x, y, z)\}$, where R is finite time decidable. We describe a program to determine whether $t \in A$.

Start with t in R_1 (register 1) and $0 \in R_2$ (register 2). The program consists of a large loop, each step of which consists of the following:

- For the current element $n \in R_2$, decode $n = \langle x, y, z\rangle$, check whether $R(t, x, y, z)$ and
$$(\forall y' < y)(\exists z')[\langle x, y', z'\rangle < n \;\&\; R(t, x, y', z')]\,, \tag{1}$$
and
$$(\forall z')[\langle x, y, z'\rangle < n \rightarrow \neg R(t, x, y, z')]\,. \tag{2}$$
- If all three of these hold, then we have evidence at step n that $(\forall y' \leq y)(\exists z')R(t, x, y, z')$ for a larger y the we had found before this step. In this case, copy x into register R_0.
- If not, then increment R_0 by 1.
- Finally, increment register R_2 by 1 and start over.

Notice that all of these tasks can be accomplished in finite time while using finitely many registers by making use of a program which computes R as a subroutine.

We run this loop ω many times. At stage ω, we compare the values in R_0 and R_2. If they are equal, then we zero the output register and halt. Otherwise, we write 1 in the output register and halt. Clearly, at the limit stage R_2 will contain ω. If R_2 and R_0 are equal at this point, then no x was was copied into register R_0 more than finitely often. Hence, $\forall x \exists y \forall z \neg R(t,x,y,z)$, i.e., $t \notin A$ and 0 is written in the output register. On the other hand, if R_0 does not contain ω, then it has some finite value x_0. This can only be the case if infinitely often more evidence was found that x_0 was a witness to the property $\forall y \exists z\, R(t,x,y,z)$. It immediately follows that $\exists x \forall y \exists z\, R(t,x,y,z)$, and hence we have that $t \in A$ and we output 1.

To decide membership in a Π^0_3 set, simply reverse the outputs. □

Using the algorithm described above, we can push on a little further into the arithmetical hierarchy:

Lemma 2. *Every Δ^0_4 set can be decided in time less than ω^2.*

Proof. Let $A \in \Delta^0_4$. Then A and $\omega \setminus A$ have Σ_4 definitions:

$$A = \{t \mid \exists x\, R(t,x)\} \tag{3}$$
$$\omega \setminus A = \{t \mid \exists x\, R'(t,x)\} \tag{4}$$

where $R, R' \in \Pi^0_3$. To decide if $t \in A$, we begin searching for an x so that either $R(t,x)$ or $R'(t,x)$. By Lemma 1, we can decide R, R' in $\omega + 2$ steps. Thus, we can decide whether t lies in A in $\omega \cdot n$ steps, for some $n \in \omega$. □

These initial results are easily extended to cover all arithmetical sets. This next lemma give the easy direction of the main theorem to come.

Lemma 3. *Every arithmetic set is ORM-decidable in time uniformly less than ω^ω. Indeed, if $A \in \Delta^0_{3n+1}$, then A is ORM-decidable in time less than ω^{n+1}.*

Proof. We proceed by induction on n. The case $n = 0$ is clear; it simply asserts the classical fact that Δ^0_1 sets are finite time decidable. In fact, the previous lemma gives the case $n = 1$ as well. Now, suppose the result holds below n and let $A \in \Delta^0_{3n+1}$. Then A and $\omega \setminus A$ have Σ_{3n+1} definitions:

$$A = \{t \mid \exists x\, R(t,x)\} \tag{5}$$
$$\omega \setminus A = \{t \mid \exists x\, R'(t,x)\} \tag{6}$$

where $R, R' \in \Pi^0_{3n}$. Then, $R = \{t \mid \forall x \exists y \forall z\, Q(t,x,y,z)\}$, where $Q \in \Sigma^0_{3n-3} \subseteq \Delta^0_{3n-2}$. By the inductive hypothesis, Q is ORM-decidable in time less than ω^n. So, we simply apply the algorithms described in Lemmas 1 and 2, using the program that decides Q as a subroutine. In this manner, we can decide A in time less than ω^{n+1}. □

3 Characterizing the Quickly Decidable Sets

We now prove the harder direction, that in time uniformly less than ω^ω, ORMs do not escape the arithmetical hierarchy. The Main Theorem provides a characterization of the sets decidable by ORMs in times strictly less than ω^ω.

Theorem 1. *The subsets of ω that are ORM-decidable (using arbitrary ordinal parameters!) in time uniformly less than ω^ω are exactly the arithmetical sets. In particular, a set A is ORM-decidable in time less than ω^{n+1} if and only if $A \in \Delta^0_{3n+1}$.*

This theorem should be contrasted with its analogue for Infinite Time Turing Machines (ITTMs) found in [2], which states that every arithmetic set can be decided by an ITTM in time uniformly less than ω^2. Of course, ITTMs are able to do this by making use of their infinite tape memory on which they are able to write out oracles that can be later referred to in order to greatly speed up later computations. ORMs, on the other hand, can store only finitely many ordinals and are thus unable to use this type of strategy. Instead, they must recompute the information required to preform a computation each time it is required.

Before we prove this main result, we provide some definitions and develop some of the key ideas involved. In order to analyze the descriptive complexity of a set that is decidable in time less than ω^n, for some $n \in \omega$, we need to be able to talk about ORM configurations in a first order fashion. Since the register contents of an ORM are arbitrary ordinals, in order to accomplish this, we require a method of coding these ordinals as natural numbers.

Definition 1. *For any ordinal $\alpha < \omega^\omega \cdot 2$, let $\ulcorner\alpha\urcorner = \langle m, n_0, \dots, n_k\rangle$ where $m \in \{0, 1\}$ and $\alpha = \omega^\omega \cdot m + \omega^k \cdot n_k + \cdots + \omega \cdot n_1 + n_0 < \omega^\omega$. Furthermore, let $\prec$ be the order on these codes such that $\ulcorner\alpha\urcorner \prec \ulcorner\beta\urcorner$ if and only if $\alpha < \beta$.*

Now, since ORM programs are finite and can make use of only finitely many registers, we may also code an ORM configuration as a natural number.

Definition 2. *An* ORM configuration *consists of a program state and the contents of each register. In the case that a computation uses only ordinals less than $\omega^\omega \cdot 2$, then we use the above coding to code each of the its configurations as some natural number $\mathcal{C} \in \omega$.*

The statement of Theorem 1 allowed for arbitrary ordinal parameters while Definition 1 only allows us to code ordinals smaller than $\omega^\omega \cdot 2$. The next result shows that this is in fact sufficient.

Lemma 4. *Given any ORM program P, finite sequence of ordinals $\overrightarrow{\beta} < \omega^\omega$, and any ordinal parameter $\beta > \omega^\omega$, $P(\overrightarrow{\beta}, \beta) \downarrow = \gamma < \omega^\omega$ in time less than ω^ω if and only if $P(\overrightarrow{\beta}, \omega^\omega) \downarrow = \gamma$, and they do so in exactly the same number of steps (where $P(\overrightarrow{\alpha}) \downarrow$ means that program P converges on inputs $\overrightarrow{\alpha}$).*

Proof. The idea of the proof is that the ordinal β is much too large for the ORM algorithm to make use of in such a short time; any such β operates identically to ω^ω itself in any computation. Specifically, we prove the result by induction on time. On one ORM, $\mathcal{M}$, we run $P(\overrightarrow{\beta}, \beta)$, while on a second ORM, $\mathcal{M}'$, we run $P(\overrightarrow{\beta}, \omega^\omega)$. It is not hard to see that at every time t, the machine state of $\mathcal{M}$ is the same as that of $\mathcal{M}'$, and if any register in $\mathcal{M}$ contains a value $< \omega^\omega$, then the corresponding register of $\mathcal{M}'$ contains the same value. Moreover, any register of $\mathcal{M}$ has value $\beta+\gamma$, with $\beta > \omega^\omega$, if and only if the corresponding register in $\mathcal{M}'$ has value $\omega^\omega+\gamma$. At limit times we use the fact that for any $\alpha < \omega^\omega$, $\alpha+\omega^\omega = \omega^\omega$. □

In fact, the above proof goes through if every occurrence of ω^ω in the statement of the lemma is replaced by any ordinal of the form ω^α. Now, given the method of Definition 2 for coding machine configurations, we now define a relation that will allow us to describe how two configurations relate to each other relative to a particular program.

Definition 3. *Let $\mathcal{C}, \mathcal{C}' \in \omega$ be codes for two ORM configurations, $\alpha < \omega^\omega$ and let $P \in \omega$ be the code for an ORM-program. Define relation $R \subseteq \omega^4$ by: $R(\mathcal{C}, \mathcal{C}', P, \ulcorner\alpha\urcorner)$ if and only if the configuration coded by $\mathcal{C}'$ follows the configuration coded by $\mathcal{C}$ in exactly α steps under the operation of the program coded by P. Also, for any ordinal $\alpha < \omega^\omega$, define the relation $R_\alpha \subseteq \omega^3$ by $R_\alpha(\mathcal{C}, \mathcal{C}', P) \leftrightarrow R(\mathcal{C}, \mathcal{C}', P, \ulcorner\alpha\urcorner)$.*

Of course, in order to make use of this relation, we need to know that it can be expressed in a first order fashion. This is taken care of by the next result.

Lemma 5. *For each ordinal $\alpha < \omega^\omega$, the relation $R_\alpha(\mathcal{C}, \mathcal{C}', P)$ is arithmetical. Indeed, if $\alpha = \omega^k \cdot n_k + \cdots + \omega \cdot n_1 + n_0$, then the statement $R_\alpha(\mathcal{C}, \mathcal{C}', P)$ is Δ^0_{3k+1}.*

Proof. By induction on k where $\alpha = \omega^k \cdot n_k + \cdots + \omega \cdot n_1 + n_0$. Clearly R_1 is finite time computable, and hence Δ^0_1. Suppose the result holds below k and that $\alpha = \omega^k \cdot n_k + \cdots + \omega \cdot n_1 + n_0$. Clearly, $R_\alpha(\mathcal{C}, \mathcal{C}', P)$ holds if and only if there is a configuration $\mathcal{C}''$ such that $R_{\omega^k \cdot n_k}(\mathcal{C}, \mathcal{C}'', P)$ and $R_\beta(\mathcal{C}'', \mathcal{C}', P)$, where $\beta = \omega^{k-1} \cdot n_{k-1} + \cdots + \omega \cdot n_1 + n_0$. The latter relation is arithmetical by assumption, so we need only consider $R_{\omega^k \cdot n_k}(\mathcal{C}, \mathcal{C}'', P)$.

Now, if $n_k > 1$, then $R_{\omega^k \cdot n_k}(\mathcal{C}, \mathcal{C}'', P)$ is equivalent to asserting that there exist configuration $\mathcal{D}_0, \ldots, \mathcal{D}_{n_k}$ such that $\mathcal{C} = \mathcal{D}_0$, $\mathcal{C}'' = \mathcal{D}_{n_k}$, and that $R_{\omega^k}(\mathcal{D}_i, \mathcal{D}_{i+1}, P)$ for $i < n_k$, i.e., each configuration follows the previous one in ω^k steps. Thus, it suffices to consider the case $n_k = 1$, which asserts that $\mathcal{C}$ leads to $\mathcal{C}''$ in ω^k steps. Using the relation for smaller ordinals, and our coding of smaller ordinals as natural numbers, we can simply write out the liminf definition as a first order sentence and see that it is indeed arithmetical.

Indeed, by the remarks above, it suffices to show that $R_{\omega^k}(\mathcal{C}, \mathcal{C}', P) \in \Pi^0_{3k}$, from which it follow that $R_\alpha(\mathcal{C}, \mathcal{C}', P) \in \Delta^0_{3k+1}$, where $\alpha = \omega^k \cdot n_k + \cdots + \omega \cdot n_1 \cdot n_0$. To see that this is so, we give the portion of the liminf definition which bounds its complexity, that is, we look closely at how to say that the value of the n^{th} register, $\mathcal{C}'(n)$, is equal to some ordinal ξ.

$$\begin{aligned}\mathcal{C}(n) = \xi \text{ iff } & (\forall \ulcorner\beta\urcorner \prec \ulcorner\xi\urcorner)(\exists \ulcorner\gamma\urcorner \prec \ulcorner\omega^k\urcorner)(\forall \mathcal{D})(\forall \ulcorner\delta\urcorner \succ \ulcorner\gamma\urcorner) \\ & \qquad \left[\left(\ulcorner\delta\urcorner \prec \ulcorner\omega^k\urcorner \;\&\; R(\mathcal{C}, \mathcal{D}, P, \ulcorner\delta\urcorner)\right) \rightarrow \ulcorner\mathcal{D}(i)\urcorner \succ \ulcorner\beta\urcorner\right] \\ & \& \; (\forall \ulcorner\beta\urcorner \prec \ulcorner\omega^k\urcorner)(\exists \mathcal{D})(\exists \ulcorner\gamma\urcorner \succ \ulcorner\beta\urcorner) \\ & \qquad [R(\mathcal{C}, \mathcal{D}, P, \ulcorner\gamma\urcorner) \;\&\; \ulcorner\mathcal{D}(i)\urcorner \preceq \ulcorner\xi\urcorner]\end{aligned} \tag{7}$$

Assuming inductively that $R(\mathcal{C}, \mathcal{D}, P, \ulcorner\gamma\urcorner) \in \Delta^0_{3k-2}$, for all $\gamma < \alpha$, it follows that the above sentence is indeed Π^0_{3k}, and hence that $R_\alpha(\mathcal{C}, \mathcal{C}', P) \in \Delta^0_{3k+1}$. □

Putting together all of these pieces, we can now go ahead and prove the main result of this section.

Proof (of Theorem 1). Suppose A is decidable in time less than ω^{n+1} (by program P, say). Then, $x \in A$ if and only if

$$(\exists \mathcal{C})(\exists n)\ [R_\alpha(x^*, \mathcal{C}, P)\ \&\ n = \ulcorner \alpha \urcorner\ \&\ \mathcal{C} \text{ halts with output } 1] \tag{8}$$

if and only if

$$(\forall \mathcal{C})(\forall n)\ [(R_\alpha(x^*, \mathcal{C}, P)\ \&\ n = \ulcorner \alpha \urcorner) \rightarrow\ \mathcal{C} \text{ halts with output } 1]\ , \tag{9}$$

where x^* is the start configuration with x in the input register. Hence, since $R_\alpha(x^*, \mathcal{C}, P)$ is Δ^0_{3n+1}, it follows that $A \in \Delta^0_{3n+1}$. □

4 Deciding Hyperarithmetical Sets

Of course, ORMs are capable of deciding membership in sets much more complex than the arithmetical sets. In this final section we present two results concerning hyperarithmetical sets. The first shows that the uniformity in time found in Theorem 1 is necessary, i.e., there are hyperarithmetical sets that can be decided in time less than ω^ω (but not uniformly so). The second result gives a characterization of the sets decidable in times uniformly less than ω_1^{CK}, the first non-recursive ordinal, as precisely the hyperarithmetical sets.

Theorem 2. *There exist hyperarithmetic sets that are ORM-decidable in time less than ω^ω (not uniformly). Indeed, the ω^{th} jump of zero, $\emptyset^{(\omega)}$, is such a set.*

Proof. We show that $\emptyset^{(\omega)}$ can be computed in time less than ω^ω by describing an algorithm that accomplishes this. Of course, Theorem 1 ensures that this can't be done in time uniformly less than ω^ω. The algorithm requires that we simulate a stack machine (as in [5] and [3]) with two stacks on an ORM. To decide whether $(n, k) \in \emptyset^{(\omega)}$, i.e., whether $k \in \emptyset^{(n)}$ we proceed as follows: First, we run the algorithm which computes whether $k \in \emptyset^{(n)}$ using an $\emptyset^{(n-1)}$-oracle. When the algorithm queries the oracle as to whether some natural number $k' \in \emptyset^{(n-1)}$, push $n-1$ onto stack 1, and push $\omega \cdot (n-1) + k'$ onto stack 2 and run this same program over again. We push $\omega \cdot (n-1) + k'$ instead of just k' because when deciding whether $k' \in \emptyset^{n-1}$ we may have to make a query about some number $l > k'$. However, pushing l onto stack 2 would violate the stack protocol, which requires that any number pushed onto a non-empty stack must be smaller than the number preceding it. This recursive process will eventually decide membership in $k \in \emptyset^{(\omega)}$. Moreover, an analysis of the algorithm shows that it always halts in time less than ω^ω, but that for every $m \in \omega$ there is some $\langle n, k \rangle \in \omega$ such that the computation that decides whether $\langle n, k \rangle \in \emptyset^{(\omega)}$ takes more than ω^m steps. □

Finally we characterize the sets decidable by an ORM in time less than some recursive ordinal.

Theorem 3. *The sets that are ORM-decidable in time less than ω_1^{CK} are exactly the hyperarithmetic sets.*

We prove this result using a sequence of lemmas. To begin with, we will show that every hyperarithmetical set is decidable in time less than some recursive ordinal. In order to this, we use a result of Shoenfield (see [9] or exercise 16-93 in [7]), which states that every hyperarithmetical set is Turing reducible to some element of a particular series of sets. The sets are defined using the usual jump operator; we denote the jump of a set D by D'.

Lemma 6 (Shoenfield). *Define a sequence of sets D_α, for $0 < \alpha < \omega_1^{CK}$, as follows:*

$$\begin{aligned} D_0 &= \omega\,; \\ D_{\alpha+1} &= D_\alpha \cap (D_\alpha)'\,; \\ D_\alpha &= \bigcap_{\beta<\alpha} D_\beta\,, \quad \textit{if } \alpha \textit{ is a limit ordinal.} \end{aligned} \tag{10}$$

Then, $A \in \Delta_1^1$ if and only if $A \leq_T D_\alpha$ for some $\alpha < \omega_1^{CK}$.

Of course, in order for Lemma 6 to be useful, we need to know that each of the D_α's is ORM-decidable in time less than some recursive ordinal, which we take care of in the next result.

Lemma 7. *For every $\alpha < \omega_1^{CK}$, D_α is ORM-decidable in time less than some $\beta < \omega_1^{CK}$.*

Proof. We proceed by induction on α. Suppose that D_α is ORM-decidable in time less than β, where $\beta < \omega_1^{\mathrm{CK}}$. For any $x \in \omega$, to determine whether $x \in D_{\alpha+1}$, we need to determine whether $x \in D_\alpha$ (requiring time $< \beta$) and whether $x \in (D_\alpha)'$, which may require up to ω queries about D_α and possibly one more step to determine that $x \notin (D_\alpha)'$. Thus, $D_{\alpha+1}$ can be decided in time at most $\beta \cdot \omega + 1$, which is again a recursive ordinal since ω_1^{CK} is closed under ordinal arithmetic.

Now suppose that α is a limit and the result holds for all $\gamma < \alpha$. Define the function $f_\alpha : \alpha \to \omega_1^{\mathrm{CK}}$ by $f(\gamma) = \delta$ if and only if δ is least such that D_γ is ORM-decidable in time less than δ. Now, if we want to decide whether some $x \in D_\alpha$, we need to decide whether it is in D_γ for each $\gamma < \alpha$. Thus, D_α should be decidable in time at most $\epsilon := \bigcup_{\gamma<\alpha} f_\alpha(\gamma)$. Thus, if can show that $\epsilon < \omega_1^{\mathrm{CK}}$, we are done.

We note that $D_\alpha \in L_{\omega_1^{\mathrm{CK}}}$ for each $\alpha < \omega_1^{\mathrm{CK}}$, and that the notion of ORM computability can be defined in a Δ_0 way in $L_{\omega_1^{\mathrm{CK}}}$. Thus, the graph of f_α is Δ_0 definable in $L_{\omega_1^{\mathrm{CK}}}$. So, since ω_1^{CK} is an admissible ordinal, and thus satisfies Σ_1 replacement, it follows that $\epsilon = \operatorname{ran} f_\alpha \in L_{\omega_1^{\mathrm{CK}}}$, and hence that $\epsilon < \omega_1^{\mathrm{CK}}$. □

Lemmas 6 and 7 essentially complete the proof of the backwards direction of Theorem 3. To see this, suppose that A is hyperarithmetical, i.e., that $A \in \Delta_1^1$. Then, by Lemma 6, we have that $A \leq_T D_\alpha$ for some $\alpha < \omega_1^{\mathrm{CK}}$. By Lemma 7, we can decide D_α is time less than some $\beta < \omega_1^{\mathrm{CK}}$. Thus, it follows that we can decide A in time at most $\beta \cdot \omega < \omega_1^{\mathrm{CK}}$. Hence, every hyperarithmetical set can be decided by an ORM in time less than some recursive ordinal.

All that remains now is to show the converse. In order to this, we will make use of Kleeene's $\mathcal{O}$, which provides natural number notations for every recursive ordinal. We will not define $\mathcal{O}$, but the standard definitions and results concerning $\mathcal{O}$ may be found in any standard text on the subject. We will follow the notation found in [1]. Thus, for any $n \in \mathcal{O}$, we write $|n|_{\mathcal{O}}$ to denote the ordinal denoted by n and we denote the standard order relation on $\mathcal{O}$ by $<_{\mathcal{O}}$ so that $a <_{\mathcal{O}} b$ if and only if $|a|_{\mathcal{O}} < |b|_{\mathcal{O}}$. We now complete the proof of Theorem 3 by proving the following lemma.

Lemma 8. *If $A \subseteq \omega$ is ORM-decidable in time less than some recursive ordinal, then $A \in \Delta_1^1$.*

Proof. Suppose that $A \subseteq \omega$ is ORM-decidable, by the program P, in time less than α for $\alpha < \omega_1^{\mathrm{CK}}$. Fix some $a \in \mathcal{O}$ so that $|a|_{\mathcal{O}} = \alpha$. Then, the set $S := \{b \mid b <_{\mathcal{O}} a\} \subseteq \omega$ is recursively enumerable and provides a set of notations for all ordinals less than α. Using S, we can mimic Definition 2 and code any ORM configuration having register contents less than α as natural numbers. For any $\mathcal{C}, \mathcal{C}' \in \omega$ coding two such configurations, let us say that $Q^P(\mathcal{C}, \mathcal{C}', n)$ holds if and only if the configuration coded by $\mathcal{C}'$ follows that coded by $\mathcal{C}$, under the operation of the program P, in exactly $|n|_{\mathcal{O}}$ many steps.

We claim that for any $n \in S$ the relation $Q^P(\mathcal{C}, \mathcal{C}', n) \in \Delta_1^1$. This is easily shown by induction. Let $n \in S$ and suppose that the result holds for all $m <_{\mathcal{O}} n$. If $|n|_{\mathcal{O}} = \beta + 1$, then $n = 2^m$ for $m \in S$, $m <_{\mathcal{O}} n$, and $|m|_{\mathcal{O}} = \beta$. In this case, $Q^P(\mathcal{C}, \mathcal{C}', n) \in \Delta_1^1$ if and only if $(\exists \mathcal{D})[Q^P(\mathcal{C}, \mathcal{D}, m) \,\&\, Q^P(\mathcal{C}, \mathcal{D}, 2)]$ (note: $|2|_{\mathcal{O}}$=1). Thus, by the induction hypothesis, it follows that $Q^P(\mathcal{C}, \mathcal{D}, n) \in \Delta_1^1$. On the other hand, if $|n|_{\mathcal{O}}$ is a limit ordinal, then we must unravel the lim inf definition as we did in the proof of Lemma 5, except that we use codes from S and the order $<_{\mathcal{O}}$ instead of $\prec$. To do this, we require only quantifiers over ω, which do not increase the complexity. Hence, the relation $Q^P(\mathcal{C}, \mathcal{C}', n) \in \Delta_1^1$ for every $n <_{\mathcal{O}} a$.

The theorem now follows immediately, since $x \in A$ if and only if

$$(\exists \mathcal{C})(\exists n \in S)\ \left[Q^P(x^*, \mathcal{C}, n) \,\&\ \mathcal{C} \text{ halts with output } 1\right] \tag{11}$$

if and only if

$$(\forall \mathcal{C})(\forall n \in S)\ \left[Q^P(x^*, \mathcal{C}, n) \rightarrow\ \mathcal{C} \text{ halts with output } 1\right]\ , \tag{12}$$

where x^* is the start configuration with x in the input register, and all other registers set to zero. Hence, $A \in \Delta_1^1$ and is thus hyperarithmetical. □

References

1. Ashe, C.J., Knight, J.: Computable Structures and the Hyperarithmetical Hierarchy. In: Studies in Logic and the Foundations of Mathematics. Elsevier, Amsterdam (2000)
2. Hamkins, David, J., Lewis, A.: Infinite Time Turing Machines. J. Symbolic Logic 65(2), 567–604 (2000)
3. Hamkins, David, J., Miller, R.: Post's Problem for Ordinal Register Machines. (To appear in this volume)
4. Jech, Thomas.: Set Theory. The Third Millenium Edition. In: Springer Monographs in Mathematics, Springer, Heidelberg (2003)
5. Koepke, Peter.: Ordinals, Computations, and Models of Set Theory: A Tutorial at Days in Logic, Coimbra, Portugal. Tutorial Material. (accessed January 2006) `http://www.mat.uc.pt/~kahle/dl06/koepke.pdf`
6. Koepke, Peter.: Turing Computations on Ordinals. J. Symbolic Logic 11(3), 377–397 (2005)
7. Rogers, Hartley Jr.: Theory of Recursive Functions and Effective Computability. The MIT Press, Cambridge (1967)
8. Sacks, G.E.: Higher Recursion Theory. In: Perspectives in Mathematical Logic, Springer, Heidelberg (1990)
9. Shoenfield, Joseph, R. (eds.): Recursion Theory. Lecture Notes in Logic. Springer, Heidelberg (1993)

On Accepting Networks of Splicing Processors of Size 3

Remco Loos*

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pça Imperial Tàrraco 1, 43005 Tarragona, Spain
remcogerard.loos@urv.cat

Abstract. In this paper, we show that accepting networks of splicing processors (ANSPs) of size 3 are computationally complete. Moreover, we prove that they can decide all languages in **NP** in polynomial time. The previous lower bound for both issues was 7. Since, by its definition, ANSPs need at least 2 nodes for any non-trivial computation, we leave only one open case. We also prove the following normal form: For any ANSP there exists an equivalent ANSP without output filters.

Keywords: Molecular Computation, Splicing, Networks of Processors.

1 Introduction

Accepting networks of splicing processors (ANSPs for short) [5,6] are a variant of networks of evolutionary processors, a well studied model of bio-inspired computing [1,2,7]. Networks of evolutionary processors consist of several processors placed in a graph. Each processor performs simple rewriting operations (insertion, deletion, substitution), after which the strings are redistributed among the nodes of the graph according to filters.

In the case of accepting networks of splicing processors, these rewriting operations are replaced by splicing rules. ANSPs were introduced in [5], where it was shown that ANSPs are computationally complete. Also, the complexity class **NP** was proved to correspond to the class of languages accepted by ASNPs in polynomial time and **PSPACE** to the class accepted by ANSPs with at most polynomial length of the stings used in the derivation. Finally, a linear time solution for SAT was presented. In [6] it was proved that that ANSPs of constant size accept all regularly enumerable languages and can solve all problems in **NP** in polynomial time.

In both cases the number of nodes needed was 7. Here we show that both things can be achieved by ANSPs of 3 nodes. Moreover, in [6] the constant size was achieved using an encoding in a 2-letter alphabet. Here we present a direct construction for any language accepted by a deterministic Turing machine. In

* Work supported by Research Grant BES-2004-6316 of the Spanish Ministry of Education and Science.

passing, we prove the following normal form: For any ANSP there exists an equivalent ANSP without output filters.

For the complexity result, we show by a more involved construction that ANSPs of size 3 can simulate the computations of a non-deterministic Turing machine in parallel. We then use this fact to show that ANSPs of size 3 can decide all languages in **NP**.

Since, by its definition, ANSPs need at least two nodes to accept any non-trivial language, these results go a long way in settling this issue, leaving just one open case. Moreover, our results suggest that research in reduced ANSPs with less features might be fruitful.

2 Basic Definitions and Notation

We assume the reader's familiarity with the basic concepts in complexity classes and formal language theory. The reader may refer to [4] and [10] for definitions.

For any set A, $|A|$ denotes the cardinality of A and for a word w, $|w|$ denotes the length of w. The smallest W such that $w \in W^*$ is denoted by $alph(w)$ and the empty word is denoted by λ.

A *splicing rule* over an alphabet V is a word of the form $u_1\#u_2\$v_1\#v_2$ such that u_1, u_2, v_1, and v_2 are in V^* and such that \$ and # are two symbols not in V.

For a splicing rule $r = u_1\#u_2\$v_1\#v_2$ and for $x, y, w, z \in V^*$, we say that r produces (w, z) from (x, y) (denoted by $(x, y) \vdash_r (w, z)$) if there exist some $x_1, x_2, y_1, y_2 \in V^*$ such that $x = x_1u_1u_2x_2$, $y = y_1v_1v_2y_2$, $z = x_1u_1v_2y_2$, and $w = y_1v_1u_2x_2$.

For a language L over V and a set of splicing rules R we define

$$\sigma_R(L) = \{z, w \in V^* \mid (\exists u, v \in L, r \in R)[(u, v) \vdash_r (z, w)]\}.$$

For two disjoint subsets P and F of V and a word x over V, we define the predicates:

$$\phi^s(x; P, F) \equiv P \subseteq alph(x) \wedge F \cap alph(x) = \emptyset$$
$$\phi^w(x; P, F) \equiv P \cap alph(x) \neq \emptyset \wedge F \cap alph(x) = \emptyset.$$

Here, P is the set of *permitting symbols* and F the set of *forbidding symbols.* The first condition (s=strong) requires all permitting symbols and no forbidding symbol to be present in x, whereas for the second (w=weak) at least one permitting and no forbidding symbol should be present in x. For a language $L \subseteq V^*$ and $\beta \in \{w, s\}$,

$$\phi^\beta(L, P, F) = \{x \in L \mid \phi^\beta(x; P, F)\}.$$

If we want to permit all strings of L, we should say $P = V^*$ for $\beta = w$ or $P = \emptyset$ for $\beta = s$. For simplicity, in those cases we simply write $P = \emptyset$, not specifying β. A *splicing processor* over V is a 6-tuple (S, A, PI, FI, PO, FO) with

- S a finite set of splicing rules over V,
- A a finite set of auxiliary words,

– $PI, FI \subseteq V$ the *input* permitting/forbidding symbols,
– $PO, FO \subseteq V$ the *output* permitting/forbidding symbols.

An *accepting network of splicing processors* (ANSP) is a construct
$\Gamma = (V, U, \langle, \rangle, G, \mathcal{N}, \alpha, x_I, x_O)$, where

- U is the network alphabet and $V \subseteq U$ is the input alphabet.
- $\langle, \rangle \in U - V$ are two special symbols.
- $G = (X_G, E_G)$ is an undirected graph with nodes X_G and edges E_G. We assume G to be complete and without loops.
- $\mathcal{N}$ is a mapping which associates with each node $x \in X_G$ the splicing processor $\mathcal{N} = (S_x, A_x, PI_x, FI_x, PO_x, FO_x)$.
- $\alpha : X_G \to \{s, w\}$ defines the type of the input/output filters, where for each node $x \in X_G$ the input filter ρ and output filter τ are defined as:

$$\rho_x(\cdot) = \phi^{\alpha(x)}(\cdot; PI_x, FI_x),$$
$$\tau_x(\cdot) = \phi^{\alpha(x)}(\cdot; PO_x, FO_x).$$

- $x_I, x_O \in X_G$ are the *input* and *output* node, respectively.

The *size* of Γ corresponds to the number of nodes in the graph, i.e. $|X_G|$. A *configuration* of an ANSP Γ is a mapping $C : X_G \to 2^U$ which associates a set of words to every node of the graph. A configuration can be seen as the sets of words which are present in any node at a given moment. For a word $w \in V^*$ the initial configuration of Γ on w is defined by $C_0^{(w)}(x_I) = \langle w \rangle$ and $C_0^{(w)}(x) = \emptyset$ for all other $x \in X_G$. By convention, the auxiliary words do not appear in any configuration.

There are two ways to change a configuration, by a splicing step or by a communication step. When changing by a splicing step, each component $C(x)$ of the configuration C is changed according to the set of splicing rules S_x, whereby the words in the set A_x are available for splicing. Formally, configuration C' is obtained in one splicing step from the configuration C, written as $C \Rightarrow C'$, iff for all $x \in X_G$

$$C'(x) = C(x) \cup \sigma_{S_x}(C(x) \cup A_x)^{1}.$$

In a communication step, each processor sends out all strings that can pass the output filter. They are received by all other nodes in the graph, provided they pass the input filter. Note that, according to this definition, strings that can leave a node are sent out even if they cannot pass any input filter. In this case we will say that the are lost. Formally, C' is obtained from C (we write $C' \models C$) iff for all $x \in X_G$

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))).$$

[1] This definition is slightly different from those in [5,6], which are also different from each other. Our definition is 'weaker' in the sense that our constructions will also work for both other definitions.

For an ANSP Γ, the computation on an input word w is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, ...,$ where $C_0^{(w)}$ is the initial configuration of Γ on w, $C_{2i}^{(w)} \Rightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \models C_{2i+2}^{(w)}$, for all $i \geq 0$. A computation *halts* if one of the following two conditions holds:

1. There exist a configuration in which the set of words existing in the output node x_O is non-empty. This is an *accepting computation.*
2. There exist two consecutive identical configurations.

The language accepted by Γ is defined as

$$L(\Gamma) = \{w \in V^* \mid \Gamma's \text{ computation on } w \text{ is an accepting computation}\}.$$

We say Γ *decides* a language L if $L(\Gamma) = L$ and Γ halts on every computation.

For a halting computation $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, ..., C_m^{(w)}$, we define the time complexity of Γ on w by $TIME_\Gamma(w) = m$. The time complexity of Γ is the partial function from $\mathbb{N}$ to $\mathbb{N}$,

$$TIME_\Gamma(n) = max\{TIME_\Gamma(x) \mid x \in V^*, |x| = n\}.$$

3 ANSPs of Three Nodes are Computationally Complete

In this section, we provide a completeness proof based on the simulation of a deterministic Turing machine. The technique used here is reminiscent of the proof of [8].

Theorem 1. *ANSPs of size 3 accept all RE languages.*

Proof. Let $M = (Q, V, \Delta, q_0, \{q_f\}, \delta, B)$ be a Turing machine, with Q is the set of states, V and Δ respectively the input and tape alphabet, q_0 the initial state, B the blank symbol and $\delta : Q \times \Delta \rightarrow Q \times \Delta \times \{L, R\}$ the transition function. We assume without loss of generality that M is deterministic, has a semi-infinite tape and a single accepting state q_f.

We construct the ANSP $\Gamma = (V, U, \langle, \rangle, G, \mathcal{N}, \alpha, 3, 1)$, where G is the complete graph with 3 nodes, $U = V \cup \Delta \cup \{L, R, Z, \langle, \rangle\}$ and (the value of α is omitted where irrelevant):

– $S_1 = \emptyset$
 $A_1 = \emptyset$
 $PI_1 = \{q_f\}$
 $FI_1 = \{L, R\}$
 $PO_1 = FO_1 = \emptyset$

In what follows $a, b, c, d \in V \cup \Delta \cup \{\langle, \rangle\}$, and $q_i, q_j \in Q$.

– $S_2 = \{$
 $c\#q_i\rangle\$L\#q_iB\rangle R$ (Right end of the tape)
 $c\#q_ia\$L\#bq_jR$ for $(q_i, a) \rightarrow (q_j, b, R) \in \delta$ (Right move)

$\lambda\#cq_ia\$L\#q_jcbR$ for $(q_i, a) \to (q_j, b, L) \in \delta\}$ (Left move)
$A_2 = \{Lbq_jR \mid (q_i, a) \to (q_j, b, R) \in \delta\} \cup \{Lq_jcbR \mid (q_i, a) \to (q_j, b, L) \in \delta\}$
$\cup\{Lq_iB\rangle R\}$
$FI_2 = \{L, R\}$
$PI_2 = PO_2 = FO_2 = \emptyset$

– $S_3 = \{$
$\langle a\#\lambda\$\langle q_0a\#Z$ for $a \in V$ (Initialisation)
$cq_iB\rangle\#R\$Lq_i\rangle\#\lambda$ (Right end of the tape)
$cbq_j\#R\$Lq_ia\#\lambda$ for $(q_i, a) \to (q_j, b, R) \in \delta$ (Right move)
$dq_jcb\#R\$Lcq_ia\#\lambda$ for $(q_i, a) \to (q_j, b, L) \in \delta\}$ (Left move)
$A_3 = \{\langle q_0aZ \mid a \in V\}$
$PI_3 = \{L, R\}$
$FI_3 = PO_3 = FO_3 = \emptyset$
$\alpha(3) = w$

The systems works as follows. Node 3 is the input node, where the input $\langle w\rangle$ is converted into $\langle q_0w\rangle$, which is sent out in the communication step. It will reach only node 2 (unless q_0 is the final state). Now, the moves of M are simulated by moving back and forth between nodes 2 and 3. When a final state is reached, that is we get to a word of the form $\langle w_1q_fw_2\rangle$, with $w_1, w_2 \in (V \cup \Delta)^*$, this word can pass the input filter of node 1, and input w is accepted.

In node 2, we can start simulating the moves of M. For a right move $(q_i, a) \to (q_j, b, R)$ on a configuration $\langle wq_iaw'\rangle$ we apply rule $c\#q_ia\$L\#bq_jR$ using Lbq_jR from A_2, giving $\langle wbq_jR$ and $Lq_iaw'\rangle$, which are passed to node 3. In node 3, the two words are combined by rule $cbq_j\#R\$Lq_ia\#\lambda$ to give $\langle wbq_jw'\rangle$, the new configuration. This word is sent back to node 2, where the simulation of the next move begins. Left moves are simulated in a similar way, using a rule $\lambda\#cq_ia\$L\#q_jcbR$ in node 2 and a rule $dq_jcb\#R\$Lcq_ia\#\lambda$ in node 3. If M needs more tape space on the right, we can insert a blank symbol B at the end of our configuration using rules $c\#q_i\rangle\$L\#q_iB\rangle R$ in node 2 and $cq_iB\rangle\#R\$Lq_i\rangle\#\lambda$ in node 3. Indeed, when we have a configuration of the form $\langle wq_i\rangle$, rule $c\#q_i\rangle\$L\#q_iB\rangle R$ yields $\langle wq_iB\rangle R$ and $Lq_i\rangle$, which combine in node 3 to give $Lq_i\rangle R$ and $\langle wq_iB\rangle$. This last string is sent out to node 2, where the simulation resumes.

It should be clear from this explanation that Γ accepts input $\langle w\rangle$ if M reaches a final state on input w. To see that Γ accepts only these words, recall that since M is deterministic, there is only one possible move at every step, thus at all times only one word representing a configuration is present in node 2 or two parts of this configuration in node 3. Moreover, all auxiliary and intermediate strings do not produce words that can interfere in the derivation. The input word $\langle w\rangle$ is sent out in the first communication step and can reach only node 2. There, no rule can be applied to it, and it is sent out in the next step. Now, it is lost, since it cannot enter either 1 or 3. The words $LvR, v \in \Delta \cup Q \cup \{\rangle\}$ produced in node 3 are sent out and cannot pass any input filter. In node 2, only strings involved in the derivation are produced and sent to node 3. The word representing the previous configuration is sent out, but cannot enter any node, so it is lost.

Thus, Γ accepts on input $\langle w\rangle$ if and only if M reaches a final state on input w. □

Note that the ANSP constructed above uses only input filters. This means we can state the following normal form for ANSPs.

Corollary 1. *For each ANSP Γ there exists an equivalent ANSP Γ' such that for each node x of Γ, $PO_x = FO_x = \emptyset$.*

4 Complexity Results

In [6] it is shown that ANSPs of size 7 can accept languages in **NP** in polynomial time. Since in our construction we simulated a deterministic Turing machine, no such claim about ANSPs of three nodes can be derived from Theorem 1. However, in this section we show that all **NP**-problems can be decided in polynomial time by ANSPs of size 3.

First, we present a more involved construction of an ANSP with 3 nodes that can simulate in parallel the computations of non-deterministic Turing machines. We will use a variation of the rotate-and-simulate technique introduced in [11] and the 'counting-down' mechanism often used in proofs for test-tube systems and time-varying H systems, e.g. [3,12,9].

Lemma 1. *For any non-deterministic Turing machine M we can construct an ANSP Γ of size 3 such that $L(\Gamma) = L(M)$ and Γ simulates all computations of M on a given word in parallel.*

Proof. Let $M = (Q, V, \Delta, q_0, \{q_f\}, \delta, B)$ be a non-deterministic Turing machine, with Q the set of states, V and Δ respectively the input and tape alphabet, q_0 the initial state, B the blank symbol and $\delta : Q \times \Delta \to Q \times \Delta \times \{L, R\}$ the transition function. We assume without loss of generality that M has a semi-infinite tape and a single accepting state q_f.

Let $|\Delta \cup V| = n$ and $K = \{\langle^i, \rangle^i, \langle^{i'}, \rangle^{i'} \mid 0 \le i \le 2n\}$. We assume some ordering such that $\Delta \cup V = \{k_1, ..., k_n\}$ and each k_i identifies a unique element of $\Delta \cup V$. We construct the ANSP $\Gamma = (V, U, \langle, \rangle', G, \mathcal{N}, \alpha, 3, 1)$, where G is the complete graph with 3 nodes, $U = V \cup \Delta \cup K \cup \{Z, E, \langle, \rangle, \langle', \rangle'\}$, all $\alpha_i = w, 1 \le i \le 3$ and:

- $S_1 = \emptyset$
  $A_1 = \emptyset$
  $PI_1 = \{q_f\}$
  $FI_1 = \{Z\} \cup \{\langle^i, \langle^{i'}, \rangle^i, \rangle^{i'} \mid 0 \le i \le 2n\}$
  $PO_1 = FO_1 = \emptyset$

In what follows $a, b, c, d \in V \cup \Delta \cup \{\langle, \rangle\}$, $m \in V \cup \Delta \cup Q$ and $q, q_i, q_j \in Q$.

- $S_2 = \{$
  (simulate)
  $\langle q_i a\#c\$\langle' b q_j \#Z$ for $(q_i, a) \to (q_j, b, R) \in \delta$

$\langle^{0'} q_i a\#c\$\langle' b q_j\#Z$ for $(q_i, a) \to (q_j, b, R) \in \delta$
$\langle c q_i a\#d\$\langle' cb\#Z$ for $(q_i, a) \to (q_j, b, L) \in \delta$
$\langle^{0'} c q_i a\#d\$\langle' cb\#Z$ for $(q_i, a) \to (q_j, b, L) \in \delta$
(right end of tape)
$c\#qE\rangle\$Z\#qBE\rangle$
(start rotate)
$\langle q\#c\$\langle^i k_i q\#Z$ for $1 \le i \le n$
$c\#k_i\rangle\$Z\#\rangle^i$ for $1 \le i \le n$
$\langle k_{i-n}\#qc\$\langle^i\#Z$ for $n+1 \le i \le 2n$
$c\#\rangle\$Z\#k_{i-n}\rangle^i$ for $n+1 \le i \le 2n$
(decrease counter)
$\langle^{i'}\#m\$\langle^{i-1}\#Z$ for $1 \le i \le 2n$
$a\#\rangle^{i'}\$Z\#\rangle^{i-1}$ for $1 \le i \le 2n\}$
$A_2 = \{\langle' b q_j Z \mid q_i, a) \to (q_j, b, R) \in \delta\} \cup \{\langle' cbZ \mid q_i, a) \to (q_j, b, L) \in \delta\}$
$\cup\{\langle^i k_i qZ, Z\rangle^i \mid 1 \le i \le n\} \cup \{\langle^i Z, Zk_i\rangle^i \mid n+1 \le i \le 2n\} \cup \{ZqBE\rangle \mid q \in Q\} \cup \{\langle^i Z, Z\rangle^i \mid 0 \le i \le 2n-1\}$
$PI_2 = \{\langle, \langle^{0'}\} \cup \{\rangle^{i'} \mid 1 \le i \le 2n\}$
$FI_2 = \{Z\}$
$PO_2 = \{\rangle\} \cup \{\langle^i \mid 0 \le i \le 2n\}$
$FO_2 = \emptyset$

– $S_3 = \{$
(initialise)
$\langle\#ab\$\langle q_0\#Z$
$\lambda\#\rangle'\$Z\#E\rangle$
(return simulate)
$\langle'\#m\$\langle\#Z$
(prime counter)
$\langle^i\#m\$\langle^{i'}\#Z$ for $0 \le i \le 2n$
$m\#\rangle^i\$Z\#\rangle^{i'}$ for $1 \le i \le 2n$
(resume computation)
$m\#\rangle^0\$Z\#\rangle\}$
$A_3 = \{\langle Z, \langle q_0 Z, ZE\rangle, Z\rangle\} \cup \{Z\rangle i' \mid 1 \le i \le 2n\}$
$\cup\, \{\langle^{i'} Z \mid 0 \le i \le 2n\}$
$PI_3 = \{\langle'\} \cup \{\rangle^i \mid 0 \le i \le 2n\}$
$FI_3 = \{Z\}$
$PO_3 = \{\rangle\} \cup \{\langle^{i'} \mid 1 \le i \le 2n\}$
$FO_3 = \emptyset$

We show that $\langle w\rangle'$ is accepted by Γ if and only if w is accepted by M[2]. We start with the 'if'-part. The computation of Γ begins with the initialisation phase. The word $\langle w\rangle'$ is converted to $\langle q_0 wE\rangle'$ by the rule $\langle\#ab\$\langle q_0\#Z$. This word cannot pass PO_3 so it stays in node 3. Next, $\lambda\#\rangle'\$Z\#E\rangle$ is applied to give $\langle q_0 wE\rangle$. The symbol E denotes the end of the tape of M. This string is passed to node 2. This concludes the initialisation phase.

[2] Note that we use $\rangle'$ for the input. This is turned into $\rangle$ in the initialisation phase. We prefer the unusual input in order to avoid a proliferation of primes in the proof.

In node 2, we have two possibilities, simulating a move of M or rotating a symbol. We can simulate a right move $(q_i, a) \rightarrow (q_j, b, R)$ by applying the rule $\langle q_i a\#c\$\langle' bq_j\#Z$. Starting from a string $\langle q_i aw\rangle$ this yields $\langle' bq_j w\rangle$. This string is sent out to node 3, where it is converted to $\langle bq_j w\rangle$ by $\langle'\#m\$\langle\#Z$, which is passed back to node 2. Thus we have the desired result of the move. If we have a word $\langle cq_i aw\rangle$ and $(q_i, a) \rightarrow (q_j, b, L)$ is a valid move in M, we obtain the desired result $\langle q_j cbw\rangle$ in node 2 by applying $\langle cq_i a\#d\$\langle' cb\#Z$ and, in node 3, $\langle'\#m\$\langle\#Z$.

Note that we simulate the moves of M only at the left end of the string. This means that if we have a sequence of two left moves or two right moves, we need to rotate symbols to make the simulation possible. For instance, from a string $\langle dqaw\rangle$ we need to go to $\langle qawd\rangle$ to allow for the simulation of a move $(q, a) \rightarrow (q_j, b, R)$. This rotating is done as follows. Suppose $d = k_i$. Then in node 2, we can apply a rule $c\#k_i\rangle\$Z\#\rangle^i$ to convert $\langle dqaw\rangle$ to $\langle dqawd\rangle^i$. This string does not pass the output filter, so it stays in node 2. In the next step, $\langle dqawd\rangle^i$ becomes $\langle^i qawd\rangle^i$ by a rule $\langle q\#c\$\langle^i k_i q\#Z$. Now the counter is progressively lowered by moving the string back and forth between nodes 2 and 3. The word $\langle^i qawd\rangle^i$ passes to node 3, where a rule $m\#\rangle^i\$Z\#\rangle i'$ gives $\langle^i qawd\rangle^{i'}$ (remains in node 3) and $\langle^i\#m\$\langle^{i'}\#Z$ yields $\langle^{i'} qawd\rangle^{i'}$ (to node 2). In node 2, rules $a\#\rangle^{i'}\$Z\#\rangle^{i-1}$ (result remains in 2) and $\langle^{i'}\#m\$\langle^{i-1}\#Z$ give $\langle^{i-1} qawd\rangle^{i-1}$, which passes to node 3. This process is repeated until arriving at $\langle^0 qawd\rangle^0$. This string arrives in node 3, where $\langle^i\#m\$\langle^{i'}\#Z$ (result remains) and $m\#\rangle^0\$Z\#\rangle$ convert the string to $\langle^{0'} qawd\rangle$. This string is passed to node 2, where a simulation rule $\langle^{0'} q_i a\#c\$\langle' bq_j\#Z$ could be applied. Rotating a symbol to the left is done similarly. To avoid interference between right and left rotation of the same symbol, for the left rotation of symbol k_i we use counter $i + n$. Note that the rotated strings begin with the symbol $\langle^{0'}$. For strings starting with $\langle^{0'}$, S_2 contains the same simulation rules as for $\langle$, but no rotation rules. This avoids fruitless rotations, since we know that the head of a Turing machine moves at most one symbol per move.

The special symbol E cannot be rotated, which enforces that the head of M never goes beyond the left end of the tape. If M needs more tape cells on the right, a rule $c\#qE\rangle\$Z\#qBE\rangle$ can be applied to insert a blank symbol between the head and the end-of-tape symbol. The result of this does not leave node 2, so simulation can continue as described above. All necessary auxiliary strings, all containing the symbol Z, for the described process are present in the sets A_2 and A_3. Finally, if M reaches a final state, the simulation in Γ will yield a word of the form $\langle wq_f w'\rangle$ in node 3. This word passes the output filters and the input filters of node 1, causing Γ to accept.

For the 'only if'-part of the proof, we show that only by the way described above we can get an accepting computation of Γ. First of all, in the initialisation phase we could apply the rule $\lambda\#\rangle'\$Z\#E\rangle$ first. The result $\langle wE\rangle$ can pass to node 2. However, since there is no state symbol, no simulation or acceptance is possible. We can only rotate, but after arriving at a string $\langle^{0'} v\rangle$ no further rule applications are possible. For the simulation steps, they can clearly only give the described result. Also in node 3, there is only one applicable rule. The old configuration can leave node 2, but not enter any other node.

For the rotation, the decreasing procedure involves two splicing steps in the same node before being passed to the other node. If we apply these two rules in the inverse order with respect to our description above, it is easy to verify that we get a string that can leave the node, but not enter the other node. Now, when starting the rotation phase, it can be that the new symbol we attach to one side does not correspond to the symbol removed on the other side. We show that these incorrect rotations do not lead to accepting computations. Suppose we have $\langle qawk_j\rangle$ and we apply $\langle q\#c\$\langle^i k_i q\#Z$ for some $i \neq j$. This means we get $\langle^i k_i qaw\rangle^j$ for $i \neq j$ in the next step. For rotation to the right we can get $\langle^i wk_j\rangle^j$ for $i \neq j$. Then we can have two cases:

$i < j$ If $i < j$, by the decreasing procedure, we arrive at a string $\langle^0 v\rangle^l$ in node 2, with $l > 0$. This string is passed to node 3. There we can apply $m\#\rangle^i\$Z\#\rangle i'$ to give $\langle^0 v\rangle^{l'}$. This string cannot pass the output filter, so it stays in node 3. Then, the only applicable rule is $\langle^i\#m\$\langle^{i'}\#Z$. This gives $\langle^{0'} v\rangle^{l'}$. This string cannot leave node 3 and no more rules can be applied to it, so it is 'trapped' and can't lead to acceptance. If the two rules are applied in the inverse order, we first get $\langle^{0'} v\rangle^l$, which cannot pass the output filter, and then the same string $\langle^{0'} v\rangle^{l'}$.

$i > j$ If $i > j$, we arrive at a string $\langle^l v\rangle^0, l > 0$ in node 2. This string passes to node 3, where two rules can apply. If we apply a rule $\langle^i\#m\$\langle^{i'}\#Z$, we get $\langle^{l'} v\rangle^0$. Applying $m\#\rangle^0\$Z\#\rangle$ gives $\langle^l v\rangle$. Both these strings can pass the output filter, but no input filter, so they are lost.

This shows that only correct simulations of M can lead to acceptance, and thus $L(\Gamma) = L(M)$. Moreover, whenever there is more than one move available to M on a given computation, Γ will simulate all moves, using the multiplicity inherent in the model.

□

Theorem 2. *All languages in **NP** can be decided by ANSPs of size 3 in polynomial time.*

Proof. If a language L is in **NP**, there exists a non-deterministic Turing machine M accepting all $w \in L$ in time $f(|w|)$, where $f(n)$ is a polynomial function. Moreover, there exists a $f(n)$-bounded Turing machine M' which simulates M but halts after $f(n)$ steps. By Lemma 1, this means that there exists an ANSP Γ deciding L. Moreover, for each move of M', Γ needs 4 steps for simulation, at most 1 step for expanding the work tape and at most $4 \cdot 2 \cdot m$ rotation steps, where $m = |V \cup \Delta|$ and V and Δ are the input and tape alphabet of M. Thus, $TIME_\Gamma(w) \leq 40m \cdot f(|w|)+1$ for all w (the term +1 comes from our assumption in Lemma 1 that M' has a single final state). This is clearly polynomial. □

5 Final Remarks

As mentioned in the introduction, acceptance in ANSPs is defined in such a way that at least two nodes are needed to accept any non-trivial language. This means that only the computational power of ANSPs of size 2 is still open.

Our normal form for ANSPs also points at other directions of research. Can we find reduced types of ANSPs which are still universal? In Theorem 1, forbidding filters are only used in one place. Can we find complete systems with only permitting filters? For such reduced systems it would be interesting to investigate which price we have to pay in terms of size of the systems, or in computational resources.

References

1. Castellanos, J., Martin-Vide, C., Mitrana, V., Sempere, J.: Networks of evolutionary processors. Acta. Informatica 39, 517–529 (2003)
2. Castellanos, J., Leupold, P., Mitrana, V.: Descriptional and computational complexity aspects of hybrid networks of evolutionary processors. Theoretical Computer Science 330, 205–220 (2005)
3. Freund, R., Csuhaj-Varjú, E., Wachtler, F.: Test Tube Systems with Cutting/Recombination Operations. In: Pacific Symposium on BIOCOMPUTING'97, pp. 163–175. World Scientific, Singapore (1997)
4. Hopcroft, J E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation (1979)
5. Manea, F., Martin-Vide, C., Mitrana, V.: Accepting networks of splicing processors: Complexity results. Theoretical Computer Science (CiE 2005 special issue) 371(1-2), 72–82 (2007)
6. Manea, F., Martin-Vide, C., Mitrana, V.: All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size. In: Mao, C., Yokomori, T. (eds.) DNA Computing. LNCS, vol. 4287, pp. 47–57. Springer, Heidelberg (2006)
7. Margenstern, M., Mitrana, V., Perez-Jimenez, M.: Accepting hybrid networks of evolutionary systems. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 235–246. Springer, Heidelberg (2005)
8. Margenstern, M., Rogozhin, Y.: Time-varying distributed H systems of degree 2 generate all recursively enumerable languages. In: Martin-Vide, C., Mitrana, V. (eds.) Where Mathematics, Computer Science and Biology Meet, Kluwer Academic, Dordrecht (2000)
9. Margenstern, M., Rogozhin, Y., Verlan, S.: Time-varying distributed H systems of degree 2 can carry out parallel computations. In: Hagiya, M., Ohuchi, A. (eds.) DNA8. LNCS, vol. 2568, pp. 326–336. Springer, Heidelberg (2003)
10. Papadimitriou, C.H.: Computational Complexity (1994)
11. Păun, Gh.: Regular extended H systems are computationally universal. J. Automata, languages, Combinatorics 1(1), 27–36 (1996)
12. Păun, A.: On time-varying H systems. Bulletin of the EATCS 67, 157–164 (1999)

Liquid Computing

Wolfgang Maass[*]

Institute for Theoretical Computer Science
Technische Universitaet Graz
A-8010 Graz, Austria
maass@igi.tugraz.at
http://www.igi.tugraz.at/

Abstract. This review addresses structural differences between that type of computation on which computability theory and computational complexity theory have focused so far, and those computations that are usually carried out in biological organisms (either in the brain, or in the form of gene regulation within a single cell). These differences concern the role of time, the way in which the input is presented, the way in which an algorithm is implemented, and in the end also the definition of what a computation is. This article describes liquid computing as a new framework for analyzing those types of computations that are usually carried out in biological organisms.

1 Introduction

The computation of a Turing machine always begins in a designated initial state q_0, with the input x (some finite string of symbols from some finite alphabet) written on some designated tape. The computation runs until a halt-state is entered (the inscription y of some designated tape segment is then interpreted as the result of the computation). This is a typical example for an *offline computation*, where the complete input x is available at the beginning of the computation, and no trace of this computation, or of its result y, is left when the same Turing machine subsequently carries out another computation for another input $\tilde{x}$ (starting again in state q_0). In contrast, the result of a typical computation in the neuronal system of a biological organism, say the decision about the location y on the ground where the left foot is going to be placed at the next step (while walking or running), depends on several pieces of information: on information from the visual system, from the vestibular system which supports balance control, from the muscles (proprioceptive feedback about their current state), from short term memory (how well did the previous foot placement work?), from long term memory (how slippery is this path at the current weather condition?), from brain systems that have previously decided where to go and at what speed, and on information from various other parts of the neural system. In general these

[*] Partially supported the Austrian Science Fund FWF, project P17229 and project S9102-N13, and by the European Union, project FP6-015879 (FACETS).

diverse pieces of information arrive at different points in time, and the computation of y has to start before the last one has come in. Furthermore, new information (e.g. visual information and proprioceptive feedback) arrives continuously, and it is left up to the computational system how much of it can be integrated into the computation of the position y of the next placement of the left foot (obviously those organisms have a better chance to survive which also can integrate later arriving information into the computation). Once the computation of y is completed, the computation of the location y' where the right foot is subsequently placed is not a separate computation, that starts again in some neutral initial state q_0. Rather, it is likely to build on pieces of inputs and results of subcomputations that had already been used for the preceding computation of y.

The previously sketched computational task is a typical example for an *online computation* (where input pieces arrive all the time, not in one batch). Furthermore it is an example for a *real-time computation*, where one has a strict deadline by which the computation of the output y has to be completed (otherwise a 2-legged animal would fall). In fact, in some critical situations (e. g. when a 2-legged animal stumbles, or hits an unexpected obstacle) a biological organism is forced to apply an *anytime algorithm*, which tries to make optimal use of intermediate results of computational processing that has occurred up to some externally given time point t_0 (such forced halt of the computation could occur at "any time"). Difficulties in the control of walking for 2-legged robots have taught us how difficult it is to design algorithms which can carry out this seemingly simple computational task. In fact, this computational problem is largely unsolved, and humanoid robots can only operate within environments for which they have been provided with an accurate model. This is perhaps surprising, since on the other hand current computers can beat human champions in seemingly more demanding computational tasks, such as winning a game of chess. One might argue that one reason, why walking in a new terrain is currently a computationally less solved task, is that computation theory and algorithm design have focused for several decades on offline computations, and have neglected seemingly mundane computational tasks such as walking. This bias is understandable, because evolution had much more time to develop a computational machinery for the control of human walking, and this computational machinery works so well that we don't even notice anymore how difficult this computational task is.

2 Liquid State Machines

A computation machine M that carries out online computations typically computes a function F that does not map input numbers or (finite) bit strings onto output numbers or bit strings, but input streams onto output streams. These input- and output streams are usually encoded as functions $u : \mathbb{Z} \rightarrow \mathbb{R}^n$ or $u : \mathbb{R} \rightarrow \mathbb{R}^n$, where the argument t of $u(t)$ is interpreted as the (discrete or continuous) time point t when the information that is encoded by $u(t) \in \mathbb{R}^n$ becomes available. Hence such computational machine M computes a function of higher type (usually referred to as operator, functional, or filter), that maps input functions u from some domain U onto output functions y. In lack of a better term we will use the term filter in this article, although filters are often associated with somewhat trivial signal processing or preprossessing devices. However, one should not fall into the trap of identifying the general term of a filter with special classes of filters such as linear filters. Rather one should keep in mind that any input to any organism is a function of time, and any motor output of an organism is a function of time. Hence biological organisms compute filters. The same holds true for any artificial behaving system, such as a robot.

We will only consider computational operations on functions of time that are input-driven, in the sense that the output does not depend on any absolute internal clock of the computational device. Filters that have this property are called time invariant. Formally one says that a filter F is *time invariant* if any temporal shift of the input function $u(\cdot)$ by some amount t_0 causes a temporal shift of the output function by the same amount t_0, i.e., $(Fu^{t_0})(t) = (Fu)(t+t_0)$ for all $t, t_0 \in \mathbb{R}$, where u^{t_0} is the function defined by $u^{t_0}(t) := u(t + t_0)$. Note that if the domain U of input functions $u(\cdot)$ is closed under temporal shifts, then a time invariant filter $F : U \to \mathbb{R}^{\mathbb{R}}$ is identified uniquely by the values $y(0) = (Fu)(0)$ of its output functions $y(\cdot)$ at time 0. In other words: in order to identify or characterize a time invariant filter F we just have to observe its output values at time 0, while its input varies over all functions $u(\cdot) \in U$. Hence one can replace in the mathematical analysis such filter F by a functional, i.e. a simpler mathematical object that maps input functions onto real values (rather than onto functions of time).

Various theoretical models for analog computing are of little practical use because they rely on hair-trigger decisions, for example they allow that the output is 1 if the value of some real-valued input variable u is ≥ 0, and 0 otherwise. Another unrealistic aspect of some models for computation on functions of time is that they automatically allow that the output of the computation depends on the full infinitely long history of the input function $u(\cdot)$. Most practically relevant models for analog computation on continuous input streams degrade gracefully under the influence of noise, i.e. they have a fading memory. *Fading memory* is a continuity property of filters F, which requires that for any input function $u(\cdot) \in U$ the output $(Fu)(0)$ can be approximated by the outputs $(Fv)(0)$ for any other input functions $v(\cdot) \in U$ that approximate $u(\cdot)$ on a sufficiently long time interval $[-T, 0]$ in the past. Formally one defines that $F : U \to \mathbb{R}^{\mathbb{R}}$ has fading memory if for every $u \in U^n$ and every $\varepsilon > 0$ there exist $\delta > 0$ and $T > 0$ so that $|(Fv)(0) - (Fu)(0)| < \varepsilon$ for all $v \in U$ with $\|u(t) - v(t)\| < \delta$ for all $t \in [-T, 0]$. Informally, a filter F has fading memory if the most significant bits of its current output value $(Fu)(0)$ depend just on the most significant bits of the values of its input function $u(\cdot)$ in some finite time interval $[-T, 0]$. Thus, in order to compute the most significant bits of $(Fu)(0)$ it is not necessary to know the *precise* value of the input function $u(s)$ for any time s, and it is also not necessary to have knowledge about values of $u(\cdot)$ for more than a finite time interval back into the past.

The universe of time-invariant fading memory filters is quite large. It contains all filters F that can be characterized by Volterra series, i.e. all filters F whose output $(Fu)(t)$ is given by a finite or infinite sum (with $d = 0, 1, \ldots$) of terms of the form $\int\limits_0^\infty \ldots \int\limits_0^\infty h_d(\tau_1, \ldots, \tau_d) \cdot u(t - \tau_1) \cdot \ldots \cdot u(t - \tau_d) d\tau_1 \ldots d\tau_d$, where some integral kernel h_d is applied to products of degree d of the input stream $u(\cdot)$ at various time points $t - \tau_i$ back in the past. In fact, under some mild conditions on the domain U of input streams the class of time invariant fading memory filters coincides with the class of filters that can be characterized by Volterra series.

In spite of their complexity, all these filters can be uniformly approximated by the simple computational models M of the type shown in Fig. 1, which had been introduced in [1]:

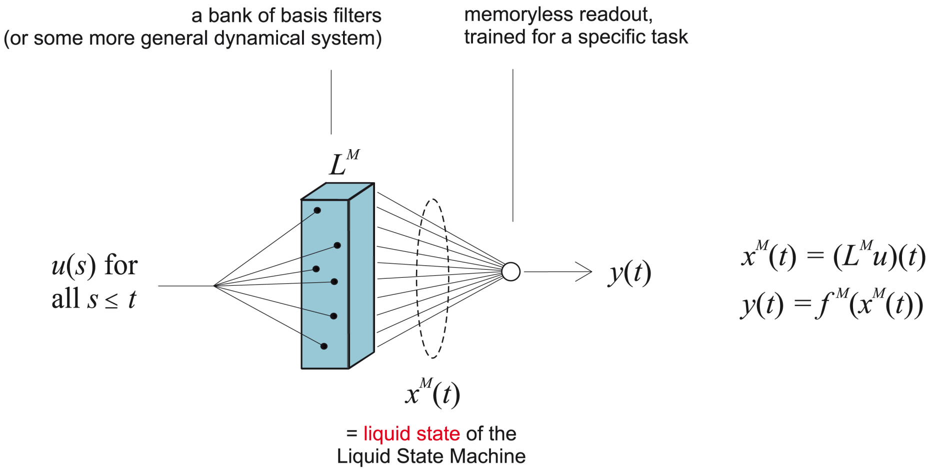


Fig. 1. Structure of a Liquid State Machine (LSM) M, which transforms input streams $u(\cdot)$ into output streams $y(\cdot)$. If L^M is a bank of k basis filters, the "liquid state" $\boldsymbol{x}^M(t) \in \mathbb{R}^k$ is the output of those k filters at time t. If L^M is a more general dynamical system, $\boldsymbol{x}^M(t)$ is that part of its current internal state that is "visible" for a readout.

Theorem 1. (based on [2]; see Theorem 3.1 in [3] for a detailed proof). *Any filter F defined by a Volterra series can be approximated with any desired degree of precision by the simple computational model M shown in Fig. 1*

– *if there is a rich enough pool* $\mathbf{B}$ *of basis filters (time invariant, with fading memory) from which the basis filters $B_1, \ldots, B_k$ in the filterbank L^M can be chosen (*$\mathbf{B}$ *needs to have the pointwise separation property) and*
– *if there is a rich enough pool* $\mathbf{R}$ *from which the readout functions f can be chosen (*$\mathbf{R}$ *needs to have the universal approximation property, i.e. any*

continuous function on a compact domain can be uniformly approximated by functions from $\mathbf{R}$*).*

Definition: *A class* $\mathbf{B}$ *of basis filters has the pointwise separation property if there exists for any two input functions* $u(\cdot), v(\cdot)$ *with* $u(s) \neq v(s)$ *for some* $s \leq t$ *a basis filter* $B \in \mathbf{B}$ *with* $(Bu)(t) \neq (Bv)(t)$*.*

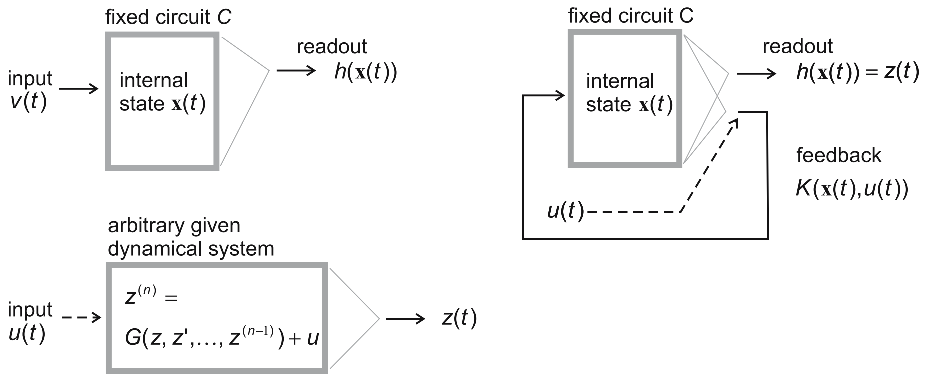
It turns out that many real-world dynamical systems (even a pool of water) satisfy (for some domain U of input streams) at least some weak version of the pointwise separation property, where the outputs $\boldsymbol{x}^M(t)$ of the basis filters are replaced by some "visible" components of the state vector of the dynamical system. In fact, many real-world dynamical systems also satisfy approximately an interesting kernel property[1], which makes it practically sufficient to use just a *linear* readout function f^M. This is particularly important if L^M is kept fixed, and only the readout f^M is selected (or trained) in order to approximate some particular Volterra series F. Reducing the adaptive part of M to the *linear* readout function f^M has the unique advantage that a learning algorithm that uses gradient descent to minimize the approximation error of M cannot get stuck in local minima of the mean-squared error. The resulting computational model can be viewed as a generalization of a finite state machine to continuous time and continuous ("liquid") internal states $\boldsymbol{x}^M(t)$. Hence it is called a Liquid State Machine (LSM)[2].

If the dynamical systems L^M have fading memory, then only filters with fading memory can be represented by the resulting LSM's. Hence they cannot approximate arbitrary finite state machines (not even for the case of discrete time and a finite range of values $u(t)$). It turns out that a large jump in computational power occurs if one augments the computational model from Fig. 1 by a feedback from a readout back into the circuit (assume it enters the circuit like an input variable).

Theorem 2. [5]. *There exists a large class* S_n *of dynamical systems* C *with fading memory (described by systems of* n *first order differential equations) that acquire through feedback universal computational capabilities for analog computing. More precisely: through a proper choice of a (memoryless) feedback function* K *and readout* h *they can simulate any given dynamical system of the form* $z^{(n)} = G(z, z', \ldots, z^{(n-1)}) + u$ *with a sufficiently smooth function* G*:*

1 A kernel (in the sense of machine learning) is a nonlinear projection Q of n input variables $u_1, \ldots, u_n$ into some high-dimensional space. For example all products $u_i \cdot u_j$ could be added as further components to the n-dimensional input vector $< u_1, \ldots, u_n >$. Such nonlinear projection Q boosts the power of any *linear* readout f applied to $Q(\mathbf{u})$. For example in the case where $Q(\mathbf{u})$ contains all products $u_i \cdot u_j$, a subsequent linear readout has the same expressive capability as quadratic readouts f applied to the original input variables $u_1, \ldots, u_n$. More abstractly, Q should map all inputs $\mathbf{u}$ that need to be separated by a readout onto a set of linearly independent vectors $Q(\mathbf{u})$.

2 Herbert Jaeger (see [4]) had simultaneously and independently introduced a very similar computational model under the name Echo State Network.


fixed circuit C
input $v(t)$
internal state $\mathbf{x}(t)$
readout $h(\mathbf{x}(t))$
arbitrary given dynamical system
input $u(t)$
$z^{(n)} = G(z, z', \ldots, z^{(n-1)}) + u$
$z(t)$
fixed circuit C
internal state $\mathbf{x}(t)$
readout $h(\mathbf{x}(t)) = z(t)$
feedback $K(\mathbf{x}(t), u(t))$
$u(t)$

This holds in particular for neural circuits C defined by differential equations of the form $x_i'(t) = -\lambda_i x_i(t) + \sigma(\sum_{j=1}^{n} a_{ij} x_j(t)) + b_i \cdot \sigma(v(t))$ (under some conditions on the λ_i, a_{ij}, b_i).

If one allows several feedbacks K, such dynamical systems C become universal for n^{th} order dynamical systems defined by a system consisting of a corresponding number of differential equations. Since such systems of differential equations can simulate arbitrary Turing machines [6], these dynamical systems C with a finite number of feedbacks become (according to the Church-Turing thesis) also universal for *digital computation*.

Theorem 2 suggests that even quite simple neural circuits with feedback have in principle unlimited computational power[3]. This suggests that the main problem of a biological organism becomes the *selection* (or learning) of suitable feedback functions K and readout functions h. For dynamical systems C that have a good kernel-property, already *linear* feedbacks and readouts endow such dynamical systems with the capability to emulate a fairly large range of other dynamical systems (or "analog computers").

3 Applications

LSMs had been introduced in [1] (building on preceding work in [7]) in the process of searching for computational models that can help us to understand the computations that are carried out in a "cortical microcircuit" [8], i.e. in a local circuit of neurons in the neocortex (say in a "cortical column"). This approach has turned out to be quite successful, since it made it possible to carry out quite demanding computations with circuits consisting of reasonably realistic models for biological neurons ("spiking neurons") and biological synapses ("dynamical synapses"). Note that in this model a large number of different readout neurons can learn to extract different information from the same circuit. One concrete benchmark task that has been considered was the classification ("recognition") of

[3] Of course, in the presence of noise this computational power is reduced to that of a finite state machine, see [5] for details.

spoken digits [9]. It turned out that already a LSM where the “liquid” consisted of a randomly connected circuit of just 135 spiking neurons performed quite well. In fact, it provided a nice example for “anytime computations”, since the linear readout could be trained effectively to guess at “any time”, while a digit was spoken, the proper classification of the digit [1,10]. More recently it has been shown that with a suitable transformation of spoken digits into spike trains one can achieve with this simple method the performance level of state-of-the-art algorithms for speech recognition [11].

The same computational task had also been considered in an amusing and inspiring study of the computational capability of LSMs where a bucket of water was used as the “liquid” L^M (into which input streams were injected via 8 motors), and video-images of the surface of the water were used as “liquid states” $\boldsymbol{x}^M(t)$ [12]. Also this realization of a LSM was able to carry out speech recognition, but “fortunately” its performance was below that of a LSM with a simulated circuit of neurons as “liquid”. This experiment raises two questions:

i) Can interesting new artificial computing devises be designed on the basis of the LSM-paradigm?
ii) Can the LSM-paradigm be used to gain an understanding of the computational role of specific (genetically encoded) details of the components and connectivity structure of circuits of neurons in the neocortex?

Research in the direction i) is currently carried out in the context of optical computing. First results regarding question ii) were recently published in [13].

Perhaps the most exciting research on the LSM-approach is currently carried out in experimental neuroscience. The LSM-approach makes specific experimentally testable predictions regarding the organization of computations in cortical circuits:

a) information from subsequent stimuli is superimposed in the “liquid state” of cortical circuits, but can be recovered by simple linear readouts (see the “separation property” in Theorem 1).
b) cortical circuits produce nonlinear combinations of different input components (kernel property)
c) the activity of different neurons within a cortical column in response to natural stimuli shows a large diversity of individual responses (rather than evidence that all neurons within a column carry out a specific common computational operation), since a “liquid” can support many different readouts.

A rather clear confirmation of predictions a) and b) has recently been produced at the Max-Plank Institute for Brain Research in Frankfurt [14]. Some earlier experimental studies had provided already evidence for prediction c), but this prediction needs to be tested more rigorously for natural stimuli.

The exploration of potential engineering applications of the computational paradigm discussed in this article were usually carried out with the closely related echo state networks (ESNs) [4], where one uses simpler non-spiking models for neurons in the “liquid”, and works with high numerical precision in the simulation of the “liquid” and the training of linear readouts (which makes a lot of sense, since artificial circuits are subject to substantial lower amounts of noise in comparison with biological circuits of neurons). Research in recent years has produced quite encouraging results regarding applications of ESNs to problems in tele-communication [4], robotics [15], reinforcement learning [16], natural language understanding [17], as well as music-production and -perception [18].

4 Discussion

We have argued in this article that Turing machines are not well-suited for modeling computations in biological neural circuits, and proposed liquid state machines (LSMs) as a more adequate modeling framework. They are designed to model real-time computations (as well as anytime computations) on continuous input streams. In fact, it is quite realistic that a LSM can be trained to carry out the online computation task that we had discussed in section 1 (see [19] for a first application to motor control). A characteristic feature of practical implementations of the LSM model is that its "program" consists of the weights $\mathbf{w}$ of a linear readout function. Since these weights can be chosen to be time invariant, they provide suitable targets for learning (while all other parameters of the LSM can be fixed in advance, based on the expected complexity and precision requirement of the computational tasks that are to be learnt). It makes a lot of sense (from the perspective of statistical learning theory) to restrict learning to such weights $\mathbf{w}$, since they have the unique advantage that gradient descent with regard to some mean-square error function $E(\mathbf{w})$ cannot get stuck in local minima of this error function (since $\nabla_{\mathbf{w}} E(\mathbf{w}) = \mathbf{0}$ defines an affine – hence connected – subspace of the weight space).

One can view these weights $\mathbf{w}$ of the linear readout of a LSM as an analogon to the code $< M >$ of a Turing machine M that is simulated by a universal Turing machine. This analogy makes the learning advantage of LSMs clear, since there is no efficient learning algorithm known which allows us to learn the program $< M >$ for a Turing machine M from examples for correct input/output pairs of M. However the examples discussed in section 3 show that an LSM can be trained quite efficiently to approximate a particular map from input – to output streams.

We have also shown in Theorem 2 that LSMs can overcome the limitation of a fading memory if one allows feedback from readouts back into the "liquid". Then not only all digital, but (in a well-defined sense) also all analog computers can be simulated by a fixed LSM, provided that one is allowed to vary the readout functions (including those that provide feedback). Hence these readout functions can be viewed as program for the simulated analog computers (note that all "readout functions" are just "static" functions, i.e. maps from $\mathbb{R}^n$ into $\mathbb{R}$, whereas the LSM itself maps input streams onto output streams). In those practically relevant cases that have been considered so far, these readout functions could often be chosen to be linear. A satisfactory characterization of the computational power that can be reached with linear readouts is still missing. But obviously the kernel-property of the underlying "liquid" can boost the richness of the class of analog computers that can be simulated by a fixed LSM with linear readouts.

The theoretical analysis of computational properties of randomly connected circuits and other potential "liquids" is still in its infancy. We refer to [20] for a very useful first step. The qualities that we expect from the "liquid" of a LSM are completely different from those that one expects from standard computing devices. One expects diversity (rather than uniformity) of the responses of individual gates within a liquid (see Theorem 1), as well as diverse local dynamics instead of synchronized local gate operations. Achieving such diversity is apparently easier to attain by biological neural circuits and by new artificial circuits on the molecular or atomic scale, than emulating precisely engineered circuits of the type that we find in our current generation of digital computers, which only function properly if all local units are identical copies of a small number of template units that respond in a stereotypical fashion. In addition, sparse random connections turn out to provide better computational capabilities to a LSM than those connectivity graphs that have primarily been considered in theoretical studies, such as all-to-all connections (Hopfield networks) or a 2-dimensional grid (which is commonly used for cellular automata).

We refer to the 2007 Special Issue on Echo State Networks and Liquid State Machines of the journal Neural Networks for an up-to-date overview of further theoretical results and practical applications of the computational ideas presented in this article.

References

1. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation 14(11), 2531–2560 (2002)
2. Boyd, S., Chua, L.O.: Fading memory and the problem of approximating nonlinear oparators with Volterra series. IEEE Trans. on Circuits and Systems 32, 1150–1161 (1985)
3. Maass, W., Markram, H.: On the computational power of recurrent circuits of spiking neurons. Journal of Computer and System Sciences 69(4), 593–616 (2004)
4. Jäger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science 304, 78–80 (2004)
5. Maass, W., Joshi, P., Sontag, E.D.: Computational aspects of feedback in neural circuits. PLOS Computational Biology 3(1), e165, 1–20 (2007)
6. Branicky, M.S.: Universal computation and other capabilities of hybrid and continuous dynamical systems. Theoretical Computer Science 138, 67–100 (1995)
7. Buonomano, D.V., Merzenich, M.M.: Temporal information transformed into a spatial code by a neural network with realistic properties. Science 267, 1028–1030 (1995)
8. Maass, W., Markram, H.: Theory of the computational function of microcircuit dynamics. In: Grillner, S., Graybiel, A.M. (eds.) The Interface between Neurons and Global Brain Function, Dahlem Workshop Report 93, pp. 371–390. MIT Press, Cambridge (2006)

9. Hopfield, J.J., Brody, C.D.: What is a moment? Transient synchrony as a collective mechanism for spatio-temporal integration. Proc. Nat. Acad. Sci. USA 98(3), 1282–1287 (2001)
10. Maass, W., Natschläger, T., Markram, H.: Fading memory and kernel properties of generic cortical microcircuit models. Journal of Physiology – Paris 98(4–6), 315–330 (2004)
11. Verstraeten, D., Schrauwen, B., Stroobandt, D., Van Campenhout, J.: Isolated word recognition with the liquid state machine: a case study. Information Processing Letters 95(6), 521–528 (2005)
12. Fernando, C., Sojakka, S.: Pattern recognition in a bucket: a real liquid brain. In: Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., Kim, J.T. (eds.) ECAL 2003. LNCS (LNAI), vol. 2801, pp. 588–597. Springer, Heidelberg (2003)
13. Häusler, S., Maass, W.: A statistical analysis of information processing properties of lamina-specific cortical microcircuit models. Cerebral Cortex 17(1), 149–162 (2007)
14. Nikolić, D., Haeusler, S., Singer, W., Maass, W.: Temporal dynamics of information content carried by neurons in the primary visual cortex. In: Proc. of NIPS 2006, Advances in Neural Information Processing Systems, vol. 19, MIT Press, Cambridge (2007)
15. Hertzberg, J., Jaeger, H., Schoenherr, F.: Learning to ground fact symbols in behavior-based robot. In: van Harmelen, F. (ed.) Proc. of the 15th European Conference on Artificial Intelligence, pp. 708–712. IOS Press, Amsterdam (2002)
16. Bush, K., Anderson, C.: Modeling reward functions for incomplete state representations via echo state networks. In: Proceedings of the International Joint Conference on Neural Networks, Montreal, Quebec (2005)
17. Tong, M.H., Bickett, A.D., Christiansen, E.M., Cotrell, G.W.: Learning grammatical structure with echo state networks. Neural Networks (in press 2007)
18. Jaeger, H., Eck, D.: Can't get you out of my head: A connectionist model of cyclic rehearsal. In: Wachsmuth, I., Knoblich, G (eds.) Modeling Communication with Robots and Virtual Humans (in press 2007)
19. Joshi, P., Maass, W.: Movement generation with circuits of spiking neurons. Neural Computation 17(8), 1715–1738 (2005)
20. White, O.L., Lee, D.D., Sompolinsky, H.: Short-term memory in orthogonal neural networks. Phys. Rev. Letters 92(14), 102–148 (2004)

Quotients over Minimal Type Theory

Maria Emilia Maietti

Dipartimento di Matematica
Università di Genova, Italy
maietti@math.unipd.it

Abstract. We consider an *extensional* version, called qmTT, of the *intensional* Minimal Type Theory mTT, introduced in a previous paper with G. Sambin, enriched with proof-irrelevance of propositions and effective quotient sets. Then, by using the construction of total setoid à la Bishop we build a model of qmTT over mTT.

The design of an extensional type theory with quotients and its interpretation in mTT is a key technical step in order to build a two level system to serve as a minimal foundation for constructive mathematics as advocated in the mentioned paper about mTT.

Keywords: Dependent type theory, intuitionistic logic, quotient completion.

1 Introduction

In [MS05] we argued for the need of a minimal foundation for constructive mathematics. We wanted this theory to be minimal among relevant constructive foundations such as the generic internal theory of a topos and Martin-Löf's type theory, besides the classical Zermelo-Fraenkel set theory. Then, being Martin-Löf's type theory *predicative*, our theory must be predicative, too.

The *constructivity* of our foundation is expressed by the fact that it satisfies the *proofs-as-programs* paradigm. In [MS05] we motivated that an essential characteristic of a proofs-as-programs theory must be its consistency with the axiom of choice and the formal Church thesis altogether. In other words, the proofs-as-programs theory must be equipped with a realizability model where the extraction of programs from proofs is internalized by validating the axiom of choice and the formal Church thesis as internal theorems.

But, then, it turned out that this requirement is so strong to be incompatible with another desirable feature that a foundation for mathematics should have, namely with the capability of representing extensional concepts as those used in everyday mathematics. In other terms, as reported in [MS05], we cannot have an extensional theory that is also proofs-as-programs in our sense. Indeed, it is well-known that extensionality of functions is inconsistent with the axiom of choice and the formal Church thesis altogether. And, for an extensional constructive theory even the consistency with the axiom of choice alone can be a problem.

Indeed, while the axiom of choice is an accepted principle in intensional type theory, it is not generally validated in an extensional constructive one since it may force the theory to be classical (see for example [Mai99], [MV99], [Car04], [ML06]). This is simply because the choice function cannot be always guaranteed to be extensional constructively.

The solution proposed in [MS05] to the problem of building an extensional proofs-as-programs foundation consists in building a *two level* theory: one should start with an intensional proofs-as-programs theory in the above sense and then build an extensional level upon it according to the forget-restore principle in [SV98]. This principle says that extensional concepts must be designed by abstracting on the intension of their representations at the intensional level in such a way that all the forgotten computational information of their representations can be restored at will. An example of this is the design of the many-sorted logic obtained from a type theory by using Martin-Löf's *true*-judgements (see [Mar84], [SV98]). In this way proofs at the extensional level are turned into proofs at the intensional level that correspond to programs. We then decided to name *programs level* the intensional one and *proofs level* the extensional one to express that the link between the two levels is also part of the proofs-as-programs transformation.

To serve as the intensional level of the minimal foundation advocated in [MS05], we there introduced Minimal Type Theory (mTT). This is obtained by extending the set constructors of intensional Martin-Löf's type theory in [NPS90] with a primitive notion of propositions. The main difference between our theory and Martin-Löf's one is that the axiom of choice and, even the axiom of unique choice are not valid theorems in mTT.

Here we assume the *extensional* level to be given by a type theory, called qmTT. This is obtained as follows. First, we take the extensional version of our Minimal Type Theory, in the same way as the type theory in [Mar84] is the extensional version of intensional Martin-Löf's type theory in [NPS90]. Then we collapse propositions into mono sets in the sense of [Mai05] and, finally, we add effective quotient sets similarly to those in [Mai05].

In order to interpret our extensional type theory qmTT in mTT we build a category Q(mTT) of total setoids, whose objects and morphisms coincide with the notion of sets and functions given by E. Bishop [Bis67].

Q(mTT) turns out to be a categorical model of qmTT. Categorically speaking, it turns out to be a lextensive list-arithmetic locally cartesian closed category with stable effective quotients of equivalence relations obtained by comprehension from a propositional fibration (for all these categorical properties see, for example, [Jac99, Mai05]).

Our total setoid model corresponds categorically to a different quotient completion from the same construction of total setoids performed over Martin-Löf's type theory [NPS90], here called MLTT, as studied in [Pal05]. In fact, the total setoid model Q(MLTT) over MLTT coincides with the exact completion of the weakly lex category [CV98, CC82] of the MLTT sets. Instead, Q(mTT) can be seen as an instance of a more general completion of quotients starting from a weakly cartesian category equipped with a suitable comprehensive fibration.

It is worth noting that in both models there are at least two ways of interpreting propositions. One consists in interpreting propositions as sets. Then, both in Q(mTT) and in Q(MLTT) the extensional version of Martin-Löf's type theory in [Mar84] is validated. Therefore, the axiom of choice where quantifiers are interpreted as dependent product and indexed sum is also valid in both. But then we know that the axiom of choice may be constructively incompatible with well-behaved quotients, in particular effectiveness of quotients (see [Mai99]). Therefore, it seems that our total setoid constructions cannot be considered as a quotient completion of a propositions-as-sets theory.

The other way of interpreting propositions consists in interpreting them in Q(MLTT) as all mono sets (as in [Mai05]), and in Q(mTT) as only some mono sets like in qmTT. Then, both models support well-behaved quotients, i.e. effective quotients. In particular, the internal language of Q(MLTT) includes that of locally cartesian closed pretopos in [Mai05]. But, by interpreting propositions as mono sets, Q(MLTT) and even more Q(mTT) loose the general validity of the propositional axiom of choice. In fact, following propositions as mono sets, the interpretation of the axiom of choice in Q(MLTT) turns out to be equivalent to what Martin-Löf calls the *extensional axiom of choice* in [ML06], known to fail constructively [ML06, Car04]. Only the validity of the axiom of unique choice survives in its generality in Q(MLTT).

The design of qmTT and its interpretation over mTT is a key technical step in order to build the extensional level of the minimal constructive foundation advocated in [MS05]. This extensional level does not exactly coincide with qmTT, since qmTT is just obtained from mTT by abstracting on the intensional equality between elements of sets and propositions. Further abstractions, like that from proof-terms of propositions or the addition of subsets, are also desirable in order to get a many-sorted logic closer to set-theoretic languages used in everyday mathematics. However these can be obtained by associating to qmTT a many sorted logic with *true*-judgements as in [Mar84, Mar85] and subsets as in [SV98, Car03]. Another important point is that the extensional level advocated in [MS05] should have predicates depending on more general types than sets, like the collection of all subsets of a set (these are particularly needed in a predicative theory to define some of common mathematical concepts, like, for example, the definition of formal topology in [Sam03]).

Here we decided to concentrate on how to interpret quotients in mTT given that various proposals of how to add quotients have been given in the literature of type theory with the notion of setoid (see for example [Hof97] and [BCP03] and references therein) and also in category theory with the notion of quotient completions of a (weakly) lex category or of a regular one (see for example [CC82, CV98, Car95, CR00]). The exact formulation of the extensional level of the minimal constructive foundation based on mTT is left to future work with G. Sambin.

2 The Extensional System

Here we briefly introduce the extensional type theory qmTT that we will interpret in a model built out of the intensional type theory mTT introduced in [MS05]. We assume that mTT includes also a boolean universe (as mentioned in [Car04] and whose rules are the same as those for U_b of qmTT in the appendix) to make the disjoint sum set really disjoint (see, for example, [Mai05] for its definition).

qmTT is obtained as follows: we first take the *extensional* version of mTT, in the same way as Martin-Löf's type theory in [Mar84] is the extensional version of that in [NPS90]; then we collapse propositions into *mono sets* according to the notion in [Mai05]; and finally we add effective quotient sets as in [Mai05]. The precise rules to form sets and propositions of qmTT are given in the appendix.

The form of judgements to describe qmTT are those of mTT. Hence, for building sets, in the style of Martin-Löf's type theory [Mar84, NPS90], we have four kinds of judgements:

$$A \ \mathsf{set}\ [\Gamma] \qquad A = B\ [\Gamma] \qquad a \in A\ [\Gamma] \qquad a = b \in A\ [\Gamma]$$

that is the set judgement, the equality between sets, the term judgement and the (definitional) equality between terms of the same set, respectively. The contexts Γ of these judgements are formed as in [Mar84] and they are telescopic [dB91] since sets are allowed to depend on variables ranging over other sets.

The set constructors of qmTT are those of mTT with the addition of *effective quotient sets* (see [Mai05]).

Moreover, to build propositions, as in mTT, we have the following judgements:

$$A \ \mathsf{prop}\ [\Gamma] \qquad A = B\ [\Gamma]$$

In order to make propositions into mono sets, namely to make propositions into sets inhabited with at most one proof according to the notion in [Mai05], we add the rules

$$\textbf{prop-into-set}\ \frac{A\ \mathsf{prop}\ [\Gamma]}{A\ \mathsf{set}\ [\Gamma]} \qquad \textbf{prop-mono}\ \frac{A\ \mathsf{prop}\ [\Gamma] \qquad p \in A\ [\Gamma] \quad q \in A\ [\Gamma]}{p = q \in A\ \ [\Gamma]}$$

The requirement that propositions are mono sets is crucial in the presence of quotient effectiveness which would otherwise become similar to a choice operator. Indeed, if we identify propositions with sets simply, quotient effectiveness may lead to classical logic (see [Mai99]) and hence it is no longer a constructive rule. The propositions of qmTT are those of mTT but their proofs are all made equal. Moreover, the *intensional* propositional equality is replaced by the *extensional* one of [Mar84], which, besides being *mono* by definition, is equivalent to the definitional equality of terms.

Our extensional theory is a variation of the internal type theory of a lextensive list-arithmetic locally cartesian closed pretopos, as devised in [Mai05]. The main difference is that we discharge the identification *propositions as mono sets* typical of a pretopos by simply taking *propositions as primitive mono sets*, without requiring that all mono sets are propositions. In this way we avoid the validity of *the axiom of unique choice*, which would instead follow under the identification of propositions with mono sets (see [Mai05]).

The mono condition for propositions makes their proof-terms irrelevant. The proof-irrelevance of propositions is helpful to implement subsets as in [SV98] without the restrictions pointed out in [Car03].

3 The Setoid Model

We define the following category of sets equipped with an equivalence relation, sometimes called "total setoids" in the literature:

Definition 1. The category **Q(mTT)** is defined as follows:
Ob**Q(mTT):** the objects are pairs $(A, =_A)$ where A is a set in mTT, called "support", and

$$x =_A y \ \mathsf{prop}\ [x \in A, y \in A]$$

is an equivalence relation on the set A. This means that in mTT there exist proof-terms witnessing reflexivity, symmetry and transitivity of the relation:

$$\begin{array}{l}\mathsf{refl}(x) \in x =_A x \ \ [x \in A]\\ \mathsf{sym}(x,y,z) \in y =_A x \ \ [x \in A,\ y \in A,\ z \in x =_A y]\\ \mathsf{trans}(x,y,z,u,v) \in x =_A z \ \ [x \in A,\ y \in A,\ z \in A,\ u \in x =_A y,\ v \in y =_A z]\end{array}$$

Mor**Q(mTT):** the morphisms from an object $(A, =_A)$ to $(B, =_B)$ are terms $f(x) \in B\ [x \in A]$ preserving the corresponding equality, i.e. in mTT there exists a proof-term

$$\mathsf{pr}_1(x,y,z) \in f(x) =_B f(y) \ \ [x \in A,\ y \in A,\ z \in x =_A y]$$

Moreover, two morphisms $f, g : (A, =_A) \to (B, =_B)$ are equal if and only if in mTT there exists a proof-term

$$\mathsf{pr}_2(x) \in f(x) =_B g(x) \ \ [x \in A]$$

This category comes naturally equipped with an indexed category (or fibration) satisfying comprehension (see [Jac99] for its definition):

Definition 2. *The indexed category:*

$$\mathcal{P}_q : \mathrm{Q(mTT)}^{\mathsf{OP}} \to \mathsf{Cat}$$

is defined as follows. For each object $(A, =_A)$ *in* Q(mTT) *then* $\mathcal{P}_q(\,(A, =_A)\,)$ *is the following category:* $\mathsf{Ob}\mathcal{P}_q(\,(A, =_A)\,)$ *are the propositions* $\phi(x)$ prop $[x \in A]$ *depending on* A *and preserving the equality on* A*, namely there exists a proof-term:*

$$\mathsf{ps}(x,y,d) \in \phi(x) \to \phi(y) \ [x \in A,\ y \in A,\ d \in x =_A y]^{1}$$

[1] Indeed, from this, by using the symmetry of $x =_A y$ it follows that $\phi(x)$ is equivalent to $\phi(y)$ if $x =_A y$ holds.

Morphisms in $\mathsf{Mor}\mathcal{P}_q(\,(A,=_A)\,)$ *are given by a partial order, namely*

$$\mathcal{P}_q(\,(A,=_A)\,)(\,\phi(x)\,,\,\psi(x)\,)\;\equiv\;\phi(x)\le\psi(x)$$
$$\textit{iff there exists a proof-term } \mathsf{pt}(x)\in\phi(x)\to\psi(x)\;[x\in A]$$

Moreover, for every morphism $f:(A,=_A)\to(B,=_B)$ *in Q(mTT) given by* $f(x)\in B\;[x\in A]$ *then* $\mathcal{P}_q(f)$ *is the substitution functor, i.e.* $\mathcal{P}_q(f)(\,\phi(y)\,)\equiv\phi(f(x))$ *for any proposition* $\phi(y)$ prop $[y\in B]$ *(recall that* $\mathcal{P}_q$ *is contravariant).*

Lemma 1. $\mathcal{P}_q$ *is an indexed category satisfying comprehension.*

Proof. To describe the comprehension adjunction, we consider the Grothendieck completion $Gr(\mathcal{P}_q)$ of $\mathcal{P}_q$ (see [Jac99] for its definition) and the functor T : Q(mTT) $\to Gr(\mathcal{P}_q)$ defined as follows:

$$T(\,(A,=_A)\,)\equiv(\,(A,=_A)\,,\,\mathsf{tt}\,)\qquad\qquad T(f)\equiv(f,id_{\mathsf{tt}})$$

where tt is the truth constant that can be represented by any tautology.
T has a right adjoint

$$Cm:Gr(\mathcal{P}_q)\to \mathrm{Q(mTT)}$$

defined as follows on the objects: $Cm(\,(\,(A,=_A)\,,\,\phi\,)\,)\equiv(\,\Sigma_{x\in A}\phi(x)\,,\,=_{Cm}\,)$ where $z_1=_{Cm}z_2\;\equiv\;\pi_1(z_1)\;=_A\;\pi_1(z_2)$ for $z_1,z_2\in\Sigma_{x\in A}\phi(x)$; and on the morphisms: $Cm(\,(\,f\,,\,\le\,)\,)\;\equiv\;\overline{f}$ where $\overline{f}(z)\;\equiv\;\langle\,f(\pi_1(z))\,,\,\mathsf{pt}(\pi_1(z))(\pi_2(z))\,\rangle$ for $z\in\Sigma_{x\in A}\phi(x)$ and $f:(A,=_A)\to(B,=_B)$.

Definition 3 ($\mathcal{P}_q$-equivalence relations). *Given an equivalence relation* $R\in\mathcal{P}_q(\,(A,=_A)\times(A,=_A)\,)$*, namely a predicate* $R(x,y)$ prop $[x\in A,y\in A]$ *that preserves* $=_A$ *on both dependencies and is also an equivalence relation, then the first component of the counit on* $(\,(A,=_A)\times(A,=_A)\,,\,R\,)$ *is a monic equivalence relation in* Q(mTT)*, and it is called a* $\mathcal{P}_q$*-equivalence relation.*

The category Q(mTT) enjoys all the categorical properties necessary to interpret qmTT (for their definitions see, for example, [Mai05]).

Theorem 1. *The category* Q(mTT) *is lextensive (i.e. with terminal object, binary products, equalizers and stable finite disjoint coproducts) list-arithmetic (i.e. with parameterized lists) and locally cartesian closed (i.e. with also dependent products) with stable effective quotients with respect to* $\mathcal{P}_q$*-equivalence relations. Moreover, the indexed category* $\mathcal{P}_q$ *validates first-order intuitionistic logic.*

Remark 1. Note that to prove theorem 1 is crucial to have explicit proof-terms witnessing that $x=_A y$ in a Q(mTT)-object is an equivalence relation, that a Q(mTT)-morphism preserves the corresponding equivalence relations and when two Q(mTT)-morphisms are equal. In other words it seems that it would not follow if we give the definition of setoid objects, morphisms and their equality via *true*-judgements of the kind $x=_A y$ *true* $[\,x\in A,y\in A\,]$ as in the first setoid model in [Hof97][2].

[2] Note that the mentioned setoid model is anyway very different from Q(mTT) since the morphism equality is simply the definitional equality of the terms between the setoid supports.

The interpretation of qmTT in Q(mTT). After theorem 1, in order to interpret qmTT in Q(mTT) we can simply use the interpretation in [Mai05] given by fibred functors (we recall that this overcomes the problem, first solved in [Hof94], of interpreting substitution correctly when following the *informal* interpretation first given by Seely in [See83] and recalled in [Joh02]). But this interpretation is not first order since it requires to quantify over fibred functors. Luckily, in our case we can give a predicative interpretation in the setoid model similar to that in the completeness proof in [Mai05]. This is because the setoid model is indeed a syntactic one!

The key point to get this interpretation is to note that the slice category of arrows in $(A, =_A)$ is equivalent to the category of dependent setoids as defined in [Bis67, Pal05].

Definition 4. Given an object $(A, =_A)$ of Q(mTT), abbreviated with $A_=$, we define a dependent setoid on the setoid $(A, =_A)$ written

$$B_=(x)\ [\,x \in A_=\,]$$

as a dependent set $B(x)$ set $[\,x \in A\,]$, called "dependent support", together with an equivalence relation

$$y =_{B(x)} y' \ \mathsf{prop}\ [x \in A,\, y \in B(x),\, y' \in B(x)].$$

Moreover, for any $x_1, x_2 \in A$ there must exist

$$\sigma_{x_1,x_2}(y) \in B(x_2)\ [x_1 \in A,\, x_2 \in A,\, d \in x_1 =_A x_2,\, y \in B(x_1)]$$

non depending on $d \in x_1 =_A x_2$ and preserving the equality on $B(x_1)$, namely there exists a proof of

$$\sigma_{x_1,x_2}(y) =_{B(x_2)} \sigma_{x_1,x_2}(y') \ \mathsf{prop}\ [x_1 \in A,\, x_2 \in A,\, d \in x_1 =_A x_2,\\ y \in B(x_1),\, y' \in B(x_1),\, w \in y =_{B(x_1)} y'].$$

Furthermore, $\sigma_{x,x}$ is the identity, namely there exists a proof of

$$\sigma_{x,x}(y) =_{B(x)} y \in B(x)\ \mathsf{prop}\ [x \in A,\, y \in B(x)]$$

and the σ_{x_1,x_2}'s are closed under composition, namely there exists a proof of

$$\sigma_{x_2,x_3}(\sigma_{x_1,x_2}(y)) =_{B(x_3)} \sigma_{x_1,x_3}(y)\ \mathsf{prop}\\ [x_1 \in A,\, x_2 \in A,\, x_3 \in A,\, y \in B(x_1),\, d_1 \in x_1 =_A x_2,\, d_2 \in x_2 =_A x_3].$$

Categorically speaking, the category having the elements of A with their equality as objects and σ_{x_1,x_2} as (the unique) morphism from x_1 to x_2 forms a groupoid, because every σ_{x_1,x_2} gives rise to an isomorphism between $B(x_1)$ and $B(x_2)$.

Definition 5. Let us call $Dep(\,(A, =_A)\,)$ the category whose objects are dependent setoids $B_=(x)$ $[x \in A_=]$ on the setoid $(A, =_A)$, and whose morphisms

$$b(x) \in B_=(x)\ [x \in A_=]$$

are dependent terms $b(x) \in B(x)\ [x \in A]$ preserving the equality on A, namely in mTT there exists a proof of

$$\sigma_{x_1,x_2}(b(x_1)) =_{B(x_2)} b(x_2)\ \mathsf{prop}\ [x_1 \in A,\ x_2 \in A,\ d \in x_1 =_A x_2].$$

Proposition 1. *The category* Q(mTT)$/(A, =_A)$ *is equivalent to Dep*$(\,(A, =_A)\,)$.

This proposition suggests that the categorical interpretation in Q(mTT) given in [Mai05] can be equivalently formulated by interpreting dependent sets into dependent setoids as follows.

Sketch of the interpretation of qmTT into dependent setoids: A dependent set $B(x_1, \ldots x_n)\ [\,x_1 \in A_1, \ldots x_n \in A_n\,]$ of qmTT is interpreted as a dependent setoid

$$B_{=}(x_1, \ldots x_n)^I\ [\,x_1 \in {A_1}^I_{=}, \ldots x_n \in {A_n}^I_{=}\,]$$

assuming to have generalized def. 4 to setoids with telescopic dependencies, where $B(x_1, \ldots x_n)^I\ [\,x_1 \in {A_1}^I, \ldots x_n \in {A_n}^I\,]$ is its dependent support.

In the following, we leave the reader to deduce himself the σ's of the various dependent setoids. Except for the extensional propositional equality, any proposition is interpreted in the corresponding one of mTT with the warning that its equality is trivial, namely if ϕ is a proposition then $z =_{\phi^I} z' \equiv \mathsf{tt}$ for all $z, z' \in \phi^I$. Hence, for example,

$$(\phi \vee \psi)^I \equiv \phi^I \vee \psi^I \qquad (\forall_{x \in B} C(x)\,)^I \equiv \forall_{x \in B^I} C^I(x).$$

Instead the extensional propositional equality is interpreted in the equality of the set to which it refers:

$$(\,\mathsf{Eq}(B, b_1, b_2)\,)^I \equiv b_1^I =_{B^I} b_2^I$$

The emptyset, the singleton and the boolean universe are interpreted in themselves with the equality given by the propositional one: for example

$$(\,U_b\,)^I \equiv U_b \qquad \text{and} \qquad z =_{U_b^I} z' \equiv \mathsf{Id}(U_b, z, z')\ \text{ for } z, z' \in U_b\,.$$

Instead the other constructors are interpreted as follows:

Strong Indexed Sum set: $(\Sigma_{x \in B} C(x))^I \equiv \Sigma_{x \in B^I} C^I(x)$
and $z =_{\Sigma_{x \in B} C(x)^I} z' \equiv \quad \pi_1(z) =_{B^I} \pi_1(z') \wedge \sigma_{\pi_1(z), \pi_1(z')}(\pi_2(z)) =_{C^I(\pi_1(z'))} \pi_2(z')$
for $z, z' \in \Sigma_{x \in B} C(x)^I$.
Disjoint Sum set: $(\,B + C\,)^I \equiv B^I + C^I$

$$\text{and } z =_{B^I + C^I} z' \equiv \begin{cases} b =_{B^I} b' & \text{if } z = \mathsf{inl}(b) \quad z' = \mathsf{inl}(b') \text{ for } b, b' \in B^I \\ c =_{C^I} c' & \text{if } z = \mathsf{inr}(c) \quad z' = \mathsf{inr}(c') \text{ for } c, c' \in C^I \\ \bot & \text{otherwise} \end{cases}$$

for $z, z' \in B^I + C^I$.

Dependent Product set: $(\,\Pi_{x\in B} C(x)\,)^I \equiv$
$\Sigma_{h\in \Pi_{x\in B^I} C^I(w)} \quad \forall_{x_1,x_2\in B^I}\, x_1 =_{B^I} x_2 \;\rightarrow \sigma_{x_1,x_2}\,(h(x_1)) =_{C^I(x_2)} h(x_2)$
and $z {=}_{\Pi_{x\in B^I} C(x)^I} z' \equiv_{\!,} \forall_{x\in B^I} \pi_1(z)(x) =_{C^I(x)} \pi_1(z')(x)$ for $z, z' \in \Pi_{x\in B^I}\, C(x)^I$.

Quotient set: $(\,A/R\,)^I \equiv A^I$
and $z =_{A/R^I} z' \equiv R^I(z, z')$ for $z, z' \in A^I$.

List set: $(\,List(C)\,)^I \equiv List(C^I)$
and $z {=}_{List(C)^I} z' \equiv_{\!,} \exists_{l\in List(R)}\;\; \mathsf{Id}(\,List(C^I),\, \overline{\pi_1}(l),\, z) \wedge \mathsf{Id}(\,List(C^I),\, \overline{\pi_2}(l),\, z')$
for $z, z' \in List(C)^I$ where $R \equiv \Sigma_{x\in C^I}\, \Sigma_{y\in C^I}\, x =_{C^I} y$ and $\overline{\pi_i} \equiv List(\pi_i)$ is the lifting on lists of the i-th projection for $i = 1, 2$.

To see that the interpretation of disjoint sum sets is well-defined, recall that in mTT the sum is disjoint thanks to the presence of the boolean universe U_b.

Remark 2. **Internal logic of Q(mTT).** We are not able to prove that Q(mTT) has qmTT as its internal language. The reason is that the interpretation, of implication, of universal quantification and of dependent product set do not seem to be preserved by the functor $\xi : \mathrm{Q(mTT)} \rightarrow \mathcal{C}(\mathrm{qmTT})$ sending a setoid into its quotient, where $\mathcal{C}(\mathrm{qmTT})$ is the syntactic category of qmTT defined as in [Mai05] (note that we can naturally interpret mTT into qmTT by sending all the constructors in the corresponding ones and, in particular, the intensional propositional equality of mTT into the extensional one of qmTT).

However, observe that the *coherent fragment* cqmTT of qmTT, obtained by cutting out implication, universal quantification and dependent product sets from qmTT, is the *internal language of the setoid model* built *over the corresponding coherent fragment* cmTT of mTT obtained by cutting out the corresponding sets and propositions.

Remark 3. **Connection with the exact completion of a weakly lex category.** The construction of total setoids on mTT corresponds categorically to an instance of the following generalization of the exact completion construction [CV98, CC82] of a weakly lex category: we start from a weakly cartesian category $\mathcal{C}$ endowed with a split comprehensive fibration $\mathcal{P}$ satisfying enough logical laws to express the notion of equivalence relation; then we simulate the exact completion of a weakly lex category by taking only those pseudo-equivalence relations coming by comprehension from equivalence relations in the fibres of $\mathcal{P}$. If $\mathcal{C}$ is weakly lex and $\mathcal{P}$ is the codomain fibration (see [Jac99]) then our construction is the exact completion of $\mathcal{C}$.

Therefore, knowing the internal language of such categorical constructions from [Mai05], we conclude that the total setoid construction coincides with the exact completion construction as in [CV98, CC82] if we perform such a construction on an extension of mTT, called MLTT, equivalent to Martin-Löf's type theory in [NPS90]. In order to get the the first order fragment of Martin-Löf's type theory it is enough to strengthen the existential quantifier of mTT to enjoy E-Σ and C-Σ of the Strong Indexed Sum set, after adding the rule that *any set A* set *is also a proposition A* prop (recall that in mTT propositions are not assumed to be proof-irrelevant!).

Then, the category Q(MLTT) of total setoids built on MLTT turns out to be a *list-arithmetic locally cartesian closed pretopos* and it *coincides with the exact completion*, as defined in [Car95, CV98], *of the weakly lex category* C(MLTT), where the category C(MLTT) is defined as follows: its objects are MLTT-sets and its morphisms from A to B are terms $b(x) \in B$ $[x \in A]$ and two morphisms $b_1(x) \in B$ $[x \in A]$ and $b_2(x) \in B$ $[x \in A]$ are equal if there exists a proof of $\mathsf{Id}(B, b_1(x), b_2(x))$ prop $[x \in A]$ in MLTT. The identity and composition are defined as in the syntactic categories in [Mai05].

However, it is important to note that the total setoid model Q(MLTT) does not seem to be closed under well-behaved quotients if we identify propositions as sets as done in MLTT. The reason is that under this identification Q(MLTT), *but also* Q(mTT), *supports the axiom of choice where the quantifiers are replaced by Π and Σ* as a consequence that they validate the extensional version of Martin-Löf's type theory in [Mar84]. Then, effectiveness of quotients, being generally incompatible with the axiom of choice (see [Mai99]), does not seem to be validated.

To gain well-behaved quotients in Q(MLTT) one possibility is to reason by identifying *propositions as mono sets* as in the logic of a pretopos (see [Mai05]). Instead, in Q(mTT) we get them by identifying propositions with only those mono sets arising from propositions in mTT as in the interpretation of qmTT. In fact, even if Q(mTT) supports quotients of all mono equivalence relations, these do not seem to enjoy effectiveness. Categorically speaking, this means that Q(mTT) does not seem to be a pretopos even if it has quotients for all monic equivalence relations because we are not able to prove that all monic equivalence relations are in bijection with $\mathcal{P}_q$-equivalence relations, for which effective quotients exist (which explains why we introduced the concept of $\mathcal{P}_q$-equivalence relation!).

As expected from [Mai99], under the identification propositions as mono sets, Q(MLTT) looses the validity of the axiom of choice. In fact it turns out that the *propositional axiom of choice* is exactly interpreted in Q(MLTT) (and also in Q(mTT)) as the *extensional axiom of choice* in [ML06, Car04] following the given interpretation of qmTT in Q(mTT). Therefore, the arguments in [ML06, Car04] exactly show that the propositional axiom of choice *fails* to be valid in the total setoid models Q(MLTT), and even more in Q(mTT), under the identification of propositions with mono sets. In Q(MLTT) the propositional axiom of choice survives only for those setoids whose equivalence relation is the propositional equality of MLTT. Only the validity of the axiom of unique choice continues to hold in its generality in Q(MLTT) (see also [ML06]).

Remark 4. In order to interpret quotients in mTT we also considered to mimic the exact completion on a regular category in [CV98, Hyl82] by taking only those equivalence relations obtained by comprehension from the propositional fibration. But we ended up just in a list-arithmetic pretopos, for example not necessarily closed under dependent products, since mTT is *predicative* and the axiom of choice is not a theorem there.

Acknowledgements. I thank Giovanni Curi, Ferruccio Guidi, Pino Rosolini, Giovanni Sambin, Thomas Streicher, Silvio Valentini for very useful discussions on the topics treated here, and Jesper Carlström about the notion of U_b.

References

[BCP03] Barthes, G., Capretta, V., Pons, O.: Setoids in type theory. J. Funct. Programming, Special issue on Logical frameworks and metalanguages 13(2), 261–293 (2003)

[Bis67] Bishop, E.: Foundations of Constructive Analysis. McGraw-Hill Book Co, New York (1967)

[Car95] Carboni, A.: Some free constructions in realizability and proof theory. J. Pure Appl. Algebra 103, 117–148 (1995)

[Car03] Carlström, J.: Subsets, quotients and partial functions in Martin-Löf's type theory. In: Geuvers, H., Wiedijk, F. (eds.) TYPES 2002. LNCS, vol. 2646, Springer, Berlin (2003)

[Car04] Carlström, J.: EM + Ext- + ACint is equivalent to ACext. Mathematical Logic Quarterly 50(3), 236–240 (2004)

[CC82] Carboni, A., Celia Magno, R.: The free exact category on a left exact one. Journal of Australian Math. Soc. 33, 295–301 (1982)

[CR00] Carboni, A., Rosolini, G.: Locally cartesian closed exact completions. J. Pure Appl. Algebra, Category theory and its applications (Montreal, QC 1997) pp. 103–116, (2000)

[CV98] Carboni, A., Vitale, E.M.: Regular and exact completions. Journal of Pure. and Applied Algebra 125, 79–116 (1998)

[dB91] de Bruijn, N.G.: Telescopic mapping in typed lambda calculus. Information and Computation 91, 189–204 (1991)

[Hof94] Hofmann, M.: On the interpretation of type theory in locally cartesian closed categories. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 427–441. Springer, Heidelberg (1995)

[Hof97] Hofmann, M.: Extensional Constructs in Intensional Type Theory. In: Distinguished Dissertations, Springer, Heidelberg (1997)

[Hyl82] Hyland, J.M.E.: The effective topos. In: The L.E.J. Brouwer Centenary Symposium (Noordwijkerhout, 1981), Stud. Logic Foundations Math. Stud. Logic Foundations Math, vol. 110, pp. 165–216. North-Holland, Amsterdam-New York (1982)

[Jac99] Jacobs, B.: Categorical Logic and Type Theory, Studies in Logic. Studies in Logic, vol. 141. Elsevier, Amsterdam (1999)

[Joh02] Johnstone, P.T.: Sketches of an Elephant: a Topos Theory Compendium. Vol. 2., volume 43 of Oxford Logic Guides. The Clarendon Press, Oxford University Press, New York (2002)

[Mai99] Maietti, M.E.: About effective quotients in constructive type theory. In: Naraschewski, W., Altenkirch, T., Reus, B. (eds.) TYPES 1998. LNCS, vol. 1657, pp. 164–178. Springer, Heidelberg (1999)

[Mai05] Maietti, M.E.: Modular correspondence between dependent type theories and categories including pretopoi and topoi. Mathematical Structures in Computer Science 15(6), 1089–1149 (2005)

[Mar84] Martin-Löf, P.: Intuitionistic Type Theory, notes by G. Sambin of a series of lectures given in Padua, June 1980. Bibliopolis, Naples (1984)

[Mar85] Martin Löf, P.: On the meanings of the logical constants and the justifications of the logical laws. In: Proceedings of the conference on mathematical logic, volume 2, pp. 203–281. Univ. Siena, Siena, 1985. Reprinted in Nordic J. Philos. Logic, 1(1):11–60 (1996)

[ML06] Martin-Löf, P.: 100 years of Zermelo's axiom of choice:what was the problem with it? The Computer Journal 49(3), 10–37 (2006)

[MS05] Maietti, M.E., Sambin, G.: Toward a minimalist foundation for constructive mathematics. In: Crosilla, L., Schuster, P. (eds.) From Sets and Types to Topology and Analysis: Practicable Foundations for Constructive Mathematics, in Oxford Logic Guides, vol. 48, pp. 91–114. Oxford University Press, Oxford (2005)

[MV99] Maietti, M.E., Valentini, S.: Can you add powersets to Martin-Löf intuitionistic type theory? Mathematical Logic Quarterly 45, 521–532 (1999)

[NPS90] Nordström, B., Petersson, K., Smith, J.: Programming in Martin Löf's Type Theory. Clarendon Press, Oxford (1990)

[Pal05] Palmgren, E.: Bishop's set theory. Slides for lecture at the TYPES summer school (2005)

[Sam03] Sambin, G.: Some points in formal topology. Theoretical Computer Science (2003)

[See83] Seely, R.A.G.: Hyperdoctrines, natural deduction and the Beck condition. Zeitschr. f. Math. Logik. und Grundlagen d. Math. 29, 505–542 (1983)

[SV98] Sambin, G., Valentini, S.: Building up a toolbox for Martin-Löf's type theory: subset theory. In: Sambin, G., Smith, J. (eds.) Twenty-five years of constructive type theory, Proceedings of a Congress held in Venice, October 1995, pp. 221–244. Oxford U. P, Oxford (1998)

Appendix: The qmTT Typed Calculus

We present here the inference rules to form sets and propositions in qmTT. Note that to write the elimination constructors of the various sets and propositions we adopt the higher order syntax as in [NPS90][3].

For brevity, in presenting formal rules we omit the corresponding equality rules that are defined as in [Mar84].

The contexts are generated by the same rules as for mTT in [MS05]. Note that the piece of context common to all judgements involved in a rule is omitted and that the typed variables appearing in a context are meant to be added to the implicit context as the last one. We also recall that the contexts are made of assumptions on sets only, and that we have the rule **prop-into-set** and **prop-mono** mentioned in section 2.

[3] For example, note that the elimination constructor of disjunction $El_\vee(w, a_B, a_C)$ binds the open terms $a_B(x) \in A\ [x \in B]$ and $a_C(y) \in A\ [y \in C]$. Hence these open terms should be then encoded into the elimination constructor given that they are needed in the disjunction conversion rules. To simplify the notation we use the higher order syntax as in [NPS90]. Thanks to this syntax from the open term $a_B(x) \in A\ [x \in B]$ we get $(x \in B)\, a_B(x)$ of higher type $(x \in B)\, A$. Then by η-conversion among higher types $(x \in B)\, a_B(x)$ is equal to a_B and we can simply write the short expression a_B to recall the open term where it comes from.

The rules to generate the propositions in qmTT are the following:

Falsum

F-Fs) $\bot\ \mathsf{prop}$ E-Fs) $\dfrac{a \in \bot \quad A\ \mathsf{prop}}{\mathsf{r_o}(a) \in A}$

Propositional Equality

Eq) $\dfrac{C\ \mathsf{set} \quad c \in C \quad d \in C}{\mathsf{Eq}(C, c, d)\ \mathsf{prop}}$ I-Eq) $\dfrac{c \in C}{\mathsf{eq_C}(c) \in \mathsf{Eq}(C, c, c)}$ E-Eq) $\dfrac{p \in \mathsf{Eq}(C, c, d)}{c = d \in C}$

Implication

F-Im $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop}}{B \to C\ \mathsf{prop}}$ I-Im $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop} \quad c(x) \in C\ [x \in B]}{\lambda_\to x^B.c(x) \in B \to C}$

E-Im $\dfrac{b \in B \quad f \in B \to C}{\mathsf{Ap}_\to(f, b) \in C}$

Conjunction

F-$\wedge$) $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop}}{B \wedge C\ \mathsf{prop}}$ I-$\wedge$) $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop} \quad b \in B \quad c \in C}{\langle b,_\wedge c\rangle \in B \wedge C}$

E$_1$-$\wedge$ $\dfrac{d \in B \wedge C}{\pi_1^B(d) \in B}$ E$_2$-$\wedge$ $\dfrac{d \in B \wedge C}{\pi_2^C(d) \in C}$

Disjunction

F-$\vee$) $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop}}{B \vee C\ \mathsf{prop}}$

I$_1$-$\vee$) $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop} \quad b \in B}{\mathsf{inl}_\vee(b) \in B \vee C}$ I$_2$-$\vee$) $\dfrac{B\ \mathsf{prop} \quad C\ \mathsf{prop} \quad c \in C}{\mathsf{inr}_\vee(c) \in B \vee C}$

E-$\vee$) $\dfrac{A\ \mathsf{prop} \qquad w \in B \vee C \quad a_B(x) \in A\ [x \in B] \quad a_C(y) \in A\ [y \in C]}{El_\vee(w, a_B, a_C) \in A}$

Existential quantification

F-$\exists$) $\dfrac{C(x)\ \mathsf{prop}\ \ [x \in B]}{\exists_{x \in B} C(x)\ \mathsf{prop}}$ I-$\exists$) $\dfrac{C(x)\ \mathsf{prop}\ \ [x \in B] \quad b \in B \quad c \in C(b)}{\langle b,_\exists c\rangle \in \exists_{x \in B} C(x)}$

E-$\exists$) $\dfrac{M\ \mathsf{prop} \qquad d \in \exists_{x \in B} C(x) \quad m(x, y) \in M\ [x \in B, y \in C(x)]}{El_\exists(d, m) \in M}$

Universal quantification

F-$\forall$ $\dfrac{C(x)\ \mathsf{prop}\ [x \in B]}{\forall_{x \in B} C(x)\ \mathsf{prop}}$ I-$\forall$ $\dfrac{C(x)\ \mathsf{prop}\ \ [x \in B] \quad c(x) \in C(x)\ [x \in B]}{\lambda_\forall x^B.c(x) \in \forall_{x \in B} C(x)}$

E-$\forall$ $\dfrac{b \in B \quad f \in \forall_{x \in B} C(x)}{\mathsf{Ap}_\forall(f, b) \in C(b)}$

The rules to generate qmTT sets are the following:

Empty set

F-Em) $N_0\ \mathsf{set}$ E-Em) $\dfrac{a \in N_0 \quad A(x)\ \mathsf{set}\ [x \in N_0]}{\mathsf{emp_o}(a) \in A(a)}$

Singleton set

S) N_1 set I-S) $\star \in \mathsf{N}_1$ C-S) $\dfrac{t \in \mathsf{N}_1 \quad M(z)\ [z \in \mathsf{N}_1] \quad c \in M(\star)}{El_{\mathsf{N}_1}(t,c) \in M(t)}$

Indexed Sum set

F-Σ) $\dfrac{C(x)\ \mathsf{set}\ \ [x \in B]}{\Sigma_{x\in B}C(x)\ \mathsf{set}}$ I-Σ) $\dfrac{b \in B \quad c \in C(b)}{\langle b,c\rangle \in \Sigma_{x\in B}C(x)}$

$$\text{E-}\Sigma)\ \frac{\begin{array}{l}M(z)\ [z \in \Sigma_{x\in B}C(x)]\\ d \in \Sigma_{x\in B}C(x) \quad m(x,y) \in M(\langle x,y\rangle)\ [x \in B, y \in C(x)]\end{array}}{El_{\Sigma}(d,m) \in M(d)}$$

$$\text{C-}\Sigma)\ \frac{\begin{array}{l}M(z)\ [z \in \Sigma_{x\in B}C(x)]\\ b \in B \quad c \in C(b) \quad m(x,y) \in M(\langle x,y\rangle)\ [x \in B, y \in C(x)]\end{array}}{El_{\Sigma}(\langle b,c\rangle,m) = m(b,c) \in M(\langle b,c\rangle)}$$

Disjoint Sum set

+) $\dfrac{C\ \mathsf{set} \quad B\ \mathsf{set}}{C+B\ \mathsf{set}}$ I$_1$-+) $\dfrac{c \in C}{\mathsf{inl}(c) \in C+B}$ I$_2$-+) $\dfrac{b \in B}{\mathsf{inr}(b) \in C+B}$

$$\text{E-+)}\ \frac{\begin{array}{l}A(z)\ [z \in C+B]\\ w \in C+B \quad a_C(x) \in A(\mathsf{inl}(x))\ [x \in C] \quad a_B(y) \in A(\mathsf{inr}(y))\ [y \in B]\end{array}}{El_{+}(w,a_C,a_B) \in A(w)}$$

$$\text{C}_1\text{-+)}\ \frac{\begin{array}{l}A(z)\ [z \in C+B]\\ c \in C \quad a_C(x) \in A(\mathsf{inl}(x))\ [x \in C] \quad a_B(y) \in A(\mathsf{inr}(y))\ [y \in B]\end{array}}{El_{+}(\mathsf{inl}(c),a_C,a_B) = a_C(c) \in A(\mathsf{inl}(c))}$$

$$\text{C}_2\text{-+)}\ \frac{\begin{array}{l}A(z)\ [z \in C+B]\\ b \in B \quad a_C(x) \in A(\mathsf{inl}(x))\ [x \in C] \quad a_B(y) \in A(\mathsf{inr}(y))\ [y \in B]\end{array}}{El_{+}(\mathsf{inr}(b),a_C,a_B) = a_B(b) \in A(\mathsf{inr}(b))}$$

Dependent Product set

F-Π $\dfrac{C(x)\ \mathsf{set}\ [x \in B]}{\Pi_{x\in B}C(x)\ \mathsf{set}}$ I-Π $\dfrac{c \in C(x)[x \in B]}{\lambda x^B.c \in \Pi_{x\in B}C(x)}$

E-Π $\dfrac{b \in B \quad f \in \Pi_{x\in B}C(x)}{\mathsf{Ap}(f,b) \in C(b)}$ βC-Π $\dfrac{b \in B \quad c(x) \in C(x)[x \in B]}{\mathsf{Ap}(\lambda x^B.c(x),b) = c(b) \in C(b)}$

βC-Π $\dfrac{c \in \Pi_{x\in B}C(x)}{\lambda x^B.\mathsf{Ap}(c,x) = c \in \Pi_{x\in B}C(x)}$

Boolean universe

F-bU U_b set I$_1$-bU $\widetilde{N_0} \in U_b$ I$_2$-bU $\widetilde{N_1} \in U_b$

E-bU $\dfrac{d \in U_b}{T(d)\ \mathsf{set}}$ βC$_1$-bU $T(\widetilde{N_0}) = N_0$ βC$_2$-bU $T(\widetilde{N_1}) = N_1$

Quotient set

$$\text{Q)}\ \frac{R(x,y)\ \mathsf{prop}\ [x \in A, y \in A] \quad \begin{array}{l}\mathsf{refl}(x) \in R(x,x)\ [x \in A] \\ \mathsf{sym}(x,y,z) \in R(y,x)\ [x \in A,\ y \in A,\ z \in R(x,y)] \\ \mathsf{trans}(x,y,z,u,v) \in R(x,z)\ [x \in A,\ y \in A,\ z \in A, \\ \hfill u \in R(x,y),\ v \in R(y,z)]\end{array}}{A/R\ \mathsf{set}}$$

$$\text{I-Q)}\ \frac{a \in A\ \ A/R\ \mathsf{set}}{[a] \in A/R} \qquad \text{eq-}Q)\ \frac{a \in A \quad b \in A \quad d \in R(a,b)\ \ A/R\ \mathsf{set}}{[a] = [b] \in A/R}$$

$$\text{E-Q)}\ \frac{\begin{array}{l}L(z)\ [z \in A/R] \\ p \in A/R \quad l(x) \in L([x])\ [x \in A] \quad l(x) = l(y) \in L([x])\ \begin{bmatrix} x \in A, \\ y \in A, d \in R(x,y)\end{bmatrix}\end{array}}{El_Q(l,p) \in L(p)}$$

$$\text{C-Q)}\ \frac{\begin{array}{l}L(z)\ [z \in A/R] \\ a \in A \quad l(x) \in L([x])\ [x \in A] \quad l(x) = l(y) \in L([x])\ [x \in A, y \in A, d \in R(x,y)]\end{array}}{El_Q(l,[a]) = l(a) \in L([a])}$$

Effectiveness

$$\frac{a \in A \quad b \in A \quad [a] = [b] \in A/R}{\mathsf{eff}(a,b) \in R(a,b)}$$

List set

$$\text{list)}\ \frac{C\ \mathsf{set}}{List(C)\ \mathsf{set}} \qquad \text{I}_1\text{-list)}\ \epsilon \in List(C) \qquad \text{I}_2\text{-list)}\ \frac{s \in List(C) \quad c \in C}{\mathsf{cons}(s,c) \in List(C)}$$

$$\text{E-list)}\ \frac{\begin{array}{l}L(z)\ [z \in List(C)] \\ s \in List(C) \quad a \in L(\epsilon) \quad l(x,y,z) \in L(\mathsf{cons}(x,y))\ \begin{bmatrix} x \in List(C), \\ y \in C, z \in L(x)\end{bmatrix}\end{array}}{El_{List}(a,l,s) \in L(s)}$$

$$\text{C}_1\text{-list)}\ \frac{\begin{array}{l}L(z)\ [z \in List(C)] \\ a \in L(\epsilon) \quad l(x,y,z) \in L(\mathsf{cons}(x,y))\ [x \in List(C), y \in C, z \in L(x)]\end{array}}{El_{List}(a,l,\epsilon) = a \in L(\epsilon)}$$

$$\text{C}_2\text{-list)}\ \frac{\begin{array}{l}L(z)\ [z \in List(C)] \\ s \in List(C) \quad c \in C \quad a \in L(\epsilon) \quad l(x,y,z) \in L(\mathsf{cons}(x,y))\ \begin{bmatrix} x \in List(C), \\ y \in C, z \in L(x)\end{bmatrix}\end{array}}{El_{List}(a,l,\mathsf{cons}(s,c)) = l(s,c,El_{List}(a,l,s)) \in L(\mathsf{cons}(s,c))}$$

Note that $List(N_1)$ corresponds to the set of natural numbers represented as lists on a singleton, with $0 \equiv \epsilon$ and $s(n) \equiv \mathsf{cons}(n,*)$ for $n \in List(N_1)$.

Hairpin Completion Versus Hairpin Reduction

Florin Manea[1] and Victor Mitrana[1,2]

[1] Faculty of Mathematics and Computer Science, University of Bucharest
Str. Academiei 14, 70109, Bucharest, Romania
flmanea@funinf.cs.unibuc.ro

[2] Research Group in Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tarraco 1, 43005, Tarragona, Spain
mitrana@fmi.unibuc.ro

Abstract. We define the hairpin reduction as the inverse operation of a formal operation on words and languages suggested by DNA biochemistry, namely the hairpin completion, introduced in [3]. We settle the closure properties of some classes in the Chomsky hierarchy as well as some complexity classes under the non-iterated version of the hairpin reduction, in comparison with the hairpin completion. Then an algorithm that decides whether or not a regular language coincides with its primitive hairpin root is presented. Finally, we discuss a cubic time algorithm for computing the common ancestors of two given words. This algorithm may be used also for computing the closest or farthest primitive hairpin root of a given word.

Keywords: DNA computing, formal languages, hairpin completion, hairpin reduction, primitive hairpin root.

1 Introduction

A DNA molecule consists of a double strand, each DNA single strand being composed by nucleotides which differ from each other by their bases: A (adenine), G (guanine), C (cytosine), and T (thymine). The two strands which form the DNA molecule are kept together by the hydrogen bond between the bases: A always bonds with T, while C with G. This paradigm is usually referred as the *Watson-Crick complementarity.* Another important biological principle is the *annealing*, that refers to fusing two single stranded molecules by complementary base. This operation of fusing two single stranded molecules by complementary base requires a heated solution containing the two strands, which is cooled down slowly. It is known that a single stranded DNA molecule might produce a hairpin structure, a phenomenon based on these two biological principles. In many DNA-based algorithms, these DNA molecules cannot be used in the subsequent computations. Hairpin or hairpin-free DNA structures have numerous applications to DNA computing and molecular genetics. In a series of papers (see, e.g., [4,6,7]) such structures are discussed in the context of finding sets of DNA sequences which are unlikely to lead to "bad" hybridizations. On the other hand, these molecules which may form a hairpin structure have been used as the basic feature of a new computational model reported in [14], where an instance of the 3-SAT problem has been solved by a DNA-algorithm whose second phase is mainly based on the elimination of hairpin structured molecules. Different types of hairpin and hairpin-free languages are defined in [12], [2], and more recently in [9], where they are studied from a language theoretical point of view.

The source of inspiration for introducing in [3] a new formal operation on words, namely *hairpin completion*, consists of three biological principles. Besides the Watson-Crick complementarity and annealing, the third biological phenomenon is that of *lengthening DNA by polymerases*. This phenomenon produces a complete double stranded DNA molecule as follows: one starts with two single strands such that one (usually called *primer*) is bonded to a part of the other (usually called *template*) by Watson-Crick complementarity and a *polymerization buffer* with many copies of the four nucleotides. Then polymerases will concatenate to the primer by complementing the template.

We now informally explain the hairpin completion operation and how it can be related to the aforementioned biological concepts. Let us consider the following hypothetical biological situation: we are given one single stranded DNA molecule z such that either a prefix or a suffix of z is Watson-Crick complementary to a subword of z. Then the prefix or suffix of z and the corresponding subword of z get annealed by complementary base pairing and then z is lengthened by DNA polymerases up to a complete hairpin structure. The mathematical expression of this hypothetical situation defines the hairpin completion operation. By this formal operation one can generate a set of words, starting from a single word. This operation is considered in [3] as an abstract operation on formal languages. Some algorithmic problems regarding the hairpin completion are investigated in [10].

In this paper, we consider the inverse operation of hairpin completion, namely hairpin reduction. Naturally, the hairpin reduction of a word x consists of all words y such that x can be obtained from y by hairpin completion. We settle the closure properties of some classes in the Chomsky hierarchy as well as some complexity classes under the non-iterated version of hairpin reduction, in comparison with the hairpin completion. A primitive hairpin root of a given word can be obtained by iterated hairpin reductions applied to the original word (including zero steps) until no hairpin reduction is applicable anymore. Then an algorithm that decides whether or not a regular language coincides with its primitive hairpin root is presented. Finally, we discuss a cubic time algorithm for computing a common ancestor of two given words. This algorithm may be used for computing the closest or farthest primitive hairpin root of a given word.

2 Basic Definitions

We assume the reader to be familiar with the fundamental concepts of formal language theory and automata theory, particularly with the notions of grammar and finite automaton [13].

An alphabet is a finite set of letters. For a finite set A we denote by $card(A)$ the cardinality of A. The set of all words over an alphabet V is denoted by V^*. The empty word is written ε; moreover, $V^+ = V^* \setminus \{\varepsilon\}$. Given a word w over an alphabet V, we denote by $|w|$ its length, while $|w|_a$ denotes the number of occurrences of the letter a in w. If $w = xyz$ for some $x, y, z \in V^*$, then x, y, z are called prefix, subword, suffix, respectively, of w. For a word w, $w[i..j]$ denotes the subword of w starting at position i and ending at position j, $1 \leq i \leq j \leq |w|$. If $i = j$, then $w[i..j]$ is the i-th letter of w, which is simply denoted by $w[i]$.

Let Ω be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of Ω. In other words, Ω is the *universe* of the languages in this paper, i.e., all words and languages are over alphabets that are subsets of Ω. An *involution* over a set S is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution σ on Ω such that $\sigma(a) \neq a$ for all $a \in \Omega$ is said to be, in this paper's context, a *Watson-Crick involution.* Despite that this is nothing more than a fixed point-free involution, we prefer this terminology since the hairpin completion defined later is inspired by the DNA lengthening by polymerases, where the Watson-Crick complementarity plays an important role. Let $\overline{\cdot}$ be a Watson-Crick involution fixed for the rest of the paper. The Watson-Crick involution is extended to a morphism from Ω^* to Ω^* in the usual way. We say that the letters a and $\overline{a}$ are complementary to each other. For an alphabet V, we set $\overline{V} = \{\overline{a} \mid a \in V\}$. Note that V and $\overline{V}$ can intersect and they can be, but need not be, equal. Remember that the DNA alphabet consists of four letters, $V_{DNA} = \{A, C, G, T\}$, which are abbreviations for the four nucleotides and we may set $\overline{A} = T$, $\overline{C} = G$.

We denote by $(\cdot)^R$ the mapping defined by $^R : V^* \longrightarrow V^*$, $(a_1a_2 \ldots a_n)^R = a_n \ldots a_2a_1$. Note that ^{R} is an involution and an *anti-morphism* ($(xy)^R = y^Rx^R$ for all $x, y \in V^*$). Note also that the two mappings $\overline{\cdot}$ and $\cdot^R$ commutes, namely, for any word x, $(\overline{x})^R = \overline{x^R}$ holds.

Let V be an alphabet, for any $w \in V^+$ we define the *k-hairpin completion* of w, denoted by $HC_k(w)$, for some $k \geq 1$, as follows:

$$HCP_k(w) = \{\overline{\gamma^R}w | w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k,\ \alpha, \beta, \gamma \in V^+\}$$
$$HCS_k(w) = \{w\overline{\gamma^R} | w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k,\ \alpha, \beta, \gamma \in V^+\}$$
$$HC_k(w) = HCP_k(w) \cup HCS_k(w)$$

The *hairpin completion* of w is defined by

$$HC(w) \ = \bigcup_{k \geq 1} HC_k(w).$$

This operation is schematically illustrated in Figure 1.

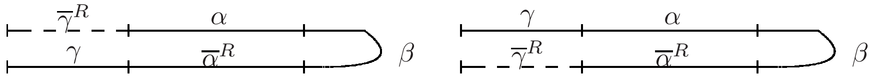


Fig. 1. Hairpin completion

Clearly, $HC_{k+1}(w) \subseteq HC_k(w)$ for any $w \in V^+$ and $k \geq 1$, hence $HC(w) = HC_1(w)$. The hairpin completion is naturally extended to languages by

$$HC_k(L) = \bigcup_{w\in L} HC_k(w) \qquad HC(L) = \bigcup_{w\in L} HC(w).$$

The iterated version of the hairpin completion is defined as usual by:

$$\begin{array}{lll} HC_k^0(w)=\{w\}, & HC_k^{n+1}(w)=HC_k(HC_k^n(w)), & HC_k^*(w)=\bigcup_{n\geq 0} HC_k^n(w) \\ HC^0(w)=\{w\}, & HC^{n+1}(w)=HC(HC^n(w)), & HC^*(w)=\bigcup_{n\geq 0} HC^n(w) \end{array}$$

$$HC_k^*(L) = \bigcup_{w\in L} HC_k^*(w) \qquad HC^*(L) = \bigcup_{w\in L} HC^*(w).$$

Of course, all these phenomena are considered here in an idealized way. For instance, we allow polymerase to extend in either end (3' or 5') despite that, due to the greater stability of 3' when attaching new nucleotides, DNA polymerase can act continuously only in the 5'$\longrightarrow$ 3' direction. However, polymerase can also act in the opposite direction, but in short "spurts" (Okazaki fragments). Moreover, in order to have a "stable" hairpin structure the subword x should be sufficiently long.

Let V be an alphabet, for any $w \in V^+$ we define the *k-hairpin reduction* of w, denoted by $HR_k(w)$, for some $k \geq 1$, as follows:

$$\begin{aligned} HRP_k(w) &= \{\alpha\beta\overline{\alpha^R\gamma^R} \mid w = \gamma\alpha\beta\overline{\alpha^R\gamma^R}, |\alpha| = k,\, \alpha,\beta,\gamma \in V^+\} \\ HRS_k(w) &= \{\gamma\alpha\beta\overline{\alpha^R} \mid w = \gamma\alpha\beta\overline{\alpha^R\gamma^R}, |\alpha| = k,\, \alpha,\beta,\gamma \in V^+\} \\ HR_k(w) &= HCP_k(w) \cup HCS_k(w) \end{aligned}$$

The *hairpin reduction* of w is defined by

$$HR(w) = \bigcup_{k\geq 1} HR_k(w).$$

The hairpin reduction is naturally extended to languages by

$$HR_k(L) = \bigcup_{w\in L} HR_k(w) \qquad HR(L) = \bigcup_{w\in L} HR(w).$$

The iterated version of the hairpin reduction is defined as usual by:

$$\begin{array}{lll} HR_k^0(w)=\{w\}, & HR_k^{n+1}(w)=HR_k(HR_k^n(w)), & HR_k^*(w)=\bigcup_{n\geq 0} HR_k^n(w) \\ HR^0(w)=\{w\}, & HR^{n+1}(w)=HR(HR^n(w)), & HR^*(w)=\bigcup_{n\geq 0} HR^n(w) \end{array}$$

$$HR_k^*(L) = \bigcup_{w\in L} HR_k^*(w) \qquad HR^*(L) = \bigcup_{w\in L} HR^*(w).$$

Note that $x \in HC_k^*(y)$ is equivalent to $y \in HR_k^*(x)$.

An element in $HC_k^*(w)$ is sometimes called *k-descendant* of w while an element of $HR_k^*(w)$ is sometimes called *k-ancestor* of w.

3 The Non-iterated Case

In this section we make a brief comparison between the non-iterated hairpin completion and reduction as formal operations on languages. A family of languages $\mathcal{F}$ is closed under hairpin completion/reduction if the hairpin completion/reduction of any language from $\mathcal{F}$ lies in $\mathcal{F}$.

First, we recall from [3]:

Theorem 1. *For any integer $k \geq 1$, a language is linear if and only if it is the morphic image of the k-hairpin completion of a regular language.*

Consequently, the class of regular languages is not closed under hairpin completion. However, the situation changes in the case of hairpin reduction.

Theorem 2. *The class of regular languages is closed under k-hairpin reduction for any $k \geq 1$.*

Proof. Assume that L is a regular language and let $A = (Q, V, q_0, \{s\}, \delta)$ a finite automaton that accepts L and has one final state only, namely s. We further assume that there is no $a \in V$ and $q \in Q$ such that $q_0 \in \delta(q, a)$. For a given integer $k \geq 1$, we show first that $HRS_k(L)$ is regular. Let $A' = (Q', V, q_0', F', \delta')$ be the non-deterministic finite automaton defined as follows:

$$\begin{aligned}
Q' &= Q \times Q \cup \{q^{(x)} \mid q \in Q, x \in V^*, 1 \leq |x| \leq k\} \times Q \cup \\
&\quad \{q^{[x]} \mid q \in Q, x \in V^*, |x| \leq k\} \times Q \\
F' &= \{(q^{[\varepsilon]}, q) \mid q \in Q\}, \\
q_0' &= (q_0, s),
\end{aligned}$$

– $\delta'((q_0, s), a) = (\delta(q_0, a) \times \{p \mid s \in \delta(p, a)\})$,

– $\delta'((q, r), a) = (\delta(q, a) \times \{p \mid r \in \delta(p, a)\}) \cup \{(t^{(a)}, r) \mid t \in \delta(q, a)\}$,

– $\delta'((q^{(x)}, r), a) = \begin{cases} \{(t^{(xa)}, r) \mid t \in \delta(q, a)\}, & \text{if } |x| \leq k-1 \\ \{(t^{[x]}, r), (t^{(x)}, r) \mid t \in \delta(q, a)\}, & \text{if } |x| = k \end{cases}$

– $\delta'((q^{[xa]}, r), a) = \{(t^{[x]}, r) \mid t \in \delta(q, a)\}$.

In the definition of δ', we take $q, r \in Q \setminus \{q_0\}$, $x \in V^*, |x| \leq k$.

It is rather easy to see that A' accepts exactly $HRS_k(L)$. Since $HRP_k(L) = (HRS(L^R))^R$, the proof is complete. □

The class of context-free languages behaves in the same way with respect to the closure under hairpin completion and reduction. Namely,

Theorem 3. *The class of context-free languages is closed under neither hairpin completion nor hairpin reduction.*

Proof. The non-closure under hairpin completion is shown in [3]. We take the context free language $L = \{ba^n\overline{ca^n b}dba^m\overline{ca^m b} \mid n, m \geq k\}$. Given any $k \geq 1$, $HR_k(L)$ intersected with the regular language defined by the regular expression $ba^+\overline{ca^+b}dba^+\overline{c}$ gives the non-context-free language $L = \{ba^n\overline{ca^n b}dba^n\overline{c} \mid n \geq k\}$. □

The behavior of the class **P** of polynomially recognizable languages with respect to the hairpin completion/reduction is the same, namely

Theorem 4. *For every $k \geq 1$, if L is recognizable in $\mathcal{O}(f(n))$ time, then both $HC_k(L)$ and $HR_k(L)$ are recognizable in $\mathcal{O}(nf(n))$ time. Therefore,* **P** *is closed under hairpin reduction.*

Proof. We consider here the hairpin reduction only; the statement for hairpin completion is proved in [10]. Let $L \subseteq V^*$ be a language recognizable in $\mathcal{O}(f(n))$ time, k be an positive integer, and w an arbitrary word over V of length n. The next function decides whether or not $w \in HRS_k(L)$ in $\mathcal{O}(nf(n))$ time.

Algorithm 1.

```
function $Rec\_HRS(w, L, k)$;
begin
for $i$=1 to $\left[\frac{n-1}{2}\right]$
    if $(w[i+1..i+k-1] = \overline{w[n-k+1..n]^R})$ and $(w\overline{w[1..i]^R} \in L)$
        then $Rec\_HRS$:=true; halt;
    endif;
endfor;
$Rec\_HRS$:=false;
end.
```

A similar function can be easily constructed for deciding whether or not $w \in HRP_k(L)$ in $\mathcal{O}(nf(n))$ time. □

In [10] one proves that the n factor is not needed for the class of regular and context-free languages in the case of hairpin completion. In the case of hairpin reduction, the n factor is still not needed for regular languages as shown below, but we do not know what happens with the class of context-free languages.

Let L be a regular language accepted by the deterministic finite automaton $A = (Q, V, q_0, F, \delta)$. The next function decides whether or not $w \in HRS_k(L)$ in $O(n)$ time.

Algorithm 2.

```
function $Rec\_HRS\_REG(w, L, k)$;
begin
$q := \delta(q_0, w)$;
$a[0] := F$;
for $i$=1 to $\left[\frac{n-1}{2}\right]$ $a[i] = \{s \in Q \mid \delta(s, \overline{w[i]}) \in a[i-1]\}$;
endfor;
for $i$=1 to $\left[\frac{n-1}{2}\right]$
    if $(w[i+1..i+k-1] = \overline{w[n-k+1..n]^R})$ and $(q \in a[i])$
        then $Rec\_HRS\_REG$:=true; halt;
    endif;
endfor;
$Rec\_HRS\_REG$:=false;
end.
```

The case of space complexity classes is slightly different, namely:

Theorem 5. *Let $f(n) \geq \log n$ be a space-constructible function.*

1. $NSPACE(f(n))$ and $DSPACE(f(n))$ are closed under k-hairpin completion for any $k \geq 1$.

2. If $f(n+n/2) \in \mathcal{O}(f(n))$, then $NSPACE(f(n))$ and $DSPACE(f(n))$ are closed under k-hairpin reduction for any $k \geq 1$.

Proof. Again, we prove here the closure under hairpin reduction only. Let $L \subseteq V^*$ be a language recognizable by an off-line Turing machine in $\mathcal{O}(f(n))$ space, k be an positive integer, and w an arbitrary word over V of length n. The function defined by Algorithm 1 can clearly be implemented on an off-line nondeterministic (multi-tape) Turing machine in $\mathcal{O}(f(n))$ space. Note that $\log n$ is needed in order to store the value of i within the input word w that indicates the current word $w\overline{w[1..i]^R}$ whose membership to L is tested. As the maximal length of $w\overline{w[1..i]^R}$ is $n+n/2$, and $f(n+n/2) \in \mathcal{O}(f(n))$, the overall space of the Turing machine is in $\mathcal{O}(f(n))$. By finite state one can keep track of whether or not the first condition of the if statement is satisfied. □

Note that the class of function satisfying the conditions required by the second statement of the previous theorem is pretty large; it contains all polynomials, all functions $\log^p n$ and is closed under addition and multiplication.

4 The Iterated Case

As shown in [3,10], Theorem 5 remains valid for iterated hairpin completion while the n factor from Theorem 4 becomes n^2 for iterated hairpin completion; however, this factor is not needed in the case of context-free languages. The situation of iterated hairpin reduction seems much more difficult: we were not able to settle even whether or not the language $HR_k^*(L)$ is always recursive provided that L is regular.

We consider here another concept that appears attractive to us. Given a word $x \in V^*$ and a positive integer k, the word y is said to be the *primitive hairpin root* of x if the following two conditions are satisfied:

$$\begin{aligned} (i)\ & y \in HR_k^*(x) (\text{or, equivalent, } x \in HC_k^*(y)),\\ (ii)\ & HR_k(y) = \emptyset. \end{aligned}$$

In other words, y can be obtained from x by iterated hairpin reduction (maybe in zero steps) and y cannot be further reduced by hairpin reduction. Clearly, a word may have more than one primitive hairpin root; the set of all primitive hairpin roots of a word x is denoted by $H_k root(x)$. Naturally, the primitive hairpin root of a language L is defined by $H_k root(L) = \bigcup_{x \in L} H_k root(x)$.

We start our investigation on primitive hairpin roots with some remarks about the primitive hairpin root of a regular language. Clearly, a given language L is its primitive hairpin root if and only if every word of L is its primitive hairpin root, if and only if $HR_k(L) = \emptyset$. Since $HR_k(L)$ is a regular language for any regular language L, it follows that one can decide whether or not a given regular language is its primitive hairpin root. Moreover, if $H_k root(L) \neq L$, then it might happen that $H_p root(L) = L$ for some $p > k$; as soon as $H_p root(L) = L$, then $H_t root(L) = L$ for all $t \geq p$. Therefore, it makes sense to compute the minimal k, if any, such that $H_k root(L) = L$.

Theorem 6. *Given a regular language L, one can decide whether or not there exists k such that $H_k root(L) = L$. Moreover, if such a k exists, then one can algorithmically compute the minimal one in polynomial time with respect to the number of states of the minimal automaton recognizing L.*

Proof. Let L be a regular language and let $A = (Q, V, q_0, F, \delta)$ be the minimal finite automaton, with n states, accepting L. The following algorithm outputs the minimal k such that $H_k root(L) = L$ or $k = \infty$ provided that there is no k such that $H_k root(L) = L$.

Algorithm 3.

```
begin
$k := 0$;
for every pair of states $(q_1, q_2)$ such that $\delta(q_1, x) = q_2$ for some $x \in V^+$
   $L_0(q_1) = \{w \in V^+ \mid \delta(q_0, w) = q_1\}$;
   $L_f(q_2) = \{w \in V^+ \mid \delta(q_2, w) \in F\}$;
   if $L_0(q_1) \cap \overline{L_f(q_2)^R}$ is infinite then $k := \infty$; halt; else
      if $max\{|x| \mid x \in L_0(q_1) \cap \overline{L_f(q_2)^R}\} > k$ then $k := max\{|x| \mid x \in L_0(q_1) \cap \overline{L_f(q_2)^R}\}$;
      endif;
   endif;
endfor
end.
```

It is clear that the automaton recognizing the intersection $L_0(q_1) \cap \overline{L_f(q_2)^R}$ can be constructed in $\mathcal{O}(n^2)$. The finiteness of this intersection can be decided in $\mathcal{O}(n^2)$. The length of the longest word in the aforementioned intersection, provided that it is finite, can be computed in $\mathcal{O}(n^2)$. Therefore, the overall computing time of the algorithm is $\mathcal{O}(n^4)$. □

By the results reported in [10], given x and y one can decide whether or not y is a descendant of x in time $\mathcal{O}(|y|^2)$; in other words, given x and y one can decide whether or not x is an ancestor of y in time $\mathcal{O}(|y|^2)$. Since one can test in linear time if $HR_k(x) = \emptyset$, given a word x, it follows that for two words x and y one can decide whether or not x is a primitive hairpin root of y in time $\mathcal{O}(|y|^2)$.

On the other hand, the problem of deciding the existence of a common descendant of two given words is left open in [10]. We show that the existence of a common ancestor of two given words is decidable in polynomial time.

Theorem 7. *Given two words x, y, one can decide the existence of a k-common ancestor in $\mathcal{O}(max(|x|, |y|)^3)$ for any $k \geq 1$.*

Proof. Assume that x is a word of length n; first we compute an $n \times n$ matrix P_x defined by

$$P_x[i][j] = \begin{cases} \max(\{t \mid x[i..i+t-1] = \overline{x[j-t+1..j]^R}\} \cup \{0\}), j-i \geq 2k \\ 0, \text{ otherwise} \end{cases}$$

This matrix can be easily computed in time $\mathcal{O}(n^2)$ by using the relation

$$P_x[i][j] = \begin{cases} P_x[i+1][j-1]+1, \text{ if } x[i] = \overline{x[j]} \\ 0, \text{ if } x[i] \neq \overline{x[j]} \end{cases}$$

Then we compute the matrix $M_x[i][j]$ defined by

$$M_x[i][j] = \min\{t \mid x[i..j] \text{ is obtained from } x \text{ by } t \text{ hairpin reductions}\}.$$

This computation can be accomplish in $\mathcal{O}(n^3)$ time as shown in the next algorithm. In this algorithm, $\mathcal{S}$ is a queue storing pairs of integers between 1 and n; by $\longrightarrow$ we denote the insertion of an element in the queue while the extraction of an element from the queue is denoted by $\longleftarrow$. We assume that initially $M_x[1][n] = 0$ and $M_x[i][j] = \infty$ for all $(i, j) \neq (1, n)$.

Algorithm 4.

```
begin
$(1,n) \longrightarrow \mathcal{S}$;
while $\mathcal{S} \neq \emptyset$
  $(i,j) \longleftarrow \mathcal{S}$;
  if $P_x[i][j] > k$ then
    for $t = 1$ to $P_x[i][j] - k$
      if $M_x[i][j-t] > M_x[i][j]+1$ then $M_x[i][j-t] := M_x[i][j]+1$
        if $(i,j-t)$ has never been in $\mathcal{S}$ then $(i,j-t) \longrightarrow \mathcal{S}$
        endif;
      endif;
      if $M_x[i+t][j] > M_x[i][j]+1$ then $M_x[i+t][j] := M_x[i][j]+1$
        if $(i+t,j)$ has never been in $\mathcal{S}$ then $(i+t,j) \longrightarrow \mathcal{S}$
        endif;
      endif;
    endfor;
  endif;
endwhile;
end.
```

We now compute P_y and M_y in the same way. The overall time needed so far is $\mathcal{O}(|x|^3 + |y|^3)$. Finally, we check for every subword $x[i..j]$ of x such that $M_x[i][j] \neq \infty$ whether there exist s, t such that $x[i..j] = y[s..t]$ and $M_y[s][t] \neq \infty$. This step can be accomplished, either using the KMP algorithm ([8]) or using the Galil-Seiferas algorithm ([5]), in time $\mathcal{O}(|x|^2(|x| + |y|))$, which concludes the proof. □

The *hairpin reduction distance* between two words x and y is defined as the minimal number of hairpin reduction which can be applied either to x in order to obtain y or to y in order to obtain x. If none of them can be obtained from the other by iterated hairpin reduction, then the distance is ∞. Clearly, this measure is not a mathematical distance. As soon as the array M_x has been constructed, one can find a primitive hairpin root of x closest or farthest to x without extra time. Also the common primitive root of x and y or a common ancestor, such that the sum of its hairpin reduction distance to x and y is minimum, can be computed in the same computational time.

References

1. Bottoni, P., Labella, A., Manca, V., Mitrana, V.: Superposition based on Watson-Crick-like complementarity. Theory of Computing Systems 39(4), 503–524 (2006)
2. Castellanos, J., Mitrana, V.: Some remarks on hairpin and loop languages. In: Ito, M., Păun, G., Yu, S. (eds.) Words, Semigroups, and Translations, pp. 47–59. World Scientific, Singapore (2001)
3. Cheptea, D., Martin-Vide, C., Mitrana, V.: A new operation on words suggested by DNA biochemistry: hairpin completion, In: Proc. Transgressive Computing, pp. 216–228 (2006)
4. Deaton, R., Murphy, R., Garzon, M., Franceschetti, D.R., Stevens, S.E.: Good encodings for DNA-based solutions to combinatorial problems, In: Landweber, L.F., Baum, E (eds). Proc. of DNA-based computers II, DIMACS Series, vol. 44, pp. 247–258 (1998)
5. Galil, Z., Seiferas, J.: Time-space optimal string matching. Journal of Computer and System Sciences 26, 280–294 (1983)
6. Garzon, M., Deaton, R., Neathery, P., Murphy, R.C., Franceschetti, D.R., Stevens, E.: On the encoding problem for DNA computing, The Third DIMACS Workshop on DNA-Based Computing, Univ. of Pennsylvania, pp. 230–237 (1997)
7. Garzon, M., Deaton, R., Nino, L.F., Stevens, Jr S.E., Wittner, M.: Genome encoding for DNA computing, In: Proc. Third Genetic Programming Conference, Madison, MI, pp. 684–690 (1998)
8. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal of Computing 6, 323–350 (1977)
9. Kari, L., Konstantinidis, S., Sosik, P., Thierrin, G.: On hairpin-free words and languages. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 296–307. Springer, Berlin (2005)
10. Manea, F., Martín-Vide, C., Mitrana, V.: On some algorithmic problems regarding the hairpin completion (submitted)
11. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing. New Computing Paradigms, Springer-Verlag, Berlin, 1998, Tokyo (1999)
12. Păun, G., Rozenberg, G., Yokomori, T.: Hairpin languages. Intern. J. Found. Comp. Sci. 12(6), 837–847 (2001)
13. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 3. Springer, Berlin, Heidelberg (1997)
14. Sakamoto, K., Gouzu, H., Komiya, K., Kiga, D., Yokoyama, S., Yokomori, T., Hagiya, M.: Molecular computation by DNA hairpin formation. Science 288, 1223–1226 (2000)

Hierarchies in Fragments of Monadic Strict NP

Barnaby Martin and Florent Madelaine

Department of Computer Science, University of Durham,
Science Labs, South Road, Durham DH1 3LE, U.K.

Abstract. We expose a strict hierarchy within *monotone monadic strict* NP *without inequalities* (MMSNP), based on the number of second-order monadic quantifiers. We do this by studying a finer strict hierarchy within a class of *forbidden patterns problems* (FPP), based on the number of permitted colours. Through an adaptation of a preservation theorem of Feder and Vardi, we are able to prove that this strict hierarchy also exists in *monadic strict* NP (MSNP). Our hierarchy results apply over a uniform signature involving a single binary relation, that is over digraphs.

1 Introduction

Answering a question of Fagin, Martin Otto proved in [1] that there is a strict hierarchy in *monadic* NP (MNP) – the monadic fragment of existential second-order logic – based on the second-order quantifier rank, i.e. the number of second-order quantifiers. The strictness of the hierarchy is proved with a uniform signature involving two binary relations. It is worth noting that this hierarchy was known to collapse to its first level in the very restricted case of word structures (strings). In fact, MNP with a single second-order quantifier is as powerful as the whole of monadic second-order logic – not just its existential fragment – on word structures, capturing exactly the class of regular languages [2,3,4].

In this paper we search for a similar, second-order quantifier-rank-based, hierarchy within *monadic strict* NP (MSNP), and its monotone inequality-free fragment (MMSNP). We note that the problems involved in Otto's proof are not monotone, and in any case require first-order existential quantification – placing them outside MSNP. We achieve our hierarchy theorems by proving a strict hierarchy within a class of *forbidden patterns problems* (FPP), introduced in [5,6] to provide a combinatorial characterisation for MMSNP, based on the number of permitted colours. Specifically, we are able to prove that the digraph colourability problem $k+1$-Col is expressible in the $k+1$th level of FPP, but is not expressible in the kth level. We can then derive that 2^{k+1}-Col is expressible in the $k+1$th level of MMSNP (respectively, MSNP), but is not expressible in the kth level. We work in FPP to expose a finer hierarchy than that in MMSNP; informally we demonstrate a complexity jump between the problems k-Col and $k+1$-Col, and not just between 2^k-Col and 2^{k+1}-Col. For the reader more familiar with the Ehrenfeucht-Fraïssé method, we provide in the appendix an overview of how the hierarchy result for MMSNP (and consequently MSNP) may be so obtained. Like Otto, we do not require extensional relations of increasing arity for our hierarchy results; we work with a uniform signature involving a single binary relation, i.e. on digraphs.

MMSNP was studied by Feder and Vardi [7] because of its close relationship with the non-uniform constraint satisfaction problem CSP(T) – for an input structure A, does A admit a homomorphism to a fixed template T? Not only can every CSP(T) be easily recast as some MMSNP query ψ_T, but it is now known that, for every ψ in MMSNP, there is a T_ψ such that the query evaluation problem for ψ and CSP(T_ψ) are polynomial-time equivalent [7,8]. If one were to consider classes of non-uniform constraint satisfaction problems CSP^k in which the template was restricted to being of size $\leq k$, then it would be virtually immediate that this hierarchy was strict and, indeed, separated by the problems k-Col. Essentially, one could not express the problem $k+1$-Col as a CSP whose template T has less than $k+1$ vertices, since then the $k+1$-clique K_{k+1}, which is plainly $k+1$-colourable, would manifest as a no-instance. The relationship between the size of a CSP template T and the number of colours required to express it as a FPP, is explored in [5]. Certainly any CSP whose template T is of size k can be expressed as a FPP whose forbidden patterns involve k colours. However, this relationship is not known to hold in the converse, and, therefore, proving the corresponding strict hierarchy in FPP does not appear to be trivial.

Following the necessary preliminaries, the paper is organised as follows. In Section 3 we prove the hierarchy result for FPP and in Section 4 we derive the related result for MMSNP. In Section 5 we demonstrate how to adapt a certain preservation theorem of Feder and Vardi to derive the same hierarchy in MSNP. At the end of the paper sits an appendix, in which we show how our results may be obtained through the ubiquitous Ehrenfeucht-Fraïssé games.

2 Preliminaries

In this paper, the only structures we consider are finite, non-empty digraphs. A *digraph* G consists of a finite *vertex set* $V(G)$ together with an *edge set* $E(G) \subseteq V(G) \times V(G)$. For a positive integer k, let $[k]$ be the set $\{0, \ldots, k-1\}$. A *k-coloured digraph* is a pair (G, c^k) where G is a digraph and c^k is a function from $V(G)$ to the colour set $[k]$ (note that we do not require that a colouring be 'proper', i.e. we do not force adjacent vertices to take different colours). If the range of c^k is the singleton $\{i\}$, then we refer to (G, c^k) as *i-monochrome*.

A homomorphism between the digraphs G and H is a function $h : V(G) \rightarrow V(H)$ such that, for all $x, y \in V(G)$, $(x, y) \in E(G)$ implies $(h(x), h(y)) \in E(H)$. A homomorphism between the k-coloured digraphs (G, c^k_G) and (H, c^k_H) is a digraph homomorphism $h : G \rightarrow H$ that also respects the colouring of G, i.e., for all $x \in V(G)$, $c^k_H(h(x)) = c^k_G(x)$. Existence (respectively, non-existence) of a homomorphism between entities P and Q is denoted $P \longrightarrow Q$ (respectively, $P \not\longrightarrow Q$).

Let K_k be the antireflexive k-clique, that is the digraph with vertex set $[k]$ and edge set $\{(i,j) : i \neq j\}$. Define the problem k-Col to be the set of digraphs G which admit a homomorphism to K_k. We describe digraphs G s.t. $G \in$ 2-Col as *bipartite*. An edge of the form (x, x) in a digraph G is described as a *self-loop*; a digraph with no self-loops is said to be *antireflexive*. It is a simple observation that a digraph with a self-loop cannot map homomorphically into an antireflexive digraph.

Let $\mathscr{R}^k$ be some finite set of k-coloured digraphs. Define the *forbidden patterns problem* $FPP(\mathscr{R}^k)$ to be the set of digraphs G for which there exists a k-colouring c^k_G such that, for all $(H, c^k_H) \in \mathscr{R}^k$, $(H, c^k_H) \not\longrightarrow (G, c^k_G)$. Intuitively, $FPP(\mathscr{R}^k)$ is the class of digraphs for which there exists a k-colouring that forbids homomorphism from all of the k-coloured digraphs of $\mathscr{R}^k$, whence we refer to $\mathscr{R}^k$ as the set of *forbidden patterns*. Define FPP^k to be the class of problems $FPP(\mathscr{R}^k)$, where $\mathscr{R}^k$ ranges over all finite sets of k-coloured digraphs, and let FPP be $\cup_{i \in \omega} \mathrm{FPP}^i$.

The logic *k-monadic* NP (MNP^k) will be considered that fragment of monadic existential second-order logic that allows at most k second-order quantifiers. The logic *k-monadic strict* NP (MSNP^k) is that fragment of MNP^k that involves prenex sentences whose first-order quantification is purely universal. We may therefore consider MSNP^k to be the the class of sentences φ of the form

$$\exists \mathbf{M} \forall \mathbf{v} \; \Phi(\mathbf{M}, \mathbf{v}),$$

where $\mathbf{M}$ is an k-tuple of monadic relation symbols and Φ is quantifier-free. In these logics, we refer to k as the *second-order quantifier rank*. The logic *k-monotone* MSNP *without inequalities* (MMSNP^k) is defined similarly, but with the additional restriction that Φ be of the form

$$\bigwedge_i \neg(\alpha_i(\mathbf{v}) \wedge \beta_i(\mathbf{M}, \mathbf{v})),$$

where: α_i is a conjunction of positive atoms, involving neither equality nor relations from $\mathbf{M}$; and β_i is a conjunction of positive or negative atoms, involving only relations from $\mathbf{M}$. Define MNP (respectively, MSNP, MMSNP) to be $\cup_{i \in \omega} \mathrm{MNP}^i$ (respectively, $\cup_{i \in \omega} \mathrm{MSNP}^i$, $\cup_{i \in \omega} \mathrm{MMSNP}^i$). The following hierarchy theorem for MNP is due to Otto.

Theorem 1 ([1]). *For all k,* $\mathrm{MNP}^k \subseteq \mathrm{MNP}^{k+1}$ *but* $\mathrm{MNP}^k \neq \mathrm{MNP}^{k+1}$.

Furthermore, the following is straightforward.

Proposition 1. *For all k, we have the inclusions* $\mathrm{FPP}^k \subseteq \mathrm{FPP}^{k+1}$*,* $\mathrm{MSNP}^k \subseteq \mathrm{MSNP}^{k+1}$ *and* $\mathrm{MMSNP}^k \subseteq \mathrm{MMSNP}^{k+1}$.

Proof. For the first part, let $FPP(\mathscr{R}^k)$ be a problem of FPP^k. Construct $\mathscr{R}^{k+1}$ from $\mathscr{R}^k$ by the addition of a k-monochrome copy of K_1. Since the extra colour is now forbidden, it is plain to see that $FPP(\mathscr{R}^{k+1}) = FPP(\mathscr{R}^k)$.

For the second part, let $\exists M_0 \ldots \exists M_{k-1} \forall \mathbf{v} \; \Phi$ be a sentence of MSNP^k, and v be one of the variables of $\mathbf{v}$. Then $\exists M_0 \ldots \exists M_{k-1} \exists M_k \forall \mathbf{v} \; \Phi \wedge \neg M_k(v)$ is an equivalent sentence of MSNP^{k+1}.

The third part may be proved in the same manner. □

The contribution of this paper will be to prove that these inclusions are strict.

The following result ties together FPP and MMSNP.

Theorem 2 ([5]). *The class of problems expressible in* MMSNP^k *coincides exactly with the class of problems that are finite unions of problems in* FPP$^{2^k}$*.*

Example 1. Let us consider the problem 2-Col. This is a forbidden patterns problem $FPP(\mathscr{R}^2)$, where $\mathscr{R}^2$ consists of two coloured digraphs (P_1, c_0^2) and (P_1, c_1^2), which are 0- and 1-monochrome, respectively, where P_1 is the digraph with vertex set $\{0, 1\}$ and a single edge $(0, 1)$. In the following depiction, we may view the white vertices as coloured 0, and the black vertices as coloured 1.

$$\mathscr{R}^2 \ := \ \{ \ \circ\!\longrightarrow\!\circ \ , \ \bullet\!\longrightarrow\!\bullet \ \}$$

2-Col may also be expressed by the sentence φ of MMSNP^1:

$$\exists M \forall u \forall v \ \neg(E(u,v) \wedge M(u) \wedge M(v)) \ \wedge \ \neg(E(u,v) \wedge \neg M(u) \wedge \neg M(v)).$$

Define the *chromatic number* of a digraph G to be the minimal k such that $G \in$ k-Col. We define the *symmetric closure* of a digraph G, denoted $Sym(G)$, over the same vertex set as G, but with edge set $\{(x,y),(y,x) : (x,y) \in E(G)\}$. For $k \geq 3$, let C_k be the undirected k-cycle, that is the digraph with vertex set $[k]$ and edge set $\{(i,j),(j,i) : j = i + 1 \bmod k\}$. Define the *odd girth* of an antireflexive, non-bipartite digraph G to be the minimal odd k s.t. C_k is (isomorphic to) an induced subdigraph of $Sym(G)$ (note that this is always defined). We define the odd girth of an antireflexive, non-bipartite coloured digraph likewise. It may be easily verified that, if two digraphs G and H have odd girth γ_G and γ_H, respectively, with $\gamma_G \leq \gamma_H$, then $G \not\longrightarrow H$.

We require the following lemma, originally proved by Erdös through the probabilistic method [9], but for which the citation provides a constructive proof.

Lemma 1 (See [10]). *For all i, one may construct a digraph B_i whose chromatic number and odd girth both strictly exceed i.*

3 A Strict Hierarchy in FPP

In this section we aim to prove that there is a strict hierarchy in FPP given by the number of colours allowed in the set $\mathscr{R}$. We will establish this through the following theorem.

Theorem 3. *For each $k \geq 1$, $k+1$-*Col $\in$ FPP^{k+1} *but $k+1$-*Col $\notin$ FPP^k*.*

Proof. ($k+1$-Col $\in$ FPP^{k+1}.) This follows similarly to Example 1. $k+1$-Col is expressed by $FPP(\mathscr{R}^{k+1})$, where $\mathscr{R}^{k+1}$ consists of $k+1$ coloured digraphs $(P_1, c_0^2), \ldots, (P_1, c_k^2)$, in which, for $0 \leq i \leq k$, (P_1, c_i^2) is i-monochrome.

($k+1$-Col $\notin$ FPP^k.) Suppose that $k+1$-Col $\in$ FPP^k, and is expressed by $FPP(\mathscr{R}^k)$ where $\mathscr{R}^k$ is a finite set of k-coloured digraphs. We are therefore claiming that, for all digraphs G:

$$(*) \quad G \in k+1\text{-Col} \ \text{ iff } \ \text{exists } c_G^k \text{ s.t. } \forall \ (H, c_H^k) \in \mathscr{R}^k \quad (H, c_H^k) \not\longrightarrow (G, c_G^k).$$

First, we aim to prove that, for every i, $\mathscr{R}^k$ must contain some i-monochrome bipartite digraph. Suppose, for some i, it does not. Let the maximum odd girth of the coloured digraphs of $\mathscr{R}^k$ be γ; if all members of $\mathscr{R}^k$ possess a self-loop or are bipartite, set $\gamma := 3$. Set μ to be $1 + \max\{k, \gamma\}$. By Lemma 1, we can construct a graph B_μ whose chromatic number and odd girth both strictly exceed μ. We now deduce from $(*)$ the absurdity $B_\mu \in k + 1$-Col, since the i-monochrome colouring of B_μ forbids homomorphism from all of the coloured digraphs of $\mathscr{R}^k$ (recall that any bipartite members of $\mathscr{R}^k$ are not i-monochrome).

Now we aim to prove that $K_{k+1} \notin FPP(\mathscr{R}^k)$. Consider any k-colouring c^k of K_{k+1}; there must be distinct vertices x and y such that $c^k(x) = c^k(y)$, let their colour be i. But we know that $\mathscr{R}^k$ contains an i-monochrome bipartite digraph, which plainly maps homomorphically into (K_{k+1}, c^k) (in fact into its i-monochrome subdigraph K_2 induced by $\{x, y\}$). By definition, we deduce that $K_{k+1} \notin FPP(\mathscr{R}^k)$.

Finally, we reach a contradiction since K_{k+1} is plainly in $k + 1$-Col. □

4 A Strict Hierarchy in MMSNP

We now show how to adapt the previous proof to generate the following[1].

Theorem 4. *For $k \geq 0$, 2^{k+1}-*Col *$\in$ MMSNP^{k+1} but 2^{k+1}-*Col *$\notin$ MMSNP^k.*

Proof. (2^{k+1}-Col $\in$ MMSNP^{k+1}.) This follows similarly to Example 1. 2^{k+1}-Col may be expressed by the following sentence of MMSNP^{k+1}:

$$\exists M_0 \ldots \exists M_k \forall u \forall v \bigwedge_{i \in [2^{k+1}]} \neg(E(u,v) \wedge \Psi_i(M_0, \ldots, M_k, u, v)),$$

where $\Psi_i(M_0, \ldots, M_k, u, v)$ is

$$\begin{aligned}(\neg)^{i_0} M_0(u) \wedge (\neg)^{i_0} M_0(v) \wedge \; & (\neg)^{i_1} M_1(u) \wedge (\neg)^{i_1} M_1(v) \wedge \\ & \ldots \wedge \; (\neg)^{i_k} M_k(u) \wedge (\neg)^{i_k} M_k(v)\end{aligned}$$

where i_j is the $j + 1$th digit in the binary expansion of i.

(2^{k+1}-Col $\notin$ MMSNP^k.) Suppose that 2^{k+1}-Col $\in$ MMSNP^k. By Theorem 2, this implies that 2^{k+1}-Col is the union, for some s, of the forbidden pattern problems $FPP(\mathscr{R}_0^{2^k})$, $\ldots$, $FPP(\mathscr{R}_{s-1}^{2^k})$. In a similar vein to before, we can deduce that, for each j ($0 \leq j < s$), $\mathscr{R}_j^{2^k}$ contains, for each i ($0 \leq i < 2^k$), an i-monochrome bipartite digraph. The proof concludes as before. □

Remark 1. Our proof can actually go further, yielding not just 2^{k+1}-Col $\notin$ MMSNP^k, but also $2^k + 1$-Col $\notin$ MMSNP^k.

5 A Strict Hierarchy in MSNP

We say that a class of finite digraphs $\mathcal{C}$ is *closed under inverse homomorphism* iff whenever we have $G \longrightarrow H$ and $H \in \mathcal{C}$ we also have $G \in \mathcal{C}$. Similarly, a class of

[1] By abuse of notation, we write that a class of digraphs belongs to a logic precisely when that class is expressible in the logic, e.g. 2^{k+1}-Col $\in$ MMSNP^{k+1}.

finite digraphs is *antireflexive* iff each digraph within it is antireflexive. It follows straight from our definition that, for each k, the class k-Col is both antireflexive and closed under inverse homomorphism. The following is from [11], and is an example of a preservation theorem.

Theorem 5 ([11]). *For every $\psi \in$ MSNP s.t. the class of finite models of ψ is closed under inverse homomorphism, there exists $\psi' \in$ MMSNP s.t. ψ and ψ' agree on all finite models.*

In fact, we will require a variant on their proof, to derive the following theorem (whose proof we defer to the end of this section).

Theorem 6. *For every $\psi \in \mathrm{MSNP}^k$ s.t. the class of finite models of ψ is both antireflexive and closed under inverse homomorphism, there exists $\psi' \in \mathrm{MMSNP}^k$ s.t. ψ and ψ' agree on all finite models.*

We are now in a position to state and prove the main result of this section.

Theorem 7. *For each $k \geq 0$, 2^{k+1}-Col $\in \mathrm{MSNP}^{k+1}$ but 2^{k+1}-Col $\notin \mathrm{MSNP}^k$.*

Proof. Membership follows as in Theorem 4.

(2^{k+1}-Col $\notin \mathrm{MSNP}^k$.) Note that 2^{k+1}-Col is an antireflexive class that is closed under inverse homomorphism. By the previous theorem that implies that it may be expressed in MSNP^{k+1} only if it may be expressed in MMSNP^{k+1}, which we know it can not – by Theorem 4. □

Proof (of Theorem 6). We show how to adapt the proof of Theorem 3 of [11]. In [11], they demonstrate how, if ψ_0 is a sentence of MSNP whose finite models form a class closed under inverse homomorphism, to construct a sequence of sentences culminating with ψ_5 in MMSNP s.t. ψ_0 and ψ_5 agree on all finite models. Unfortunately, it is not the case that the second-order quantifier rank is preserved: in their construction of ψ_2 from ψ_1 it may be necessary to introduce new second-order monadic relations. In all other of their translations, the second-order quantifier rank is preserved. Our proof uses the additional constraint of antireflexivity to amend the translation from ψ_1 to ψ_2 into something very simple that does preserve the second-order quantifier rank. This is the only area in which our proof differs from theirs. We now sketch the proof.

Starting with a sentence ψ_0 of MSNP^k whose finite models form an antireflexive class that is closed under inverse homomorphism, we will describe a sequence of sentences culminating in ψ_5 that is in MMSNP^k and agrees with ψ_0 on all finite digraphs. We may assume that ψ_0 is in prenex form with its quantifier-free part in conjunctive normal form, where we interpret each clause as a negated conjunction. That is, ψ_0 is of the form

$$\exists \mathbf{M} \forall \mathbf{v} \bigwedge_i \neg(\bigwedge_j \alpha_{ij}(\mathbf{M}, \mathbf{v}))$$

where each α_{ij} is atomic.

From ψ_0 we generate ψ_1 by enforcing that, if distinct u and v occur in some negated conjunct, then $u \neq v$ also occurs in that conjunct. If this is not already the case, then we split the negated conjunct in two, one involving $u \neq v$ and the other involving $u = v$, whereupon, in the latter case, we may substitute all occurrences of v with u, dispensing with the equality.

From ψ_1 we generate ψ_2 by removing any atomic instances of $\neg E(v, v)$. From ψ_2 we generate ψ_3 by removing any negated conjuncts that contain either an instance $v \neq v$ or both atoms $E(u, v)$ and $\neg E(v, u)$. From ψ_3 we generate ψ_4 by removing all negative atoms. Finally, from ψ_4 we generate ψ_5 by removing all inequalities. It is transparent that ψ_5 is in MMSNP^k. It remains for us to settle the following.

Lemma 2. *Let ψ_0 be a sentence of* MSNP^k *in the required form. Then, on the class of finite digraphs,*

(i) ψ_0 is equivalent to ψ_1,
(ii) ψ_1 is equivalent to ψ_2 (since ψ_1 describes an antireflexive class),
(iii) ψ_2 is equivalent to ψ_3,
(iv) ψ_3 is equivalent to ψ_4, and
(v) ψ_4 is equivalent to ψ_5 (since ψ_4 describes a class closed under inverse homomorphism).

(*i*) and (*iii*) are trivial (and appear in [11]). Part (*ii*) is transparent. Parts (*iii*) and (*iv*) are non-trivial and appear as Lemmas 8 and 7, respectively, in [11]. □

6 Further Work

The problems in Otto's proof of the strict hierarchy in MNP are colouring problems of a kind. However, they demand highly regular structures that, in some sense, make them less natural than the problems k-Col. It would be interesting to know whether the problems 2^k-Col separate the hierarchy in MNP; that is, whether 2^{k+1}-Col can be proved inexpressible in MNP^k. Our attempts to use Ehrenfeucht-Fraïssé games (even Ajtai-Fagin games) to settle this have not, thus far, succeeded.

References

1. Otto, M.: A note on the number of monadic quantifiers in monadic Σ^1_1. Information Processing Letters 53(6), 337–339 (1995)
2. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschrift für mathematische Logik und Grundladen der Mathematik 6, 66–92 (1960)
3. Elgot, C.: Decision problems of finite-automata design and related arithmetics. Trans. Amer. Math. Soc. 98, 21–51 (1961)
4. Thomas, W.: Classifying regular events in symbolic logic. Journal of Computer and System Sciences 25, 360–376 (1982)
5. Madelaine, F.: Constraint satisfaction problems and related logic. PhD thesis, University of Leicester (2003)
6. Madelaine, F., Stewart, I.A.: Constraint satisfaction, logic and forbidden patterns (SIAM Journal of Computing) 33 pp (To appear)

7. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. SIAM J. Comput. 28 (1999)
8. Kun, G.: Constraints, MMSNP and expander structures (2006)
9. Erdös, P.: Graph theory and probability. Canad. J. Math. 11, 34–38 (1959)
10. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. OUP (2004)
11. Feder, T., Vardi, M.: Homomorphism closed vs existential positive (2003)
12. Fagin, R.: Monadic generalized spectra. Z. Math. Logik Grund. Math. 21, 89–96 (1975)

Appendix

An Ehrenfeucht-Fraïssé Game for MMSNP

Ehrenfeucht-Fraïssé games traditionally provide the preferred method for separating logics. Here we state the relevant Ehrenfeucht-Fraïssé game for MMSNP and give its methodology theorem. We are then able to give an alternative proof of Theorem 4.

For digraphs G and H, the game $\mathcal{G}^m_q(G,H)$ is played between two players, Spoiler and Duplicator, and proceeds as follows.

– Spoiler chooses a 2^m-colouring $c^{2^m}_G$ of G;
– Duplicator responds with a 2^m-colouring $c^{2^m}_H$ of H.
– Spoiler places q pebbles $a_0,\ldots,a_{q-1}$ on H;
– Duplicator responds with q pebbles $b_0,\ldots,b_{q-1}$ on G.

Duplicator wins iff the resultant relation $\{(a_0,b_0),\ldots,(a_{q-1},b_{q-1})\}$ is a partial homomorphism from $(H,c^{2^m}_H)$ to $(G,c^{2^m}_G)$.

Let MMSNP^m_q be that fragment of MMSNP^m in which the first-order part of the sentences has quantifier-rank bounded by q (owing to the restricted syntax of MMSNP, we may equivalently consider q to be a bound on the number of first-order variables). The next theorem ties together the game and the logic; a proof, based on the connection between FPP and MMSNP appears at the end of the section (although it is possible to derive a more conventional proof similar to that given by Fagin for his original game for MNP [12]).

Theorem 8 (Methodology). *For digraphs G and H the following are equivalent.*

– *Duplicator has a winning strategy in the game $\mathcal{G}^m_q(G,H)$.*
– *For all $\varphi \in \text{MMSNP}^m_q$, $G \models \varphi$ implies $H \models \varphi$.*

We are now in a position to give another proof of Theorem 4.

Theorem 9 (a.k.a. Theorem 4). *For each $m \geq 0$, 2^{m+1}-*Col *$\notin \text{MMSNP}^m$.*

Proof. Suppose that 2^{m+1}-Col were expressible by a sentence $\psi^m_q \in \text{MMSNP}^m_q$, for some q. Set μ to be $1+\max\{2^{m+1},q\}$. Note that the digraph B_μ, constructed as in Lemma 1, is not in 2^{m+1}-Col. We aim to prove that Duplicator has a winning strategy in the game $\mathcal{G}_q^m(K_{2^{m+1}}, B_\mu)$, which, taken with the previous methodology theorem, leads to a contradiction.

Let Spoiler give a 2^m-colouring of $K_{2^{m+1}}$, and let x and y be some distinct vertices that are given some same colour i (such vertices clearly must exist). Duplicator chooses the i-monochrome colouring of B_μ. Now Spoiler places q pebbles on B_μ. Crucially, because of the enormous odd girth of B_μ, the subdigraph induced by these q pebbles must be bipartite. It therefore homomorphically maps onto the subdigraph K_2 of $K_{2^{m+1}}$ induced by the set $\{x, y\}$, and we are done. □

Proof (of Theorem 8). Let FPP_q^m be that subclass of FPP^m in which all the forbidden patterns in $\mathscr{R}^m$ have size bounded by q. We require the following, more sophisticated, version of Theorem 2, also proved in [5].

- The class of problems expressible in MMSNP_q^m coincides exactly with the class of problems that are finite unions of problems in $\text{FPP}_q^{2^m}$.

By this result, it suffices to prove that the following are equivalent.

(i) Duplicator has a winning strategy in the game $\mathcal{G}_q^m(G, H)$.
(ii) For all $\mathscr{R}_0^{2^m}, \ldots, \mathscr{R}_{s-1}^{2^m}$, whose members are of size bounded by q, we have that $G \in \bigcup_{i\in[s]} FPP(\mathscr{R}_i^{2^m})$ implies $H \in \bigcup_{i\in[s]} FPP(\mathscr{R}_i^{2^m})$.

$[(i) \Rightarrow (ii)]$ Consider a winning strategy for Duplicator in the game $\mathcal{G}_q^m(G, H)$, and any sequence of sets of forbidden patterns,each of whose members is bounded in size by q, $\mathscr{R}_0^{2^m}, \ldots, \mathscr{R}_{s-1}^{2^m}$. Further assume that $G \in \bigcup_{i\in[s]} FPP(\mathscr{R}_i^{2^m})$. It follows that there is some $i \in [s]$ s.t. $G \in FPP(\mathscr{R}_i^{2^m})$. We will prove that $H \in FPP(\mathscr{R}_i^{2^m})$ whereupon $H \in \bigcup_{i\in[s]} FPP(\mathscr{R}_i^{2^m})$ is immediate. Take the 2^m-colouring $c_G^{2^m}$ of G that witnesses its membership of $FPP(\mathscr{R}_i^{2^m})$, and consider Duplicator's response c_H^{2m} on H to it in her winning strategy in the game $\mathcal{G}_q^m(G, H)$. We claim that this witnesses the membership of H in $FPP(\mathscr{R}_i^{2^m})$; for, otherwise, if some forbidden pattern – of size bounded by q – of $\mathscr{R}_i^{2^m}$ mapped homomorphically into $(H, c_H^{2^m})$ then it would also map homomorphically into $(G, c_G^{2^m})$, by the winning strategy of Duplicator, which is a contradiction.

$[\neg(i) \Rightarrow \neg(ii)]$. Given a winning strategy for Spoiler in the game $\mathcal{G}_q^m(G, H)$, we will construct a set of forbidden patterns $\mathscr{R}^{2^m}$, each of whose size is bounded by q, such that $G \in FPP(\mathscr{R}^{2^m})$ but $H \notin FPP(\mathscr{R}^{2^m})$. Taking Spoiler's winning strategy, consider the size $\leq q$ induced subdigraph H' of H that he pebbles with $a_0, \ldots, a_{q-1}$. Let $\mathscr{R}^{2^m}$ be the set of all 2^m-colourings of H'. Now, $G \in FPP(\mathscr{R}^{2^m})$ and this is witnessed by Spoiler's initial colouring $c_G^{2^m}$ of G in his winning strategy. But $H \notin FPP(\mathscr{R}^{2^m})$ since any colouring of H admits homomorphism from itself, and consequently from the same colouring restricted to its induced subdigraph H'. □

Membrane Systems and Their Application to Systems Biology

Giancarlo Mauri

University of Milano-Bicocca
mauri@disco.unimib.it

1 Introduction

P-systems, or membrane systems [1], were introduced by George Păun as a class of unconventional computing devices of distributed, parallel and nondeterministic type, inspired by the compartmental structure and the functioning of living cells. The basic model consists of a membrane structure, described by a finite string of well matching parentheses, and graphically represented as regions on the plane, hierarchically embedded within an external region. Each membrane contains a multiset of *objects* (representing chemical substances) that evolve according to given *evolution rules* (representing reactions). Objects are described as symbols or strings over a given alphabet, evolution rules are given as rewriting rules. The rules act on objects, by modifying and moving them, and they can also affect the membrane structure, by dissolving the membranes. A computation in P systems starts from an initial configuration, identified by the membrane structure, the objects and the rules initially present inside each membrane, and then letting the system evolve. Assuming an universal clock, rules are applied in a nondeterministic and maximal parallel manner: all the applicable rules are used at each step to modify all objects which can be the subject of a rule, and this is done in parallel for all membranes; the evolved objects are then communicated to the regions specified by the rules. When no rule can be further applied, the computation halts and the output is defined in terms of the objects sent out of the external membrane or, alternatively, collected inside a specified membrane. No output is obtained if the computation never halts (that is, whenever a rule can be continuously applied). A comprehensive overview of basic P systems and of other classes appeared in [1], an updated bibliography can be found in the P systems Web Page ([2]). As a model of computation inspired by biological mechanisms, P systems have been extensively studied in the area of Natural Computing from the point of view of their computational power, and compared with other models like DNA computing or splicing systems. However, they can also be considered as a powerful tool to model complex systems and to simulate processes taking place inside them. In this view, they have been applied in various research areas, ranging from Biology to Linguistics to Computer Science (see, e.g., [3]), but very promising results have been obtained in simulating cellular phenomena, hence returning meaningful and useful information to biologists, in the frame of systems biology.

2 Stochastic Modelling

In this view, it is important to take into account the role of stochastic noise in modelling and simulating coupled chemical reactions (see the classical Gillespie algorithm [4]) and, in a more general frame, in cellular processes involving few molecules as, e.g., signal transduction pathways, and the working of transcription or translation machinery [5]. For this reason, in [6] the class of dynamical probabilistic P systems (DPPs) has been introduced for the analysis and simulation of the behavior of complex systems. DPPs are discrete and stochastic models, where probability values are associated with the rules, and such values change during the evolution according to the current state of the system. A different approach to stochastic modeling of biological systems has been given in [7].

3 Simulation Results

In order to check both the effectiveness and the correctness of the models, we designed and implemented a software simulator, that hopefully will become a tool for biologists for testing known data, predicting unknown scenarios and returning meaningful information. The last version of the simulator uses a novel method, called *tau leaping*, introduced by Gillespie et al. [8] and then adapted by Cazzaniga et al. [9] to work in the framework of P Systems (this new method is named tau-DPPs). Using tau-DPPs, we can simulate systems structured by several volumes, tracing the simulated time of the compartments as well as time line of the whole system. This gives us the possibility to quantitatively and qualitatively describe biological systems. Our model was able to simulate properly the Ras protein cycle, the activation of adenylate cyclase, the production of cyclic AMP and the activation of cAMP-dependent protein kinase in a single yeast cell of the yeast *Saccharomyces cerevisiae*. The results are compared with the experimental data and give information on the key regulatory elements of this signalling network. Another application was to metapopulation modeling [10]. A slightly different approach to modeling biological systems with P-systems can be found in [11].

References

1. Păun, Gh.: Membrane Computing. An Introduction. Springer, Berlin (2002)
2. The P Systems Web Page: `http://psystems.disco.unimib.it/`
3. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Berlin (2005)
4. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journ. Phys. Chem. 81, 2340–2361 (1977)
5. Meng, T.C., Somani, S., Dhar, P.: Modelling and simulation of biological systems with stochasticity. In Silico Biology 4, 293–309 (2004)
6. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. Int. J. of Foundations of Computer Science 17, 183–204 (2006)

7. Bernardini, F., Gheorghe, M., Krasnogor, N., Muniyandi, R.C., Pérez-Jiménez, M.J., Romero-Campero, F.J.: On P systems as a modelling tool for biological systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 114–133. Springer, Berlin (2005)
8. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. Journ. Chem. Phys. 124, 44–109 (2006)
9. Cazzaniga, P., Pescini, D., Besozzi, D., Mauri, G.: Tau leaping stochastic simulation method in P systems. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 298–313. Springer, Berlin (2007)
10. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Modelling metapopulations with stochastic membrane systems. BioSystems (To appear)
11. Bianco, L., Fontana, F., Franco, G., Manca, V.: P systems for biological dynamics. In: [3], pp. 81–126

Some Aspects of a Complexity Theory for Continuous Time Systems

Marco Gori[1] and Klaus Meer[2,⋆]

[1] Dipartimento di Ingegneria dell'Informazione
Università di Siena, Via Roma 56, I-53100 Siena, Italy
[2] Syddansk Universitet Odense, Dept. of Mathematics and Computer Science
Campusvej 55, DK-5230 Odense M, Denmark

Abstract. In this paper we survey previous work by the authors defining a complexity measure for certain continuous time systems. Starting point are energy functions of a particular structure. Global minimizers of such energies correspond to solutions of a given problem, for example an equilibrium point of an ordinary differential equation. The structure of such energies is used to define complexity classes for continuous problems and to obtain completeness results for those classes. We discuss as well algorithmic aspects of minimizing energy functions.

1 Introduction

The use of analog systems as computational models has attracted increasing interest in recent years. One way to formalize computation in this framework is to consider a differential equation and follow a trajectory until a solution, e.g., an equilibrium point, is reached. There are many interesting and open problems related to such an approach, ranging from the question of setting up a complexity theoretic framework for such dynamical systems (including notions of complexity classes, reducibility, completeness etc.) to concrete solution algorithms. For an excellent up to date survey on related questions see [3] and the literature cited in there. An older yet very readable survey is [6].

In this paper we discuss a general framework for measuring the complexity of analog systems introduced in [4].

Based on the notion of a *problem* we define complexity classes in dependence of the structural complexity of certain energy functions. Those functions are related to the solutions of a problem instance through their global minimizers. This gives a way to introduce complexity classes which mimic classical P and NP as well as the polynomial hierarchy, and to obtain completeness results.

Both the strength and weakness of this approach may be are its abstractness. On the negative side one might expect a complexity theory for continuous time

⋆ Partially supported by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778 and by the Danish Agency for Science, Technology and Innovation FNU. This publication only reflects the author's views.

systems to be more concrete. However, on the positive side the approach is not based on how to specify a complexity measure for following a trajectory.

Section 2 recalls the definition of a problem and introduces the above mentioned complexity classes in our model. We then discuss the main results concerning completeness for the introduced classes. In Section 3 we outline how the approach can be made more concrete by adding as well a measure for following trajectories. As examples we consider linear system solving and the perceptron learning algorithm.

Proof details can be found in [4].

2 The General Framework

A problem in our setting is defined as a binary relation over the space $\mathbb{R}^\infty := \bigoplus_{i\geq 1} \mathbb{R}^i$ of finite sequences of real numbers.

Definition 1. *A* PROBLEM Π *is a relation in* $\mathbb{R}^\infty \times \mathbb{R}^\infty$. SOLVING A PROBLEM *means that on input* $d \in \mathbb{R}^n$ *for some* $n \in \mathbb{N}$, *a vector* $y \in \mathbb{R}^k$ *for some* $k \in \mathbb{N}$ *is computed such that* $(d, y) \in \Pi$. *Usually we require the output dimension* k *to be polynomially related to* n, *i.e. there exists a polynomial* p *such that* $k = p(n)$ *for all* $n \in \mathbb{N}$.

Remark 1. We shall frequently use the notation $\Pi(d)$ to denote a solution y such that $(d, y) \in \Pi$, even though Π may not be a function.

Example 1. The following examples are typical for our framework:

a) The problem of solving linear equations is given by

$$\Pi := \{(A, b, y) | A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, y \in \mathbb{R}^n \text{ such that } A \cdot y = b\}.$$

In terms of Definition 1 we thus have $d := (A, b)$.

b) The problem of finding a separating hyperplane for two classes $X^+ \subset \mathbb{R}^n$, $X^- \subseteq \mathbb{R}^n$ of patterns is given by

$$\Pi := \{(X^+, X^-, w, \delta) | w^T x \geq \delta \ \forall\ x \in X^+, w^T x \leq -\delta\ \forall x \in X^-\}.$$

Again, in terms of Definition 1 it is $d := (X^+, X^-)$ and $y := (w, \delta)$.

c) Consider a dynamical system $\frac{dx}{dt} = F(x(t))$ for which an equilibrium point is searched. This can be formalized in different ways. One appropriate possibility is to consider $d := F$ as the first component of a problem Π and to look for a y which is an equilibrium point of F. Thus

$$\Pi = \{(F, y) \in \mathbb{R}^\infty \times \mathbb{R}^\infty \mid y \text{ is an equilibrium of } \frac{dx}{dt} = F(x(t))\}\ .$$

The above definition requires F to be representable in a certain way as a point in $\mathbb{R}^\infty$. This is, for example, the case if F is given as a rational function.

We are interested in characterizing the complexity of a problem through the structure of certain energy functions in the following sense. Such energies are associated to the problem in a uniform way by considering a family $\{E_n\}_{n\in\mathbb{N}}$ of functions for each dimension n. Every E_n is a function depending on two blocks d and w of variables. The block $d \in \mathbb{R}^n$ is taken to represent an input instance of a problem Π. The block $w \in \mathbb{R}^m$ will be related to a solution y of Π for input d. Once more, the dimension m should be polynomially related to n, that is $m = q(n)$ for a polynomial q. This polynomial is given together with the family $\{E_n\}$.

Moreover, for each n and fixed $d \in \mathbb{R}^n$ the function $w \to E_n(d, w)$ is supposed to have a global minimum. Such a minimum w^* can be used by an additional (computationally easy to perform) algorithm to yield a solution of the particular instance d for our problem.

Another main point of the definition is the way how these energies can be computed. This will be crucial for defining a complexity measure later on. We use straight-line programs for this purpose.

Definition 2. *(Straight-line programs) For an operation set $\mathcal{O}$ a* STRAIGHT-LINE PROGRAM *of input dimension T over $\mathcal{O}$ is a sequence $\beta_1, \ldots, \beta_\ell$ of operations defined as follows. Every β_i is either of the form $\beta_i := c$ for a constant $c \in \mathbb{R}$ or of the form $\beta_i := \beta_j \circ \beta_k$, where $\circ \in \mathcal{O}$ and $j, k \in \{-T+1, \ldots, 0, 1, \ldots i-1\}$. For any T-dimensional real input $x_1, \ldots, x_T$, to the first T registers $\beta_{-T+1}, \ldots, \beta_0$ we assign the values $\beta_{-T+1} := x_1, \beta_{-T+2} := x_2, \ldots, \beta_0 := x_T$. A computation of the program then proceeds in the obvious manner assigning the corresponding values to the $\beta_i, i \geq 1$. The result is supposed to be computed in β_ℓ. The* SIZE *or* COMPUTATION TIME *of the program is the number ℓ of operations performed.*

In our framework, input variables for an SLP are chosen as $d_1, \ldots, d_n$ and $w_1, \ldots, w_{q(n)}$. Thus, $T = n + q(n)$. We are interested in computing a real valued function $E_n : \mathbb{R}^{n+q(n)} \to \mathbb{R}$. We want to combine SLPs in a uniform way in order to relate them to functions from $\mathbb{R}^\infty \to \mathbb{R}^\infty$.

Definition 3. *A family $\mathcal{E} := \{E_n\}_{n\in\mathbb{N}}$ of SLPs, every E_n of input dimension $n + q(n) \in \mathbb{N}$ for a fixed polynomial q, is called to be* UNIFORMLY POLYNOMIALLY BOUNDED *if there exists an algorithm which on input $n \in \mathbb{N}$, computes a description of E_n and runs in polynomial time with respect to n. We call such a family an SLP energy family.*

In this paper we restrict the operation set $\mathcal{O}$ to be $\{+, -, *\}$, but more general sets are thinkable, see [4]. Thus, our energies basically are multivariate polynomials and can be treated in the framework of the BSS model of computation, see [2]. We suppose the reader to be familiar with this model.

2.1 Complexity Classes

For defining the complexity of a problem we now look for the structure of related families of energy functions. At this level the complexity of a problem will be independent of the question how to find a global minimum of an energy. The latter problem is addressed in the next section.

Next, we introduce certain complexity classes denoted by $\mathbf{U}, \mathbf{NU}$, and $\mathbf{PU}$ that are relevant in our framework.

Definition 4. *Let Π be a problem.*

a) Π belongs to the class $\mathbf{U}$ if there exists an SLP energy family $\{E_n\}_n$ together with another family $\{N_n\}$ of SLPs which is uniformly given by a polynomial time BSS machine such that the following is true:
  i) There is a fixed polynomial q such that every E_n is a map $E_n : \mathbb{R}^n \times \mathbb{R}^{q(n)} \to \mathbb{R}$;
  ii) For any fixed $d \in \mathbb{R}^n$ the function $w \to E_n(d, w)$ is unimodal;
  iii) If w^* is a global minimizer of $w \to E_n(d, w)$ for given d, then we can compute a solution $\Pi(d)$ using the SLP $N_{q(n)}$, i.e. $(d, N_{q(n)}(w^*)) \in \Pi$.

b) Problem Π belongs to class $\mathbf{NU}$ if items i) and iii) above hold, but $w \to E_n(d, w)$ has not to be unimodal. Clearly, it is $\mathbf{U} \subseteq \mathbf{NU}$.

c) Π belongs to the continuous-time polynomial hierarchy $\mathbf{PU}$ if the following holds: there exist an SLP energy family $\{E_n\}_n$ and a function $N : \mathbb{R}^\infty \to \mathbb{R}^\infty$ computable by a uniform family of SLPs in polynomial time such that:
  i) There is a fixed polynomial q such that every E_n is a map $E_n : \mathbb{R}^n \times \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \ldots \times \mathbb{R}^{n_k} \to \mathbb{R}$, where $\sum_{i=1}^{k} n_i = q(n)$.
  ii) If for given d the point w_1^* is a solution for the choice of variables w_1 in the optimization problem

$$\min_{w_1} \max_{w_2} \ldots \min_{w_k} \; E_n(d, w_1, \ldots, w_k),$$

then we can compute a solution of Π for input d as $(d, N(w_1^*))$. The same should hold w.r.t. every $\widehat{w}_1$ such that $\max_{w_2} \ldots \min_{w_k} E_n(d, \widehat{w}_1, \ldots, w_k)$ does not exist. Above, the last optimization operation is min *if k is odd and* max *if k is even. The problem belongs as well to $\mathbf{PU}$ if the optimization starts with* max.

The classes $\mathbf{U}$ and $\mathbf{NU}$ can be seen as a natural counterparts of $\mathbf{P}$ and $\mathbf{NP}$ in our framework. We thus conjecture the obvious inclusion $\mathbf{U} \subset \mathbf{NU}$ to be proper.

In order to speak about complete problems finally the following definition is needed.

Definition 5

a) Let Π_1 and Π_2 be two problems. We say that Π_1 is SLP-REDUCIBLE IN POLYNOMIAL TIME *to Π_2 if there exist two functions ϕ and ϕ^* from $\mathbb{R}^\infty \to \mathbb{R}^\infty$, both computable in polynomial time by a uniform SLP in the BSS model of computation, such that*

$$\forall d \in \mathbb{R}^\infty \quad \Pi_1(d) = \phi^* \left(\Pi_2 \left(\phi(d) \right) \right)$$

Note that since $\Pi_2(\phi(d))$ might not be unique (cf. Remark 1) we require Φ^* to compute a solution of $\Pi_1(d)$ for any possible value of $\Pi_2(\phi(d))$.

*b) A problem $\Pi \in \mathbf{NU}$ is $\mathbf{NU}$-*COMPLETE *if every other problem in $\mathbf{NU}$ is SLP-reducible in polynomial time to Π. Similarly for $\mathbf{PU}$-*COMPLETENESS.

2.2 Completeness Results

The following results show the existence of complete problems for **NU** and **PU**. For proofs we refer to [4].

Theorem 1

a) *There exist* **NU***-complete problems with respect to the operation-set* $\mathcal{O} := \{+, -, *\}$ *and SLP-reducibility.*

b) *The following quadratic optimization problem is* **NU***-hard with respect to the operation set given in a): Given a linear objective function* $f : \mathbb{R}^n \to \mathbb{R}$ *together with finitely many constraints* $h_1(x) = 0, \ldots, h_m(x) = 0, x \in \mathbb{R}^n$*, where the* h_i *are polynomials of degree at most* 2*, find a solution point of* $\min\{f(x)|h_i(x) = 0, 1 \leq i \leq m\}$*. Thus,* **QP** *is the problem defined by tuples* $(f, h_1, \ldots, h_m, x)$ *such that* x *is a global minimizer of the constrained optimization problem* $\min f(x)$ *subject to* $h_i(x) = 0, 1 \leq i \leq m$*.*

For the polynomial hierarchy it can be shown

Theorem 2. *The following problem is* **PU***-complete under SLP reductions: Given a polynomial* f *of degree* 4 *in* n *blocks* $w, X_2, \ldots, X_n$ *of variables, compute a minimizer of the function*

$$w \to \max_{X_2} \min_{X_3} \ldots \max_{X_n} f(w, X_1, \ldots, X_n)$$

Again, the last optimization operation is min *if* n *is odd and* max *if* n *is even.*

To get a rough idea of how the proof works consider a problem in **NU** with an input d and attached energy $w \to E_n(d, w)$. The decision problem: Given z and d, does z minimize $w \to E_n(d, w)$ is in co-NP$_\mathbb{R}$ over the reals. Using the common reduction arguments from [2] one realizes that by means of a max-min problem such a minimizer can be found (if existing). In general, a related argument establishes the problem in the statement to be **PU**-complete by finding a reduction of a given problem in the hierarchy to the former that increases the number of alternations of max and min by one.

We consider it to be important in our framework to extend the list of complete problems.

3 A Discretization for Trajectory Following: Guiding Examples

The framework analyzed above results in a split of the relevant parts contributing to the complexity of continuous time problems. The first deals with the structure and the SLP complexity of an underlying energy function. As mentioned in the introduction, this is advantageous in that the approach is independent of concrete measures for following trajectories of the given ODE. On the other side, this approach remains abstract when it comes to approximating concrete solutions. Thus, a second part contributes to the complexity of a problem. There is so far no real consensus about what a unifying approach for measuring this second part should be. For a much deeper discussion of different approaches we refer once more to [3].

In this final section we discuss briefly one natural such attempt. It adds a typical measure for the steepest descent algorithm finding minima of energy functions occuring in the above approach. We outline the resulting complexity statements for the first two problems of Example 1. Not surprising, the results obtained that way resemble well known properties for numerical solutions of those problems. Since most of the calculations are quite standard we just outline them.

Let $\{E_n\}_n$ be an SLP energy family as above. W.l.o.g. we assume that 0 is the minimal value for each E_n. For the moment we fix the input $d \in \mathbb{R}^n$ and suppress it and its size n notationally, i.e., instead of $E_n(d, w)$ we write $E(w)$. Choosing a start value w_0 we consider (see [8] for more on terminal dynamics) the terminal attractor equation

$$\frac{d}{dt} w(t) \;=\; -\frac{E_0}{\sigma} \cdot \frac{D_w E(w(t))}{\|D_w E(w(t))\|^2} \ , \tag{1}$$

where $E_0 := E(w_0)$ and $\sigma > 0$ is fixed, together with its Euler-discretization

$$w_{k+1} \;=\; w_k - \tau \cdot \frac{E_0}{\sigma} \cdot \frac{D_w E(w_k)}{\|D_w E(w_k)\|^2} \tag{2}$$

with step size $\tau > 0$. It is easy to see that the exact solution $\tilde{w}(t)$ of (1) for $t < \sigma$ satisfies

$$E(\tilde{w}(t)) = E_0 \cdot (1 - \frac{t}{\sigma}), t \in [0, \sigma).$$

Thus, the terminal attractor approaches a point with energy value 0 in finite time $t = \sigma$. The following theorem addresses the complexity of the discretization procedure (2).

We shall first study the number of discretization steps necessary in order to achieve a point w^* such that the energy value satisfies $E(w^*) \le \epsilon$ for a given precision $\epsilon > 0$. The theorem below is proven by a straightforward calculation using Taylor's formula.

Theorem 3. *Let E be an energy function as above and let E_0 denote the energy value for a starting point w_0 (see below). For a pair $(\epsilon_1, \epsilon_2), \epsilon_1 > \epsilon_2 > 0$ consider the differential equation*

$$\frac{d}{dt} w(t) \;=\; -\frac{E_0}{\sigma} \cdot \frac{D_w E(w(t))}{\|D_w E(w(t))\|^2} \ ,$$

and its Euler-discretization

$$w_{k+1} \;=\; w_k - \tau \cdot \frac{E_0}{\sigma} \cdot \frac{D_w E(w_k)}{\|D_w E(w_k)\|^2}$$

Suppose that there exist bounds $L(\epsilon_1, \epsilon_2)$ and $H(\epsilon_1, \epsilon_2)$ on the set $\Omega(\epsilon_1, \epsilon_2) := \{\tilde{w} \in \Omega | \epsilon_1 \ge E(\tilde{w}) \ge \epsilon_2\}$ such that:

(i) $\|D_w E(\tilde{w})\| \geq L(\epsilon_1, \epsilon_2) > 0$ *for all* $\tilde{w} \in \Omega(\epsilon_1, \epsilon_2)$ *as well as*
(ii) $\|D_w^2 E(\tilde{w})\| \leq H(\epsilon_1, \epsilon_2)$ *for all* $\tilde{w} \in \Omega(\epsilon_1, \epsilon_2)$*, where the norm is the operator norm corresponding to the Euclidean vector norm.*

Then starting from a point w_0 *in* $\Omega(\epsilon_1, \epsilon_2)$ *a point* w^* *such that* $E(w^*) \leq \epsilon_2$ *can be reached in*

$$k(\epsilon_1, \epsilon_2) := O\left(\frac{\epsilon_1^2 \cdot H(\epsilon_1, \epsilon_2)}{\epsilon_2 \cdot L(\epsilon_1, \epsilon_2)^2}\right)$$

many discretization steps of step size

$$\tau(\epsilon_1, \epsilon_2) = O\left(\frac{\epsilon_2 \cdot L(\epsilon_1, \epsilon_2)^2 \cdot \sigma}{\epsilon_1^2 \cdot H(\epsilon_1, \epsilon_2)}\right).$$

After having reduced the energy value below ϵ_2 we let ϵ_2 play the role of ϵ_1 and choose a new target value for the energy. This results in considering a sequence $\{\epsilon_k\}_{k \in \mathbb{N}}$ which is used in the steepest descent algorithm until we have reduced the energy below a given accuracy ϵ. The number of iteration steps then is given by

$$\sum_{k=1}^{K^*} \frac{1}{2} \cdot \frac{H(\epsilon_{k-1}, \epsilon_k) \cdot \epsilon_{k-1}^2}{L(\epsilon_{k-1}, \epsilon_k)^2 \cdot \epsilon_k}, \tag{3}$$

where $\epsilon_0 := E_0$ is the first energy value we start with. The goal now is to determine how this sum depends on the required accuracy ϵ and to choose reasonable sequences $\{\epsilon_k\}$ to keep it as small as possible.

A typical choice is given by $\epsilon_k := \frac{E_0}{2^k}$. Then K^* in the above formula has to be taken such that $\frac{E_0}{2^{K^*}} \leq \epsilon$ for a given precision $\epsilon > 0$, i.e. $K^* := \lceil \log(\frac{E_0}{\epsilon}) \rceil$.

Remark 2.
(a) It should be clear from the above arguments that the ratio $\kappa(E, \epsilon_{k-1}, \epsilon_k) := \frac{H(\epsilon_{k-1}, \epsilon_k)}{L(\epsilon_{k-1}, \epsilon_k)^2}$ can be interpreted as a condition number for the problem of minimizing the energy on the set $\Omega(\epsilon_{k-1}, \epsilon_k)$. The limit behaviour of $\kappa(E, \epsilon_{k-1}, \epsilon_k)$ for $k \to \infty$ for an optimal sequence $\{\epsilon_k\}$ is a measure of the conditioning of minimizing the energy.
(b) The above analysis can be carried out for other numerical procedures as well (e.g., for higher degree discretizations). Since we want to focus on outlining the general framework we restrict ourselves to the Euler-method.

Clearly, with respect to applying a concrete numerical algorithm like the Euler method additional requirements related to the energies are necessary. One such is that an approximation of a global minimum still can be used to obtain (for example through the SLP family $\{N_n\}$ of Definition 4) a suitable approximation of a solution of the given problem. Here follow two classical problems that can be treated completely that way.

3.1 Example: Linear Systems

Let us consider a square linear system $A \cdot x = b$ with regular matrix $A \in \mathbb{R}^{n \times n}$. Define an energy

$$E(A, b, w) \; = \frac{1}{\|b\|^2} \cdot \|A \cdot w - b\|^2 \; ,$$

where $\| \bullet \|$ denotes the Euclidean norm. We have

$$D_w E(\tilde{w}) = 2 \cdot \frac{A^T \cdot (A \cdot \tilde{w} - b)}{\|b\|^2} \;\text{ and }\; D_w^2 E(\tilde{w}) = 2 \cdot \frac{A^T \cdot A}{\|b\|^2} \; .$$

A is regular, so E is unimodal and the only critical point w^* of E is the solution. It clearly satisfies $E(w^*) = 0$. Moreover,

$$\begin{aligned} \|D_w E(A, b, \tilde{w})\| &= 2 \cdot \frac{\|(A^T)^{-1}\| \cdot \|A^T \cdot (A \cdot \tilde{w} - b)\|}{\|(A^T)^{-1}\| \cdot \|b\|^2} \\ &\geq \frac{2 \cdot \|A \cdot \tilde{w} - b\|}{\|(A^T)^{-1}\| \cdot \|b\|^2} \\ &= \frac{2 \cdot \sqrt{E(A, b, \tilde{w})}}{\|A^{-1}\| \cdot \|b\|} \end{aligned}$$

Thus, in the terminology of Theorem 3 we get for $\epsilon_1 > \epsilon_2 > 0$ the bounds

$$H(\epsilon_1, \epsilon_2) \leq \frac{2 \cdot \|A^T\| \cdot \|A\|}{\|b\|^2} \;\text{ and }\; L(\epsilon_1, \epsilon_2) \geq \frac{2 \cdot \sqrt{\epsilon_2}}{\|A^{-1}\| \cdot \|b\|}.$$

For an application of Theorem 3 we choose the sequence $\epsilon_k := \frac{1}{2^k}, \epsilon_0 := 1 = E(0) =: E_0$, a step size $\tau(\epsilon_{k-1}, \epsilon_k) := \frac{4\sigma \cdot \epsilon_k^2}{\epsilon_{k-1}^2 \cdot \|A\|^2 \cdot \|A^{-1}\|^2}$ and get as a bound on the number of steps

$$\begin{aligned} &\frac{1}{2} \cdot \sum_{k=1}^{K^*} \frac{2 \cdot \|A^T\| \cdot \|A\|}{\|b\|^2} \cdot \left(\frac{1}{2^{k-1}} \right)^2 \cdot \frac{(2^k)^2}{4} \cdot \|A^{-1}\|^2 \cdot \|b\|^2 \\ &= \sum_{k=1}^{K^*} \|A\|^2 \cdot \|A^{-1}\|^2 \\ &= \|A\|^2 \cdot \|A^{-1}\|^2 \cdot \lceil \log(\tfrac{1}{\epsilon}) \rceil \end{aligned}$$

since $K^* := \lceil \log(\frac{1}{\epsilon}) \rceil$ iterations are sufficient to reduce the energy to a value $\leq \epsilon$.

Several remarks are in charge. The quantity $\|A\| \cdot \|A^{-1}\|$ of course is well known as the condition number of a square matrix. So it is no surprise that in comes into our analysis. The complexity of the algorithm to minimize the energy also depends (besides on the number of iterations) on the complexity of evaluating $D_w E$. Thus, the approach taken in the previous section is important as well here. The latter evaluation complexity is bounded by the complexity of performing two matrix-vector multiplications, which is $O(n^2)$. Thus, for well-conditioned families of matrices, i.e, if the condition number can be bounded by a (known) constant, we get a number $O(n^2 \cdot \log \frac{1}{\epsilon})$ of arithmetic operations. Note that in this case the step sizes can be easily computed as well.

A related example can be found in [1], where the ranking problem for webpages is considered.

3.2 Example: Separating Hyperplane

In this subsection we want to show how our general framework gives back qualitatively the results obtained by the well known Perceptron learning algorithm, see [5]. Relations between the perceptron algorithm and steepest descent methods have been studied previously, see, for example, [7].

Given two finite sets X^+, X^- of points in $\mathbb{R}^n$ and a $\delta > 0$, the task is to find a $w \in \mathbb{R}^n$ such that

$$w^T \cdot x \geq \delta \;\; \forall\, x \in X^+ \text{ and } \; w^T \cdot x \leq -\delta \;\; \forall\, x \in X^- \; . \tag{4}$$

An energy for this problem can be defined as:

$$E(X^+, X^-, w) := \sum_{x \in X^+} \beta(w^T \cdot x - 2 \cdot \delta) + \sum_{x \in X^-} \beta(-w^T \cdot x - 2 \cdot \delta) \; ,$$

where

$$\beta(t) := \begin{cases} t^4 & t \leq 0 \\ 0 & t > 0 \end{cases}$$

The idea behind using this energy is as follows: First, it is not hard to see that E is twice differentiable and unimodal. If w is a hyperplane doing a correct separation, then for $x \in X^+$ we obtain $w^T \cdot x \geq \delta > 0$; similarly for $x \in X^-$. The energy is not necessarily vanishing in such a separating hyperplane w; however, E is vanishing in $2 \cdot w$. Note that β is penalizing those hyperplanes that do not separate the test sets sufficiently good, even though such a hyperplane might solve the initial problem. Note as well that any w satisfying $E(w) < \delta^4$ *is* a separating hyperplane, even though it might not be a global minimizer of E.

We compute upper and lower bounds according to Theorem 3.

i) Let $\epsilon_1 > \epsilon_2 > 0$. We compute an upper bound for $\|D_w^2 E(X,w)\|$ on the set $\Omega(\epsilon_1, \epsilon_2)$. Instead of the operator norm $\|D_w^2 E\|_2$ induced by the Euclidean vector norm we use the well-known estimation

$$\|D_w^2 E\|_2 \leq \sqrt{\|D_w^2 E\|_1 \cdot \|D_w^2 E\|_\infty} \; ,$$

where $\| \bullet \|_\infty$ denotes the maximal sum of absolute values of row entries and $\| \bullet \|_1$ does the same for the column sums.

Suppose that for $X := X^+ \cup X^-$ it is $|X| =: m$, i.e. there are m many test points; let $B_X > 0$ denote a bound such that $\|x\|_\infty \leq B_X$ for all $x \in X$. For $i, j \in \{1, \ldots, n\}$ we get

$$\frac{\partial^2 E}{\partial w_i \partial w_j} = \sum_{\substack{x \in X^+ \\ w^T \cdot x < 2 \cdot \delta}} 12 \cdot (w^T \cdot x - 2 \cdot \delta)^2 \cdot x_i \cdot x_j + \\ + \sum_{\substack{x \in X^- \\ w^T \cdot x > -2 \cdot \delta}} 12 \cdot (w^T \cdot x + 2 \cdot \delta)^2 \cdot x_i \cdot x_j .$$

Using norm equivalence in $\mathbb{R}^n$: $\|z\|_2 \leq \sqrt{n} \cdot \|z\|_4 \ \forall \ z \in \mathbb{R}^n$ together with the assumption that $w \in \Omega(\epsilon_1, \epsilon_2)$ easy calculations result in

$$\|D_w^2 E(X, w)\|_\infty \ \leq \ 12 \cdot n^2 \cdot \sqrt{\epsilon_1} \cdot B_X^2$$

as well as

$$\|D_w^2 E(X, w)\|_1 \ \leq \ 12 \cdot n^2 \cdot \sqrt{\epsilon_1} \cdot B_X^2$$

and thus the same bound holds for $\|D_w^2 E(X, w)\|_2$.

ii) For obtaining a lower bound for $\|D_w E(X, w)\|$ on $\Omega(\epsilon_1, \epsilon_2)$ consider once more a separating hyperplane $\tilde{w}$ and apply the Cauchy-Schwartz inequality

$$\|D_w E(X, w)\|_2 \ \geq \|\tilde{w}\|_2^{-1} \cdot |\tilde{w}^T \cdot D_w E(X, w)| \ .$$

Now

$$|\tilde{w}^T \cdot D_w E(X, w)| \geq 4\delta \cdot \sum_{\substack{x \in X^+ \\ w^T \cdot x < 2\delta}} |w^T \cdot x - 2\delta|^3 + \sum_{\substack{x \in X^- \\ w^T \cdot x > -2\delta}} |w^T \cdot x + 2\delta|^3 \geq 4 \cdot \delta \cdot E(w)^{\frac{3}{4}} .$$

using the norm inequality $\|z\|_3 \geq \|z\|_4$ for any $z \in \mathbb{R}^n$, where $\|z\|_p := \left(\sum_{i=1}^{n} z_i^p \right)^{\frac{1}{p}}$ for $p \in \mathbb{N}$.

Altogether, we obtain as lower bound on $\Omega(\epsilon_1, \epsilon_2)$:

$$\|D_w E(w)\|_2 \ \geq \ 4 \cdot \|\tilde{w}\|_2^{-1} \cdot \delta \cdot \epsilon_2^{\frac{3}{4}} .$$

iii) With these bounds we can compute the number of steps necessary to get an ϵ-approximate solution. We apply Theorem 3 with the following quantities: $\epsilon_0 := E(X, 0) = 16 \cdot m \cdot \delta^4; \epsilon_k := \frac{E(\epsilon_0)}{2^k}, \epsilon > 0$ fixed and

$$H(\epsilon_{k-1}, \epsilon_k) \leq 12 \cdot n \cdot B_X^2 \cdot \sqrt{\epsilon_{k-1}} \ , \ L(\epsilon_{k-1}, \epsilon_k) \geq 4 \cdot \delta \cdot \|\tilde{w}\|_2^{-1} \cdot \epsilon_k^{\frac{3}{4}}$$

for any separating hyperplane $\tilde{w}$; the latter should be taken in the analysis so to minimize the norm (note that $\tilde{w}$ is not used in the algorithm).

Thus, the number of steps to reduce the energy from ϵ_{k-1} to ϵ_k is bounded by

$$\frac{1}{2} \cdot \frac{H(\epsilon_{k-1}, \epsilon_k) \cdot \epsilon_{k-1}^2}{L(\epsilon_{k-1}, \epsilon_k)^2 \cdot \epsilon_k} \le \frac{3}{\sqrt{2}} \cdot \frac{n^2 \cdot B_X^2 \cdot \|\tilde{w}\|_2^2}{\delta^2}.$$

With $K^* := O(\log \frac{E_0}{\epsilon})$ many iterations we thus need

$$O\left(\frac{n^2 \cdot B_X^2 \cdot \|\tilde{w}\|_2^2}{\delta^2} \cdot \log \frac{E_0}{\epsilon}\right)$$

many steps. Finally, recalling that $\epsilon < \delta^4$ is a sufficient choice for obtaining a separating hyperplane we end up with

$$O\left(\frac{n^2 \cdot B_X^2 \cdot \|\tilde{w}\|_2^2}{\delta^2} \cdot \log m\right)$$

many steps for the Euler-discretization. For the computation of the gradient in each step an upper bound of order $O(n \cdot m)$ is obvious.

These bounds pretty well correspond to the bounds known from the perceptron convergence theorem, see [5]. The important difference is that our algorithm results as just one specific example from a much more general framework, that hopefully can be applied to a larger class of problems as well.

4 Conclusion and Acknowledgement

We have studied a framework for measuring the complexity of analog systems. The latter is based on the notion of a problem. In its more abstract part the approach allows to define complexity classes independently of particular trajectory following algorithms. The existence of complete problems for such classes was established. In a second part we considered steepest descent algorithms for discretized versions of our problems resulting in a concrete running time analysis. Two such examples were discussed.

We still believe that a lot of questions have to be investigated. To get a better overview what has been done so far and which problems are waiting to be solved let us finally refer once again to the survey by Bournez and Campognolo [3].

We would like to thank the anonymous referees for some helpful remarks.

References

1. Bianchini, M., Gori, M., Scarselli, F.: Inside PageRank. ACM Transactions on Internet Technology 5(1), 92–128 (2005)
2. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, Heidelberg (1998)
3. Bournez, O., Campagnolo, M.L.: A Survey on Continuous Time Computations. Preprint (2006)
4. Gori, M., Meer, K.: A step towards a complexity theory for analog systems. Mathematical Logic Quarterly 48(1), 45–58 (2002)

5. Minsky, M., Papert, S.: Perceptrons. The MIT Press, Cambridge (1969)
6. Orponen, P.: A survey of continuous-time computation theory. In: Du, D.-Z., Ko, K.-I. (eds.) Advances in Algorithms, Languages, and Complexity, pp. 209–224. Kluwer Academic Publishers, Dordrecht (1997)
7. Shynk, J.J.: Performance Surfaces of a Single-Layer Perceptron. IEEE Transactions on Neural Networks 1(3), 268–274 (1990)
8. Zak, M.: Introduction to terminal dynamics. Complex Systems 7, 59–87 (1993)

Enumerations and Torsion Free Abelian Groups*

Alexander G. Melnikov

Sobolev Institute of Mathematics, Novosibirsk, Russia
vokinlem@bk.ru

Abstract. We study possible spectrums of torsion free Abelian groups. We code families of finite sets into group and set up the correspondence between their algorithmic complexities.

1 Introduction

Studying model theory and theory of algorithms gives us another branch of science - computable model theory. We say that the model is computable if it's main set, predicates and functions are recursive, and all functions and predicates are effectively enumerated. We may think these models as "the only ones that can be applied in computer science and that can be presented on some computer" or "the ones we can exactly imagine" etc. Note that we don't think about time or space complexity of algorithms - this is another subject for studying.

Starting with abstract computable models, we try to apply some results or their variations to effective algebra. In particular, computable fields, Boolean algebras and groups are widely studying.

We generalize the notion of computable model replacing in it's definition all words "recursive" by "X-recursive", where X is some countable set. That means that we can ask someone on some steps of given program whether $x \in X$ or not, for arbitrary x. It's not easy to imagine, how can we apply it in computer science. We can think that this "oracle" is some physical experiment - but is there any physicist who knows everything about halting problem?.. That means that we need some methods to answer the question:

Question. *Let $\mathcal{A}$ be a model, and suppose that $\mathcal{A}$ has copies, computable in a fixed family of Turing degrees respectively. Does it necessarily follow that $\mathcal{A}$ has a computable copy?*

If $\mathcal{A}$ is a Boolean algebra, and it has a low copy, then the answer is "yes"[1]. There is another related question:

Question. *Given a structure $\mathcal{A}$ what can we say about $\{deg(\hat{\mathcal{A}}) : \hat{\mathcal{A}} \simeq \mathcal{A}\}$?*

For an arbitrary structure, this family of degrees (called *degree spectrum*) can be enough complicated and reach, but does not contain **0**-degree or low degrees. Wehner [7] built a graph that has presentations exactly in non-recursive degrees

* Partially supported by President grant of Scientific School NSh-4413.2006.1.

(see also [4] for alternative proof). Miller R. [3] built a linear order, that has all noncomputable Δ_2^0 - copies, but does not have a computable one. These results give us examples of algorithmic "anomaly" and shows the variety and richness of pure and applied computable algebra.

We study possible spectrums of torsion free Abelian groups. For torsion free Abelian group the key notion is its rank. In this paper we study algorithmic properties of groups in the case of infinite rank, and we obtain the following:

Theorem. *For any family R of finite sets there exists a torsion free Abelian group G_R of infinite rank, such that G_R has X-computable copy iff R has Σ_2^X-computable enumeration.*

Interpretation of a family, that can be obtained by relativization of Wehner's result, gives us the corollary:

Theorem. *There exists a torsion free Abelian group G of infinite rank, such that G has X-computable copy iff $X' >_T 0'$, i.e. has exactly nonlow copies.*

2 Basic Notions

We need some basic notions and facts from computability theory, theory of groups and computable model theory. For better background see also [5], [6] and [2]. We suppose that the reader knows the elementary properties of recursive functions and recursively enumerable sets.

Definition 1. *A set A is recursive with respect to a set B ($A \leq_T B$), if its characteristic function is B-recursive. That means that it can be computed by Turing machine with "oracle" B. If $A \leq_T B$ and $B \leq_T A$ then $A \equiv_T B$. It's obvious that $\equiv_T$ is the relation of equivalence. The equivalence classes for $\equiv_T$ are called degrees.*

Definition 2. *Let $K = \{x \colon \Phi_x^A(x)\downarrow\} = \{x \colon x \in W_x^A\}$. This set is denoted by A' and called the jump of a set A.*

Index A in $\Phi_x^A(x)$ means that Φ is (partially) recursive with respect to a set A. It's clear how to define the n-th jump of A ($A^{(n)}$) using the same construction for $A^{(n-1)}$. Jump is well-defined on degrees, and iteration of jumps induces hierarchy, that is called arithmetical:

$$X \in \Sigma_n^Y \leftrightarrow \text{X is r.e. in } Y^{(n-1)}.$$

Definition 3. *We say that a set $A \leq_T \emptyset'$ is low if $A' \equiv_T \emptyset'$, and it is n-low if $A^{(n)} \equiv_T \emptyset^{(n)}$.*

Definition 4. *Let $R = \{R_i | i \in \omega\}$ and $\nu : \omega \to^{on} R$. The set $S_\nu \leftrightharpoons \{\langle n, i\rangle | n \in \nu(i)\}$ is called enumeration of R. We also refer to ν as an enumeration of R, using a simple fact, that having S_ν we can recollect the map ν and vice versa.*

We will follow the tradition of enumeration theory in defining computable enumeration:

Enumeration ν is called Σ_n^X-computable if $S_\nu \in \Sigma_n^X$ (and X-computable if $S_\nu \in \Sigma_1^X$).

Now let G be a countable group.

Definition 5. *A group $\langle G, \cdot\rangle$ is called computable group if $\mid G \mid \subseteq N$ is a recursive set and the operation $\cdot$ is presented by some recursive function.*

We define A-computable groups by substitution of the word "recursive" by "A-recursive" in the definition above.

Definition 6. *Let $\langle G, +, 0\rangle$ be a torsion free Abelian group (i.e for all $a \neq 0$, $0 \neq n \cdot a \leftrightharpoons \underbrace{a + a + ... + a}_{n}$). The elements $g_0, ..., g_n \in G$ are linearly independent if, for all $c_0, ..., c_n \in Z$, the equality $c_0g_0 + c_1g_1 + ... + c_ng_n = 0$ implies that $c_i = 0$ for all i. An infinite set is linearly independent if every finite subset is linearly independent. A maximal linearly independent set is called a basis, and the cardinality of any basis is called the rank of G.*

As for vector spaces, it can be proved that the notion of rank is proper, i.e. all maximal linearly independent sets have the same cardinality.

Fix a canonical listing of prime numbers:

$$p_1, p_2, ..., p_n, ...$$

Definition 7. *Let $g \in G$. Then $p^k|g \leftrightharpoons (\exists h \in G)(p^k h = g)$ and*

$$h_p(g) = \begin{cases} max\{k : p^k|g\}, \textit{if this maximum exists,} \\ \infty, \textit{else.} \end{cases}$$

The infinite sequence $\chi(g) = (h_{p_1}(g), ..., h_{p_n}(g), ...)$ is called the characteristic of element g.

Now we are ready to define one of the basic notions in Abelian groups theory.

Definition 8. *Given two characteristics, $(k_1, ..., k_n, ...)$ and $(l_1, ..., l_n, ...)$, we say that they are equivalent, $(k_1, ..., k_n, ...) \simeq (l_1, ..., l_n, ...)$, if $k_n \neq l_n$ only for finite different n, and only if these k_n and l_n are finite. This relation is obviously an equivalence relation, and the corresponding equivalence classes are called types.*

It can be easily proved that linear dependant elements has the same type. That means that we can give a proper definition of *type of group* in the case of *rank 1.* The following theorem is the key result for torsion free Abelian groups of rank 1:

Theorem 1 (Baer, see [6]). *Let G and H be torsion free Abelian groups of rank 1. Then G is isomorphic to H iff they have the same type.*

Proof (sketch).
We can choose any nonzero $g \in G$ and $h \in H$, and it will be necessarily $\chi(g) \simeq \chi(h)$. Then we extract the finite number of roots receiving $g' \in G$ and $h' \in H$ with identical characteristics. We define isomorphism $\varphi : G \to H$ starting with $g' \to h'$.

3 Constructing The Group

Theorem 2. *For any family R of finite sets there exists a torsion free Abelian group G_R of infinite rank, such that G_R has X-computable copy iff R has Σ_2^X-computable enumeration.*

Proof.
Notation. *We fix the family of finite sets denoted by R and it's Σ_2^X-computable enumeration ν^X with corresponding $S_\nu^X \leftrightharpoons \{\langle n, i\rangle | n \in \nu^X(i)\} \in \Sigma_2^X$. Without loss of generality, we can assume that $\emptyset \in R$.*

We will use the fact, that every Σ_2^X-relation can be presented as $\{\langle i, k\rangle : (\exists^{<\infty} x)P^X(x, \langle i, k\rangle)\}$, where P^X is some recursive in X relation. We fix T^X such that

$$S_\nu^X = \{\langle i, k\rangle : (\exists^{<\infty} x)T^X(x, \langle i, k\rangle)\}.$$

The **scheme of proof** is the following:

Given a Σ_2^X-computable enumeration ν^X of R we build a r.e. in X presentation $G^X \subseteq Q^\omega$ of group $G_R = \bigoplus_{k\in\omega} \bigoplus_{m\in\omega} G_{k,m}$, where $rank(G_{k,m}) = 1$. Since Q^ω is computable, then G^X must have a computable copy as a r.e. subgroup (see [2]).

Then, to make inverse step, we need to construct some Σ_2^X-computable enumeration if we have some X-computable presentation of G_R.

This is the **idea** of building a group:

1. Fix a computable listing of prime numbers $\{p_n\}_{n\in\omega}$ and canonical enumeration of (nonempty) finite sets $\{D_n\}_{n\in\omega}$.

2. Build a group such that any $\nu^X(k)$ corresponds to ω linearly independent elements $g_{k,m}$, and for all m, n, k:

$$\neg(p_n^\infty | g_{k,m}) \iff (D_n \subseteq \nu^X(k)).$$

3. To build a group, enumerate $T^X(x, \langle i, k\rangle)$ until a *new* x for some pair $\langle i, k\rangle$ appeared. If we have found such x, add p_n-roots to elements $g_{k,m}$ for all n, such that $i \in D_n$.

First we define a procedure **Root(**$\langle i, k\rangle, t^n_{\langle x,i,k\rangle}, Y^n_{\langle x,i,k\rangle}$**)**, that adds prime roots to elements $g_{k,m}$. $Y^n_{\langle x,i,k\rangle}$ is the "memory" of this procedure, and $t^n_{\langle x,i,k\rangle}$ is it's "counter of steps".

Root($\langle i, k\rangle, t^n_{\langle x,i,k\rangle}, Y^n_{\langle x,i,k\rangle}$**)** :
For all $n' \in Y^n_{\langle x,i,k\rangle}$, add $p_{n'}$-root to $g_{k,t^n_{\langle x,i,k\rangle}}$. If $i \in D_{t^n_{\langle x,i,k\rangle}}$, then $Y^{n+1}_{\langle x,i,k\rangle} := Y^n_{\langle x,i,k\rangle} \cup \{t^n_{\langle x,i,k\rangle}\}$ and add $p_{t^n_{\langle x,i,k\rangle}}$-root to $g_{k,m}$, $m \le t^n_{\langle x,i,k\rangle}$. If $i \notin D_{t^n_{\langle x,i,k\rangle}}$, then $Y^{n+1}_{\langle x,i,k\rangle} := Y^n_{\langle x,i,k\rangle}$.

Finally, let $t^{n+1}_{\langle x,i,k\rangle} := t^n_{\langle x,i,k\rangle} + 1$.

End of procedure.

Let **Search(**$\langle i, k\rangle, l$**)** $\leftrightharpoons \mu_x(x \ge l \wedge T^X(x, \langle i, k\rangle))$.

Construction.

Step 0. Fix a computable presentation of Q^ω and numbers for $g_{k,m}$ (we can suppose that $g_{k,m}$ is an element of a form $(\underbrace{0,0,...,0,1}_{p_k^m},0,0,...))$.

For all i,k,x, let $l^0_{\langle i,k\rangle}=0$, $t^0_{\langle x,i,k\rangle}=0$, $Y^0_{\langle x,i,k\rangle}=\emptyset$.

Step s. We denote by G^X_s the part of G^X that has been constructed by the step s. For all $\{(g_1,...,g_n): g_i\in G^X_s, g_i\le s, n\le s\}$, add to G^X_s linear combinations $\{m_1g_1+...+m_ng_n : m_i\le s\}$ (if they were not already added).

Make s steps in computation of $Search(\langle i,k\rangle, l^s_{\langle i,k\rangle})$, $\langle i,k\rangle\le s$.

If $Search^s(\langle i,k\rangle, l^s_{\langle i,k\rangle})\downarrow = x$ for some i,k,x, then $l^{s+1}_{\langle i,k\rangle}:=x+1$, and $\langle x,i,k\rangle$ *gets attention.* Suppose $R^s_{\langle x,i,k\rangle}\leftrightharpoons Root(\langle i,k\rangle, t^s_{\langle x,i,k\rangle}, Y^s_{\langle x,i,k\rangle})$, and

$$\langle x_1,i_1,k_1\rangle,...,\langle x_j,i_j,k_j\rangle$$

be the listing of all triples, that have got attention by this moment. Perform $R^{x_1,s}_{\langle i_1,k_1\rangle}$, then perform the next, ..., and finally $R^{x_j,s}_{\langle i_j,k_j\rangle}$[1].

End of construction.

Lemma 1. *G^X, built by construction, is torsion free Abelian group and has computable in X copy.*

Proof. The first statement is clear: $G\subseteq Q^\omega$ by construction. The second is true because the algorithm of building G^X is effective with oracle X, i.e. G is X-r.e., and $G\subseteq Q^\omega$, that is computable (again see [2]).

Lemma 2. *For any k and m, $\neg(p_n^\infty|g_{k,m})$ in G^X iff $(D_n\subseteq\nu^X(k))$.*

Proof. $\nu^X(k)$ is finite, and $i\in\nu^X(k)$ iff

$$(\exists^{<\infty}x)T^X(x,\langle i,k\rangle),$$

That means that in the procedure **Search** *after some moment* no "new" x for $\langle i,k\rangle$ will appear **for all** $i\in\nu^X(k)$.

We notice that the existence of such step follows from two key properties: R contains only finite sets and $i\in\nu^X(k)$ iff *"there is only finitely many x, such that $T^X(x,\langle i,k\rangle)$."*

We can make a conclusion that after this step, roots that correspond to subsets of $\nu^X(k)$, will not be added by procedures **Root** to elements of a form $g_{m,k}$.

Now let $n\in\{l : D_l\nsubseteq\nu^X(k)\}$. That means that D_n contains $i\notin\nu^X(k)$ and

$$(\exists^\infty x)T^X(x,\langle i,k\rangle),$$

i.e. infinitely many triples of a form $\langle x,i,k\rangle$ will get attantion. Therefore procedures **Root** will add infinetly many p_n-roots to elements $g_{m,k}$. This completes the proof of lemma.

[1] Remember that the procedure $Root(\langle i,k\rangle, s, Y^s_{\langle x,i,k\rangle})$ defines the value of $Y^{s+1}_{\langle x,i,k\rangle}$ and $t^{s+1}_{\langle x,i,k\rangle}$.

Lemma 3. *Let ν^X and ν^Y be enumerations of R. Then two groups G^X and G^Y (built using construction for ν^X and ν^Y respectively) are isomorphic.*

Proof. First fix enumeration ν^X. Notice that ω identic elements $\{g_{k,m}\}_{m\in\omega}$ (in G^X), corresponds to one $\nu^X(k)$, and g_{k_1,m_1} and g_{k_2,m_2} are linearly independent for $\langle k_1, m_1\rangle \neq \langle k_2, m_2\rangle$.

We add roots to $g_{k,m}$ in such a way that G^X is a direct sum:

$$G^X = \bigoplus_{k\in\omega}\bigoplus_{m\in\omega} G^X_{k,m},$$

where $G^X_{k,m}$ corresponds to element $g_{k,m}$ (and therefore codes $\nu^X(k)$, i.e. $G^X_{k,m} \simeq G^X_{k,n}$ for all m, n and fixed k), and $rank(G^X_{k,m}) = 1$.

Now we fix ν^Y, and build G^Y. We receive a group of the similar form

$$G^Y = \bigoplus_{k\in\omega}\bigoplus_{m\in\omega} G^Y_{k,m}.$$

All sets from $R_i \in R$ are finite, therefore $\nu^X(k)$ and $\nu^Y(t)$, coding the same $R_i \in R$ in enumerations, give us elements of the same type: these elements have only finitely many finite roots, and these roots correspond to the same prime numbers. By Baer Theorem we have the isomorphism of groups of rank 1.

The last problem is to show that the direct sum has the same structure. But by construction we always have exactly ω subgroups, coding the same $R_i \in R$, even if there are repetitions in coding of R_i in enumeration.

We showed that both G^X and G^Y are isomorphic to

$$G_R = \bigoplus_{k\in\omega} G_k,$$

where $R_k \in R$ is coded by $G_k = \bigoplus_{m\in\omega} G_{k,m}$, $G_{k,m} \cong G_{k,m'}$

Given Σ^X_2 enumeration of R we can build a r.e. group G^X that has a computable copy. It is a presentation of

$$G_R = \bigoplus_{k\in\omega}\bigoplus_{m\in\omega} G_{k,m}.$$

Now we need an inverse step, i.e. to construct some Σ^X_2-computable enumeration if we have some X-computable presentation of G_R.

Proposition 1. *There exists an algorithm, that for any computable in X presentation G^X of G_R (defined above for R) gives Σ^X_2-enumeration of R.*

Proof.

We define X-p.r.f. r as follows:

$$r(g, n, k) = \begin{cases} 1, \text{ if } G^X \models p_n^k | g, \\ r(g, n, k)\uparrow, \text{ else.} \end{cases}$$

We define also X'-recursive function $\hat{r}$:

$$\hat{r}(g,n,k) = \begin{cases} 1, & \text{if } r(g,n,k) \downarrow = 1, \\ 0, & \text{if } r(g,n,k) \uparrow . \end{cases}$$

Using $\hat{r}$ we can check (with oracle X') the existence of prime roots for any $g \in G^X$. If there is a pair $\langle n,k \rangle$, such that $\hat{r}(g,n,k) = 0$, then g has only finitely many p_n-roots.

We identify elements from G with there codes in G^X.

Construction.

Step 0: Let all m_t^0 be undefined.

Step s:
Substep s,1: For $g \in G^X$, such that $g \leq s$ and m_g^{s-1} is undefined, compute $\hat{r}(g,m,k)$ for $m,k \leq s$. If there exist $g_i, m_i, k_i \leq s$, such that $\hat{r}(g_i,m_i,k_i) = 0 \wedge (\forall n < k_i)(\hat{r}(g_i,m_i,n) = 1)$, then suppose $m_{g_i}^s = m_i$ [2]. For every such g_i add to the enumeration all pairs

$$\{\langle j, g_i \rangle : j \in D_{m_{g_i}^s}\}.$$

Substep s,1: For $g \in G^X$, such that $g \leq s$ and m_g^{s-1} is defined, compute $\hat{r}(g,m,k)$ for $m,k \leq s$. If there exist $g_i, m_i, k_i \leq s$, such that $\hat{r}(g_i,m_i,k_i) = 0 \wedge (\forall n < k_i)(\hat{r}(g_i,m_i,n) = 1 \wedge D_{m_{g_i}^{s-1}} \subset D_{m_i}$, then for every such g_i, add to the enumeration pairs

$$\{\langle j, g_i \rangle : j \in D_{m_i} \setminus D_{m_{g_i}^{s-1}}\},$$

and then suppose $m_{g_i}^s = m_i$.

End of Construction.

Lemma 4. *Described algorithm builds the enumeration of R.*

Proof. Remember that

$$G_R = \bigoplus_{k \in \omega} \bigoplus_{m \in \omega} G_{k,m},$$

where $rank(G_{k,m}) = 1$ and $G_{k,m} \cong G_{k,m'}$ (for any m, m'). For every $g_{k,m} \in G_{k,m}$ we have the following:

$$\neg(p_n^\infty | g_{k,m}) \iff (D_n \subseteq R_k^X).$$

Let $g \in G$. Then $g = r_{k_1,m_1} g_{k_1,m_1} + ... + r_{k_t,m_t} g_{k_t,m_t}$ for some $g_{k_1,m_1} \in G_{k_1,m_1}, ..., g_{k_t,m_t} \in G_{k_t,m_t}$. But G is the direct sum, and g is the linear combinayion of linear independent elements. It is easy to see that

$$(\forall k)((\neg p_k^\infty | g) \iff \bigvee_{j=1,\ldots,t} (D_k \subseteq R_{k_j})),$$

[2] We can suggest (for every i) m_i be the minimal one with this property.

i.e. the characteristic of g is the g.l.b. of characteristics of components. That means that g codes the *union* of all subsets, coded by it's components.

Algorithm let us to move higher and higher along the subsets of some R_{k_j}, until we reach this R_{k_j}. After we reach it, there will be no new pairs of a form $\langle l, g\rangle$ added in enumeration. That means that we enumerate R_{k_j}, and we do it for all elements of R, and only this elements could be enumerated.

Lemma 5. *Enumeration built by algorithm (for G^X) is Σ_2^X.*

Proof. The function $\hat{r}(g, n, k)$ is recursive in X'. That means that the Procedure is effective in X', and enumeration is Σ_2^X.

We set up a correspondence between Σ_2^X-enumerations of R and computable in X presentations of G_R. This completes the proof of theorem.

The following result is one of the possible applications of the previous theorem:

Theorem 3. *There exists a torsion free Abelian group G of infinite rank, such that G has X-computable copy iff $X' >_T 0'$, i.e. has exactly nonlow copies.*

Proof (sketch). First we relativize the result of Wehner [7]. This gives us the family of finite sets that has Σ_2^X $(X' >_T 0')$ enumerations, but has no Σ_2 enumeration. Then we apply construction from previous theorem for this family of sets.

4 Questions

We suggest some related problems.

Question 1. Is it possible to build a torsion free Abelian group with copies exactly in none-recursive degrees? Can we generalize our second theorem for the case of low_n-degrees, for arbitrary n?

At the case of finite rank the answer for the first part is "no" (for the second part it is also natural to get "no"). But in general case the question is open - there is no uniform procedure for coding of any given property into torsion free Abelian groups, especially coding respecting effectiveness.

Question 2. What can we say about Abelian p-groups? Can we build a p-group with any of these properties?

Studying p-groups from these point of view is very interesting and needs some new ideas and methods to be developed.

References

1. Downey, R., Jockusch, G.: Every low Boolean algebra is isomorphic to a recursive one. Proc. Amer. Math. Soc 122, 871–880 (1994)
2. Ershov, Y., Goncharov, S.: Constuctive models. Novosibirsk, Nauchnaya kniga (1999)

3. Miller, R.: The Δ_2^0-spectrum of a linear order, preprint.
4. Slaman, T.: Relative to ani non-recursive set. Proc. of the Amer. Math. Soc 126, 2117–2122 (1998)
5. Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, Berlin Heidelberg (1987)
6. Fuchs, L.: Infinite Abelian Groups. vol. II. Academic Press, San Diego (1973)
7. Wehner, S.: Enumerations, countable structures and Turing degrees. Proc. of the Amer. Math. Soc 126, 2131–2139 (1998)

Locally Computable Structures

Russell G. Miller[1,2,⋆]

[1] Department of Mathematics
Queens College – C.U.N.Y.
65-30 Kissena Blvd.
Flushing, New York 11367 U.S.A.
[2] Doctoral Program in Computer Science
The Graduate Center of C.U.N.Y.
365 Fifth Avenue
New York, New York 10016 U.S.A.
Russell.Miller@qc.cuny.edu

Abstract. We introduce the notion of a *locally computable structure*, a natural way of generalizing the notions of computable model theory to uncountable structures $\mathcal{S}$ by presenting the finitely generated substructures of $\mathcal{S}$ effectively. Our discussion emphasizes definitions and examples, but does prove two significant results. First, our notion of m-extensional local computability of $\mathcal{S}$ ensures that the Σ_n-theory of $\mathcal{S}$ will be Σ_n for all $n \leq m+1$. Second, our notion of perfect local computability is equivalent (for countable structures) to the classic definition of computable presentability.

Keywords: computability, computable model theory, extensional, locally computable, perfectly locally computable.

1 Introduction

Turing computability has always been restricted to maps on countable sets. This restriction is inherent in the nature of a Turing machine: a computation is performed in a finite length of time, so that even if the available input was a countable binary sequence, only a finite initial segment of that sequence was actually used in the computation. Thus only that finite segment was relevant to the computation. To be sure, there are approaches that have defined natural notions of computable functions on the real numbers. These include the bitmap model, detailed in [2], and the Blum-Shub-Smale model (see [1]). These are elegant in several respects, but also omit certain basic functions, and moreover, each was built with the real numbers (viewed either as 2^ω or as the real line) specifically in mind, rather than arbitrary uncountable structures.

Nevertheless, mathematicians are hardly daunted by the prospect of doing actual computations on $\mathbb{R}$. When faced with a real number whose binary expansion is not immediately accessible, they do not flinch; they simply call that real

⋆ The author was partially supported by PSC-CUNY grants numbered 67182-00-36, 68470-00-37, and 80209-04-12.

"x." All field operations can then be performed with ease within the subfield of $\mathbb{R}$ generated by x; the mathematician only needs to know whether x is algebraic or transcendental, and, in the former case, what its minimal polynomial over $\mathbb{Q}$ is. Similar devices handle the situation of several unknown reals at once. The binary expansions of these reals are not required for the algebraic operations.

In this paper we formalize this process, and generalize it to arbitrary structures in finite languages. Starting with the notion of a *computable model*, which is entirely in keeping with Turing's notion of computability, we will view the real numbers and other fields as locally computable structures. No claim is made that operations on the reals can be performed globally, but we develop a definition in which countable objects are used to describe all finitely generated substructures of a (potentially uncountable) structure $\mathcal{S}$. Then the *local computability* of $\mathcal{S}$ is determined by the computability of the countable objects. In cases such as the field $\mathbb{R}$, where every finitely generated substructure is computably presentable, we will say that we have a *computable cover* of the structure. Indeed, for $\mathbb{R}$, a single algorithm can list out all elements of this cover.

The term "cover" is borrowed from the definition of a manifold, and the analogy, while imprecise, can be useful for intuitions about our definitions. For instance, for a topological space M, being a manifold does not just require the existence of a cover by open subsets of $\mathbb{R}^n$, but also that the charts within M given by the cover should fit together in a nice way: the transition functions between open subsets of $\mathbb{R}^n$, defined whenever two charts in M intersect, should be continuous (or differentiable, or C^∞, depending on how nicely we wish the manifold to behave). In short, it is not sufficient just to describe the local behavior of M; one must ensure that where the descriptions overlap, they agree with one another in a reasonable way.

For us, it will certainly be true that finitely generated substructures of a structure $\mathcal{S}$ can overlap. Therefore, our description of finitely generated substructures of $\mathcal{S}$ will include an account of which such substructures extend to others. Since any two finitely generated substructures of $\mathcal{S}$ lie within a single larger finitely generated substructure, it is sufficient for our purposes to consider the question of extensions among them. Topological notions do not fit our setting very well, but embeddings among finitely generated computable structures are themselves inherently computable, since they are determined by their values on the generators of the domain. (This is our main reason for considering only finitely generated substructures of $\mathcal{S}$, in fact, rather than all countable substructures.) In order for a structure to be called *locally computable*, we will require not only that the finitely generated substructures be computably presentable in a uniform way, but also that there be a computable enumeration of the embeddings among them corresponding to extensions in the structure $\mathcal{S}$. Various strengthenings of this requirement will allow us to prove stronger theorems about certain of the structures.

The technical content of this paper is not especially high, and for reasons of space we have emphasized our definitions and new concepts, and omitted many of the proofs. When computability-theoretic notions arise, we refer the reader to [4], the standard source, for notation and definitions. A good overview of the field of computable model theory is given in [3].

2 Local Computability

Let T be a $\forall$-axiomatizable theory in a language with n symbols. (If T is finitely axiomatizable but not $\forall$-axiomatizable, we can Skolemize to give it a set of $\forall$-axioms, while keeping the language finite.) We first consider simple covers of a model $\mathcal{S}$ of T. These describe only the finitely generated substructures of $\mathcal{S}$, with no attention paid to any relations between those substructures.

Definition 1. *A* simple cover *of $\mathcal{S}$ is a (finite or countable) collection $\mathfrak{A}$ of finitely generated models $\mathcal{A}_0, \mathcal{A}_1, \ldots$ of T, such that:*

- *every finitely generated substructure of $\mathcal{S}$ is isomorphic to some $\mathcal{A}_i \in \mathfrak{A}$; and*
- *every $\mathcal{A}_i \in \mathfrak{A}$ embeds isomorphically into $\mathcal{S}$.*

A simple cover $\mathfrak{A}$ is computable *if every $\mathcal{A}_i \in \mathfrak{A}$ is a computable structure whose domain is an initial segment of ω. $\mathfrak{A}$ is* uniformly computable *if the sequence $\langle(\mathcal{A}_i, \overline{a}_i)\rangle_{i\in\omega}$ can be given uniformly: there must exist a computable function which, on input i, outputs a tuple of elements $\langle e_1, \ldots, e_n, \langle a_0, \ldots, a_{k_i}\rangle\rangle \in \omega^n \times \mathcal{A}_i^{<\omega}$ such that $\{a_0, \ldots, a_{k_i}\}$ generates $\mathcal{A}_i$ and φ_{e_j} computes the j-th function, relation, or constant in $\mathcal{A}_i$.*

The intention is that $\mathcal{S}$ itself should not be finitely generated, of course, although the definition is still valid in this case. Indeed, $\mathcal{S}$ is not at all required to be countable, since a single $\mathcal{A}_i$ may be isomorphic to many substructures of $\mathcal{S}$. For countable structures $\mathcal{S}$, a related notion is Fraïssé's concept of the *age* of $\mathcal{S}$, i.e. the set of all finitely generated substructures of $\mathcal{S}$. We note that all elements of $\mathfrak{A}$ will be models of T; this was the reason for which we demanded that T be $\forall$-axiomatizable.

Notice that the definition requires that the generators of $\mathcal{A}_i$ be given as a tuple $\langle a_0, \ldots, a_{k_i}\rangle$, so that k_i is computable uniformly in i and we know how many values from $\mathcal{A}_j$ are needed to define an embedding in $I_{ij}^{\mathfrak{A}}$. (In the language of [4], the definition requires the *canonical index* for the set $\{a_0, \ldots, a_{k_i}\}$.)

As an example, it is straightforward to show that the best-known uncountable structure in mathematics is locally computable. We omit the proof, since most of its ingredients have long been established.

Proposition 1. *The field $\mathcal{R} = (\mathbb{R}, +, \cdot, -, r, 0, 1)$ of real numbers is locally computable.* □

Notice that we have added the operations of negation and inversion (r, for *reciprocal*) to the usual language of fields, in order to get a Π_1 axiom set.

It is also useful to see a negative example. Although the real numbers form a locally computable field, Adding the usual $<$ relation to the field $\mathcal{R}$ of Proposition 1 destroys local computability.

Proposition 2. *The ordered field $(\mathcal{R}, <)$ of real numbers, with $\mathcal{R}$ as in Proposition 1, has no computable simple cover, uniform or otherwise.*

Proof. Let b be any noncomputable real number. We claim that the ordered subfield $\mathcal{B}$ of $\mathcal{R}$ generated by b has no computable presentation. Suppose $\mathcal{A}$ were a computable presentation of $\mathcal{B}$, with $a \in \mathcal{A}$ representing b. Then just from knowing the additive and multiplicative identity elements in $\mathcal{A}$, we could compute the representation in $\mathcal{A}$ of any rational number $\frac{p}{q}$. But then we could compute the n-th bit of the binary expansion of a, uniformly in n, just by using the computable relation $<$ in $\mathcal{A}$ to compare a to various dyadic rationals. But this is also the n-th bit of the binary expansion of b, which was assumed to be noncomputable. Therefore no such $\mathcal{A}$ can exist. $\square$

We will be concerned mainly with the full definition of a cover, in which we also describe how the substructures of $\mathcal{S}$ fit together.

Definition 2. *An embedding $f : \mathcal{A}_i \hookrightarrow \mathcal{A}_j$* lifts *to the inclusion $\mathcal{B} \subseteq \mathcal{C}$, via isomorphisms $\beta : \mathcal{A}_i \twoheadrightarrow \mathcal{B}$ and $\gamma : \mathcal{A}_j \twoheadrightarrow \mathcal{C}$, if if the diagram below commutes:*

$$\begin{tikzcd}
\mathcal{B} \arrow[r, "\subseteq"'] & \mathcal{C} \\
\mathcal{A}_i \arrow[u, "\beta", "\cong"'] \arrow[r, "f"] & \mathcal{A}_j \arrow[u, "\gamma", "\cong"']
\end{tikzcd}$$

with $\gamma \circ f = \beta$.

A cover of $\mathcal{S}$ *consists of a simple cover $\mathfrak{A} = \{\mathcal{A}_0, \mathcal{A}_1, \ldots\}$ of $\mathcal{S}$, along with sets $I^{\mathfrak{A}}_{ij}$ (for all $\mathcal{A}_i, \mathcal{A}_j \in \mathfrak{A}$) of injective homomorphisms $f : \mathcal{A}_i \hookrightarrow \mathcal{A}_j$, such that:*

- *for all finitely generated substructures $\mathcal{B} \subseteq \mathcal{C}$ of $\mathcal{S}$, there exist $i, j \in \omega$ and an $f \in I^{\mathfrak{A}}_{ij}$ which lifts to $\mathcal{B} \subseteq \mathcal{C}$ via some isomorphisms $\beta : \mathcal{A}_i \twoheadrightarrow \mathcal{B}$ and $\gamma : \mathcal{A}_j \twoheadrightarrow \mathcal{C}$; and*
- *for every i and j, every $f \in I^{\mathfrak{A}}_{ij}$ lifts to an inclusion $\mathcal{B} \subseteq \mathcal{C}$ in $\mathcal{S}$ via some isomorphisms β and γ.*

This cover is uniformly computable *if $\mathfrak{A}$ is a uniformly computable simple cover of $\mathcal{S}$ and there exists a c.e. set W such that for all $i, j \in \omega$,*

$$I^{\mathfrak{A}}_{ij} = \{\varphi_e \restriction \mathcal{A}_i : \langle i, j, e\rangle \in W\}.$$

A structure $\mathcal{B}$ is locally computable *if it has a uniformly computable cover.*

If $\mathfrak{A}$ is a computable simple cover, then every embedding of any $\mathcal{A}_i$ into any $\mathcal{A}_j$ is determined by its values on the generators of $\mathcal{A}_i$. Since $\mathcal{A}_i$ is finitely generated, all such embeddings are computable, and therefore it is reasonable to call $\mathfrak{A}$ a computable cover without any further requirements on the sets $I^{\mathfrak{A}}_{ij}$. For a uniformly computable cover, on the other hand, the sets $I^{\mathfrak{A}}_{ij}$ will play a key role in our development of the subject, and it should be kept in mind that $I^{\mathfrak{A}}_{ij}$ need not contain every possible embedding of $\mathcal{A}_i$ into $\mathcal{A}_j$.

It is an easy exercise to see that the second condition of Definition 2 follows trivially from the definition of a simple cover, for any embedding $f : \mathcal{A}_i \hookrightarrow \mathcal{A}_j$.

We include this second condition here because it is the dual of the first, and in the rest of our study of local computability, this duality between inclusion maps within $\mathcal{S}$ and embeddings among structures in $\mathfrak{A}$ will appear repeatedly.

Lemma 1. *A structure $\mathcal{S}$ has a uniformly computable cover (i.e. is locally computable) iff $\mathcal{S}$ has a uniformly computable simple cover.*

Proof. Given a uniformly computable simple cover $\mathfrak{A} = \{\mathcal{A}_0, \mathcal{A}_1, \ldots\}$, we adjoin all finitely generated substructures of each $\mathcal{A}_i$. The embeddings are precisely the inclusion maps from each substructure of $\mathcal{A}_i$ into $\mathcal{A}_i$. It is quickly seen that this yields a uniformly computable cover. □

In light of this lemma, one naturally asks why we bothered to give Definition 2. The answer is that local computability will be the $m = 0$ case in the following definition, which uses the enumeration of the sets $I_{ij}^{\mathfrak{A}}$ extensively. Indeed, it is the enumeration of the embeddings, rather than that of the finitely generated substructures of $\mathcal{S}$, which will be the heart of our study of local computability.

Definition 3. *Let $\mathfrak{A}$ be a cover of a structure $\mathcal{S}$. Every embedding β of any $\mathcal{A}_i \in \mathfrak{A}$ into $\mathcal{S}$ will be called* 0-extensional. *For each $m \geq 0$, we say that such an embedding β, with image $\mathcal{B} \subseteq \mathcal{S}$, is* $(m+1)$-extensional *if:*

- *for every $j \in \omega$, every $f \in I_{ij}^{\mathfrak{A}}$ lifts to an inclusion $\mathcal{B} \subseteq \mathcal{C}$ in $\mathcal{S}$ via β and some m-extensional match γ with domain $\mathcal{A}_j$; and*
- *for every finitely generated $\mathcal{C}$ with $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{S}$, there exists $j \in \omega$ and $f \in I_{ij}^{\mathfrak{A}}$ which lifts to $\mathcal{B} \subseteq \mathcal{C}$ via β and some m-extensional match γ with domain $\mathcal{A}_j$.*

A uniformly computable cover $\mathfrak{A}$ of $\mathcal{S}$ is m-extensional *if every $\mathcal{A}_i \in \mathfrak{A}$ m-extensionally matches some substructure of $\mathcal{S}$ and every finitely generated substructure of $\mathcal{S}$ m-extensionally matches some $\mathcal{A}_i \in \mathfrak{A}$. If such a cover exists, we say that $\mathcal{S}$ is* m-extensionally locally computable *(or just* m-extensional*). A structure is* ω-extensionally locally computable *if it is m-extensionally locally computable for every $m \in \omega$.*

Notice that $\mathcal{S}$ is 0-extensional iff $\mathcal{S}$ is locally computable, iff $\mathcal{S}$ has a uniformly computable simple cover (by Lemma 1). Definition 3 will be used in Proposition 4 to derive results about the complexity of the theory of $\mathcal{S}$. The idea of 1-extensionality is that the embeddings in the sets $I_{ij}^{\mathfrak{A}}$ (for all j) correspond precisely to the finitely generated superstructures of $\mathcal{B}$ in $\mathcal{S}$, rather than just to some possible extension of some $\mathcal{B}' \cong \mathcal{B}$ within $\mathcal{S}$ to some superstructure of $\mathcal{B}'$ in $\mathcal{S}$. The distinction is best illustrated by the negative example of Proposition 3 below. However, Proposition 5 will show that the definition holds for the field of complex numbers. Indeed, the set $I_{ij}^{\mathfrak{A}}$ of embeddings of one $\mathcal{A}_i$ in the cover into another $\mathcal{A}_j$ is just the set of all embeddings of $\mathcal{A}_i$ into $\mathcal{A}_j$, and this set is actually computable, uniformly in i and j.

m-extensionality is the obvious iteration of this notion. The extra conditions for extensionality strengthen the idea that each finitely generated substructure of $\mathcal{S}$ is represented by some $\mathcal{A}_i \in \mathfrak{A}$: not only are they isomorphic, but the embeddings (given by $I^{\mathfrak{A}}$) of $\mathcal{A}_i$ into other structures in $\mathfrak{A}$ coincide exactly with the extensions of $\mathcal{B}$ to larger finitely generated substructures of $\mathcal{S}$.

Proposition 3. *The field $\mathcal{R}$ of real numbers is not 1-extensionally locally computable.*

Proof. Suppose that $\mathfrak{A}$ were a 1-extensional cover of $\mathcal{R}$. Fix any noncomputable real number $t \in \mathbb{R}$. Definition 3 gives an $\mathcal{A}_i \in \mathfrak{A}$ which 1-extensionally matches (via some isomorphism β) the subfield $\mathcal{B}$ of $\mathcal{R}$ generated by t, and we may assume we know i and $\beta^{-1}(t)$, since they constitute finitely much information.

Now we can enumerate the lower cut of rationals $q < t$ in $\mathcal{R}$, knowing that extensions of $\mathcal{B}$ in $\mathcal{R}$ correspond to embeddings $f \in I_{ij}^{\mathfrak{A}}$ (for all j) in the 1-extensional cover. For any rational $q \in \mathcal{R}$:

$$\begin{aligned}\models_{\mathcal{R}} q < t \iff & \models_{\mathcal{R}} (\exists x) x^2 = t - q \\ \iff & (\exists \text{ f.g. } \mathcal{C})[\mathcal{B} \subseteq \mathcal{C} \subset \mathcal{R} \ \& \ \models_{\mathcal{C}} (\exists x) x^2 = t - q] \\ \iff & (\exists j)(\exists f \in I_{ij}^{\mathfrak{A}}) \models_{\mathcal{A}_j} (\exists x) x^2 = f(\beta^{-1}(t - q)) \\ \iff & (\exists j)(\exists f \in I_{ij}^{\mathfrak{A}})(\exists a \in \mathcal{A}_j) \models_{\mathcal{A}_j} a^2 = f(\beta^{-1}(t)) - f(\beta^{-1}(q)).\end{aligned}$$

A similar argument holds for the upper cut of rationals $q > t$, using square roots of $(q - t)$ in $\mathcal{R}$. This contradicts the noncomputability of t. (Of course, β and f fix the rationals, so $f(\beta^{-1}(q)) \in \mathcal{A}_j$ is just the element of $\mathcal{A}_j$ representing q. This $f(\beta^{-1}(q))$, lying in dom$(\mathcal{A}_j)$, is a natural number, but the element of $\mathcal{A}_j$ representing any particular rational q can easily be computed from the numerator and denominator of q, uniformly in j, by using the functions of $\mathcal{A}_j$.) □

So the extensional local computability of $\mathcal{C}$ in Proposition 5 does not follow solely from the existential closure of the structure; after all, $\mathcal{R}$, viewed as a real closed field, is also existentially closed. The difficulty for $\mathcal{R}$ is that real closed fields have an implicit order on their elements, whether it is included in the language of the structure or not, and as we saw in Proposition 2, adding the order relation to $\mathcal{R}$ destroys local computability. $\mathcal{R}$ itself can still be locally computable, because the relation $<$ cannot be defined in $\mathcal{R}$ without quantifiers (even though it is both Σ_1-definable and Π_1-definable!) and existential questions about $\mathcal{R}$ can be left unanswered by a uniformly computable cover. An extensional cover, on the other hand, answers all such questions, as we now see.

Proposition 4. *For $m \in \omega$, any m-extensionally locally computable structure $\mathcal{S}$, and any $n \leq m+1$, the Σ_n-theory of $\mathcal{S}$,*

$$\{\varphi \in Th(\mathcal{S}) : \varphi \text{ is a } \Sigma_n \text{ sentence}\},$$

is itself a Σ_n set in the arithmetic hierarchy. (For $n > 0$, this means that the Σ_n-theory is 1-reducible to $\emptyset^{(n)}$, and for $n = 0$, the Σ_0-theory is computable.)

Proof. Let $\langle \mathcal{A}_i \rangle_{i \in \omega}$ be an m-extensionally computable cover of $\mathcal{S}$. For arbitrary $n \leq m+1$, the key fact is simply that for any formula $\varphi(\overline{x})$,

$$\models_{\mathcal{S}} (\exists \overline{x}) \varphi(\overline{x}) \ \text{ iff } \ (\exists \text{ f.g. } \mathcal{B} \subseteq \mathcal{S})(\exists\, \overline{b} \in \mathcal{B}^j) \ \models_{\mathcal{S}} \varphi(\overline{b}).$$

When we have alternating quantifiers, we need to take superstructures at each step. For an arbitrary formula $\varphi(\overline{x}, \overline{y})$,

$$\begin{aligned} & \models_{\mathcal{S}} (\exists \overline{x})(\forall \overline{y})\varphi(\overline{x}, \overline{y}) \\ \text{iff} \quad & (\exists \text{ f.g. } \mathcal{B} \subseteq \mathcal{S})(\exists\, \overline{b} \in \mathcal{B}^k) \models_{\mathcal{S}} (\forall \overline{y})\varphi(\overline{b}, \overline{y}) \\ \text{iff} \quad & (\exists \text{ f.g. } \mathcal{B} \subseteq \mathcal{S})(\exists\, \overline{b} \in \mathcal{B}^k)(\forall \text{ f.g. } \mathcal{C} \text{ s.t. } \mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{S})(\forall \overline{c} \in \mathcal{C}^p) \models_{\mathcal{S}} \varphi(\overline{b}, \overline{c}) \end{aligned}$$

If the original sentence was Σ_2, then the matrix (after all the quantifiers) will be the truth in $\mathcal{S}$ of the quantifier-free formula $\varphi(\overline{b}, \overline{c})$. In this case, $\varphi(\overline{b}, \overline{c})$ holds in $\mathcal{S}$ iff it holds in $\mathcal{C}$, so we can add the following:

$$\begin{aligned} \text{iff} \quad & (\exists \text{ f.g. } \mathcal{B} \subseteq \mathcal{S})(\exists\, \overline{b} \in \mathcal{B}^k)(\forall \text{ f.g. } \mathcal{C} \text{ s.t. } \mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{S})(\forall \overline{c} \in \mathcal{C}^p) \models_{\mathcal{C}} \varphi(\overline{b}, \overline{c}) \\ \text{iff} \quad & (\exists i)(\exists\, \overline{a} \in \mathcal{A}_i^k)(\forall j)(\forall f \in I_{ij}^{\mathfrak{A}})(\forall \overline{d} \in \mathcal{A}_j^p) \models_{\mathcal{A}_j} \varphi(f(\overline{a}), \overline{d}). \end{aligned}$$

The definition of 1-extensional cover shows these last two lines to be equivalent. Specifically, if the last line holds, then the witness $\mathcal{A}_i$ has a 1-extensional match β onto some $\mathcal{B} \subseteq \mathcal{S}$, and Definition 3, applied to any $\mathcal{A}_j$ and $f \in I_{ij}^{\mathfrak{A}}$, provides a 0-extensional match γ from $\mathcal{A}_j$ onto some $\mathcal{C} \supseteq \mathcal{B}$ such that $\gamma \circ f = \beta$. Then $\varphi(\gamma(f(\overline{b})), \gamma(\overline{c}))$ must hold in $\mathcal{C}$, since $\varphi(f(\overline{b}), \overline{c})$ holds in $\mathcal{A}_j$ and γ is an isomorphism. Conversely, if the next-to-last line holds, a similar argument applies, since there is some 1-extensional match onto the witness $\mathcal{B}$ from some $\mathcal{A}_i \in \mathfrak{A}$. This completes the proof of the result on 1-extensional structures.

The obvious iteration of this process, applied to any Σ_n sentence about $\mathcal{S}$, yields a statement consisting of a Σ_n-sequence of quantifiers over structures in $\mathfrak{A}$, their elements, and the sets $I_{ij}^{\mathfrak{A}}$, followed by a quantifier-free statement about an $\mathcal{A}_j \in \mathfrak{A}$. The argument requires that each $\mathcal{A}_i$ correspond to some $\mathcal{B}$ via an $(n-1)$-extensional map, so that the extensions must then correspond via $(n-2)$-extensional maps, and so on down to 0-extensional maps once all the quantifiers have been moved outside the turnstile $\models$. Therefore, for an m-extensionally locally computable $\mathcal{S}$ with $m \geq (n-1)$, the Σ_n statement yielded by iterating the process holds iff the original Σ_n sentence held in $\mathcal{S}$. Since the structures in $\mathfrak{A}$, the sets $I_{ij}^{\mathfrak{A}}$ and the atomic diagram of such an $\mathcal{A}_j$ are all computable uniformly in i and j, the truth of the original Σ_n-sentence in $\mathcal{S}$ is itself a Σ_n fact. Moreover, this process is entirely uniform in n. □

Notice that this argument does not extend to values $n > m+1$. For $m = 0$ a specific counterexample appears in Proposition 3.

Theorem 1. *For $m \in \omega$, any m-extensionally locally computable structure $\mathcal{S}$, any finite tuple $\overline{p}$ of parameters from $\mathcal{S}$, and any $n \leq m$, the Σ_n-theory of $\mathcal{S}$ over $\overline{p}$,*

$$\{\varphi \in Th(\mathcal{S}, \overline{p}) : \varphi \text{ is a } \Sigma_n \text{ sentence}\},$$

is itself Σ_n, uniformly in n and in an appropriate description of the parameters (as discussed after the proof).

Proof. Let $\mathcal{B}$ be generated by $\overline{p}$ in $\mathcal{S}$, and fix an m-extensional match $\beta : \mathcal{A}_l \twoheadrightarrow \mathcal{B}$ for some $\mathcal{A}_l \in \mathfrak{A}$. As before, we give an example by evaluating the truth in $\mathcal{S}$ of an arbitrary Σ_2 sentence with the parameters $\overline{p}$, assuming now that $m \geq 2$. Seeting $a_i = \beta^{-1}(p_i)$ and applying an argument similar to that in the proof of Proposition 4, we see that the Σ_2 sentence $(\exists\overline{x})(\forall\overline{y})\varphi(\overline{p},\overline{x},\overline{y})$ holds in $\mathcal{S}$ iff

$$\begin{aligned}&(\exists i)(\exists h \in I^{\mathfrak{A}}_{li})(\exists \overline{b} \in \mathcal{A}^k_i)(\forall j)(\forall f \in I^{\mathfrak{A}}_{ij})(\forall \overline{c} \in \mathcal{A}^m_j)\\ &\models_{\mathcal{A}_j} \varphi(f(h(\overline{a})), f(\overline{b}), \overline{c}),\end{aligned}$$

which is a Σ_2 condition, uniformly in $\overline{a}$ and l. The obvious iteration works for any $n \leq m$, but no longer applies when $n = m + 1$. In the example above, since $\mathcal{S}$ is 2-extensional, we can find a 2-extensional match β onto $\mathcal{A}_l$, so that $\mathcal{A}_i$ will have a 1-extensional match in its turn. Adding the parameters forces us to start by fixing an $\mathcal{A}_l \in \mathfrak{A}$ and a β, whereas in Proposition 4 we were allowed simply to search for any $\mathcal{A}_i$ and a single embedding into an $\mathcal{A}_j$. Hence parameters require one more level of extensionality.

Of course, knowing an original parameter $p_i \in \mathcal{B}$ is useless to us; we need to know l and the value $a_i = \beta^{-1}(p_i)$ in $\mathcal{A}_l$. For finitely many parameters, this constitutes only finitely much information, but we also wish to consider uniformity. It does not make sense to ask that parameters from a potentially uncountable structure $\mathcal{S}$ be given uniformly. Instead, our formal statement of uniformity is that if we are given an l and finitely many parameters $\overline{a}$ from $\mathcal{A}_l$, then for any n and any n-extensional match β mapping $\mathcal{A}_l$ into $\mathcal{S}$, the Σ_n-theory of $\mathcal{S}$ over the parameters $\beta(a_1), \ldots, \beta(a_k)$ is Σ_n uniformly in l and $\overline{a}$ and n. □

We view Theorem 1 as the strongest argument yet that local computability, and in particular perfect local computability (see Definition 4), is the appropriate analogue in uncountable structures to computable presentability in countable structures. The point of a computable presentation of a structure is not just that it allows us to compute the atomic theory and enumerate the Σ_1-theory and so on, but that it actually allows us to do over specific elements of the structure: the atomic *diagram* is computable, and the Σ_n diagram is Σ_n, uniformly in n. For an uncountable $\mathcal{S}$, of course, there is no effective way to name all individual elements, so it is hopeless to expect the entire atomic diagram to be computable. An m-extensional cover, however, gives us a way of describing individual elements and tuples of them: using the cover, we name an $\mathcal{A}_l$ which m-extensionally matches the substructure of $\mathcal{S}$ generated by the tuple, and specify which elements of $\mathcal{A}_l$ correspond to the tuple.

To state the same fact differently, having an m-extensional cover tells us exactly what information we need about the tuple $\overline{p}$ from $\mathcal{S}$ in order to compute the atomic theory of $\mathcal{S}$ over $\overline{p}$, or to enumerate its Σ_1 theory over $\overline{p}$, etc. In Proposition 5, for instance, for the field of complex numbers, an $\mathcal{A}_i$ is given by its transcendence degree and the minimal polynomial of a single additional element generating the rest of $\mathcal{A}_i$ over a transcendence basis. If we can determine this information for the subfield $\mathbb{Q}(\overline{p}) \subset \mathbb{C}$, and know which elements correspond to $\overline{p}$, then without further information we can give a Σ_n description of the Σ_n-theory of $(\mathbb{C}, \overline{p})$. Indeed, there is another perfect cover of the field $\mathbb{C}$, more difficult to describe, in which every finite tuple $\overline{p}$ from $\mathbb{C}$ corresponds, via a perfect match, to the generators of a particular $\mathcal{A}_i$ in the cover. Using this cover, one would only need to know the minimal polynomial of each p_{i+1} over $\mathbb{Q}(p_1, \ldots, p_i)$, or else to know that no such minimal polynomial exists. Similarly, each m-extensional cover of any $\mathcal{S}$ says, "if you tell me this particular information about your tuple $\overline{p}$ from $\mathcal{S}$, I will give you a Σ_n-presentation of the Σ_n facts about $\overline{p}$ in $\mathcal{S}$, for each $n \leq m$."

Corollary 1. *For any ω-extensionally locally computable structure $\mathcal{S}$, and any finite parameter set $\overline{p}$ from $\mathcal{S}$, the Σ_n-theory of $(\mathcal{S}, \overline{p})$ is Σ_n for every n, uniformly in n and in $\overline{p}$ (as described above). In particular, this holds for any $\mathcal{S}$ with a correspondence system, including any perfectly locally computable $\mathcal{S}$.* □

Corollary 2. *Any two structures with the same ω-extensional cover are elementarily equivalent and realize the same types. Also, any two structures with the same m-extensional cover have the same Σ_{m+1}-theory.* □

We add one more version of local computability, even stronger than ω-extensional local computability, whose main interest stems from Theorem 2.

Definition 4. *Let $\mathfrak{A}$ be a uniformly computable cover for a structure $\mathcal{S}$. A set M is a* correspondence system *for $\mathfrak{A}$ and $\mathcal{S}$ if it satisfies all of the following:*

1. *Each element of M is an embedding of some $\mathcal{A}_i \in \mathfrak{A}$ into $\mathcal{S}$; and*
2. *Every $\mathcal{A}_i \in \mathfrak{A}$ is the domain of some $\beta \in M$; and*
3. *Every finitely generated $\mathcal{B} \subseteq \mathcal{S}$ is the image of some $\beta \in M$; and*
4. *For every i and j and every $\beta \in M$ with domain $\mathcal{A}_i$, every $f \in I_{ij}^{\mathfrak{A}}$ lifts to an inclusion $\beta(\mathcal{A}_i) \subseteq \gamma(\mathcal{A}_j)$ via β and some $\gamma \in M$; and*
5. *For every i, every $\beta \in M$ with domain $\mathcal{A}_i$, and every finitely generated $\mathcal{C} \subseteq \mathcal{S}$ containing $\beta(\mathcal{A}_i)$, there exist a j and an $f \in I_{ij}^{\mathfrak{A}}$ which lifts to $\beta(\mathcal{A}_i) \subseteq \mathcal{C}$ via β and some $\gamma \in M$ mapping $\mathcal{A}_j$ onto $\mathcal{C}$.*

The correspondence system is perfect *if it also satisfies*

6. *For every finitely generated $\mathcal{B} \subseteq \mathcal{S}$, if $\beta : \mathcal{A}_i \twoheadrightarrow \mathcal{B}$ and $\gamma : \mathcal{A}_j \twoheadrightarrow \mathcal{B}$ both lie in M and have image $\mathcal{B}$, then $\gamma^{-1} \circ \beta \in I_{ij}^{\mathfrak{A}}$.*

If a perfect correspondence system exists, then its elements are called perfect matches *between their domains and their images. $\mathcal{S}$ is then said to be* perfectly locally computable*, with* perfect cover *$\mathfrak{A}$.*

This concept is related to extensionality, clearly, and any element β of a correspondence system M is quickly seen to be an m-extensional match for every $m \in \omega$. However, perfect local computability is stronger than ω-extensional local computability in two distinct ways. First, for the map β to be an m-extensional match, we only needed the existence of $(m-1)$-extensional matches γ to relate the embeddings $f \in I_{ij}^{\mathfrak{A}}$ (for all j) to the finitely generated extensions of the image of β in $\mathcal{S}$, and for different values of m, we could use different maps β.

Here Conditions 4 and 5 require that the isomorphisms γ be in M themselves, hence that they satisfy the same conditions. The second difference is Condition 6, which is not related to Definition 3, but appears necessary for Theorem 2.

Proposition 5. *Every algebraically closed field of characteristic* 0*, and in particular the field* $\mathbb{C}$*, is perfectly locally computable.* □

Theorem 2. *Let* $\mathcal{S}$ *be a countable structure. Then* $\mathcal{S}$ *is perfectly locally computable iff* $\mathcal{S}$ *is computably presentable.*

We omit these proofs for reasons of space. Proposition 5 is not difficult to prove, but the $\Longrightarrow$ direction of Theorem 2 requires a detailed construction. To close, we state several more relevant results, also without proof.

Proposition 6. *Let* $\mathcal{R}_c$ *be the ordered field containing all computable real numbers. Then* $\mathcal{R}_c$ *has a computable cover, but no uniformly computable cover.* □

Lemma 2. *There exist countable structures* $\mathcal{S}$ *and* $\mathcal{S}'$ *with the same uniformly computable cover, such that* $\mathcal{S}$ *is computable (and hence perfectly locally computable), but* $\mathcal{S}'$ *is not computably presentable, indeed not even* 1*-extensional.* □

Lemma 3. *There exist* 2^ω*-many countable structures, all with the same uniformly computable cover, which are pairwise elementarily non-equivalent. Indeed, these structures all have distinct* Σ_2*-theories, and every Turing degree is the degree of the* Σ_2*-theory of some such structure.* □

Finally, the converse of each statement in Theorem 1 is false.

Proposition 7. *There exists a tree* T *which is not* 1*-extensionally locally computable, yet such that for every* m *and every finite tuple* $\overline{p}$ *from* T*, the* Σ_m*-theory of* $(T, \overline{p})$ *is itself* Σ_m*.*

References

1. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions, and universal machines. Bulletin of the AMS 21, 1–46 (1989)
2. Braverman, M., Cook, S.: Computing over the Reals: Foundations for Scientific Computing. Notices of the AMS 53(3), 318–329 (2006)
3. Harizanov, V.S.: Pure computable model theory, Handbook of Recursive Mathematics, vol. 1, pp. 3–114. Elsevier, Amsterdam (1998)
4. Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, New York (1987)

Logic and Control

Anil Nerode

Cornell University,
Ithaca, New York, USA

Abstract. I will give a brief account of how I came to introduce the term Hybrid System and to encourage it as a subject of study. I will also discuss what one should understand to pursue the subject fully, and also discuss the lines of development of my joint research with Wolf Kohn in this area. This is an area at the interface of mathematics, computer science, and control engineering, which has been pursued by many in all three areas since its inception. It can be thought of as the subject which deals with the interaction of discrete and continuous processes. This paper is intended to be legible to those without advanced analysis or differential geometry in their background. We concentrate on control, not on logics of hybrid systems, and aim at an audience unfamiliar with control. I make no apologies for not being technical. This is not a survey of the field. For a year 2000 survey see [1]. A survey made now would be very large.

Relevant Subjects. To advance hybrid systems one really ought to acquire a firm grasp of

- PROLOG and Horn Clause Logic
- Temporal logics used in Computer Science
- Automata logics of Büchi, Rabin
- Automata theory for finite state transducers of Eilenberg and Schutzenberger
- Linear Control
- Pontryagin's optimal control theory
- Optimization Theory
- Calculus on Banach spaces
- Relaxed calculus of variations of L. C. Young and E. J. McShane (1937-40)
- Tensor calculus and Cartan exterior differential forms
- Differential geometry of geodesics, connections, sprays
- Finsler geometry
- Variational inequalities
- Lie Theory

This was a bit much for most engineers, many mathematicians, and all businessmen. So to prove that our mathematical technology can be implemented for real time systems, Wolf Kohn and I founded a company in 1995 to develop real time hybrid control software for specific applications. We developed general software, hoping the user could simply dump their particular models into that software, letting the software extract the desired control automaton. But it turns out in practice that each important application requires extensive development of special models and speed-up algorithms based on the particular model in addition to the universal software. This is not really surprising. Generally, if a physicist wants information about how to solve a new system of partial differential equations, the theory of PDE's and the general algorithms we have for them, while necessary, is the least of the problem. It often takes years to work out practical algorithms for a single problem, especially real time algorithms. In the course of development often the underlying model is changed to one which is more tractable but still represents the problem closely enough to be useful.

Complex Systems
Complex systems can be viewed as consisting of:

- "Plants" whose evolution in time is to be constrained
- "Sensors", which sense current plant state and can transfer measurements of plant state to "controllers"
- "Controllers" which use this information to issue control orders to
- "Actuators" which act to change plant state.

Fundamental Problem of Control
Given a plant and a plant peformance specification,

- Design and place sensors of plant state.
- Design and place controllers of plant state.
- Design, analyze, simulate, prototype
- Test to verify that the completed system satisfies the performance specifications.

Some Grand Challenge Problems for Control

- Air traffic
- Automated battle command
- Supply chain
- Robotic cooperation
- Distributed simulation
- Network routing
- Banking systems
- Financial prediction
- The Internet

Kohn and I and our co-workers have published proposals for applying our methods to problems in these and other areas. We have produced software products in some of them.

My Background in Control. I am best known as a logician. But in 1954, when I was a graduate student in pure mathematics at the University of Chicago, I had a position at the Institute for Systems Research. This organization was created by the by the Dean of Physical Sciences of the University of Chicago, Walter

Bartky, at the request of the Wright Air Development Command. I learned control theory and practice there on the job, mostly by reading the whole MIT Radiation Laboratory red series and also from senior engineer James Corbett. I worked on design and evaluation of air to air weapon systems. I have been a consultant for over fifty years for many military and civilian engineering projects.

Pacifica Meeting and Col. Erik G. Mettala
In 1990 I was Director of the Mathematical Sciences Institute (MSI) at Cornell, sponsored by the Army Research Office (ARO). My program director, Dr. Jagdish Chandra, requested that I attend a DARPA meeting in Pacifica, California, July 17-19, 1990, near Monterrey, California. The meeting was entitled "Domain Specific Software Architectures". I had no idea what the organizer, Col. Erik G. Mettala, had in mind. I could not make heads nor tails out of the preliminary announcement. When I got there, most of the participants were from the control industry. Colonel Mettala explained that the purpose of the meeting was to explore how to clear a major bottleneck, the control of large military systems such as air-land-sea forces in a battle space. Such problems occur widely in government, business and industry as well. How can one control highly non-homogeneous non-linear systems in which many digital processes and continuous processes interact in real time?

First Day. Many control industry representatives said that if their community were given several hundreds million per year, they could scale up their design software for control systems to extract control for such large systems. They outlined the tools they intended to scale. I was astounded that these were the SAME tools that I and others had always used for design of control systems, but now implemented in software. These tools have been extremely successful for the control of linear and mildly nonlinear systems throughout business and industry. They are still the bread and butter of practical control engineering. But they can not handle complex systems.

Second Day. Participants were asked to submit their analyses for the second day of the meeting. I wrote a one page paper [11] to explain that as the state of the system evolves, the digital and continuous systems interact. A change of state of the digital subsystem causes a change of state of the continuous subsystem. A change of state of the continuous subsystem causes a change of state in the digital subsystem, and so on ad infinitum. I said that although one can simulate on a computer the evolution of state induced by a specific proposed control, there exist no mathematics or algorithms which, applied to a model of the complex system and its performance requirements, could extract a control program which enforces that the system satisfy its performance requirements. I left this paper on the table of handouts the second morning, next to the comments of others. The second day the group had a lively argument trying to determine what we were talking about. Someone suggested that my formulation be taken as the definition of the problem, a motion which was adopted. I suggested that DARPA fund a modest research program to determine if appropriate mathematics and algorithms could be developed to extract such controls, and in this I was seconded by Wolf Kohn of Boeing.

First Workshop. I organized a first Hybrid Systems Workshop on June 10-12, 1991 at MSI at Cornell. I brought together all researchers I could locate world wide who had worked on logical (digital) control of physical plants, some from the control point of view, some from the logic and expert systems point of view. As a result of the enthusiasm of the workshop participants, I organized a second workshop.

Second Workshop. This was the first International Hybrid Systems Conference, held at Cornell in 1992. I proposed a simple representation for Hybrid Systems. This representation did not contribute to extracting controls, but it did provide a common framework for many who subsequently have published work on hybrid systems. It has been widely reproduced and elaborated. The conference volume [2] was the first volume devoted to Hybrid Systems.

The Simple Hybrid Systems Model

- a plant
- a digital control automaton
- a sensor of plant state
- an actuator to change plant state
- an analog to digital (AD) converter
- a digital to analog (DA) converter.
- The sensor measures current plant state, sends this measurement to the AD converter.
- The AD converter outputs a digitized measurement sent to the control automaton.
- The control automaton sends a digitized command to the AD converter.
- The AD converter converts this to a continuous input to the actuator.
- The actuator changes the plant state.

We can model hybrid systems with many plants, sensors, control automata, actuators in the same way.

In retrospect I would say that the significant relevant literature for hybrid systems that existed at that time was as follows.

- First, there was linear control with switching. I and many others had used this over forty years. One can approximate to almost any non-linear control by piecewise linear control. If an engineer devises a scheme for when to switch linear controls based on sensed system state, that engineer can implement the scheme. The problem was, there was no methodology for extracting such a control scheme to meet performance specifications.
- Second, there was Wonham and Ho's Discrete Event Systems. They generally replace physical models by automata models, and then control those models. This leads to difficult questions as to how well any controls extracted will control the original physical process, in particular how robust such control

schemes are in the real world. Verification by simulation has not proved to be practical for large non-linear systems. The number and length of simulation runs needed is huge due to the vast variety of dynamical phenomena which may be encountered.

– Third, there was the Declarative Control of Wolf Kohn at Boeing. At the time I was unaware of Kohn's work [8]. Kohn's supporting comments at the Pacifica meeting and at the subsequent 1991 Cornell workshop were so acute that I wanted to know what he did and so I asked him to send me his papers. I found them very difficult to decipher. Kohn's method was described to me in 1990 as one of the few successful uses of AI methods for control. It looked like AI because it used PROLOG. When I got his papers, it took me a year or two to figure out what the proper underpinning was for the method. It was not AI. More about that below.

A Growing Literature. There were three more workshops in that series [4], [5], [6]. After that many others have held Hybrid Systems workshops, conferences, and engineering and computer science meetings too numerous to list. A majority of the literature investigates verification problems. Our interest is in extraction of control algorithms. In a sense these do not require formal verification since they are the result of a mathematical derivation. But they have to be tested extensively anyway because the models used may not reflect the dynamics with sufficient accuracy and may not be sufficiently robust.

Capsule Historical Summaries

Linear Control. These algorithms extract linear controls (such as PID controllers) for linear systems. They are a 20th century triumph of linear algebra and Laplace transforms. I think of serious linear control as having its inception in the 1916 patents of Armstrong for linear amplifiers, the boost that moved us from crystal sets to loudspeakers and AM radio and a new industry. Then H. Bode and his colleagues at Bell Laboratories developed high performance amplifiers for telephone lines in the 1920's and 1930's, developing frequency response curves, the Nyquist diagram and Hilbert transforms. At MIT Norbert Weiner coined the name Linear Servomechanisms for negative feedback linear control of linear systems. In World War II at the MIT radiation laboratories there was systematic development of Laplace transforms and transfer functions for the design of linear servomechanisms to keep RADAR on its moving targets. Much of the further development of linear control stemmed from MIT. The subject was codified neatly by Kalman in about 1960.

Digital Control. The 1900-1950 period was the golden age of magnetic relay circuits. Two big companies developing and vending them were Western Electric (for AT&T) and Kellogg Switch (for elevators and subways). A widely publicized advance was Claude Shannon's 1937 master's thesis on using the language and theorems of Boolean algebra to analyze switching circuits. Another was Harvard's Howard Aiken and his Mark I computer (1939-44) and Germany's Konrad Zuse's Z1-Z4 (1938-44). To minimize cost was then to minimize the number of relays used. Each relay could open and close many circuits. This is a very complicated minimization since the connectives a relay represented had many arguments. Next we have the evolution of logic based control: programmable controllers, real time operating systems, robotics, digital signal processing, and digital circuit design. Nowadays embedded digital control is ubiquitous in our consumer products from cameras to automobiles.

Expert Systems and Control. At the end of the 1950's, as time on mainframes became available for pure research, the first programs for automated deduction were written. Very limited forms of some of these systems were used to write the first medical diagnosis systems such as Mycin at Stanford. In 1962-3, Alan Robinson at Syracuse, building on the work of Herbrand in the 1920's, developed his unification resolution method for automatic theorem proving for first order logic. This is still the main paradigm in automated deduction. In 1970 Kowalski, observing the effectiveness of Colmerauer's planning language PROLOG for extracting plans from constraints and goals, gave that language an overhaul with a complete theoretical underpinning, an elegant reduction of Robinson's unification-resolution method for universal Horn sentence logic. For the sake of improving running times, all commercial applications of PROLOG are incomplete and semantically incorrect. They use such shortcuts as depth first search, the cut rule, and a unification algorithm without an occurs check. This has led to a specialized profession of PROLOG Programmers who avoid these pitfalls. The credo "the program specification is the program" does not hold for the commercial PROLOG's. Hundreds of PROLOG based planning systems have been written by professional PROLOG programmers. How do expert systems relate to control? By the 1970's expert systems had been constructed to give advice on courses of action, and generally for planning. There is now a large logic based planning community. A description of the current system state is fed into the expert system, which then deduces a recommended action to lead to a prescribed goal state on the basis of its facts and rules. Most of the facts and rules are the constraints describing the system, a system model reflected in rules and facts. If the expert system is written in Colmeraur-Kowalski's PROLOG, the course of action recommended to get to the goal is part of the answer substitution that proves the goal state can be obtained from current system state. The answer substitution contains the sequence of actions needed to get to the goal provided the rule base is kept constant and the states of the controlled system have the transitions expected by the rule base. However, if due to system dynamics unmodeled in the expert system, the state of the system evolves in a way unexpected by the rule base, and if this is detected, the rule base has to be modified, and a new answer substitution has to be deduced from a changed rule base. This form of belief revision is carried out by extralogical assert and retract operations which change the rule base. If one automates input of system state, execution of recommended action, and the belief revision mechanism, so that human intervention is eliminated, we have a control system. A planning policy can be viewed as a control program to steer the system to a specified goal, while preserving constraints.

Limitations of Expert Systems for Control. In the 1970's and early 1980's, attempts were made to use this technology for real time control of physical systems, such as aircraft or missles or tank formations. But, given a goal and the physical model of a complex non-linear system, there existed no general tools for extracting a control program leading to a prescribed goal, if one indeed exists. Ad hoc methods for figuring out transition table representations of such controls were not very successful because of the size of state spaces and because of a lack of tools to analyze the effect of a control imposed on a non-linear system. If expert system software has to carry out online deductions to decide what control to use in real time we are in trouble. Automated deduction does not yet execute fast enough to control a rapidly changing physical process. For instance, if we use PROLOG and the present state and the goal state are specified, PROLOG carries out a backwards chaining search for a branch connecting two nodes of a graph, the present state and a goal state. If we are working "off line" to produce a control map, the time required to compute the control map may not matter. But this does not meet the reuirements of real time online control of life-critical processes, when the control law has to be changed on line due to unexpected changes in constraints or intentional changes in goal.

Automata and Automata Logics. In 1943 McCulloch and Pitts developed a theory of neural nets in a form of predicate temporal logic. Between 1950 and 1965 the theory of finite automata and their languages was developed by Kleene, Moore, Mealy, McNaughton, Büchi, Nerode, Myhill, Scott, Rabin, Schutzenberger, Eilenberg. In the 1960's and 1970's Büchi (S1S) and Rabin (S2S) developed the theory of automata on infinite strings, extensions of which are active areas of research now in logic and computer science.

State Charts and Fuzzy Control. In the 1980's piecewise linear control was explored for non-linear systems. A generic description of the methodology is, break up the state space into a finite number of regions; as a controlled trajectory enters a region, change the linear control being used by consulting a transition table. The problem is that there was no systematic methodology either for breaking the state space into regions or constructing the required transition table. A main point in developing hybrid systems theory is to explore methodologies for doing just this.

Static Optimization. In both the World Wars maximizing the throughput of supply lines for troops and ships and aircraft was of cardinal importance. So was operational planning for the distribution of men, troops, ships. Driven by these needs, mathematical optimization tools were developed with consequent wide impact on business, government, and industry. Optimization deals with finding an optimal state or an optimal path of system states to get to a desired goal relative to some definition of optimality. Important early contributions to static optimization were Van Dantzig's linear programming(1949) and Gomory's integer programming (1957).

Connections with Logic. Optimization theory did not arise from logic. But the linear programming problem gives a solution to the decision problem for the first order theory of the additive group of reals under addition and order, and conversely. In fact the simplex method can replace the usual logician's decision method of elimination of quantifiers. The mixed integer linear programming problem is equivalent to the decision problem for the first order theory of the additive group of reals under addition, order, and with a distinguished predicate for the integers; and indeed the usual algorithm supplies such a decision method. These structures are representable by finite and Büchi automata, which gives alternate automata theoretic decision methods.

Dynamic Optimization. At the same time, the theory of optimal policies for dynamic (time dependent) optimization problems was developed, including Bellman's dynamic programming (1949), Issacs's differential games for pursuit (1951), and Pontryagin's optimal control theory (1960). These subjects have some fragmentary connections with logic through temporal logics and automata. We would expect that future hybrid systems developments will reveal deeper connections. Generally speaking, logics which deal with function spaces and functional analysis have not been extensively developed except for **O**-minimal theories, which indeed give some hybrid system results.

Kohn-Nerode
Control Maps. Kohn and I model the usual space of states x of the system as a manifold which is defined by the constraints, some of which may be differential and some of which may be logical. We refer to the usual system state x as "position x" and use x' as the variable for possible "velocities", rates of change of state x along trajectories on the manifold x' . The pair (x, x') may as well be taken as a point in the tangent manifold of the state manifold of x's. For the uninitiated, this use of x' as a free variable ranging over the possible tangent directions is a standard source of confusion in trying to read the traditional calculus of variations texts. Here x' is just another variable, it is not a derivative at a point on a curve (which would be denoted $y'(t)$ for a curve $y(t)$.) The intuition for a control map is that the control map determines the new x' when your trajectory passes through x; that is, the control map determines the direction field along the controlled trajectory. We view a control map as a map $\mathbb{M}$ from the tangent manifold of the system state manifold to controls. A control map $\mathbb{M}$ and a state A induce a *controlled trajectory* of states starting at A. Suppose that we can define a non-negative cost function (Lagrangian) on controlled trajectories starting at state A ending at state B such that if we add up (integrate) costs along a controlled trajectory, we get the total cost of going from A to B along that controlled trajectory using the prescribed control map. An optimal control map from A to B, if such exists, is one yielding a controlled trajectory whose cost is miminal among the costs of all controlled trajectories from A to B. Bellman called this the *cost-to-go* from A to B. Finding controls as a function of state was called by Pontryagin the synthesis problem. The vast majority of the control literature represents controls as functions of time. It is very desirable to have

controls as a function of state because there are always unmodeled dynamics. At any time in the real system you may be in an unexpected state. You do not wish to use a control scheduled at that time which assumes you are in another state at that time. (For relaxed control maps as discussed below, the direction x' assigned by the control map is the expected value of x' using a probability measure.)

Necessary Conditions for Optimal Controls. There may be a wide choice of cost functions which would serve for a given problem, and which would give rise to different optimal control maps. Also, for a given cost function there may be one, or a whole family, or no optimal control maps. If we search for an optimal control map relative to a prescribed cost function, we need only search among control laws that satisfy all necessary conditions that all optimal control maps must satisfy. Those necessary conditions most used and investigated assume the control maps and controlled trajectories are fairly smooth. They are generalizations to function spaces of the multivariable calculus first derivative test for minima relative to constraints. This test is the Lagrange multiplier test; a well-behaved function can attain a minimum subject to constraints only if Lagrange multipliers exist. We need only to search for a minimal control map subject to constraints in a function space of control maps. Differentiation of functions of functions and computing Lagrange multipliers is the domain of the 300 year old calculus of variations, and is where functional calculus enters. For many complex systems there is not sufficient smoothness to ensure the applicability of the Lagrange test. Even if applicable, the problem of finding the multipliers reduces to solving a complicated system of ordinary differential equations, and is often intractable. Let us examine the alternative provided by existence proofs (sufficient conditions).

Sufficient Conditions for Optimal Controls. Hilbert, in a famous short paper at the turn of the twentieth century on the direct method of the calculus of variations (the granddaddy of the finite element method and weak solutions) opined that all calculus of variations problems have solutions, provided the notion of solution is properly defined. The editors of his collected works changed this to “all regular calculus of variations problems”, but that is not what he said, and I think he meant what he said. I view L. C. Young’s theory as the control theory realization of Hilbert’s remark. L. C. Young and E. J. McShane proved in 1937-40 that most variational problems asking for a curve minimizing a cost function had solutions, provided that the notion of solution was extended properly. Their early existence proof led L. C. Young in 1969 to introduce control maps whose values are probability measures on the usual values of control maps. These are the so-called *relaxed* control maps. His existence proof (sufficient condition) showed that relative to a wide variety of cost functions, generally speaking control problems have optimal relaxed control maps. The relaxed control maps induce controlled trajectories of states, but these controlled trajectories are often far from smooth. The Pontryagin theory alluded to above dealt with necessary conditions, not sufficient. The proof that Young optimal relaxed control maps exist is non-constructive due to using the Arzela-Ascoli lemma, which is a compactnss theorem for function spaces.

ϵ-Optimal Control. It is my belief that this non-constructivity was the reason that, as far as I know, though the Young theory has been available since 1969 in several books, no one before Kohn and I seriously contemplated approximately implementing the Young theory. From our point of view, approximating to Young's relaxed optimal controls is the point at which automata and hybrid systems enter. For us the saving grace is that, although the existence proof for optimal relaxed controls is non-constructive, we need only the constructive part of the proof, the part that verifies that the hypothesis of the Ascoli-Arzela lemma hold. We do not need the non-constructive compactness argument that concludes from the constructive part that the optimal control laws exist. The part that is constructive is defining control maps that yield controlled trajectories achieving within ϵ of minimum cost. These we call the ϵ-optimal controlled trajectories. They put us in the domain of ϵ-variational inequalities. The reason constructing ϵ-optimal controlled trajectories is enough for us is that engineers who design systems are never required to produce optimal control maps. Here is why. Suppose you want a control map to get to a goal. Usually, if the control map puts you close enough to the goal at an affordable cost, you are happy. If the incremental cost of designing a system getting much closer to the goal is high, and the return is low, why bother to get closer? Another way of putting it is that as positive number ϵ decreases, the cost of building an implementation that controls the process to achieve the goal within ϵ of the minimal cost increases. It is not the cost of computing a control map I am referring to, it is the cost of constructing physical devices precise enough to implement the control map in the real world, which increases as ϵ decreases. So the engineer has to ask the client: "how much are you willing to spend"? We build to accuracy ϵ if it is possible to do that within budget and the result is good enough to satisfy the client's needs. Thus the ϵ used is the result of a negotiation between client and system architect.

The non-constructive proof of the Arzela-Ascoli lemma for proving there exist optimal controls reflects itself in the following limitation: if you have computed an ϵ-optimal relaxed control and now want a (say) $\frac{\epsilon}{2}$-optimal relaxed control, you have to start over; you cannot get it by a uniform method of successive approximations, refining the previous control to get the next, more accurate, one. Successive approximation is useful only for fairly smooth problems. Most large practical problems are not smooth and are not convex.

Another reason that ϵ-optimal controls suffice is that true optimality is not an issue. System models are always imperfect; there are always unmodeled dynamics not incorporated into the model used. An optimal control map for a model is usally not perfectly optimal in the real world. The concentration has to be on extracting robust ϵ-optimal controlled trajectories, that is, which remain ϵ-optimal for small perturbations of initial conditions, constraints, goals. Our experience at the company Kohn and I founded is that for each specific real time control problem one has to refine the general algorithms for computing robust

ϵ -optimal trajectories in order to to obtain efficient real time algorithms. The tweaks needed are highly model and goal dependent, that is they depend on knowledge specific to the domain to be controlled. Building a good model can entail years of work on algorithms for one application. Physics and engineering in general have the same problem. General theory and algorithms are all very well, but many years may have to be spent on a single equation to understand and compute solutions. Think of the hundreds of years for special cases of the Navier-Stokes equation.

$\epsilon-$**Optimal Control Maps.** We use the fact that control maps with finite probability measures as values are dense, in the appropriate topology, in the space of relaxed controls. These are what we want to implement. How does one (approximately) implement a finite probability measure valued control acting from time t_0 to time $t_0 + \delta t$? If the finite probability measure assigns probability p_1 to control value c_1, ..., probability p_k to control value c_k, then divide up the time interval from t_0 to t_1 into k successive subintervals of lengths $p_1\delta t, ...,$ $p_k\delta t$, and use c_i as control value in the $i-$th interval of length $p_i\delta t$. This is a finite *chattering* control law for the time interval from t to $t + \delta t$. A general algorithm based on the constructive part of the Arzela-Ascoli lemma allows us to compute a δt and divide time into successive intervals of length δt and compute a finite chattering control to apply during each such interval, and to obtain an ϵ-optimal control that way.

Open Covers and Automata. In approximating to a continuous control map, whether by piecewise linear controls or piecewise constant controls or chattering controls, the question is: when do we change from using one such simple control map to using another as we move along a trajectory on the manifold? The abstract answer Kohn-Nerode gave is that we cover the control map, regarded as a closed set in the space of appropriate control maps, by a finite cover of small open sets correctly chosen. Then we convert the cover into the desired control automaton. This was introduced in Kohn-Nerode [13], Appendix II, and used in papers of Davoren and of Diaz [14]. This is possible for relaxed controls as well. We emphasize that extracting arbitrarily close finite control automata approximations to a continuous control map is possible independent of optimality. It depends only on assumptions of compactness and continuity. In concrete applications in which we want to extract $\epsilon-$optimal controls, we need better algorithms based on additional information which turns out to be of a differential geometric character.

Continualization. When presented with discrete constraints, such as Boolean constraints or PROLOG constraints, we convert them into continuous form, replacing a hybrid system by a continuous system. There are a variety of methods to continualize, see [14]. So we convert finding optimal controls for a hybrid system problem to a purely continuous problem on an appropriately constructed manifold determined by all constraints, discrete and continuous. Then we extract approximate optimal relaxed control by a finite control automaton, ending up with a hybrid system. From discrete to continuous and back again. There are many unsolved problems associated with choosing continualizations.

Finsler Geometry in Control. With appropriate assumptions, finding an optimal control map is finding a control map which minimizes total cost to the goal from present position. If we think of cost as length, the goal is a point, and we assume such controlled trajectories always exist and are unique, we can think of the optimal controlled paths as forming a geodesic field. Many systems have states which evolve according to ordinary differential equations which are a function of time. Raising time to be an independent variable, we find that we are in the domain of a Finsler geometry, and are dealing with Finsler geodesics. This makes the Finsler apparatus of tensor calculus, Cartan exterior differential forms, and Finsler connections apply. This is the main set of tools for obtaining additional relations which reduce the labor of computing optimal controls and their approximations. One can express optimal controlled trajectories as determined by either their geodesic fields or a connection, or a Cartan form. The method incorperates Pontryagin's necessary conditions as indirect descriptions of the geodesic field. When feasible, errors and tolerances are computed using the second variation. We conclude this paragraph with the following description. We are working with a Finsler connection; the notion of optimality recedes to the background. We choose a connection whose geodesic field we try to follow by always heading in the direction indicated by the connection at your current position. The connection is computed from the goal and the constraints. If, in following a controlled trajectory, unmodeled dynamics put you off course, you head in the direction of the geodesic through that point. If the goal point is altered or the constraints change midcourse, you follow the geodesic field of the connection associated with the new goal and new constraints. In practice you follow an $\epsilon-$optimal approximation implemented by a finite automaton controller. As a concluding remark, our Finsler models lead to new algorithms for linear, mixed integer-linear, and dynamic programming problems.

References

[1] Davoren, J.M., Nerode, A.: Logics for Hybrid Systems. Proceedings of the IEEE 88(7), 985–1010 (2000)

[2] Hybrid control systems. Antsaklis, P.J., Nerode, A. (eds.) IEEE Transactions Automatic Control 43, no. 4. Institute of Electrical and Electronics Engineers, Inc (1998)

[3] Hybrid Systems, Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) LNCS, vol. 736, Springer, Heidelberg (1993)

[4] Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.): Hybrid systems II. Papers from the Third Workshop held at Cornell University. LNCS, vol. 999, pp. 28–30. Springer, Heidelberg (1995)

[5] Alur, R., Sontag, E.D., Henzinger, T.A. (eds.): Hybrid Systems III. LNCS, vol. 1066. Springer, Heidelberg (1996)

[6] Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.): Hybrid Systems IV. LNCS, vol. 1273. Springer, Heidelberg (1997)

[7] Antsaklis, P., Kohn, W., Nerode, A., Lemmon, M., Sastry, S. (eds.): Hybrid Systems V. LNCS, vol. 1567. Springer, Heidelberg (1999)

[8] Kohn, W.: A Declarative Theory for Rational Controllers. Proceedings of the 27th IEEE Conference on Decision and Control 1, 131–136 (1988)

[9] Kohn,, Wolf,, Nerode,, Anil,, Remmel,, Jeffrey, B.: Hybrid systems as Finsler manifolds: finite state control as approximation to connections. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) Hybrid Systems II. LNCS, vol. 999, Springer, Heidelberg (1995)

[10] Kohn, W., Brayman, V., Cholewinski, P., Nerode, A.: Control in Hybrid Systems, International Journal of Hybrid Systems vol. 3 (2003)

[11] Kohn, W., Brayman, V., Nerode, A.: Control Synthesis in Hybrid Systems with Finsler Dynamics, Houston Journal of Mathematics vol. 28(2), pp. 353–375

[12] Nerode, A.: Modeling Intelligent Control, In: Proc. DARPA Workshop on Software Tools for the Domain Specific Software Initiative, Pacifica, Ca, July 17-19 (1990)

[13] Kohn, W., Nerode, A.: Parallel Computer Architectures. LNCS, vol. 732. Springer, Heidelberg (1993)

[14] Kohn, W., Nerode, A., Remmel, J.B.: Continualization: A Hybrid Systems Control Technique for Computing, In: Proceedings of CESA'96 IMACS Multiconference, vol 2. pp. 507–511

[15] Diaz, A.: Extraction of Finite State Controllers, AD and DA maps, and Associated Small Topologies for Measure Valued Control Laws International Journal of Hybrid Systems (September 2003)

Nash Stability in Additively Separable Hedonic Games Is NP-Hard[⋆]

Martin Olsen

Department of Computer Science
University of Aarhus
mo@daimi.au.dk

Abstract. Ballester has shown that the problem of deciding whether a Nash stable partition exists in a hedonic game with arbitrary preferences is NP-complete. In this paper we will prove that the problem remains NP-complete even when restricting to additively separable hedonic games.

Bogomolnaia and Jackson have shown that a Nash stable partition exists in every additively separable hedonic game with *symmetric* preferences. We show that *computing* Nash stable partitions is hard in games with symmetric preferences. To be more specific we show that the problem of deciding whether a *non trivial* Nash stable partition exists in an additively separable hedonic game with *non-negative* and *symmetric* preferences is NP-complete. The corresponding problem concerning individual stability is also NP-complete since individually stable partitions are Nash stable and vice versa in such games.

Keywords: Nash Stability, Hedonic Games, NP-Completeness.

1 Introduction

In a *Coalition Formation Game* a set of players splits up in coalitions so that each player belongs to exactly one coalition. Each player prefers certain partitions[1] of the players to other partitions. If all players are satisfied with the partition in some formalized sense - or not able to move - the partition is said to be *stable*. A stable partition is called an *equilibrium*. For an overview of the field of *Coalition Formation Games* we refer to the report [7] by Hajdukova.

A given notion of stability can have limitations in terms of computability. For some types of games it might be impossible to effectively compute equilibriums on a computing device under the assumption NP$\neq$P. If a real world system is modeled using Coalition Formation Games and equilibriums with such limitations you should not expect to be able to calculate the equilibriums using a computer if the model is large. It is also an interesting question whether a real system is able to find an equilibrium if a computer can not find it effectively.

[⋆] The research is partly sponsored by the Danish company Cofman (www.cofman.com).

[1] A partition of a set N is a collection of non empty disjoint subsets of N with union N.

This is the motivation for analyzing the computational complexity for a given notion of stability as also pointed out by Daskalakis and Papadimitriou in [4] and Chen and Rudra in [3]. This paper deals with proving limitations for the notion of *Nash stability* in *Additively Separable Hedonic Games.*

1.1 Additively Separable Hedonic Games

In a *hedonic game* we are given a set $N = \{1, 2, \ldots, |N|\}$ of *players*. Each player $i \in N$ has a reflexive, complete and transitive *preference relation* $\preceq_i$ on the set $N_i = \{S \subseteq N : i \in S\}$. In this way we know which *coalitions* player i prefers to be a member of. The game is *additively separable* if there exists a *utility function* $v_i : N \to \mathrm{I\!R}$ for each $i \in N$ such that

$$\forall S, T \in N_i : T \preceq_i S \Leftrightarrow \sum_{j \in T} v_i(j) \leq \sum_{j \in S} v_i(j) \ .$$

In an additively separable hedonic game the *payoff* $v_i(j)$ of player i for belonging to the same coalition as player j is independent of how other players are forming coalitions. Changing the value $v_i(i)$ has no effect on $\preceq_i$ so we assume $v_i(i) = 0$.

1.2 An Example

We would like to give an example of an additively separable hedonic game. We will use biological terminology metaphorically to ease the understanding for the game. The game does *not* represent a serious attempt to model a biological system.

Assume that there are two buffaloes b_1 and b_2 in an area with n waterholes w_1, w_2, ... , w_n. Each waterhole w_i has a capacity $c(w_i)$ specifying how much water a buffalo can drink from that hole per year. There are also two parasites p_1 and p_2 in the area. The only possible host for p_1 is b_1 and b_1 must drink a lot of water if p_1 is sitting on its back. The same goes for p_2 and b_2. Now assume that b_1 and b_2 are enemies and that a buffalo must drink water corresponding to half the total capacity C of the waterholes if it is the host of a parasite. This system can be viewed as an additively separable hedonic game depicted as a weighted directed graph in Fig. 1 where the weight of edge (i, j) is $v_i(j)$ - if there is no edge (i, j) then $v_i(j) = 0$. We have added two edges (b_1, b_2) and (b_2, b_1) with capacity $-C - 1$ to model that b_1 and b_2 are enemies. Please note that the waterholes are also players in the game. The waterholes do not care to which coalitions they belong.

1.3 Stability

In this paper we will focus on one type of stability: *Nash stability.* We refer to [7] for more information on other types of stability.

A partition $\Pi = \{S_1, S_2, \ldots, S_K\}$ of N is *Nash stable* if it is impossible to find a player who would be strictly better of if the player left his coalition and

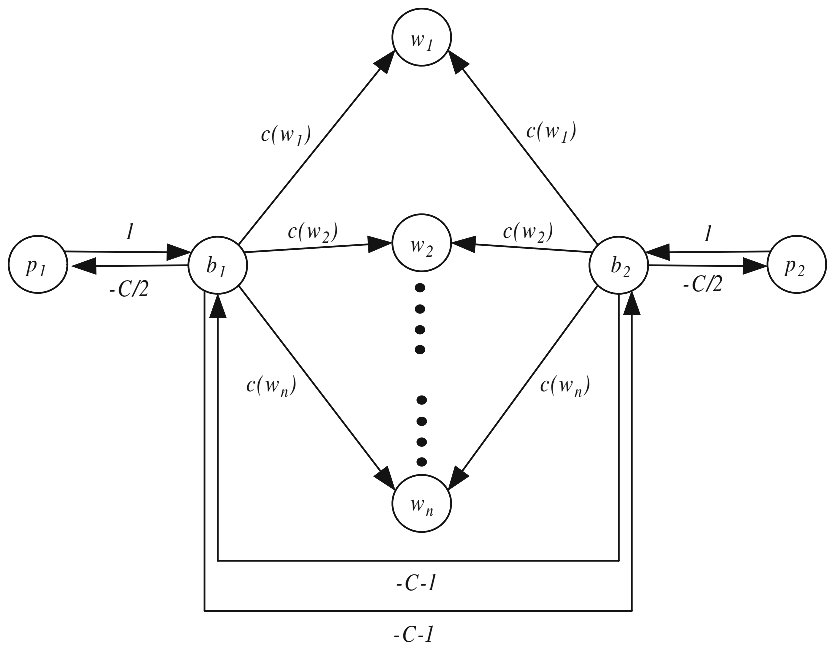


Fig. 1. An additively separable hedonic game

joined one of the other coalitions in the partition. If $S_{\Pi}(i)$ denotes the set in the partition Π such that $i \in S_{\Pi}(i)$ then Π is Nash stable if and only if

$$\forall i \in N, \forall S_k \in \Pi \cup \{\emptyset\} : S_k \cup \{i\} \preceq_i S_{\Pi}(i) \ .$$

Now consider the game in Fig. 1. A partition of the players is not Nash stable if b_1 is not the host of p_1 - in this case p_1 would be strictly better off by joining $S_{\Pi}(b_1)$. This fact can be expressed more formally: $S_{\Pi}(b_1) \cup \{p_1\} \succ_{p_1} S_{\Pi}(p_1)$ if $S_{\Pi}(p_1) \neq S_{\Pi}(b_1)$. As an exercise the reader is invited to figure out necessary and sufficient conditions for the existence of a Nash stable partition of this game. We will return to this problem in Sect. 2.

1.4 Related Work

Ballester has shown in [1] that the problem of deciding whether a Nash stable partition exists in a hedonic game with *arbitrary* preferences is NP-complete.

On the other hand Bogomolnaia and Jackson show in [8] that a Nash stable partition exists in every additively separable hedonic game with *symmetric* preferences. The preferences are symmetric if $\forall i, j \in N : v_i(j) = v_j(i)$. If v_{ij} is the common value for $v_i(j)$ and $v_j(i)$ in a symmetric game then Bogomolnaia and Jackson show that any partition Π maximizing $f(\Pi) = \sum_{S \in \Pi} \sum_{i,j \in S} v_{ij}$ is Nash stable.

Flake et al. work with so called *communities* in [5] where the objective is to divide a network into clusters. The web graph and the CiteSeer graph are examples of networks that are processed in [5]. Using the terminology from coalition formation games a *community* is a subset of players $C \subseteq N$ in an additively separable game with symmetric preferences such that $\forall i \in C : \sum_{j \in C} v_{ij} \geq \sum_{j \in N-C} v_{ij}$. In other words each player in C gets at least half the total possible payoff by belonging to C. Flake et al. show that the problem of deciding whether it is possible to partition N into k communities is NP-complete. Such a partition is Nash stable but a Nash stable partition is not necessarily a partition into communities. The proof techniques used in this paper are similar to those used in [5].

Burani and Zwicker introduces the concept of *descending separable* preferences in [2]. Burani and Zwicker show that descending separable preferences guarantees the existence of a Nash stable partition. They also show that descending separable preferences do not imply and are not implied by additively separable preferences.

1.5 Our Results

In Sect. 2 we restrict our attention to additively separable hedonic games compared to Ballester [1]. Compared to Bogomolnaia and Jackson [8], we also allow asymmetric preferences. Informally we show that things are complicated even when looking at additively separable hedonic games. With an intuitively clear proof based on the example in Sect. 1.2 we show that the problem of deciding whether a Nash stable partition exists in a hedonic game remains NP-complete when restricting to additively separable preferences.

In Sect. 3 we show that *computing* Nash stable and individually stable partitions is hard in games with symmetric preferences. To be more specific we show that the problem of deciding whether a *non trivial* Nash stable partition exists in an additively separable hedonic game with *non-negative* and *symmetric* preferences is NP-complete. This result also applies to individually stable partitions since individually stable partitions are Nash stable and vice versa in such games.

2 Restricting to Additively Separable Games

We will now formally define the problem of deciding whether a Nash stable partition exists in an additively separable hedonic game:

Definition 1. *The ASHNASH problem:*

- *Instance: A set $N = \{1, 2, \ldots, n\}$ and a function $v_i : N \to \mathbb{R}$ such that $v_i(i) = 0$ for each $i \in N$.*
- *Question: Does a partition Π of N exist such that*

$$\forall i \in N, \forall S_k \in \Pi \cup \{\emptyset\} : \sum_{j \in S_{\Pi}(i)} v_i(j) \geq \sum_{j \in S_k \cup \{i\}} v_i(j) \ . \qquad (1)$$

We are now in a position to prove that this problem is intractable.

Theorem 1. *ASHNASH is NP-complete.*

Proof. It is easy to check in polynomial time that Π is a partition satisfying (1) thus ASHNASH is in NP.

We will transform an instance of the NP-complete problem PARTITION [6] into an instance of ASHNASH in polynomial time such that the answers to the questions posed in the two instances are identical - if such a transformation exists we will write PARTITION $\propto$ ASHNASH following the notation in [6]. This means that we can solve the NP-complete problem PARTITION in polynomial time if we can solve ASHNASH in polynomial time thus ASHNASH is NP-complete since it is a member of NP. The rest of the proof explains the details of the transformation.

An instance[2] of PARTITION is a finite set $W = \{w_1, w_2, \ldots, w_n\}$ and a capacity $c(w) \in Z^+$ for each $w \in W$. The question is whether a subset $W' \subset W$ exists such that $\sum_{w \in W'} c(w) = \frac{C}{2}$ where $C = \sum_{w \in W} c(w)$.

Now suppose we are given an instance of PARTITION. The PARTITION instance is easily transformed into the buffalo-parasite-game from Sect. 1.2 in polynomial time. All we have to do to translate this as an ASHNASH instance is to perform a simple numbering of the players in the game.

Now we only have to show that a Nash stable partition of the game in Fig. 1 exists if and only if W' exists. This can be seen from the following argument:

– The partition $\Pi = \{\{b_1, p_1\} \cup W', \{b_2, p_2\} \cup W - W'\}$ is Nash stable if W' exists.
– Now assume that a Nash stable partition Π exists and define $W_1 = S_\Pi(b_1) \cap W$ and $W_2 = S_\Pi(b_2) \cap W$. The set $S_\Pi(b_1)$ must contain p_1. Due to the stability we can conclude that $\sum_{w \in W_1} c(w) \geq \frac{C}{2}$ - otherwise b_1 would be better off by its own. By a symmetric argument we have $\sum_{w \in W_2} c(w) \geq \frac{C}{2}$. The two nodes b_1 and b_2 are not in the same coalition so the two sets W_1 and W_2 are disjoint, so we can conclude that $\sum_{w \in W_1} c(w) = \sum_{w \in W_2} c(w) = \frac{C}{2}$. We can take $W' = W_1$. □

3 Non-negative and Symmetric Preferences

In this section we will restrict our attention to additively separable games with *non-negative* and *symmetric* preferences. The trivial partition where all players cooperate is optimal for all players in such games. If all players form a clique where the payoffs are identical then the trivial partition is the only Nash stable partition.

Now let us on the other hand assume that two disjoint communities S and T of players exist as defined in Sect. 1.4. If we collapse these communities to two players s and t then we can effectively calculate the s-t minimum cut in the underlying graph for the game. This cut defines a non trivial Nash stable

[2] The objects constituting an instance in [6] are renamed to match the example in Sect. 1.2.

partition. We will denote a non trivial Nash stable partition as an *inefficient equilibrium*. In this section we will prove that inefficient equilibriums generally are hard to compute. To be more specific we will prove that the problem of deciding whether they exist is NP-complete.

As in the proof of Theorem 1 we need a known NP-complete problem in the proof of the theorem of this section. The "base" problem of the proof in this section is the following problem:

Definition 2. *EQUAL CARDINALITY PARTITION*

– *Instance: A finite set $W = \{w_1, w_2, \dots, w_n\}$ and a capacity $c(w) \in Z^+$ for each $w \in W$*
– *Question: Does a non trivial partition of W exist such that $|W_i| = |W_j|$ and $\sum_{w \in W_i} c(w) = \sum_{w \in W_j} c(w)$ for all sets W_i and W_j in the partition?*

EQUAL CARDINALITY PARTITION is closely related to the balanced version of PARTITION where we are looking for a set $W' \subset W$ such that $\sum_{w \in W'} c(w) = \frac{C}{2}$ and $|W'| = \frac{|W|}{2}$. The balanced version of PARTITION is known to be NP-complete [6]. An instance of the balanced version of PARTITION is transformed into an equivalent instance of EQUAL CARDINALITY PARTITION by adding two more elements to the set W - both with capacity $C + 1$. This shows that EQUAL CARDINALITY PARTITION is NP-complete since it is easily seen to belong to NP.

We will now formally define the problem of deciding whether a non trivial Nash stable partition exists in an additively separable hedonic game with non-negative and symmetric preferences:

Definition 3. *The INEFFICIENT EQUILIBRIUM problem:*

– *Instance: A set $N = \{1, 2, \dots, n\}$ and a function $v_i : N \to \mathbb{R}^+ \cup \{0\}$ such that $v_i(i) = 0$ for each $i \in N$ and $v_i(j) = v_j(i)$ for each $i, j \in N$.*
– *Question: Does a non trivial partition Π of N exist such that*

$$\forall i \in N, \forall S_k \in \Pi \cup \{\emptyset\} : \sum_{j \in S_\Pi(i)} v_i(j) \geq \sum_{j \in S_k \cup \{i\}} v_i(j) \ .$$

Theorem 2. *INEFFICIENT EQUILIBRIUM is NP-complete.*

Proof. We will show that EQUAL CARDINALITY PARTITION $\propto$ INEFFICIENT EQUILIBRIUM. By the same line of reasoning as in the proof of Theorem 1 we conclude that INEFFICIENT EQUILIBRIUM is NP-complete since INEFFICIENT EQUILIBRIUM is easily seen to belong to NP.

We will now show how to transform an instance of EQUAL CARDINALITY PARTITION into an equivalent instance of INEFFICIENT EQUILIBRIUM. All the members of W are players in the instance of INEFFICIENT EQUILIBRIUM and the payoff for w_i and w_j for cooperating is $c(w_i) + c(w_j) + C$. For each player w_i we also add a player z_i. Player z_i only gets a strictly positive payoff by cooperating with w_i - in this case the payoff is $2c(w_i) + C$. Figure 2 depicts a

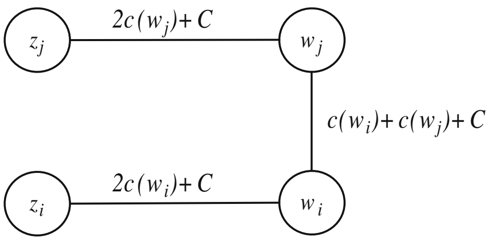


Fig. 2. A part of a game with positive and symmetric preferences

part of the INEFFICIENT EQUILIBRIUM instance as an undirected weighted graph. The members of W are fully connected but z_i is only connected to w_i in the graph.

We will now prove that the two instances are equivalent:

– Suppose that we have a non trivial Nash stable partition Π of the players in Fig. 2. For $S_k \in \Pi$ we define $W_k = S_k \cap W$. The player z_i cooperates with w_i - otherwise Π would not be stable. The total payoff of $w_i \in W_k$ is $|W_k|(C + c(w_i)) + \sum_{w \in W_k} c(w)$.
  - $|W_i| = |W_j|$: If $|W_i| < |W_j|$ then all the players in W_i would be strictly better off by joining W_j. This contradicts that Π is stable.
  - $\sum_{w \in W_i} c(w) = \sum_{w \in W_j} c(w)$: Now assume $\sum_{w \in W_i} c(w) < \sum_{w \in W_j} c(w)$. Once again the players in W_i would be strictly better off by joining W_j since $|W_i| = |W_j|$. Yet another contradiction.

– Suppose that we have a non trivial partition of W into sets with equal cardinality and capacity. For a set W_i in this partition let S_i be the union of W_i and the corresponding z-members. The set of S_i's is easily seen to be a non trivial Nash stable partition of the game in Fig. 2. □

4 Conclusion

We have shown that the problem of deciding whether a Nash stable partition exists in an additively separable hedonic game is NP-complete. For additively separable games with non-negative and symmetric preferences we have shown that the problem of deciding whether a non trivial Nash stable partition exists is NP-complete.

Acknowledgments. The author of this paper would like to thank Peter Bro Miltersen from University of Aarhus for valuable comments and constructive criticism.

References

1. Ballester, C.: NP-completeness in Hedonic Games. Games and Economic Behavior 49(1), 1–30 (2004)
2. Burani, N., Zwicker, W.S.: Coalition formation games with separable preferences. Mathematical Social Sciences 45(1), 27–52 (2003)

3. Chen, N., Rudra, A.: Walrasian equilibrium: Hardness, approximations and tractable instances. In: WINE, pp. 141–150 (2005)
4. Daskalakis, K., Papadimitriou, C.H.: The complexity of games on highly regular graphs. In: ESA, pp. 71–82 (2005)
5. Flake, G., Tarjan, R., Tsioutsiouliklis, K.: Graph clustering and minimum cut trees. Internet Mathematics 1(4), 385–408 (2004)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
7. Hajdukova, J.: On Coalition Formation Games. Technical report, Institute of Mathematics, P.J. Safarik University (2004)
8. Jackson, M.O., Bogomolnaia, A.: The Stability of Hedonic Coalition Structures. Games and Economic Behavior 38(2), 201–230 (2002)

Comparing Notions of Computational Entropy

Alexandre Pinto[⋆]

DCC-FC & LIACC
R. Campo Alegre 1021/1055
4169-007 Porto
alx@dcc.fc.up.pt

Abstract. In the information theoretic world, entropy is both the measure of randomness in a source and a lower bound for the compression achievable for that source by any encoding scheme. But when we must restrict ourselves to efficient schemes, entropy no longer captures these notions well. For example, there are distributions with very low entropy that nonetheless look random for polynomial-bound algorithms.

Different notions of computational entropy have been proposed to take the role of entropy in such settings. Results in [GS91] and [Wee04]) suggest that when time bounds are introduced, the entropy of a distribution no longer coincides with the most effective compression for that source.

This paper analyses three measures that try to capture the compressibility of a source, establishing relations and separations between them and analysing the two special cases of the uniform and the universal distribution $\mathbf{m}^t$ over binary strings of a fixed size. It is shown that for the uniform distribution the three measures are equivalent and that for $\mathbf{m}^t$ there is a clear separation between metric type entropy, and thus pseudo-entropy, and the maximum compressibility of a source.

Keywords: Computational Entropy, Compressibility, Kolmogorov Complexity.

1 Introduction

Randomness is an essential concept in computer science. It is fundamental in cryptography and has been used extensively to provide quick algorithms for otherwise difficult problems. Unfortunately, it is beyond the abilities of classical computers to produce true random bits. For this reason, it is necessary to simulate true randomness by deterministic methods, but the distributions thus generated are only useful if they are 'random enough', or pseudo-random. The definition of pseudo-randomness is based on computational indistinguishability from a truly random distribution.

[⋆] Partially supported by KCrypt (POSC/EIA/60819/2004), the grant SFRH/BD/13124/2003 from FCT and funds granted to LIACC through the Programa de Financiamento Plurianual, FCT and Programa POSI.

The randomness of a distribution is measured by the notion of entropy, introduced by Shannon in [Sh48]. A related notion is that of min-entropy, which is a lower bound on the entropy of some distribution.

It is natural to try to extend these objective measures to the pseudo-random case, thus defining notions of computational entropy. The first such definition is due to Yao ([Yao88]), but the most used one is due to Hastad, Impagliazzo, Levin and Luby ([HILL99]). Barak, Shaltiel and Wigderson provide definitions for computational analogues of min-entropy in [BSW03].

In Information Theory, randomness is tighly related to compression, in the sense that the output of a random distribution can not be noticeably compressed. The ultimate notion of compression is Kolmogorov complexity, introduced by Solomonoff, Kolmogorov and Chaitin [Sol64, Kol65, Cha66]. For an in-depth analysis of this notion, see [LV97].

Kolmogorov defended this measure characterized the intrinsic randomness of a string, independently of the distribution from which it was sampled. One of the major results in the theory of Kolmogorov complexity is precisely the tight relation that exists between this and entropy, namely the fact that the latter is assymptotically equal to the average value of the former. This relation has been studied in [GV03] and suggests that a similar one might exist between resource-bounded Kolmogorov complexity and some notion of computational entropy.

Yao's definition of effective entropy has not been much studied. Goldberg and Sipser ([GS91]) analyse languages that can be compressed efficiently by a probabilistic machine, which is the computation model considered by Yao, but do not mention any notion of computational entropy. The notion of pseudo-entropy appears in [HILL99]. In [BSW03], the authors introduce an analogue of min-entropy that closely resembles that in [HILL99]. For this reason, the authors call it HILL-type entropy. They introduce a similar notion, metric entropy, and present a measure based on a compression idea which they called Yao-type pseudo-entropy. They show that for some models of computation, metric entropy is equivalent to HILL-type entropy.

Hoeteck Wee studies compressibility, improving on a result from [GS91] that separated compressibility from pseudo-entropy in an oracle setting and giving a separation between BSW's metric-entropy and Yao-type entropy. Hoeteck says that this result suggests pseudo-entropy is not the right lower bound for the size of the compression on samplable sources.

The present paper focuses on these measures, comparing Yao-type entropy, effective entropy and the expected value of the time-bounded Kolmogorov complexity. Effective entropy is by definition the lower bound for compression of a samplable source and because time-bounded Kolmogorov complexity is the lower bound for compression of a single word in a fixed time, it seems reasonable that its expected value be related to effective entropy.

We show that all three are equivalent for the uniform distribution. We also show that for the distribution $\mathbf{m}^t$, Yao-type entropy and effective entropy have very different values. This result depends on a standard complexity assumption and it can be applied to $H_\epsilon^{\text{Metric}}$, thus giving a clear separation between pseudo-entropy (by a connection of $H_\epsilon^{\text{Metric}}$ to H_ϵ^{HILL} given in [BSW03]) and effective entropy. We note that if we remove the randomness in the encoder then the result is unconditional.

Our results are as follows: in Sec. 3, we show that effective entropy is at least as large as Yao-type entropy when considering efficiency to be represented either by **FP** or the equivalent class for **BPP**. In Sec. 4, it is shown that the effective entropy is at least as large as the average of the resource-bounded Kolmogorov complexity. Sec. 5 shows that the latter is not necessarily larger than Yao-type entropy, although it is always larger than a variant thereof. Finally, Sec. 6 analyses two special distributions, uniform and $\mathbf{m}^t$, and shows that for the uniform distribution, the three notions are equivalent. It is also shown that for $\mathbf{m}^t$ there is a difference between the values of metric and Yao type entropy of [BSW03] on the one hand and that of the expected Kolmogorov complexity, and under a reasonable complexity assumption, Yao's effective entropy of [Yao88], on the other.

Some proofs were omitted in the main text and presented in the appendix at the end of the paper.

2 Preliminaries

Throughout this paper, all logs are taken base 2. Throughout the paper, for some random variable, we use $x \in X$ as a shorthand to "x is in the support of X".

We use the same notations used in the original papers: H_c for Yao's effective entropy, $H_\epsilon^{\text{Metric}}$ and H_ϵ^{Yao} for BSW's metric and Yao type entropies respectively. Sometimes we use the notation $\Pr[X \in D]$ where X is a random variable and D is a set. This signifies the probability that a random value of X belongs to D, and is shorthand to $\sum_{x \in X} \Pr[x \in D]$.

We use the notation $O(f(n))$ in several places when computing the complexity of a given string, say x, according to the standard practice in the theory of Kolmogorov Complexity. This term represents a quantity not greater than $c \cdot f(n)$ for some constant c. This constant is always considered to be positive, since the term usually represents the size of some part of a program for x. For this reason, this quantity is usually signed.

The whole theory of Kolmogorov Complexity is based on this kind of approximations. This is a consequence of its cornerstone theorem of invariance, which says the universal Turing machine chosen as reference affects $K(x)$ only in a constant term independent of x. Since the theory is generally concerned with assymptotic results when n is big enough, this constant is represented by $O(1)$ even though it may be big.

We now give the basic definitions of computational entropy used.

2.1 Yao's Effective Entropy

Definition 1. *Consider a fixed finite alphabet Σ and some language $L = \Sigma^+$. A source S is a random variable defined over L with probability distribution p subject to the restriction that $\sum_{x \in S} p(x)|x| < \infty$.*

A source ensemble S is a sequence of sources $S_1, S_2, \ldots$ with probability distributions $p_1, p_2, \ldots$ such that for some fixed t every element $y \in [S_n]$ has $|y| < n^t$. □

To simplify the notation in the rest of the paper, we define a constant $\lambda = n^k$ when n and k are understood from the context.

Definition 2. *Let $S_n^\lambda = \underbrace{S_n \times \cdots \times S_n}_{\lambda\,\text{times}}$ be the random variable over $\lambda = n^k$ independent draws of S_n.*

Definition 3. *A $(t, k)-$encoding scheme for S is a triple of probabilistic polynomial algorithms $M = (M_A, M_B, M_C)$ such that:*

- *M_A receives as input a parameter n and a "text" composed of λ words of L. These words are independently identically sampled from S_n. Then, $x = (w_1, w_2, \ldots, w_\lambda)$ has probability of occurring equal to $p_n^k(x) = p_n(w_1) \cdot p_n(w_2) \cdot \ldots \cdot p_n(w_\lambda)$, i.e., x is sampled from S_n^λ.*
- *Let $M_A(n, x) = y$. Let $M_B(n, y) = u$. Then, the probability that $u \neq x$ must be smaller than $1/n^t$ for sufficiently large n.*
- *Let $b > 0$ be any fixed constant, and let $u = O(n^b)$. Pick $x_1, x_2, \ldots, x_u$ where each x_i is independently distributed from S_n^λ with probability $p_n^k(x)$. Then, let $M_A(n, x_i) = z_i$ and $z = z_1 z_2 \ldots z_u$ be the concatenation of the u outputs of M_A. Evaluate $M_C(n, z) = a$. Then, for sufficiently large n, it must happen that $a = (z_1, z_2, \ldots, z_u)$ with probability at least $1 - 1/n^t$.*

The last property states that to code a whole text we can encode different blocks of it and concatenate the results. If the number of blocks is polynomial in n, then M_C will be able to separate the blocks such that each of them can be correctly decoded. Roughly, the code is uniquely decipherable. □

Definition 4. *Let $L_n(M; S)$ represent the average number of bits per symbol of S that encoding $M = (M_A, M_B, M_C)$ achieves. Since all the algorithms are probabilistic, for a fixed input the algorithm M_A may return different outputs. Let then $|M_A(n, x)|$ represent the expected length of $M_A(n, x)$ over all possible random values that M_A uses internally. Then,*

$$L_n(M; S) = \frac{\sum_{x \in S_n^\lambda} p_n^k(x) |M_A(n, x)|}{\lambda}$$ □

Definition 5. *A $(t, k)-$entropy-sequence for S is a sequence $s_1, s_2, \ldots$ such that there exists a $(t, k)-$encoding scheme M for S with $L_n(M; S) = s_n$.* □

Definition 6 (Effective Entropy). *We say that a source S has effective entropy $H_c(S; n) \leq g(n)$ if there exist t and k and a $(t, k)-$entropy sequence such that $s_n \leq g(n)$ for every sufficiently large n.*

Similarly, $H_c(S; n) \geq g(n)$ if for every pair (t, k) and every $(t, k)-$entropy sequence $\langle s_n \rangle$ satisfies $s_n \geq g(n)$. □

This definition is very general, since it not only allows the encoder and the decoder to be probabilistic but it also considers the effect of patterns occurring in a longer text that would not be detected if the compression acted on each word individually. For this to be feasible, the text must have at most a polynomial number of words.

This makes H_c hard to compare with the other notions of computational entropy, which are concerned with individual words. As such, $H_c(S;n)$ will be compared with $H_\epsilon^{\text{Yao}}(S_n^\lambda)$ and $E(K^t(S_n^\lambda))$. All these functions are based on inputs of length $n \cdot \lambda$, but H_c is scaled down by dividing by λ. We feel that to properly compare these measures, we should consider the unscaled version of H_c instead, that is, $\lambda \cdot H_c(S;n)$. That's what is done in the statemente of the theorems in the following sections.

2.2 BSW's Yao-Type Entropy

Definition 7. *Let $\mathcal{C}$ be a class of efficiently computable functions, for example,* **FP**. *For any two functions $(c,d) \in \mathcal{C}$ defined as $c : S \to \{0,1\}^\ell$ and $d : \{0,1\}^\ell \to S$, let their codeset be $D = \{x : d(c(x)) = x\}$. Denote by $\mathcal{C}(\ell)$ the class of all such functions and by $\mathcal{D}(\ell)$ the class of all such D.*

A random variable S has computational Yao-like entropy at least k, written $H_\epsilon^{\text{Yao}}(S) \geq k$, if for all $\ell < k$ and all $D \in \mathcal{D}(\ell)$ it happens that $\Pr[S \in D] \leq 2^{\ell-k} + \epsilon$. □

3 Relations Between H_c and H_ϵ^{Yao}

Given that these two definitions are analogues of entropy and min-entropy, it is interesting to find if the order relation between them still holds in the computational setting. That is, is it true that $H_\epsilon^{\text{Yao}} \leq H_c$?

This section investigates the answer to this question.

We begin by giving a deterministic analog of H_c.

Definition 8. *$H_c^d(S;n)$ is defined as $H_c(S;n)$, with the difference that M_A, M_B are not allowed to fail. That is, they are deterministic.* □

It is clear that if a deterministic encoding scheme achieves entropy $g(n)$ for some source S, then adding randomness to this scheme can only improve the compression. Therefore,

$$H_c(S;n) \leq H_c^d(S;n)$$

The following theorem will be needed in the sequel:

Theorem 1. *Given any language L over $\{0,1\}^{<n}$, it is possible to build another language L' isomorphic to this such that all words in L' have length n. Furthermore, the isomorphism can be computed and reverted in linear time.*

Proof. For any string $x \in L$, define $f\{0,1\}^{<n} \mapsto \{0,1\}^n : f(x) = 0^{n-|x|-1}1x$. It is easy to check that this construction can be made and reverted quickly.

Now, it is necessary to show that all words thus formed are distinct. For any two words $x_1, x_2 \in L$ of distinct sizes, $f(x_1)$ and $f(x_2)$ have the leftmost 1 at different positions, so they are different. For any two distinct strings x_1, x_2 of the same size, the prefix appended to both strings is equal, so $f(x_1) \neq f(x_2)$ because $x_1 \neq x_2$. □

Theorem 2. *Consider Definition 7. Considering that efficient functions are those that can be computed in polynomial time, $\mathcal{C} = \mathbf{FP}$. Then, for any integer $i > 0$, $H_\epsilon^{\text{Yao}}(S_n^\lambda) \geq \lambda \cdot g(n) + (i+1) \Rightarrow H_c^d(S;n) \geq \left(1 - \frac{1}{2^i}\right) g(n)$. Equivalently, $\lambda \cdot H_c^d(S;n) \geq \left(1 - \frac{1}{2^i}\right)\left(H_\epsilon^{\text{Yao}}(S_n^\lambda) - (i+1)\right)$.*

The case $i = 0$ can be solved with a different technique.

Theorem 3. *Consider Definition 7 and let $\mathcal{C} = \mathbf{FP}$. Then, $H_\epsilon^{\text{Yao}}(S_n^\lambda) \geq \lambda \cdot g(n) + 1 \Rightarrow H_c^d(S;n) \geq g(n) - 1/\lambda$ or equivalently $\lambda \cdot H_c^d(S;n) \geq H_\epsilon^{\text{Yao}}(S_n^\lambda) - 2$.*

The previous results concern the deterministic version of H_c. That is due to considering $\mathcal{C} = \mathbf{FP}$. It is commonly accepted today that the class of probabilistic polynomial algorithms may be a better representation of real-world efficiency. Letting $\mathcal{C}$ equal to that class, the previous results all hold for H_c using essentially the same proofs.

Theorem 4. *Consider the class $\mathcal{C}$ referred in Definition 7 and let $\mathcal{C}$ be the class of probabilistic polynomial algorithms. Then, for any integer $i > 0$, $H_\epsilon^{\text{Yao}}(S_n^\lambda) \geq \lambda \cdot g(n) + (i+1) \Rightarrow H_c(S;n) \geq \left(1 - \frac{1}{2^i}\right) g(n)$, which is equivalent to $\lambda \cdot H_c(S;n) \geq \left(1 - \frac{1}{2^i}\right)\left(H_\epsilon^{\text{Yao}}(S_n^\lambda) - (i+1)\right)$.*

Theorem 5. *Consider the class $\mathcal{C}$ referred in Definition 7 and let $\mathcal{C}$ be the class of probabilistic polynomial algorithms. Then, $H_\epsilon^{\text{Yao}}(S_n^\lambda) \geq \lambda \cdot g + 1 \Rightarrow H_c(S;n) \geq g - 1/\lambda$ or equivalently $\lambda \cdot H_c(S;n) \geq H_\epsilon^{\text{Yao}}(S_n^\lambda) - 2$.*

These theorems establish the expected relation, that up to a multiplicative constant, the effective entropy is at least as great as Yao-type entropy.

4 Relations Between H_c and K^t

By definition, Yao's effective entropy identifies the probabilistic polynomial encoding scheme that achieves the best compression. This section shows that despite the randomness available to the encoding schemes, the average time-bounded Kolmorogov complexity is still a lower bound for H_c.

Since K^t is defined for deterministic Turing Machines, this section first compares K^t with the deterministic version of H_c. Then it addresses the general case.

Theorem 6. $\lambda \cdot H_c^d(S;n) \geq E(K^p(S_n^\lambda)) - O(1)$

Proof. Let $H_c(S;n) = g(n)$ and $M = (M_A, M_B, M_C)$ be any $(t,k)-$encoding scheme for S that achieves $L_n(M;S) = g(n)$. Then every $x \in S$ can be computed by M_B and a codeword output by M_A. Furthermore, these algorithms all run in time polynomial on n, say $p(n)$. Since M_B is fixed, its size is a constant. Then, for all $x \in S_n^\lambda$, $K^p(x) \leq |M_A(n,x)| + K^p(M_B) = |M_A(n,x)| + O(1)$.

But then,

$$\begin{aligned} E(K^p(S_n^\lambda)) &= \sum_{x \in S_n^\lambda} p_n^k(x) K^p(x) \leq \sum_{x \in S_n^\lambda} p_n^k(x) |M_A(n,x)| + O(1) \\ &= \lambda \cdot L_n(M;S) + O(1) = \lambda \cdot g(n) + O(1) \end{aligned}$$

Since for all n such an M exists, the theorem follows. □

To allow for probabilistic machines, we consider a program for x that has access to a pseudo-random generator (PRG) and to a random permutation of size $O(\log n)$. With these auxiliary constructions, we can show the following theorem.

Theorem 7. *If cryptographically-secure pseudo-random generators exist, then* $\lambda \cdot H_c(S;n) \geq E(K^p(S_n^\lambda)) - O(\log n)$.

5 Relations Between H_ϵ^{Yao} and K^t

The objective of this section is to analyse whether a meaningful relationship can be found between Yao-type entropy and the average value of $K^t(X)$. To begin, we give a relaxation of Definition 7 that can be compared to $E(K^t(X))$.

Definition 9. *Let $\mathcal{C}$ be as in Definition 7. Let c, d be a pair of computable functions as before, but only d is required to belong to $\mathcal{C}$. Let $\mathcal{D}^+$ be the corresponding class of codesets.*

A random variable X has inefficient computational Yao-type entropy at least k, written $H_\epsilon^{\text{Yao}+}(X) \geq k$, if $\Pr[X \in D] \leq 2^{\ell - k} + \epsilon$ for all $\ell < k$ and all $D \in \mathcal{D}^+(\ell)$. □

This definition includes at least all the functions allowed by the definition of $H_\epsilon^{\text{Yao}}(X)$. Thus if a certain property holds for all the functions allowed by Definition 9, then it also holds for all the functions allowed by Definition 7. Hence the following lemma:

Lemma 1. *For any random variable X, the following holds:*

$$H_\epsilon^{\text{Yao}+}(X) \leq H_\epsilon^{\text{Yao}}(X).$$

The relation between $H_\epsilon^{\text{Yao}+}$ and $E(K^t(X))$ can be shown by a theorem that uses the same technique of Theorem 2, and so the proof is omitted.

Theorem 8. *Consider the class $\mathcal{C}$ referred in Definition 9 and let $\mathcal{C} = $ **FP**. Then, for any integer $i > 0$, $H_\epsilon^{\text{Yao}+}(X) \geq k + i \Rightarrow E(K^t(X)) \geq \left(1 - \frac{1}{2^i}\right) k$.*

If the inverse implication were true, then we would be able to relate $E(K^t(X))$ to $H_\epsilon^{\text{Yao}}(X)$. Unfortunately, that is not the case.

Example 1. Let $P(x)$ be the probability function associated with a random distribution X defined as this:

$$\begin{aligned}
&\Pr[K^t(x) = 1] = 5/2^k \\
&\Pr[K^t(x) \in \{0, 2, \dots, k-1\}] = 0 \\
&\Pr[K^t(x) \geq k] = 1 - 5/2^k
\end{aligned}$$

for some $k \geq 5$. For this distribution, $H_\epsilon^{\mathrm{Yao}+}(X) \leq k - 1$. However, $E(K^t(X)) \geq k \cdot (1 - 5/2^k) + 5/2^k \geq k - 5k/2^k$, which, for $k \geq 5$, is greater than $k - 1$. □

The results in the previous sections can be summed up in the following inequalities, up to the approximations shown in the Theorems:

$$\begin{aligned}
H_\epsilon^{\mathrm{Yao}+}(X^\lambda) &\leq H_\epsilon^{\mathrm{Yao}}(X^\lambda) \leq \lambda \cdot H_c(X) \\
H_\epsilon^{\mathrm{Yao}+}(X^\lambda) &\leq E(K^t(X^\lambda)) \leq \lambda \cdot H_c(X)
\end{aligned}$$

It remains an open question to find the relation between $E(K^t(X))$ and $H_\epsilon^{\mathrm{Yao}}(X)$. As shown later, for the universal distribution $\mathbf{m}^t$, $E(K^t(X))$ is markedly superior to $H_\epsilon^{\mathrm{Yao}}(X)$, but there can be other distributions for which the relation is inverted.

6 Relations for Specific Distributions

This section analyses the uniform and the $\mathbf{m}^t$ distributions. Throughout this section, U_n denotes the uniform distribution over binary strings of length n.

6.1 Uniform Distribution

Theorem 9. *$H_\epsilon^{\mathrm{Yao}}(U_n) = n$.*

Proof. First we show that $H_\epsilon^{\mathrm{Yao}}(U_n) \geq n$. In fact, for every pair of functions $c, d \in \mathcal{C}(\ell)$ the set $D = \{x : c(d(x)) = x\}$ has at most 2^ℓ different words. Since all of them have probability equal to 2^{-n}, we get $\Pr[X \in D] = |D|/2^n \leq 2^\ell/2^n$. Then by definition, $H_\epsilon^{\mathrm{Yao}}(U_n) \geq n$.

To prove the converse, we have to show that there is some $\ell < n + 1$ and a pair of functions $c, d \in \mathcal{C}(\ell)$ such that for $D = \{x : c(d(x)) = x\}$, $\Pr[X \in D] > 2^\ell/2^{n+1}$. There are exactly 2^n strings in this distribution, all of them with length equal to n. Then we can code them by the identity function. Therefore, $|D| = 2^n$ and $\Pr[X \in D] = 1 > 2^n/2^{n+1}$. Then, $H_\epsilon^{\mathrm{Yao}} < n + 1$. □

Next we prove a similar result for H_c. First, we state an easy lemma.

Lemma 2. *For any random variable S over binary strings of length at most n, $E(K^t(S)) \leq n + O(1)$.*

Theorem 10. *For the uniform distribution, $E(K^t(U_n)) = \Theta(n)$.*

Corollary 1. *If cryptographically-secure pseudo-random generators exist, then* $H_c(U_n; n) = \Theta(n)$.

The previous theorems show that for the uniform distribution, all three notions are assymptotically equivalent. Since in that case [BSW03] has shown that Yao-type entropy is equivalent to Hill-type entropy, a distribution is pseudo-random only if the expected time-bounded Kolmogorov complexity and the effective entropy are about the logarithm of its support set.

6.2 Universal Distribution $\mathbf{m}^t$

This section analyses the relation between these notions under the universal distribution $\mathbf{m}^t$.

Definition 10. *The universal distribution* $\mathbf{m}^t$ *is defined as* $\mathbf{m}^t(x) = 2^{-K^t(x)}$ *(see [LV97], pg 506).* $\square$

This distribution is computable in time $t(n)2^{n+1}$ for $n = |x|$. In the remainder of this section, we consider only the restriction to binary strings of length n, $\mathbf{m}^t_n(x)$. Unlike the case for the uniform distribution, there is a noticeable difference between $H^{\text{Yao}}_\epsilon(\mathbf{m}^t_n(X))$ and $H_c(\mathbf{m}^t_n(X); n)$.

Theorem 11. *For any constant* $c' > 0$, $H^{\text{Yao}}_\epsilon(\mathbf{m}^t_n(X)) < 2c' \log n + 1$.

This can be generalized to the following theorem:

Theorem 12. *If there is a polynomial-time computable distribution* X *with probability function* $P(x)$ *such that* $H^{\text{Yao}}_\epsilon(X) < f(n)$, *then* $H^{\text{Yao}}_\epsilon(\mathbf{m}^t_n(X)) < f(n) + K^{t(n)}(P)$.

It is possible to prove a result for H_c opposite to Theorem 11 that is a consequence of the following Theorem:

Theorem 13. *For the universal distribution* $\mathbf{m}^t$, $E(K^t(\mathbf{m}^t_n(X))) = \Theta(n)$.

Corollary 2. *If cryptographically-secure pseudo-random generators exist, then* $H_c(\mathbf{m}^t_n(X); n) = \Theta(n)$.

The same idea of the proof of Theorem 11 can be used to show the separation between $H_c(\mathbf{m}^t_n(X); n)$ and metric type entropy. We first give the definition presented in Lemma 3.3 of [BSW03] and then list the corresponding theorem. The proof can be found in the Appendix.

Definition 11. *Let* X *be a random variable over a set* S. *For every class* $\mathcal{C}$ *which is closed under complement and for every* $k \leq \log|S| - 1$ *and* ϵ, $H^{\text{Metric}}_\epsilon(X) \geq k$ *if and only if for every set* D *whose characteristic function belongs to* $\mathcal{C}$, $\Pr[X \in D] \leq \frac{|D|}{2^k} + \epsilon$.

Theorem 14. *For any constant* $c' > 0$, $H^{\text{Metric}}_\epsilon(\mathbf{m}^t_n(X)) < c' \log n + 1$.

It is known that $\mathbf{m}^{t'}(X)$ dominates all polynomial-time computable distributions $P(X)$, where $t'(n) = nt(n)$. This can be used to prove the next theorem:

Theorem 15. *Suppose there is a polynomial-time computable distribution X such that $H_c(X;n) \geq g(n)$. Then $H_c(\mathbf{m}_n^{t'}(X);n) \geq g(n)/2^{c_P}$, for $c_P = 2^{K^{nt(n)}}(P) + O(1)$.*

The results in this section show that there is a computable distribution that clearly separates H_c from H_ϵ^{Yao} and especially $H_\epsilon^{\text{Metric}}$. Since $H_\epsilon^{\text{Metric}}$ is equivalent to HILL-type entropy, this establishes a separation between pseudo-entropy and the maximum compressibility of a samplable source. This result is similar to Wee's, which is relative to an oracle and depends on the existence of good PRGs. This assumption is needed only because the coding function may be a probabilistic algorithm. If we allow only deterministic encodings, the result follows unconditionally.

7 Conclusion

In this paper we made an analysis of three measures that try to capture the maximum compression of a samplable source: effective entropy, Yao-type entropy, and Time-Bounded Kolmogorov complexity. We showed that effective entropy dominates the other two, in the sense that it is always greater than them up to some multiplicative parameter. We gave an inefficient definition of Yao-type entropy which is shown to be less than both Kolmogorov complexity and the traditional Yao-type entropy, but we could not relate these two notions, which remains an open question. We showed that for the uniform distribution all three notions are equivalent.

Also we gave an example of a distribution for which Yao-type and metric Entropies are very different from the other two notions, showing that they are not the right measure for the maximum compressibility of a samplable source and thus giving another evidence that pseudo-entropy and maximum compressibility are not necessarily related in a computationally-limited setting. It seems reasonable to believe that the average of bounded Kolmogorov Complexity is a strict lower-bound for the maximum compressibility of a source. Yao's effective entropy can not be easily computed. We'd have to consider all possible encoding schemes and run them over all strings to compare their averages. The problem is that the number of possible schemes is infinite.

However, Kolmogorov Complexity theory gives as a framework for computing this value, if one can spend a large amount of time: we can use the standard trick of, for every string in the support of S_n^λ, enumerating all programs of size up to $\lambda \cdot n$ in dovetailing fashion, from the shortest to the longest, and running each of them only up to a fixed number of steps polynomial on n. As soon as the shortest program is found for x, keep the its size and advance for the next string. This is a finite computation, since the number of strings, of programs and execution time are all finite quantities. Thus, computing the average bounded Kolmogorov complexity is at least theoretically possible, and for that reason it might be a better candidate for the true measure of maximum compressibility of a random source.

References

[BSW03] Barak, B., Shaltiel, R., Widgerson, A.: Computational Analogues of Entropy. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 200–215. Springer, Heidelberg (2003), Available at http://www.math.ias.edu/∼avi/PUBLICATIONS/MYPAPERS/BSW03/bsw03.ps

[Cha66] Chaitin, G.J.: On the length of programs for computing finite binary sequences. Journal of the ACM 13(4), 145–149 (1966)

[GS91] Goldberg, A., Sipser, M.: Compression and Ranking. SIAM Journal On Computing 20(3), 524–536 (1991)

[GV03] Grünwald, P., Vitányi, P.: Kolmogorov Complexity and Information Theory. Journal of Logic, Language and Information 12(4), 497–529 (2003), Available at http://citeseer.ist.psu.edu/565384.html

[HILL99] Hastad, J., Impagliazzo, R., Levin, L., Luby, M.: A Pseudorandom Generator from any One-way Function. SIAM Journal On Computing 28(4), 1364–1396 (1999), Available at http://citeseer.ist.psu.edu/hastad99pseudorandom.html

[Kol65] Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Problems Inform. Transmission 1(1), 1–7 (1965)

[LV97] Li, M., Vitányi, P.M.B.: An introduction to Kolmogorov complexity and its applications, 2nd edn. Springer, Heidelberg (1997)

[Sh48] Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal, vol. 27, pp. 379–423 and 623–656, July and October (1948)

[Sol64] Solomonoff, R.: A formal theory of inductive inference, part i. Information and Control, 7(1) 1–22, 1964.

[Wee04] Wee, H.: On Pseudoentropy versus Compressibility. IEEE Conference On Computational Complexity, pp. 29–41, (2004) Available at http://ieeexplore.ieee.org/iel5/9188/29139/01313782.pdf

[Yao88] Yao, A.: Computational Information Theory. In: Complexity in Information Theory, pp. 1–15. Springer, Heidelberg (1988)

A Appendix: Omitted Proofs

This section presents the proofs omitted in earlier Sections.

Theorem 2. *Consider Definition 7. Considering that efficient functions are those that can be computed in polynomial time, $\mathcal{C} = \mathbf{FP}$. Then, for any integer $i > 0$, $H_\epsilon^{\text{Yao}}(S_n^\lambda) \geq \lambda \cdot g(n) + (i+1) \Rightarrow H_c^d(S;n) \geq \left(1 - \frac{1}{2^i}\right) g(n)$. Equivalently, $\lambda \cdot H_c^d(S;n) \geq \left(1 - \frac{1}{2^i}\right)\left(H_\epsilon^{\text{Yao}}(S_n^\lambda) - (i+1)\right)$.*

Proof. Suppose for contradiction that $H_c^d(S;n) < \left(1 - \frac{1}{2^i}\right) g(n)$. Then, there is a $(t,k)-$encoding scheme $M = (M_A, M_B, M_C)$ that, for sufficiently large n, satisfies $L_n(M;S) < \left(1 - \frac{1}{2^i}\right) g(n)$. Fix some sufficiently large n and let $g = g(n) \geq L_n(M;S)$. Then, this condition is equivalent to

$$\sum_{x \in S_n^\lambda} p_n^k(x)|M_A(n,x)| < \lambda \cdot \left(1 - \frac{1}{2^i}\right) g$$

Define the set $D = \{x : |M_A(n,x)| \leq \lambda \cdot g\}$. Then,

$$\lambda \cdot \left(1 - \frac{1}{2^i}\right) g > \sum_{x \in S_n^\lambda} p_n^k(x)|M_A(n,x)| \geq \sum_{x \in S_n^\lambda \setminus D} p_n^k(x)|M_A(n,x)|$$
$$> \lambda \cdot g \cdot \Pr[S_n^\lambda \notin D] \Rightarrow \Pr[S_n^\lambda \notin D] < \left(1 - \frac{1}{2^i}\right)$$

This means that $\Pr[S_n^\lambda \in D] > \frac{1}{2^i}$. To show that $H_\epsilon^{\text{Yao}}(S_n^\lambda) < \lambda \cdot g + i + 1$, it suffices to give a pair of functions $c', d' \in \mathcal{C}(\lambda \cdot g + 1)$ and respective codeset D' such that $\Pr[S_n^\lambda \in D'] > \frac{2^{\lambda \cdot g + 1}}{2^{\lambda \cdot g + i + 1}} + \epsilon = \frac{1}{2^i} + \epsilon$.

Since every $x \in D$ can be compressed to a string of length at most $\lambda \cdot g$, by Theorem 1 we can efficiently compute a set D' that is isomorphic to D and such that every element $y \in D'$ has length $\lambda \cdot g + 1$. The reverse operation is also efficient.

Let f be this isomorphism. Then, let $c'(x) = f(M_A(n,x))$ and $d'(y) = M_B(f^{-1}(y))$. Since $\Pr[S_n^\lambda \in D] = \Pr[S_n^\lambda \in D']$, the theorem follows. □

Theorem 3. *Consider Definition 7 and let $\mathcal{C} = \mathbf{FP}$. Then, $H_\epsilon^{\text{Yao}}(S_n^\lambda) \geq \lambda \cdot g(n) + 1 \Rightarrow H_c^d(S;n) \geq g(n) - 1/\lambda$ or equivalently $\lambda \cdot H_c^d(S;n) \geq H_\epsilon^{\text{Yao}}(S_n^\lambda) - 2$.*

Proof. Fix n and let $g = g(n)$. Fix any efficient deterministic encoding scheme $M = (M_A, M_B, M_C)$. Let $c(x)$ be the restriction of $M_A(n,x)$ to the domain $R = \{x \in S_n^\lambda : |M_A(n,x)| < \lambda \cdot g\}$. Then, $d(x) = M_B(x)$. Using Theorem 1, there are functions c', d' and $D = \{x : d'(c'(x))\}$ such that all and only elements in R are in D and $|c'(x)| = \lambda \cdot g$ for all $x \in R$. Then, by definition, $\Pr[X \in D] \leq \frac{2^{\lambda \cdot g}}{2^{\lambda \cdot g + 1}} = 1/2 \Leftrightarrow \Pr[|M_A(n,x)| \geq \lambda \cdot g] \geq 1/2$, and $L_n(M;S)$ can be estimated:

$$\lambda \cdot L_n(M;S) = \sum_{x \in S_n^\lambda} p(x)|M_A(n,x)|$$
$$\geq \sum_{x \in R} p(x)|M_A(n,x)| + 1/2 \cdot \lambda \cdot g \quad (1)$$

The objective of the proof now is to show a lower bound for $L_n(M;S)$. The worst case happens when the $x \in S_n^\lambda$ that have shortest $|M_A(n,x)|$ descriptions have maximal probability. The first half of (1) is studied next.

Since $\Pr[|M_A(n,x)| < \lambda \cdot g]$ is at most $1/2$, in the worst case it is $1/2$. For any $\ell < \lambda \cdot g$ the function $M_A(n,x)|_{|x|=\ell}$ is also an efficient coding for the strings in the domain that have length ℓ. The result for other strings is not defined, for example we can assume they are all transformed into the empty string. Then we have

$$\Pr[|M_A(n,x)| = \ell] \leq 2^l / 2^{\lambda \cdot g+1} \tag{2}$$

$$\Pr[|M_A(n,x)| < \ell] \leq 2^l / 2^{\lambda \cdot g+1}. \tag{3}$$

the first by assumption and the second using Theorem 1.

We can write $\sum_{x \in R} p(x)|M_A(n,x)| = \sum_{i=0}^{\lambda \cdot g-1} i \cdot \Pr[|M_A(n,x)| = i]$.

To find a lower bound, we let the probabilities in the sum be maximum. Starting with $i = 0$, let $\Pr[|M_A(n,x)| = i]$ be $2^i/2^{\lambda \cdot g+1}$. This satisfies both (2) and (3). Now,

$$\begin{aligned}\sum_{x \in R} p(x)|M_A(n,x)| &= \sum_{i=0}^{\lambda \cdot g-1} i \cdot \Pr[|M_A(n,x)| = i] = \sum_{i=0}^{\lambda \cdot g-1} i \cdot \frac{2^i}{2^{\lambda \cdot g+1}} \\ &= \frac{1}{2^{\lambda \cdot g+1}} \cdot \left[2^{\lambda \cdot g} \cdot (\lambda \cdot g - 2) + 2\right] \geq \frac{\lambda \cdot g - 2}{2}\end{aligned}$$

But then, $\lambda \cdot L_n(M;S) \geq \lambda \cdot g - 1$. Since this must happen for any encoding scheme, it follows that

$$\lambda \cdot H_c^d(S;n) \geq \lambda \cdot g(n) - 1 \Leftrightarrow H_c^d(S;n) \geq g(n) - 1/\lambda \qquad \square$$

Theorem 7. *If cryptographically-secure pseudo-random generators exist, then* $\lambda \cdot H_c(S;n) \geq E(K^p(S_n^\lambda)) - O(\log n)$

Proof. Let G be a cryptographically secure PRG that stretches $O(\log n)$ bits of randomness into n^δ bits of pseudo-randomness for some δ. Let H be a fixed permutation over $\{0,1\}^{O(\log n)}$ chosen at random.

Let $y = M_A(n,x)$. We show how to compute x from y with small randomness with high probability.

Take some uniformly random string s_1 of size $O(\log n)$. Then use this to obtain $r_1 = G(s_1)$ and use r_1 as the randomness in one execution of $x' = M_B(n,y)$. The output x' may be correct or not. If it is wrong, then let $s_2 = H(s_1)$ and $r_2 = G(s_2)$. Run M_B again using r_2 as the random choices and repeat this process until the answer is correct.

Since H is a random permutation, s_i is uniformly random over $\{0,1\}^{O(\log n)}$ and so $G(s_i)$ returns a pseudorandom output over $\{0,1\}^{n^\delta}$. Therefore, eventually $M_B(n,y)$ will output the correct x. The number of times M_B has to be run with this scheme is sufficient to give a description of x. Let this number be m. Then, for some polynomial p

$$K^p(x) \le |M_A(n,x)| + K^p(M_B) + K^p(H) + K^p(G) + |s_1| + \log m \Rightarrow$$
$$K^p(x) \le |M_A(n,x)| + \log m + O(\log n) + O(1)$$

where we lumped the sizes of M_B, H and G in one constant term.

We compute the expected value of m: $E(m) = \sum_{i\ge 0}(i+1)\cdot\left(1-\frac{1}{n^t}\right)\cdot(1/n^t)^i = 1+\frac{1}{n^t-1}$. Since by definition n^t is big enough to make $1/n^t$ negligible, then also $1/(n^t-1)$ is negligible, so in average $M_B(n;x)$ is correct in the first try. Therefore, for each x the expected value of $K^p(x)$ is $K^p(x) \le |M_A(n,x)| + O(\log n)$ and so we get $\lambda \cdot H_c(S;n) \ge E(K^p(S_n^\lambda)) - O(\log n)$. □

Theorem 10. *For the uniform distribution, $E(K^t(U_n)) = \Theta(n)$.*

Proof. Lemma 2 shows $E(K^t(U_n)) \le n + O(1)$.

Now we prove the converse by giving a lower bound for $E(K^t(U_n))$.

It is known that $K^t(x) \le n + 2\log n$, where $n = |x|$, so we can write $E(K^t(U_n)) = \frac{1}{2^n}\sum_{i=0}^{n+2\log n} i\cdot f(i)$ where $f(i)$ is the number of strings x with $K^t(x) = i$. This sum is minimum when the complexities $K^t(x)$ are minimum, this is, exhausting the programs of short length. Thus

$$\begin{aligned} E(K^t(U_n)) &\ge \frac{1}{2^n}\left(n + \sum_{i=0}^{n-1} i\cdot 2^i\right) \\ &\ge \frac{1}{2^n}\cdot(2^n\cdot(n-2)+2) \ge n-2 \end{aligned}$$ □

Theorem 11. *For any constant $c' > 0$, $H_\epsilon^{\text{Yao}}(\mathbf{m}_n^t(X)) < 2c'\log n + 1$.*

Proof. Let $b = 2c'\log n + 1$. We find some ℓ, and a pair of functions $f, g \in \mathcal{C}(\ell)$ such that for $D = \{x : g(f(x)) = x\}$, it happens that $\Pr[X \in D] > 2^\ell/2^b + \epsilon$.

Consider the following algorithm. When f receives $x \in \{0,1\}^n$, it executes some universal prefix-free Turing machine U in dovetail fashion with all programs p_i of length up to $c'\log n$ for at most $t(n)$ steps, where c' is some constant. If there is some i such that $U^t(p_i) = x$, then $f(x)$ is the shortest such program. Otherwise, it returns the empty string. This program runs in time polynomial in $t(n)$ and returns some shortest prefix-free program for x that runs in time $t(n)$. Therefore, $|p_i| = K^t(x)$. The set D associated with this function contains only and all strings x of size n with $K^t(x) \le c'\log n$. Since all these strings form a prefix-free code, they can be padded with zeroes so that they all have length $c'\log n$.

There certainly is at least one string in this set, for example, the string x_0 composed of n zeroes has $K^t(x_0) \le \log n + O(1)$. Therefore, $|D| > 0$. For all $x \in D$, $K^t(x) \le c'\log n \Leftrightarrow \mathbf{m}_n^t(x) \ge 1/n^{c'}$. Then $\Pr[X \in D] \ge |D| \,/\, n^{c'} \ge 1 \,/\, n^{c'}$. Let $\ell = c'\log n$. Then, $\Pr[X \in D] \ge 1/2^\ell$. Since $b = 2\ell + 1$, it follows that $1/2^\ell > 2^\ell/2^b$. Therefore, $H_\epsilon^{\text{Yao}}(\mathbf{m}_n^t(X)) < 2c'\log n + 1$. □

Theorem 14. *For any constant $c' > 0$, $H_\epsilon^{\text{Metric}}(\mathbf{m}_n^t(X)) \le c' \log n$.*

Proof. The proof is similar to the one for Theorem 11, and it even uses some constructions from it.

Let $\mathcal{C} = \mathbf{P}$ and $\mathcal{C}(\ell)$ be the restriction of this class as outlined in Definition 7. Now, consider the pair of functions $c, d \in \mathcal{C}(\ell)$ and the corresponding codeset D given in that proof.

We build a predicate $A \in \mathcal{C}$ such that $\{x : A(x) = 1\} = D$. For that, let $A(x)$ evaluate $d(c(x))$ and output 1 if and only if the result is x. Since both c and d are efficient and deterministic, so is $A(x)$. Then, all and only elements in D satisfy A. As was seen in the proof of Theorem 11, $\Pr[X \in D] \ge |D|/n^{c'}$. Plugging $c' \log n$ for k in Definition 11, we get that $H_\epsilon^{\text{Metric}}(\mathbf{m}_n^t(X)) \le c' \log n$. □

Theorem 13. *For the universal distribution $\mathbf{m}^t$, $E(K^t(\mathbf{m}_n^t(X))) = \Theta(n)$.*

Proof. By Lemma 2 we have $E(K^t(\mathbf{m}_n^t(X))) \le n + O(1)$.

Consider the minimal prefix-free programs for all strings of length n. It is known that $K^t(x) \le n + 2\log n$. However, given that $\mathbf{m}_n^t(X)$ ranges only over the strings of length n, then n is implicit in it. Therefore, we can use instead $K^t(x|n) \le n + O(1)$. Let δ be the constant represented by $O(1)$. We can write $E(K^t(\mathbf{m}_n^t(x))) = \sum_{i=0}^{n+\delta} \frac{f(i)}{2^i} \cdot i$ where $f(i)$ is the number of strings that have $K^t(x) = i$. Since 2^i is the dominant term in the fraction, the minimum sum is achieved when all the strings have complexity as large as possible. There are 2^n strings of length n, so $f(i) = 2^n$ for $i = n + \delta$ and 0 everywhere else.

$$E(K^t(\mathbf{m}_n^t(X))) \ge 2^n/2^n \cdot (n + \delta) = n + O(1)$$ □

Theorem 15. *Suppose there is a polynomial-time computable distribution X such that $H_c(X; n) \ge g(n)$. Then $H_c(\mathbf{m}_n^{t'}(X); n) \ge g(n)/2^{c_P}$, for $c_P = 2^{K^{nt(n)}}(P) + O(1)$.*

Proof. For any $(t,k)-$encoding scheme $M = (M_A, M_B, M_C)$ for X, we have, by definition, that $L_n(M; X) \ge g(n) \Leftrightarrow \frac{\sum_x p_n^k(x)|M_A(x)|}{\lambda} \ge g(n)$. Recall that $x = \langle x_1, \ldots, x_\lambda \rangle$ is a word distributed according to X^λ. Since X is polynomial-time computable, then for $c_P = 2^{K^{nt(n)}(P)} + O(1)$, $\mathbf{m}_n^{t'}(x_i) \ge p_n(x_i)/2^{c_P} \Leftrightarrow K^{t'}(x_i) \le \log 1/p_n(x_i) + c_P$. The proof of this theorem in [LV97] uses the description of P to reconstruct the distribution and thence the string that is sought.

This reasoning can also be applied to x as a whole, but now the reconstruction of P need be done only once. Thus, c_P is not multiplied by λ. Therefore,

$$K^{t'}(x) \le \log 1/p_n^k(x) + c_P \Leftrightarrow \mathbf{m}_n^{t'}(x) \ge p_n^k(x)/c_P$$

and so

$$L_n(M; \mathbf{m}_n^{t'}(X)) \ge \frac{\sum_x \frac{p_n^k(x)}{2^{c_P}}|M_A(x)|}{\lambda} \ge \frac{g(n)}{2^{c_P}}$$

and the theorem follows. □

From Logic to Physics: How the Meaning of Computation Changed over Time

Itamar Pitowsky

The Edelstein Center, Levi Building, The Hebrew University, Jerusalem, Israel

1 The Church-Turing Thesis and the Meaning of 'Computable Function'

The common formulation of the Church-Turing thesis runs as follows:

Every computable partial function is computable by a Turing machine

Where by partial function I mean a function from a subset of natural numbers to natural numbers. As most textbooks relate, the thesis makes a connection between an intuitive notion (computable function) and a formal one (Turing machine). The claim is that the definition of a Turing machine captures the pre-analytic intuition that underlies the concept computation. Formulated in this way the Church-Turing thesis cannot be proved in the same sense that a mathematical proposition is provable. However, it can be refuted by an example of a function which is not Turing computable, but is nevertheless calculable by some procedure that is intuitively acceptable.

What exactly is our intuition about computation, and where does it come from? Turing was interested in the decision problem for formal systems, that of arithmetic in particular[1]. He was therefore concerned with a sense of "computation" that is tightly related to the formal concept of *proof*. Formal proofs by their very nature can be validated, at least in principle, by checking whether each step follows the mechanical rules of inference. This means that computations in the sense that is relevant for logic should also have that character, they should be idealizations of (symbolic) calculation with pencil and paper. To put it in Enderton's (1977) words; "One can picture an industrious and diligent clerk, well supplied with scratch paper, tirelessly following his instructions". If there is only a finite list of instructions and no bound on the supply of paper and pencils, and no preassigned limit on time, we have a Turing machine. In this context the Church-Turing thesis is simply the assertion that the language of Turing machines (or any other universal programming language) is the correct idealization *for that purpose*. Turing (1936) provided a very powerful argument to that effect, which is not less convincing than any informal mathematical proof. To sum, what the Church-Turing thesis meant initially can be formulated as follows

Any function that is computable by an ideal "industrious tireless and diligent clerk" is Turing machine computable.

With the advance of technology computer science has left its place of birth in mathematical logic and became an independent enterprise. It was pointed out

[1] Much of what I say here applies also to Church.

by Shagir (2002) that this development also led to a change in the concept of computation. The stress is no longer put on idealized human activity (proofs, calculations) but shifts to the abilities of "discrete finite automata"[2]. With this change we obtain *the physical Church-Turing thesis* that reads

Any function that is computable by a discrete finite automaton is Turing machine computable.

However, with this change some of the transparency of the thesis is gone. While every schoolchild knows what a pencil and paper calculation is, and is able to imagine how it can be idealized, no equivalent understanding exists with regard to finite automata. It is not completely clear what kind of contraptions should be included as legitimate finite discrete automata, and what are the grounds for inclusion. Of course, we do have *some* intuition, most of it in the form of physical constraints of a very general kind. For example, we imagine the automaton to be composed of discrete parts distributed in space, such that only finitely many of them participate in each step. We believe that there is a limit to the speed with which one part can influence other parts, so that in each step a part can only induce a change in its spatial neighbors, and so on. Using intuitions of this kind Gandy (1980) tried to give a very general (and abstract) characterization of finite discrete automata and prove the thesis. His work is explicated in Sieg and Byrnes (1999).

It seems to me that this issue is nevertheless not settled. What we mean by finite discrete automaton depends too heavily on our theories of space and time, and more importantly, on what we mean by "physical state". These issues are contingent and can change quite radically with the advance of empirical science. In this context we should perhaps refer to the physical Church-Turing *hypothesis*. However, it must be emphasized that even in this contingent form the thesis has not yet been seriously challenged. Neither by any physically realizable machine model, nor even by an ideal physical automaton working in *our* universe.

2 Challenges to the Thesis

Consider two attempted counterexamples to the Church-Turing thesis in its physical, finite automaton version. The first, I think, does not get off the ground, and the second is so highly idealized that it never gets to the ground.

2.1 Kieu's Quantum Computer

Kieu (2005) claimed that given a Diophantine equation there is a quantum adiabatic computer that decides if the equation is solvable. Since this problem is generally undecidable we have an (alleged) automaton that can decide a non-Turing computable predicate. Kieu's proposal has been widely criticized on various counts. My aim here is not to repeat or assess previous points of criticism

[2] The word *discrete* serves to exclude analog computers that use continouus space-time processes. By *finite* I mean that every instance of computation requires a finite amont of resources. However, the amount can grow with the size of the instances.

but rather to make a simple general point which does not require a detailed understanding of the model. The gist is as follows: Kieu's design assumes tacitly that we are already in possession of some physical process that computes another non-recursive function. In other words, it begs the question. This point is also made in Hagar and Korolev (2006), which contains a wide reference list on the subject.

Consider the set of all Diophantine equations, this set can be effectively enumerated, so let $E_1, E_2, ..., E_n, ...$ be such an enumeration. Fix a universal Turing machine U and let $d_U(n)$ be the number of steps it takes U to decide that E_n has a solution *in case it is solvable*, and $d_U(n) = 0$ in case E_n is not solvable. Put $D_U(n) = \max\limits_{1 \le j \le n} d_U(j)$. The function $D_U(n)$ is not computable, it grows asymptotically faster than any recursive function, like the famous Busy Beaver (Boolos and Jeffrey, 1974). Moreover, this property of $D_U(n)$ is independent of the universal Turing machine U and the enumeration of the equations.

Kieu is employing a version of the adiabatic quantum computation. Suppose that we set to decide the Diophantine equation E_n. Then one takes a time dependent Hamiltonian of the form

$$H(t) = (1 - \frac{t}{T_n})H_0 + \frac{t}{T_n}H_1$$

Where H_0^n and H_1^n are stationary Hamiltonians (which depend on the equation E_n), they do not commute, and the spectrum of H_0^n is known. The adiabatic theorem states that if T_n is sufficiently large, and if we start from the ground state of H_0^n we end after time T_n in the ground state of H_1^n. The trick is to encode the solution to the computational problem as the ground state of H_1^n which is then discovered by a measurement after time T_n.

But how large should T_n be? According to the adiabatic theorem it is a polynomial in the inverse of the spectral gap of the Hamiltonian. However, this needs not concern us here. What is important is that in the worst case T_n grows with n like $D_U(n)$. Or, to put it more precisely: *No matter what translation is chosen between "number of Turing machine steps" and the physical "number of seconds", so long that this translation is a monotone recursive function, the worst case* $\max\limits_{1 \le j \le n} T_j$ *is non-recursive.*

Kieu does not assume that we explicitly know from the outset the value of T_n, which would directly beg the question. He devises a probabilistic algorithm which is executed in steps. We run a few copies of the computer for some time T and then performs a measurement on all copies. If more than half the copies give the same outcome we are done. Otherwise we have to increase the running time and try again, and so on. Assuming that we accept the mathematical correctness of the Kieu's procedure, we are still left with the puzzle. Suppose that we are in one of the worst cases and the actual running time is $\backsim T_n$ seconds. Suppose that we ran the process for "only" $\frac{1}{2}T_n$ seconds and got the wrong result, so in the next step we have to slow the process, say by a factor of $\frac{1}{2}$. How do we do that? *How do we know that when we start the process afresh it is running at half the paste of the previous one, or even that it is just slower than the previous one?*

It requires a "speedometer" that is capable of distinguishing between a process that will end after $\frac{1}{2}T_n$ seconds and one that will end after T_n seconds.

The model of adiabatic quantum computation assumes that the speed of the process is just a parameter over which we have physical control. We can run it slower or faster at will. But as we take more and more complicated Diophantine equations E_n the speeds involved become asymptotically slower then $\frac{1}{f(n)}$ for every monotone increasing recursive function $f(n)$[3]. To assume that we have such a degree of control over a physical parameter begs the question.

2.2 Supertasks

A machine performs a supertask if it performs an infinite number of steps in a finite span of time. It turns out that the existence of such devices is compatible with the principles of general relativity (Pitowsky 1990, Hogarth 1994, Earman and Norton, 1993). In some space-time manifolds which are solutions to the Einstein's equation we can find a point p, and a time-like future directed half curve γ (with $\int_\gamma d\tau = \infty$) such that the entire stretch of γ lies in the chronological past of p. If we let a regular computer which is equipped with a transmitter move along γ, while a reciever moves along another future directed time-like curve that connects a beginning point, call it q, with p. The time of travel of the receiver from q to p is finite, while "at the same time" the computer and transmitter complete the infinite time trip along γ. The computer can check an undecidable proposition of the form $\forall n \Phi(n)$ case by case, to infinity. If a counter-example is found the computer transmits a message. The receiver will get it before it arrives at p, and the result of the computation is that $\forall n \Phi(n)$ is false. If, on the other hand, no message is received by the time the computer arrives at p, then $\forall n \Phi(n)$ is true.

On one level the model comprises a counter-example to the Church-Turing thesis in its finite automaton version[4]. In particular, it shows that some of the physically inspired constraints on finite automata (for example, that there is a limit to the speed of information transmission) do not prevent the machine from completing a supertask (roughly, because when such a limit exists the length of the time interval becomes observer dependent). The question is how significant this counter-example really is?

This model is not free of conceptual problems, some of which were pointed out by Shagrir and Pitowsky (2001). Here I will concentrate on some physical observations made by Earman and Norton (1993) that shed light on the notion of "physical possibility" assumed by this model. We apply here a rule that may be formulated as follows: *If there is a solution to Einstein's equation in which P holds, then P is physically possible.* As a methodological dictum it serves as a

[3] That is, for every such f there is N such that if $n > N$, then the (worst case) process is slower than $f^{-1}(n)$.

[4] If $\forall n \Phi(n)$ turns out to be true, more powerful computers should be built to check larger instances. However, at no finite time along γ an actual infinite machine is needed.

useful tool to expose tacit assumptions behind claims of impossibility. However, it provides a very weak notion of physical possibility. For many physicists the existence of Einsteinian manifolds on which supertasks and other strange physical effects are possible is an indication of a handicap of the theory. One way to handle this handicap is to formulate physical principles that rule out undesirable solutions on a wholesale basis. A significant example is the (strong version) of cosmic censorship which eliminates the kind of hypercomputation described above, as well as many other strange effects (Penrose 1972).

Another problem with the model is that it entails the existence of a singularity, namely, a process through which a physical magnitude grows without bound. Suppose that the message from the transmitter to the receiver is an electromagnetic wave. One way the singularity is expressed in this case is in terms of the frequency of the message received on the way from q to p. The longer it takes to find a counter example to $\forall n \Phi(n)$ (assuming there is one) the shorter is the wave length that arrives from γ. Beyond a certain point such hard photons with huge energies are no longer described by the familiar laws of physics. To delay this effect the transmitter can lower its frequency as it goes along γ, but beyond a certain point it will be masked by thermal noise. Hence, like other more famous cases of singularity, the model actually points out a limit to the validity of the general theory of relativity rather than a physical possibility, even in a weak sense of the term.

To sum, despite the fact that "discrete finite automaton" is not a precisely defined notion, the intuition the lies behind the physical Church -Turing thesis seems so far to be standing on a solid ground. Although Turing, Church and the other founders of the theory of computation did not mean their work to apply to physics, they may have inadvertently made a great discovery in this field.

3 Exponential vs. Polynomial

3.1 Quantum Computation

A common way to formulate the physical Church-Turing thesis is the following

The distinction between computable and non-computable functions is independent of the machine model.

Indeed, one method to gather support for the thesis is to demonstrate for as many machine-models as possible that they are equivalent to Turing machines. An unexpected feature of these equivalences is that they are of polynomial time complexity, that is

$$(\text{computation time of } f(x) \text{ on machine model } I) \leq$$

$$\leq C(\text{computation time of } f(x) \text{ on machine model } II)^k$$

for some constant $C > 0$ and a fixed natural number k which are independent of f and x. Computation time is measured by the number of steps the machine is performing, assuming that it is a finite discrete automaton. The polynomial time dependence has been noticed quite early (see Cobham, 1964; Edmonds, 1965). Let $\mathcal{P}$ be the class of functions which are polynomial time Turing computable. As the evidence about the transformations between machine models accumulated the following thesis became more established

The Polynomial Church -Turing thesis: The class $\mathcal{P}$ is independent of the machine model.

In other words, no automaton can reduce the complexity of an exponential time[5] Turing computable function, and compute it in polynomial time. It is an interesting historical fact that the robustness of $\mathcal{P}$, although independent of the Church-Turing thesis, was noticed while evidence for the latter was mounting.

The polynomial Church-Turing thesis has no counterexample, but there is a very serious candidate: Shor's (1994) quantum algorithm for the factorization of numbers. (Recall that the classical complexity of this problem is not known to be exponential.) In the following I will assume that the classical factorization of integers is not in $\mathcal{P}$ and discuss the question: What is it about quantum computers which is responsible for the speed-up? Note that this question can still be of some interest even if factorization turns out to be classically in $\mathcal{P}$, because more modest speed-up is gained in other circumstances (e.g., Grover's 1997 search algorithm).

The robustness of $\mathcal{P}$ has been conjectured because all computer models available at the time could be simulated in polynomial time by a Turing machine. When we try to figure out why quantum computers should fail to be polynomially simulated an immediate answer seems to suggest itself: Because of the principle of superposition, that is, entanglement. However, this is at best a very partial answer (see Jozsa and Linden, 2003). In the following I will point out why this answer is only partially true. I will also compare the state of quantum computation with the situation in other fields of quantum information theory, where proven advantages are gained by quantum resources, and where the reasons for the enhancement are better understood.

But first, why should entanglement be the immediate suspect? Let L be a natural number, and let $a = \sum_{k=0}^{L-1} a_k 2^k$ be a natural number where $a_0, a_1, ..., a_{L-1} \in \{0,1\}$. Choose, as usual, an orthonormal basis $|0\rangle$, $|1\rangle$ in the two dimensional complex space $\mathbb{C}^2$, and represent the number a in terms of L qbits on the L-times tensor product of $\mathbb{C}^2$ as follows: $|a\rangle = |a_{L-1}\rangle\ |a_{L-2}\rangle ... |a_0\rangle$. By the principles of quantum mechanics any superposition of the form

$$|\phi\rangle = \sum_{c=0}^{2^L-1} c(a)\, |a_{L-1}\rangle\, |a_{L-2}\rangle ... |a_0\rangle\,,$$

where $c(a)$ are complex numbers and $\sum_{c=0}^{2^L-1} |c(a)|^2 = 1$, corresponds to a possible (pure) state of a quantum system. Storing the generic state $|\phi\rangle$ (or its approximation) in the memory of a classical computer requires more than 2^L bits. This may seem as the source of difficulty in simulating quantum computers

[5] Here by "exponential" I mean anything higher than polynomial (including the subexponential).

efficiently on a classical automaton. However, one has to remember that in the model of quantum computation the arbitrary state $|\phi\rangle$ has to be generated from $|0\rangle_L = |0\rangle\ |0\rangle \dots |0\rangle$, and there seems to be no general way of doing it in a number of steps which is bounded by a polynomial in L. On the other hand, even if a specific genuinely entangled state can be generated in polynomial time from $|0\rangle_L$ it does not necessarily entail that it can serve to speed-up computation. To achieve speed-up one needs specific entangled states which can be generated in polynomial time (a certain amount of noise can of course be permitted). So entanglement seems to be only a part of the answer and more details should be added.

Consider two examples, one positive and one negative. The crucial step in Shor's algorithm is the proof that the Fourier transform $\mathcal{F}_L\, |a\rangle$, which is defined on the computational basis vectors by

$$\mathcal{F}_L\, |a\rangle = \frac{1}{\sqrt{2^L}} \sum_{c=0}^{2^L-1} \exp\left(\frac{2\pi i a c}{2^L}\right) |c\rangle$$

can be computed in $poly(L)$ steps on a quantum computer. It is safe to say that apart from Born's rule this is the only physical aspect of the algorithm (the rest is number theory). The actual entangled state required for the calculation is generated by the application of $\mathcal{F}_L$ to a state that was previously prepared by a measurement.

The negative example concerns the graph isomorphism problem. It is an open problem whether it is classically solvable in polynomial time, although it is believed to be simpler than the NP-complete problems. A (simple) graph is a pair $G = (V, E)$ where V is the set of *vertices*, which we take to be $V = \{1, 2, ..., n\}$, and E the set of *edges* is a subset of the set of pairs $E \subseteq \{\{i, j\}\ ;\ 1 \leq i < j \leq n\}$. Two graphs $G = (V, E)$ and $G^{`} = (V, E^{`})$, with the same set of vertices V, are called *isomorphic* if there is a permutation π of V such that $\{i, j\} \in E$ if and only if $\{\pi i, \pi j\} \in E^{`}$. The computational task is to decide whether two given graphs are isomorphic.

To represent a graph $G = (V, E)$ by a quantum state we can use $\frac{1}{2}n(n-1)$ qbits and label them by the pairs $\{i, j\}$ with $1 \leq i < j \leq n$. Define $\chi(i, j) = 1$ when $\{i, j\} \in E$, and $= 0$ otherwise, and denote

$$|G\rangle = |\chi(1,2)\rangle\, |\chi(1,3)\rangle \dots |\chi(i,j)\rangle \dots |\chi(n-1,n)\rangle$$

Then the graph G is uniquely characterized by the state $|G\rangle$. Similarly define the state $\left|G^{`}\right\rangle$ corresponding to the graph $G^{`}$ and its function $\chi^{`}(i, j)$ which equals 1 when $\{i, j\} \in E^{`}$ and 0 otherwise. Now, consider the entangled state

$$|SG\rangle = \frac{1}{\sqrt{n!}} \sum_{\pi \in S_n} |\chi(\pi 1, \pi 2)\rangle\, |\chi(\pi 1, \pi 3)\rangle \dots |\chi(\pi i, \pi j)\rangle \dots |\chi(\pi(n-1), \pi n)\rangle\,,$$

where the sum ranges over all permutations π of V, and define a similar state $\left|SG^{`}\right\rangle$ for the graph $G^{`}$. It is easy to see that if G is isomorphic to $G^{`}$then

$\left|SG^{'}\right\rangle = \left|SG\right\rangle$. Moreover, in case G and $G^{'}$ are not isomorphic $\left|SG\right\rangle$ is orthogonal to $\left|SG^{'}\right\rangle$. If we could create the state $\left|SG\right\rangle$ in a number of steps that is polynomial in n then we would have solved graph isomorphism in polynomial time (Beckman, 2004) . However, nobody knows how to do that.

It seems therefore that the possibility of quantum speed-up depends on the ability to create *specific* entangled states fast. Both examples can be described in group theoretic terms, the first relating to a commutative, and the second to a non-commutative group. The notion of entanglement seems to be too general to capture the structure that underlies fast quantum algorithms, and a more complete understanding is not yet available.

3.2 Quantum Communication

While the causes of quantum computational speed-up are elusive the advantages of some quantum mechanical communication protocols are better understood. The reason is that those protocols are built on the general concept of entanglement and the associated quantum correlations.

To understand the difference we shall concentrate on communication between two parties. Connsider pairs of objects sent from a source, one in the direction of Alice, and one in Bob's direction. Alice can perform either one of two measurements on her object; she can decide to detect the event A_1 or its absence (which means detecting the event $\overline{A}_1$). Alternatively, she can decide to detect the event A_2 or $\overline{A}_2$. So each of these two possible measurements has two possible outcomes. Similarly, Bob can test for event B_1 or use a (possibly different) test to detect B_2. We have three possibilities:

(1) Assume the four events A_1, A_2, B_1, and B_2 are events in a classical probability space, and let p be a probability measure. Consider the eight dimensional real vector:

$$v = (p(A_1), p(A_2), p(B_1), p(B_2), p(A_1B_1), p(A_1B_2), p(A_2B_1), p(A_2B_2)).$$

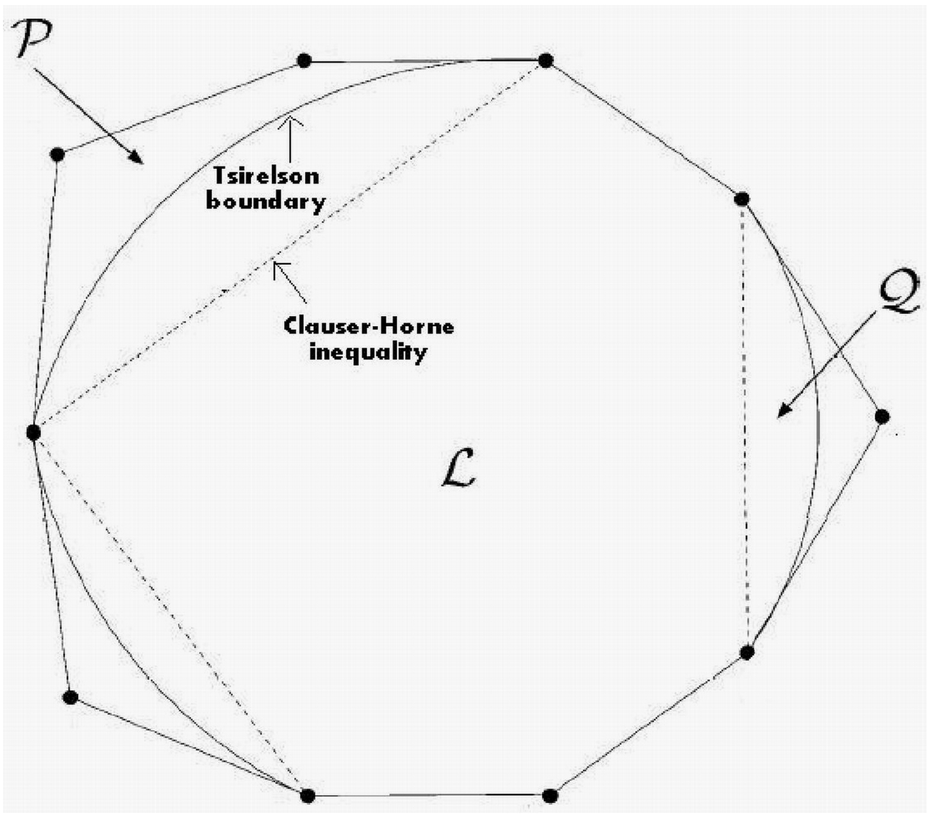
Let the A's and B's range over any events in any classical event space, and p range over all possible probability measures on that space. Then the vector v ranges over a convex polytope $\mathcal{L}$ in $\mathbb{R}^8$. The 16 vertices of $\mathcal{L}$ are just the extreme zero-one assignments to the probabilities of the A's and B's, and the non-trivial facets of $\mathcal{L}$ are just the Clauser Horne (1974) inequalities of the form

$$-1 \leq p(A_1B_1) + p(A_1B_2) + p(A_2B_2) - p(A_2B_1) - p(A_1) - p(B_2) \leq 0.$$

All the non trivial facets of $\mathcal{L}$ are obtained by permuting the events in this inequality (Pitowsky, 1989).

(2) Consider what happens if we let the A's and B's range over quantum events. In other words, suppose that A_i and B_j are projection operators in some Hilbert space $\mathbb{H}$, and ρ a (pure or mixed) state on $\mathbb{H} \otimes \mathbb{H}$, then the corresponding quantum probabilities are $p(A_i) = tr[\rho(A_i \otimes I)]$, $p(B_j) = tr[\rho(I \otimes B_j)]$, and $p(A_iB_j) = tr[\rho(A_i \otimes B_j)]$. The set of all probability assignments of this kind, which is obtained as we vary $\mathbb{H}$, A_i, B_j and ρ, is a convex subset $\mathcal{Q}$ of $\mathbb{R}^8$, and it satisfies $\mathcal{L} \subsetneqq \mathcal{Q}$. In particular, the Clauser Horne Inequality is violated in some cases. The boundary of $\mathcal{Q}$ is a complicated set which was described by Tsirelson (1980).

(3) Furthermore, one can theoretically consider possibilities that even transcend quantum theory (and therefore not known to be physically realizable). Popescu and Rohrlich (1996) introduced such extended correlations, which are constrained only by the condition of no-signaling[6]. These correlations 'live' within yet another convex polytope $\mathcal{P}$ which is still larger than $\mathcal{Q}$. The situation is schematically described in the figure.



If ρ is a *pure* entangled quantum state one can always find A_i, B_j such that the correlation is quantum (that is, lies in $\mathcal{Q} \setminus \mathcal{L}$). This is not true for some entangled mixed states (Werner, 1989), so entanglement does not always imply the existence of quantum correlations. Quantum correlations can be used in a variety of tasks. The number of articles published on these topics is staggering:

[6] The no-signaling condition says that a change in Bob's measurement does not influence Alice's marginals $p(A_i)$, and vice versa.

from Ekert's (1991) application of quantum correlations to secure quantum key distribution, to a recent applications of the so-called pseudotelepathy (Gisin et al., 2006). Some of these applications are provably impossible without quantum correlations.

In addition, entanglement sometimes provide exponential gain in communication complexity over classical correlations (Buhrman, Cleve and Wigderson, 1998), and assuming the existence of the unphysical regime of the Popescu Rohrlich correlations in $\mathcal{P}$ trivialize communication complexity entirely (van Dam, 2005; Brassard et al., 2005). Barrett (2005) developed a general framework in which a variety of probabilistic theories can be defined, including the classical and quantum theories, but also others, which allow the inclusion of the Popescu Rohrlich correlations. Each theory of this kind gives rise to certain information theoretic possibilities. It will be an interesting excercise to develop a theory of computation within such a wide framework and see what are the possibilities for speed-up. This may shed some light on the power of quantum computers, and on the relations between physics and computational complexity.

References

Barrett, J.: Information processing in generalized probabilistic theories (2005) http://arxiv.org/quant-ph/0508211

Beckman, D.E.: Investigations in quantum computing, causality and graph isomorphism PhD thesis, California Institute of Technology (2004)

Boolos, J.S., Jeffrey, C.J.: Computability and Logic. Cambridge University Press, Cambridge (1974)

Brassard, G., Buhrman, H., Linden, N., Methot, A.A., Tapp, A., Ungerquant, F.: A limit on nonlocality in any world in which communication complexity is not trivial (2005) http://arxiv.org/abs/quant-ph/0508042

Buhrman, H., Cleve, R., Wigderson, A.: Quantum vs. Classical Communication and Computation (1998) http://arxiv.org/abs/quant-ph/9802040

Clauser, J.F., Horne, M.A.: Experimental consequences of objective local theories. Physical Review D 10, 526–535 (1974)

Cobham, A.: The intrinsic computational difficulty of a function. In: Bar-Hillel, Y. (ed.) Proc. 1964 International Congress for Logic, Methodology, and Philosophy of Science, North Holland, Amsterdam (1964)

Earman, J., Norton, J.D.: Forever is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes. Philosophy of Science 60, 22–42 (1993)

Edmonds, J.: Paths trees and flowers. Canadian Journal of Mathematics 17, 449–467 (1965)

Ekert, A.: Quantum cryptography based on Bell' s theorem. Physical Review Letters 67, 661–664 (1991)

Enderton, H.B.: Elements of recursion theory In: Barwise, J. (ed.) Handbook of Mathematical Logic North Holland, Amsterdam, pp. 527–566

Gandy, R.O.: Church's Thesis and Principles of Mechanisms. In: Barwise, J., Keisler, J.J., Kunen, K. (eds.) The Kleene Symposium, pp. 123–145. North Holland, Amsterdam (1980)

Gisin, N., Methot, A.A., Scarani, V.: Pseudo-telepathy: input cardinality and Bell-type inequalities (2006) http://arxiv.org/quant-ph/0610175

Grover, L.K.: Quantum Mechanics helps in searching for a needle in a haystack. Physical Reveiw Letters 78, 325–328 (1997)

Hagar, A., Korolev, A.: Quantum hypercomputability? Minds and Machines 16, 87–93 (2006)

Hogarth, M.L.: Non-Turing Computers and Non-Turing Computability. Proceedings of the Philosophy of Science Association (PSA) 1, 126–138 (1994)

Jozsa, R., Linden, N.: On the role of entanglement in quantum computational speed-up. In: Proceedings of the Royal Society of London A vol. 459, pp. 2011–2032 (2003)

Kieu, T.D.: An Anatomy of a Quantum Adiabatic Algorithm that Transcends the Turing Computability. International Journal of Quantum Information 3, 177–183 (2005)

Penrose, R.: Gravitational collapse. In: De Witt-Morette, C. (ed.) Gravitational Radiation and Gravitational Collapse, pp. 82–91. Reidel, Dordrecht (1974)

Pitowsky, I.: Quantum Probability, Quantum Logic. Lecture Notes in Physics, vol. 321. Springer, Heidelberg (1989)

Pitowsky, I.: The Physical Church Thesis and Physical Computational Complexity, Iyun vol. 39, pp. 161–180 (1990)

Popescu, S., Rohrlich, D.: Action and Passion at a Distance: An Essay in Honor of Professor Abner Shimony (1996) http://arxiv.org/abs/quant-ph/9605004

Shagrir, O.: Computations by Humans and Machines. Minds and Machines 12, 221–240 (2002)

Shagrir, O., Pitowsky, I.: The Church-Turing Thesis and Hypercomputation. Minds and Machines 13, 87–101 (2003)

Shor, P.W.: Polynomial Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Journal of Computing 26, 1484–1509 (1994)

Sieg, W., Byrnes, J.: An Abstract Model for Parallel Computations: Gandy's Thesis. The Monist 82, 150–164 (1999)

Tsirelson, B.S.: Quantum generalizations of Bell's inequality. Letters in Mathematical Physics 4, 93–100 (1980)

Turing, A.M.: On Computable Numbers with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 45(2), 115–154 (1936)

van Dam, W.: Implausible Consequences of Superstrong Nonlocality (2005) http://arxiv.org/abs/quant-ph/0501159

Werner, R.F.: Quantum states with Einstein-Podolsky-Rosen correlations admitting a hidden-variable model. Physical Review A 40, 4277–4281 (1989)

Theories and Ordinals: Ordinal Analysis

Michael Rathjen

Department of Pure Mathematics, University of Leeds, Leeds LS2 9JT, UK

1 Introduction

How do ordinals gauge the strength and computational power of theories and what kind of information can be extracted from this correlation? This will be the guiding question of this talk. The connection between ordinal representation systems and theories is established in ordinal analysis, a central area of proof theory. The origins of proof theory can be traced back to the second problem on Hilbert's famous list of problems, which called for a proof of consistency of the arithmetical axioms of the reals. In the 1920s, Ackermann [1] and von Neumann [15], in pursuit of Hilbert's Programme, were working on consistency proofs for arithmetical systems. Ackermann's 1924 [1] dissertation gives a consistency proof for a second-order version of primitive recursive arithmetic which explicitly uses a finitistic version of transfinite induction up to the ordinal $\omega^{\omega^{\omega}}$. The employment of transfinite induction on ordinals in consistency proofs came explicitly to the fore in Gentzen's [9] 1936 consistency proof for Peano arithmetic, **PA**. He showed that transfinite induction up to the ordinal

$$\varepsilon_0 \;=\; \sup\{\omega, \omega^{\omega}, \omega^{\omega^{\omega}}, \ldots\} \;=\; \text{least } \alpha \text{ such that } \; \omega^{\alpha} = \alpha$$

suffices to prove the consistency of Peano Arithmetic, **PA**. To assess Gentzen's result at its true worth it is important to note that he applied transfinite induction up to ε_0 solely to primitive recursive predicates and besides that his proof used only finitistically justified means. Hence, a more precise rendering of Gentzen's analysis reads as follows:

$$\mathbf{F} + \mathrm{PR} - \mathrm{TI}(\varepsilon_0) \vdash \mathrm{Con}(\mathbf{PA}), \tag{1}$$

where **F** notates a theory that is acceptable from a finitist's viewpoint (e.g. $\mathbf{F} = \mathbf{PRA}$ = *Primitive Recursive Arithmetic*) and PR-TI(ε_0) stands for transfinite induction up to ε_0 for primitive recursive predicates (while Con(**PA**) formalizes the consistency of **PA**). Gentzen also showed that his result is best possible in that **PA** proves transfinite induction up to α for arithmetic predicates for any $\alpha < \varepsilon_0$. The compelling picture emanating from this is that the non-finitist part of **PA** is encapsulated in PR-TI(ε_0) and therefore "measured" by ε_0, thereby suggesting the following definition of *proof-theoretic ordinal* of a theory T:

$$|T|_{\mathrm{Con}} = \text{least } \alpha.\ \mathbf{PRA} + \mathrm{PR} - \mathrm{TI}(\alpha) \vdash \mathrm{Con}(T). \tag{2}$$

Note, however, that the definition of $|T|_{\mathrm{Con}}$ depends on the representation of an intial segment of ordinals in the theory **PRA** (i.e. an ordinal representation system). In the case of **PA** what is required is a representation $\mathcal{O}(\varepsilon_0)$ of the ordinals below and including ε_0 (which is usually engineered by the Cantor normal form).

A caveat to be uttered here is that ordinal analysis establishes a relationship in which both parts have to be "natural" in that a "natural" theory (e.g. **PA**) is related to a "natural" ordinal representation system (e.g. $\mathcal{O}(\varepsilon_0)$). It is a striking empirical fact that many "natural" theories, i.e. theories which have something like an "idea" to them, are comparable with regard to consistency strength. This has actually been proved in many cases, for theories whose ideas and motivations have nothing at all to do with one another. A plethora of results in proof theory and set theory seems to provide compelling evidence that the ordering of consistency strength, $\leq_{\mathrm{Con}}$, is a linear ordering on "natural" theories. To illustrate this by way of examples from set theory, with a few exceptions, large cardinal axioms have been shown to form a well-ordered hierarchy when ordered as follows:

$$\phi \leq_{\mathrm{Con}} \psi \quad := \quad \mathbf{ZFC} + \phi \leq_{\mathrm{Con}} \mathbf{ZFC} + \psi,$$

where ϕ and ψ are large cardinal axioms. This has not been established for all of the large cardinal axioms which have been proposed to date; but there is strong conviction among set theorists that this will eventually be accomplished (cf. [13,28]). The mere fact of linearity of $\leq_{\mathrm{Con}}$ is remarkable. But one must emphasize "natural" here, because one can construct a pair of self-referential sentences to engender incomparable theories (cf. [24], 2.17, 2.18). In the same vein, for of a given theory T, it is possible (as Kreisel liked to emphasize) to write down a "self-referential" ordinal representation system $\mathcal{O}(T)$ of sorts (utilizing T's proof predicate) to the effect that the consistency of T is trivially implied by the well-foundedness of $\mathcal{O}(T)$.

2 Natural Wellorderings

Proof theorists have concerned themselves with the problem of discerning the difference between the examples of well-orderings that arise naturally in the proof theory of formal systems and pathological examples. Feferman [6,7,8] put forward the idea "that what distinguishes such orderings are certain intrinsic mathematical properties that are independent of their possible use in proof-theoretical work" ([8], p.10). Uniqueness up to recursive isomorphism has been suggested by Kreisel as a criterion for naturalness (cf. [14]) who also proposed that naturalness can be found in algebraic characterizations of ordered structures. Feferman, in [6], chose the properties of completeness, repleteness, relative categoricity and preservation of these under iteration of the critical process as significant features of systems of natural representation.

The representation of all ordinals $< \varepsilon_0$ used by Gentzen is based on the Cantor normal form with respect to base ω, i.e. every ordinal $0 < \alpha < \varepsilon_0$ can be uniquely represented in the form

$$\alpha = \omega^{\alpha_1} \cdot k_1 + \ldots + \omega^{\alpha_n} \cdot k_n$$

with $\varepsilon_0 > \alpha_1 > \ldots > \alpha_n$ and $0 < k_1, \ldots, k_n < \omega$. To capture the abstract essence of a representation system for ordinals, Girard [10] proposed a generalized Cantor-normal-form-type of representations, dubbed *denotation systems*. He characterized them in categorical terms as derived from certain structure preserving functors on the category of ordinals. Let **Ord** be the category whose objects are the ordinals and whose arrows are the strictly increasing functions between ordinals. A *dilator* F is an endofunctor of the category **Ord** preserving direct limits and pullbacks. Since any ordinal is a direct limit $\lim_{\rightarrow}(x_i, f_{ij})$ of a system of finite ordinals x_i, a dilator F is completely determined by its behaviour on the subcategory $\mathbf{Ord}_\omega$ of finite ordinals. By utilizing the latter property and the fact that F preserves pullbacks one can assign a unique denotation of the form $(\gamma; \alpha_0, \ldots, \alpha_{n-1})$ to every ordinal $\beta < F(\alpha)$ such that $\alpha_0, \ldots, \alpha_{n-1} < \alpha$ and $\gamma < F(n)$ for some finite ordinal n.

Definition 2.1 ([10])**.** Let ON be the class of ordinals and $F : \mathrm{ON} \rightarrow \mathrm{ON}$. A *denotation-system* for F is a class $\mathcal{D}$ of ordinal *denotations* of the form

$$(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$$

together with an assignment $D : \mathcal{D} \rightarrow \mathrm{ON}$ such that the following hold:

1. If $(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$ is in $\mathcal{D}$, then $\alpha_0 < \ldots < \alpha_{n-1} < \alpha$ and $D(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha) < F(\alpha)$.
2. Every $\beta < F(\alpha)$ has a unique denotation $(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$ in $\mathcal{D}$, i.e. $\beta = D(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$.
3. If $(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$ is a denotation and $\gamma_0 < \ldots < \gamma_{n-1} < \gamma$, then $(c; \gamma_0, \ldots, \gamma_{n-1}; \gamma)$ is a denotation.
4. If $D(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha) \leq D(d; \alpha'_0, \ldots, \alpha'_{m-1}; \alpha)$, $\gamma_0 < \ldots < \gamma_{n-1} < \gamma$, $\gamma'_0 < \ldots < \gamma'_{m-1} < \gamma$, and for all $i < n$ and $j < m$, $\alpha_i \leq \alpha'_j \Leftrightarrow \gamma_i \leq \gamma'_j$, then
$$D(c; \gamma_0, \ldots, \gamma_{n-1}; \gamma) \leq D(d; \gamma'_0, \ldots, \gamma'_{m-1}; \gamma).$$

In a denotation $(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$, c is called the *index*, α is the *parameter* and $\alpha_0, \ldots, \alpha_{n-1}$ are the *coefficients* of the denotation. If $\beta = D(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$ the index c represents some 'algebraic' way of describing β in terms of the ordinals $\alpha_0, \ldots, \alpha_{n-1}, \alpha$.

Dilators and denotation systems are basically the same thing (cf. [10]) in that every dilator F gives rises to a denotation system D_F in the way described above and every denotation system D induces a dilator F_D by letting $F_D(\alpha)$ be the least ordinal η that does not have a denotation of the form $D(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)$ and for any arrow $f : \alpha \rightarrow \delta$ of the category **Ord** letting $F_D(f) : F_D(\alpha) \rightarrow F_D(\delta)$ be defined by

$$F_D(f)(D(c; \alpha_0, \ldots, \alpha_{n-1}; \alpha)) \ := \ D(c; f(\alpha_0), \ldots, f(\alpha_{n-1}); \delta).$$

Several of the hierarchies of ordinal functions used to engender representation systems in proof theory have been shown to fit the categorical framework.

Girard's approach has been applied to give a functorial reconstruction of the Veblen hierarchy (see [11]). In a technically difficult and dense paper [12] this work has been extended to some part of the Bachmann hierarchy up to the Bachmann-Howard ordinal (the proof-theoretic ordinal of Kripke-Platek set theory and the theory of positive inductive arithmetic definitions). Bachmann [4] defines a hierarchy of functions $(\varphi_\alpha^{\mathfrak{B}})_{\alpha\in\mathfrak{B}}$ simultaneously with a set of ordinals $\mathfrak{B}$ closed under successor such that with each limit $\lambda \in \mathfrak{B}$ is associated an increasing sequence (called a *fundamental sequence*) $\langle\lambda[\xi] : \xi < \tau_\lambda\rangle$ of ordinals $\lambda[\xi] \in \mathfrak{B}$ of makes the functions length $\tau_\lambda \leq \mathfrak{B}$ and $\lim_{\xi<\tau_\lambda} \lambda[\xi] = \lambda$. Keeping track of fundamental sequences makes the Bachmann functions $\varphi_\alpha^{\mathfrak{B}}$ particularly difficult to deal with. After the work of Bachmann, the story of ordinal representations became very complicated. Significant papers (by Isles, Bridge, Pfeiffer, Schütte, Gerber to mention a few) involved quite horrendous computations to keep track of the fundamental sequences. Feferman (cf. [7]) then proposed an entirely different method for generating a Bachmann-type hierarchy of normal functions which does not involve fundamental sequences. This led to the Aczel-Buchholz-Feferman hierarchy of fixed point free functions $\bar{\theta}_\alpha$. Weiermann [27] showed that these functions are dilators.

The ordinal representation systems that have been recast in functorial form are rather weak compared to the ones used in proof theory today. However, it is quite likely that many of them can eventually be reconstructed as based on dilators. While the theory of dilators and denotation system has been of value in illuminating an abstract property shared by many representation systems for ordinals in proof theory, it could very well turn out that this property merely scratches the surface of what defines a good ordinal representation system. It is also worth stating that the theory of dilators has not contributed to the goal of finding an ordinal analysis of Π_2^1-comprehension. It has not led to any new ideas of attaining stronger ordinal representation systems nor has it simplified existing ones.

One of the central ideas used in devising strong representation systems is to use collapsing functions that mimic reflection properties of large cardinals (cf. [18,19,22,23,24,16,25,26]). I have said something about this in [23] so I won't repeat any details here. Another interesting way of defining ordinal representation systems has been pursued by Tim Carlson [5]. In this approach the class of ordinals gets furnished with a relation of $\exists_1$ elementary substructurehood and the ordinal representations correspond to finite substructures of this class structure.

There are several other aspects of representation systems used in proof theory that make them special. Due to space limitations this cannot be discussed here (but see [24]).

3 Plan of the Talk

In the first part of the talk, I shall outline the general form of an ordinal analysis and discern some crucial properties that ordinal representation systems used in proof theory always have, one being their versatility in establishing proof-theoretic equivalences between classical non-constructive theories and intuitionistic constructive theories based on radically different ontologies.

The second part is earmarked for applications of ordinal analysis. To mention a few, this will include the characterization of provable functions and functionals, Π^0_2-conservativity between theories, and independence results, notably in connection with Kruskal's tree theorem and the graph minor theorem.

Ordinal analyses have been obtained for mathematically strong theories up to and including the subsystem of second order arithmetic based on Π^1_2-comprehension (cf. [18,21,22,17,2,3,25,26]). In the final part of the talk I will address the problem of extending this ordinal analysis to the full system of second order arithmetic.

References

1. Ackermann, W.: Begründung des, tertium non datur mittels der Hilbertschen Theorie der Widerspruchsfreiheit. Dissertation (Göttingen, 1924)
2. Arai, T.: Wellfoundedness proofs by means of non-monotonic inductive definitions. I. Π^0_2-operators. Journal of Symbolic Logic 69, 830–850 (2004)
3. Arai, T.: Proof theory for theories of ordinals. II. Π_3-reflection. Annals of Pure and Applied Logic 129, 39–92 (2004)
4. Bachmann, H.: Die Normalfunktionen und das Problem der ausgezeichneten Folgen von Ordinalzahlen. Vierteljahresschrift Naturforsch. Ges. Zürich 95, 115–147 (1950)
5. Carlson, T.: Elementary patterns of resemblance. Annals of Pure and Applied Logic 108, 19–77 (2001)
6. Feferman, S.: Systems of predicative analysis II. Representations of ordinals. Journal of Symbolic Logic 33, 193–220 (1968)
7. Feferman, S.: Proof theory: a personal report. In: Takeuti, G. (ed.) Proof Theory, 2nd edn, pp. 445–485. North-Holland, Amsterdam (1987)
8. Feferman, S.: Three conceptual problems that bug me. Unpublished lecture text for 7th Scandinavian Logic Symposium (Uppsala, 1996)
9. Gentzen, G.: Die Widerspruchsfreiheit der reinen Zahlentheorie. Mathematische Annalen 112, 493–565 (1936)
10. Girard, J.-Y.: A survey of Π^1_2-logic. Part I: Dilators. Annals of Mathematical Logic 21, 75–219 (1981)
11. Girard, J.-Y., Vauzeilles, J.: Functors and Ordinal Notations. I: A Functorial Construction of the Veblen Hierarchy. Journal of Symbolic Logic 49, 713–729 (1984)
12. Girard, J.-Y., Vauzeilles, J.: Functors and Ordinal Notations. II: A Functorial Construction of the Bachmann Hierarchy. Journal of Symbolic Logic 49, 1079–1114 (1984)
13. Kanamori, A.: The higher infinite. Springer, Berlin (1995)
14. Kreisel, G.: A survey of proof theory. Journal of Symbolic Logic 33, 321–388 (1968)
15. von Neumann, J.: Zur Hilbertschen Beweistheorie. Mathematische Zeitschrift pp. 1–46 (1926)
16. Pohlers, W.: Proof theory and ordinal analysis. Archive for Mathematical Logic 30, 311–376 (1991)
17. Pohlers, W.: Subsystems of set theory and second order number theory. In: Buss, S. (ed.) Handbook of proof theory, pp. 209–335. North-Holland, Amsterdam (1998)
18. Rathjen, M.: Ordinal notations based on a weakly Mahlo cardinal. Archive for Mathematical Logic 29, 249–263 (1990)

19. Rathjen, M.: Proof-Theoretic Analysis of KPM. Archive for Mathematical Logic 30, 377–403 (1991)
20. Rathjen, M.: How to develop proof–theoretic ordinal functions on the basis of admissible sets. Mathematical Quarterly 39, 47–54 (1993)
21. Rathjen, M.: Collapsing functions based on recursively large ordinals: A well–ordering proof for KPM. Archive for Mathematical Logic 33, 35–55 (1994)
22. Rathjen, M.: Proof theory of reflection. Annals of Pure and Applied Logic 68, 181–224 (1994)
23. Rathjen, M.: Recent advances in ordinal analysis: Π^1_2-CA and related systems. Bulletin of Symbolic Logic 1, 468–485 (1995)
24. Rathjen, M.: The realm of ordinal analysis. In: Cooper, S.B., Truss, J.K. (eds.) Sets and Proofs, pp. 219–279. Cambridge University Press, Cambridge (1999)
25. Rathjen, M.: An ordinal analysis of stability. Archive for Mathematical Logic 44, 1–62 (2005)
26. Rathjen, M.: An ordinal analysis of parameter-free Π^1_2 comprehension. Archive for Mathematical Logic 44, 263–362 (2005)
27. Weiermann, A.: A Functorial Property of the Aczel-Buchholz-Feferman Function. Journal of Symbolic Logic 59, 945–955 (1994)
28. Woodin, W.H.: Large cardinal axioms and independence: The continuum problem revisited. The Mathematical Intelligencer 16(3), 31–35 (1994)

Computable Riemann Surfaces
(Extended Abstract)

Robert Rettinger

FernUniversität Hagen
LG Komplexität und Algorithmen
Universitätsstrasse 1
D-58095 Hagen
robert.rettinger@fernuni-hagen.de

Abstract. In this paper we introduce computable and time bounded Riemann surfaces, based on the classical abstract definition by charts. Building upon this definition we discuss computable versions of several classical results, such as the existence of complete continuations of holomorphic functions, universal coverings and the uniformization theorem (for some cases).

Though we state most of our results for computable surfaces, many of them can also be transformed to a uniform version, i.e. based on representations of the class of Riemann surfaces (modulo conformal equivalence).

Keywords: computable Riemann surface, computable uniformization.

1 Introduction

The importance and influence of Riemann surfaces to many areas of mathematics, physics and even computer graphics can hardly be overestimated. Not surprisingly, there exists a vast literature on this topic even giving constructive results and implementations in many cases.

Surprisingly, however, a common computability notion for and on Riemann surfaces seems not to exist (beside a recent definition by [Hoe06] which is different to the definition we present, from the technical and philosophical point of view, see below).

We will introduce our definitions based on the abstract definition of Riemann surfaces by charts and the well developed type-2-theory of computability. This definitions immediately give computability and complexity notions in a very natural way which are in many respects compatible with the classical results.

Type-2-theory is based on representations of topological spaces, i.e. names for the elements of the topological spaces on which we want to introduce computability or complexity. We will introduce and discuss such representations for Riemann surfaces and classes of Riemann surfaces in Section 2 (computability) and Section 3 (complexity), where we restrict ourself to single Riemann surfaces. Nevertheless we give a basic construction for representations of the class of Riemann surfaces itself. The presented results can in most cases be translated to the (uniform) case, i.e. based on the representation of the class of Riemann surfaces.

A central tool of type-2 theory are computable (or continuous) reductions which lead to different equivalence classes on to the structure of representations. The usual definition of computable reductions, however, seems not totally accurate for the given representations of Riemann surfaces. To this end we relax the definitions to computable conformal reductions.

To prove basic properties of the given representations and reductions we will follow the classical line of results on homotopy and uniformization. Of course, we can only scratch on these results instead of giving the fully adequate treatment they'd deserve. Thus, this paper is rather meant to be a starting point and invitation for further research on this topics than a full treatment of the classical theory in a computable way.

Beside the translation of known results, the strong connection between analytic and algebraic aspects of Riemann surfaces should be of interest, especially because negative results on holomorphic functions are usually hard to prove. Finally there is a third important application, which we won't discuss here. Riemann surfaces seem useful to extend results, which work well for compact subsets, to a global scope.

As already mentioned there is a large literature on Riemann surfaces. For the purpose of this paper any standard book on the topic will do. An introduction to type-2-theory is given in [Wei00].

There are several results related to the presented ones. An early treatment of holomorphic functions based on type-2-theory can be found in [Mue87],[Mue93]. More recent results (on complex dynamics) can be found in a line of publications [BY06],[Bra06],[BBY07],[RW03],[Ret05]. The latter paper is one of our motivations to start a more systematic investigation of Riemann surfaces, as an adequate treatment of the dynamics even of rational functions can be done on Riemann surfaces only. A treatment of the computability of Riemann mappings theorem can be found in [Her99], a complexity treatment is given in [BBY06]. The only known strict definition of computable Riemann surfaces is given by a recent paper of van der Hoeven [Hoe06] . This definition is, however, based on a different notion of computability and excludes non-computable numbers. Even if we consider generalization of this definition (or restrict our definition to computable points) the two definitions are not equivalent.

In Section 2 we will give the definitions and representations of computable Riemann surfaces and give some basic facts on these surfaces. Furthermore we discuss reductions and the induced equivalence relation on (representations of) Riemann surfaces.

A variant of these definition adequate for complexity will be given in Section 3. There we also discuss continuations of holomorphic functions and their complete surface.

In Section 4 we prove the computability of universal Riemann surfaces and Deck transformations, ending up with computable versions of the uniformization theorem.

Because of the restricted space we give only proof ideas rather than full proofs. These will be found in the full version of this paper.

2 Computable Riemann Surfaces

In this section we will introduce computable Riemann surfaces. Actually we will give two (non-equivalent) definitions of computable Riemann surfaces (with computable and r.e. domains) to fit compact as well as non-compact Riemann surfaces. The definitions are in some way standard adoptions of abstract Riemann surfaces (given by atlases). In addition any computability structure of a Riemann surface defines a (standard) representation of the underlying manyfold.

Before we give the exact definitions we will introduce a few notations and definitions which we will need throughout this paper. For further discussions on the theory of holomorphic functions see e.g. [Ahl81]. For an adequate treatment of type-2-theory of computation see [Wei00].

A Riemann surface is a two dimensional (real) manyfold with the additional property that the transitions are holomorphic (i.e. the complex derivative exists on the open domain), that is, a Riemann surface is a connected Hausdorff space M together with a maximal atlas A. Here an atlas is a set of bijective mappings (charts) $\phi : U \to M$ so that U is a connected, bounded and open subset of $\mathbb{C}$, M is covered by the images of the charts and any two mappings $\phi_0 : U_0 \to M$ and $\phi_1 : U_1 \to M$ are holomorphically compatible. That means, that the transition $\phi_1^{-1} \circ \phi_0$ is holomorphic wherever it is defined. The atlas A is maximal iff we cannot add any holomorphically compatible chart $\phi : U \to M$ not already in A.

To simplify things in the sequel we assume that all atlases considered in this paper are given by a sequence $(\phi_i)_i$ of charts $\phi_i : U_i \to M$. Notice, that for any Riemann surface there exists such a representation of countably many charts. Furthermore we can restrict U_i to be $\mathbb{D} = \{z \in \mathbb{C} : |z| < 1\}$ and $U_i \cap U_j$ to be either empty or simply connected. We will assume that charts are normed in this way.

Examples of Riemann surfaces are $\mathbb{C}$, $\mathbb{D}$ and $\mathbb{C}_{\inf}$ where the latter is given by $\mathbb{C}_{\inf} = \mathbb{C} \cup \{\infty\}$ and the charts $\phi_1(z) = z$ and $\phi_2(z) = \frac{1}{z}$. More complex examples are the tori T_{w_1,w_2} defined as $\mathbb{C}$ modulo the group $\mathbb{Z}w_1 \times \mathbb{Z}w_2$.

To introduce computability on $\mathbb{C}$ and open subsets of $\mathbb{C}$ we will shortly repeat some notions and notations of discrete and type-2-computability theory. Let Σ be some finite set (alphabet). Then partial computable functions $f :\subseteq \Sigma^* \to \Sigma^*$ on the free monoid Σ^* can be defined by Turing-machines. This model gives naturally complexity classes by restricting the time and space the Turing-machines are allowed to use. Here the complexity is measured by means of the input length. These notions can be extended to the set Σ^∞ of infinite sequences of elements of Σ as follows: A function $f :\subseteq \Sigma^\infty \to \Sigma^\infty$ is computable iff there exists a computable function $g : \Sigma^* \times \mathbb{N} \to \Sigma^*$ so that for all u, w with $f(u) = w$ and all $n \in \mathbb{N}$ there exists exactly one $m \in \mathbb{N}$ so that $g(u|_m, n) = w_n$ is defined, where $w|_n$ denotes the initial word of w of length n. Here and henceforth we assume that natural numbers are given by their binary representation. Time and space complexity is measured by means of the parameter n.

To define computability on other spaces M we need representations, i.e. surjections $\nu :\subseteq \Sigma^\infty \to M$. A word w with $\nu(w) = m$ is also called $(\nu-)$name of m. A function f between represented spaces (M_1, ν_1) and (M_2, ν_2) is computable if we can realize f by a computable g in the sense that $\nu_2 \circ g$ equals $f \circ \nu_1$ wherever the latter is defined. Products, even for sequences of representations or names, and representations of integers and rational numbers can be define straightforwardly, for example by using linear time computable pairing functions $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. (Standard) representation of $\mathbb{R}$ and thus $\mathbb{C}$ (identified by $\mathbb{R} \times \mathbb{R}$) can be introduced as follows: A name of a real x is a sequence $(q_i)_i$ of rational numbers so that $|q_i - x| \leq 2^{-i}$ for all i. The corresponding representation is denoted by ρ.

Finally we have to introduce r.e. and computable open subsets of $\mathbb{C}$: We will call an open subset U of $\mathbb{C}$ r.e. iff there exist computable sequences $(z_i)_i$ and $(q_i)_i$ of numbers in $\mathbb{Q} + i\mathbb{Q}$ ($\mathbb{Q}(i)$ for short) and rational numbers, respectively, so that $U = \bigcup_i B_{q_i}(z_i)$ where $B_r(z)$ denotes the open ball of diameter r and center z. We call U computable, iff the distance function $d_U : \mathbb{C} \to \mathbb{R}$, $d_U(z) = \inf\{d(z, z') : z' \notin U\}$ and $d_U(z) = \inf\{d(z, z') : z' \in U\}$ for $z \in U$, $z \notin U$, respectively, where $d(z, z')$ denotes the euclidean distance on $\mathbb{C}$.

It's time to define computable Riemann surfaces:

Definition 1. *A Riemann surface* (M, A) *is* **computable with r.e. domains** *iff there exists a computable atlas with r.e. domains. That means that there exists an atlas* $(\phi_i)_i$, $\phi_i : \mathbb{D} \to M$, *and a computable mapping* $\psi :\subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{D} \to \mathbb{D}$ *so that dom*(ψ) *is r.e. and* $\psi(i, j, x) = \phi_i \circ \phi_j^{-1}(x)$ *whenever this value is defined.*

If, in addition, dom(ψ) *is computable we call* M *a* **computable Riemann surface with computable domains**.

For computable Riemann surfaces with computable domains we have full control about the domains of the charts. We will see later on that this definition will allow us to unify certain Riemann surfaces, whereas it seems hopeless to find such kind of theorems for surfaces of the relaxed kind.

By the uniformization theorem we know that any simply connected Riemann surface is computable. The first question is, wether there exist non-computable Riemann surfaces?

Theorem 1. *There exists hyperbolic and parabolic Riemann surfaces, which are not computable.*

To prove this theorem we can use a simple countability argument: Any computable atlas is based on at least one Turing-machine realizing ψ. Furthermore, any two (possibly) different Riemann surfaces defined by atlases based on the same Turing-machine are conformally equivalent, i.e. there exists a holomorphic bijection between these surfaces. However, there are uncountably many hyperbolic and parabolic Riemann surfaces, which are not conformally equivalent (in contrast to countably many Turing-machines).

One easily shows that manyfolds can have many different complex structures and each of these structures can have many different computability structures (i.e. there are computable atlases which cannot be translated into each other computably). To give a simple example consider the disk $\mathbb{D}$ with $\phi_1 = id$ and $\mathbb{D}$ together with some non-computable automorphism.Thus, once we realize that a Riemann surface is computable, we have to fix a computable atlas to talk about computability in a reasonable way. Any such atlas defines a (standard) representation of M (as a topological space) in the sense of type-2 theory. More generally, any atlas of a Riemann surface determines a representaion in the following way:

Definition 2. *The (standard) representation ν_M of a Riemann surface (defined by the atlas $(\phi_i)_i$) is defined by $\nu_M(i,\omega) = \phi_i(\rho(\omega))$.*

From now on we assume that every (computable) Riemann surface is given by a (computable) atlas and the above representation (even if not explicitly mentioned). Thus we can freely adopt computability on the considered surfaces.

In type-2-theory we usually compare representations by reductions, i.e translations of names of the represented objects. Wheras two standard representations of a Riemann surface (in the sense of the above definition) are always continuously equivalent (the representations can be translated into each other by continuous mappings) this no longer holds for computational reductions.

On the other hand, if the exact definition of M is not of importance, we can use another interesting kind of (computable) equivalence on Riemann surfaces: We call two (atlases of) Riemann surfaces **computably (conformally) equivalent** iff there exists a computable conformal mapping between them (with respect to their standard representations). Thus the next interesting question arising is wether two (computable) representations are always computably equivalent.

Theorem 2. *Let M be a computable Riemann surface. Then any two computability structures of M are computably equivalent iff M is compact.*

There is another aspect of conformal equivalence: Although there obviously does not exist a representation of all Riemann surfaces, such representations do exist if we consider the class of Riemann surfaces modulo conformal equivalence. The only ingredient we need is a proper representation of holomorphic functions, which can be done in many ways. As we need only to consider holomorphic functions $f :\subseteq U \to \mathbb{C}$ where U is an open subset of $\mathbb{D}$, we can represent U by a dense and computable sequence $(r_i)_i$ of numbers in $\mathbb{D} \cap \mathbb{Q}(i)$, where $\mathbb{Q}(i) = \mathbb{Q} + i\mathbb{Q}$, and a second sequence $(d_i)_i$ of rational numbers so that $U = \bigcup_i U_i$, where U_i is the open ball of diameter d_i and center r_i ($d_i = 0$ implies $U_i = \emptyset$). To represent the values we take a third sequence $(v_i)_i$ of numbers in $\mathbb{Q}(i)$ so that $|f(r_i) - v_i| \leq 2^{-i}$. A name of f is then given by these three sequences.

As a simple application of this definition one can show that a Riemann surface is computable iff it has a computable name (with respect to the above representation).

3 Continuation

A fundamental fact about holomorphic functions is that they are determined already by their values on a non-discrete subset of its (connected) domain. In this section we will give the corresponding result for computable Riemann surfaces with rough complexity bounds. To this end we will introduce linear and polynomial time bounded Riemann surfaces.

Finally we show the computability of complete continuations of computable holomorphic functions under a reasonable extension of the computability of holomorphic functions.

It is a simple consequence of a result by Norbert Müller ([Mue93]) for power series that, if a holomorphic function is computable on an open subset of its domain (inside $\mathbb{C}$), this holomorphic function is computable everywhere on its connected domain (containing these subset). By continuation along arcs (continues functions $\gamma : [0,1] \to M$) one can easily prove that this also holds for computable Riemann surfaces. If we assume, that the transitions $\phi_i^{-1} \circ \phi_j$ are polynomial time computable, the result in [Mue93] also shows that if a holomorphic mapping h is polynomial time computable locally, it is polynomial time computable everywhere. This only shows, that for any compact subset of the domain of h there exists a polynomial p so that h can be computed in time $O(p)$ on this compact set. To get uniform results on the time complexity we will use the uniformization theorem (see Section 4). Before we state this result we will first introduce time bounded Riemann surfaces.

Definition 3. *Let an atlas $(M, (\phi_i)_i)$ and ψ be given as in Definition 1. Then we call this atlas t-time-bounded iff ψ can be computed in time $O(t)$.*

We call $(M, (\phi_i)_i)$ locally t-time-bounded if $\psi(i, j, \cdot)$ can be computed in time $O(t)$ for all fixed pairs (i, j).

For $t(n) = n$ and t a polynomial we also say, that an atlas is linear time or polynomial time bounded, respectively. As already mentioned, continuation is easy as long as we are only interested in computability (for example by applying the results of the next section). However, if complexity comes into play, we have to do a little more. Before we give the central ideas we will first state the main result of this section. Although we will give a much more general result (concerning computability) we will also state the result on computable continuation.

Theorem 3. *Let $(M, (\phi_i)_i)$ be computable and h be a holomorphic mapping on M. If h can be computed on some open subset of the connected domain dom(h), then h can be computed on every compact subset of dom(h).*

If in addition $(M, (\phi_i)_i)$ is locally linear time bounded and h is $O(t)$-time computable on some open subset of the connected domain dom(h), then h can be computed in time at most $O(n \cdot t(n) + n^3)$ on every compact subset of dom(h).

To prove this result we want to reduce the problem to a continuation problem inside $\mathbb{D}$. The main ingredient here is the uniformization theorem: Assume we have two charts $\phi_1 : \mathbb{D} \to U$ and $\phi_2 : \mathbb{D} \to U'$ and an arc γ from $\phi_1(0)$ to $\phi_2(0)$. Furthermore let h be computable on U in time $O(t)$. Then we can choose a covering of a neighborhood of γ by a sequence ϕ_3, ϕ_4,... of charts. These charts can be replaced by a sequence ϕ'_1, ϕ'_2, ... so that the image of ϕ'_i is contained in the image of ϕ_i, the union V of these images still cover the arc γ, the transitions are polynomials over $\mathbb{Q}(i)$ and $\phi' \circ \phi^{-1}$ is linear time computable. By the uniformization theorem we can find a conformal mapping $f : \mathbb{D} \to V$. f can be approximated by a conformal mapping p over $\mathbb{Q}(i)$ so that p defines a bijection of $\mathbb{D}$ onto $V' \subset V$ so that V' still covers γ and furthermore p is a polynomial on $V' \cap \phi_1'(\mathbb{D})$. Then p is a polynomial even on $V' \cap \phi_2'(\mathbb{D})$. Thus we can translate any holomorphic mapping to a mapping on $\mathbb{D}$ and moreover this can be done efficiently.

Notice that the above result can be improved to some $O(n \cdot t(n) + n^2 \cdot p(n))$, where p is a polylogarithmic function ([**?**]). Whether the $n \cdot t(n)$ term can be improved is open.

Lets move to a more global discussion of continuation. One of the roots of Riemann surfaces is, that complete continuations of holomorphic functions (like $log(z)$) do usually not exist inside $\mathbb{C}$, leading in a natural way to Riemann surfaces.

First we need a more general definition of computable holomorphic functions: For $z_1, z_2, ..., z_n$ in $\mathbb{Q}(i)$ let $\langle z_1, ..., z_n \rangle$ denote the arc defined by the concatenation of linesegments $[z_i, z_{i+1}]$. Now we call a holomorphic function h completely computable iff there exists a computable $z \in dom(h)$ and a neighborhood U of h so that h is computable on U, and there exists a computable enumeration of all arcs $< z, z_1, ..., z_n >$ and corresponding open sets U_i with $z_i \in U_i$ for which h can be continued along these arcs so that h is defined on the U_i.

Theorem 4. *Let h be a completely computable holomorphic function. Then there exists a computable Riemann surface M and a complete continuation of h in M so that the embedding of any r.e. subset of $dom(h)$ in $\mathbb{C}$ into M and the translation of h to its continuation is computable.*

Notice however that this is not the most general statement of this kind. One easily adopts the classical construction to give the above result (and more general ones).

4 The Universal Covering Map and Uniformization

In this section we will present some results on the computability of the uniformization of simply connected computable Riemann surfaces. It is quite obvious that there exist computable Riemannn surfaces for which the uniformization is not computable. Such surfaces can for example be defined by subsets of $\mathbb{C}$ using the result of Peter Hertling ([Her99]) on Riemann mappings. Thus it seems to be important to find conditions under which the uniformization is computable.

Concerning non-simply connected computable Riemann surfaces we will show that the universal covering map of a computable Riemann surface is again computable and even the Deck-transformations are computable in this case. To this end we will also formulate a computable version of liftings.

Whereas the results of the previous sections don't depend on the kind of computability of Riemann surfaces (with computable or r.e. domains), most results in this section do.

The uniformization theorem states that any simply connected Riemann surface is conformally equivalent either to $\mathbb{C}_\infty$ (elliptic case), $\mathbb{C}$ (parabolic case) or $\mathbb{D}$ (hyperbolic case).

We will start with a negative result on uniformization for computable Riemann surfaces with r.e. domains.

Theorem 5. *There exists computable Riemann surfaces M with r.e. domains and only two charts, for which the uniformization is not computable.*

In contrast to this result we can prove that

Theorem 6. *The uniformization of any simply connected computable Riemann surface with computable domains and finitely many charts is computable.*

The main step in proving this theorem is an algorithm to merge two charts, which can be done by potential theory methods or simple approximation using computable Riemann mappings. As an immediate consequence we get that any elliptic Riemann surface is computable. For more general Riemann surfaces the existence is less simple to answer. Nevertheless we will give a first characterization of those hyperbolic computable Riemann surfaces, which do admit computable uniformizations. At the end of this section we will in addition show that any computable Riemann surfaces which is the universal covering of a computable compact Riemann surface admits computable uniformizations.

Theorem 7. *Let $(M, (\phi_i)_i)$, $\phi_i : \mathbb{D} \to U_i$, be a computable and simply connected hyperbolic Riemann surface. Then the uniformization of M is computable iff there exists a r.e. subset R of $\mathbb{N} \times \mathbb{N}$ so that for all i the set $U = \bigcup_{(i,j)\in R} U_j$ covers some neighborhood of ∞ and we have for all $c \in U$: $|c| > 2^i$, where distances are measured in the hyperbolic metric.*

We will leave the uniformization theorem for the moment and show how to get hand on simply connected coverings. It is well known, that any Riemann surface M has a unique (up to conformal equivalence) universal covering, i.e. a simply connected Riemann surface N and a locally conformal mapping $f : N \to M$ so that for any $z \in M$ there exists a neighborhood U of z so that f is a bijection of each connected component of $f^{-1}(U)$ onto U. The classical construction of the universal covering (adopted to our situation) is based on a procedure deciding wether two arcs are homotopic. For computable Riemann surfaces with finitely many arcs this can be done for both kinds of computability, i.e with r.e. or computable domains. In the general case, however, we get only a computable Riemann surface with r.e. domains:

Theorem 8. *Let M be a computable Riemann surface. Then there exists a computable universal covering with r.e. domains and the covering map is computable.*

If in addition M has finitely many charts and computable domains, then also the universal covering has computable domains.

Once we have the universal covering we can identify the original Riemann surface by the universal covering modulo the group of Deck transformations, which form a subgroup of the automorphism group of the universal covering. The main tool in the construction of deck transformations are liftings (of arcs). Let $h : (M, (\phi_i)_i) \to (M', (\phi'_i)_i)$ be a locally conformal bijective map. It is well known that any arc γ' in M' can be lifted to an arc γ in M so that $h \circ \gamma = \gamma'$.

A first (simple) version of computable liftings is stated below.

Theorem 9. *Let $h : (M, (\phi_i)_i) \to (M', (\phi'_i)_i)$ be a computable, locally conformal map of computable Riemann surfaces. Furthermore let $z' \in M'$ be computable and $z \in M$ with $h(z) = z'$ be given and γ' be a computable arc on M' with $\gamma'(0) = z'$. Then the uniquely determined arc γ on M with $\gamma(0) = z$ and $h \circ \gamma = \gamma'$ is computable.*

If M, M', z', z, γ' and h are all linear time computable then γ is linear time computable.

Similar results can be also shown for homotopies. As here are only finitely many charts involved, this can be easily proven along the classical construction. In the construction of Deck transformations we face, however, the problem that we have to find reasonable neighborhoods on which the covering map is conformal. A simple algorithm to find such neighborhoods uses tests to ensure injectivity of the covering maps. Such tests can for example be implemented using the Koebe 1/4 theorem. Before we can state this result formally we have to define computability of groups: We call a group $(G, \cdot)$ computable iff G is finite or there exists an enumeration $g_1, g_2, ...$ of all elements in G so that the sets $\{(i, j) : g_i = g_j)$ and $\{(i, j, k) : g_i \cdot g_j = g_k\}$ are r.e., i.e. are domains of computable functions.

Theorem 10. *Let N, h be a computable covering and computable covering map, respectively. Then the group of Deck transformations and thus the fundamental group of the covered Riemann surface is computable.*

In the previous situation even the set of Deck transformations are computable in a sense similar to the computability of groups. To end this section we will state a result on uniformization of computable compact Riemann surfaces. A proof can be given by using the Gauss-Bonnet theorem:

Theorem 11. *For compact computable Riemann surfaces with computable domains the uniformization of its universal covering is computable.*

References

[Ahl81] Ahlfors, L.: Complex Analysis. McGraw-Hill, New York (1981)

[BBY06] Binder, I., Braverman, M., Yampolsky, M.: On computational complexity of Riemann mapping, unpublished manuscript

[BBY07] Binder, I., Braverman, M., Yampolsky, M.: On computational complexity of Siegel Julia sets, Commun. Math. Physics

[Bra06] Braverman, M.: Parabolic Julia Sets are Polynomial Time Computable, Nonlinearity 19 (2006)

[BY06] Braverman, M., Yampolsky, M.: Non-Computable Julia Sets, Journ. Amer. Math. Soc. 19(3) (2006)

[Her99] Hertling, P.: An effective Riemann mapping theorem. Theor. Comp. Sci. 219, 225–265 (1999)

[Hoe06] van der Hoeven, J.: On Effective Analytic Continuation, unpublished manuscript (2006)

[Mue87] Müller, N.: Uniform computational complexity of Taylor series. In: Ottmann, T. (ed.) Automata, Languages and Programming. LNCS, vol. 267, pp. 435–444. Springer, Berlin (1987)

[Mue93] Müller, N.: Polynomial-Time Computation of Taylor Series In: Proc. 22 JAIIO - PANEL '93, Part 2, Buenos Aires, pp. 259–281 (1993)

[RW03] Rettinger, R., Weihrauch, K.: The computational complexity of some julia sets, STOC, pp. 177–185 (2003)

[Ret05] Rettinger, R.: A Fast Algorithm for Julia Sets of Hyperbolic Rational Functions. ENTCS 120, 145–157 (2005)

[Ret07] Rettinger, R.: On continuations of holomorphic mappings, unpublished manuscript (2007)

[Wei00] Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)

Rank Lower Bounds for the Sherali-Adams Operator

Mark Rhodes

Durham University, Department of Computer Science,
South Road, Durham, Co. Durham, DH1 3LE, UK
m.n.c.rhodes@dur.ac.uk
http://www.dur.ac.uk/m.n.c.rhodes

Abstract. We consider the Sherali-Adams (SA) operator as a proof system for integer linear programming and prove linear lower bounds on the SA rank required to prove both the pigeon hole and least number principles. We also define the size of a SA proof and show that that while the pigeon hole principle requires linear rank, it only requires at most polynomial size.

Keywords: Propositional Proof Complexity, Lift and Project Proof Systems, Rank Lower Bounds, Sherali-Adams Relaxation.

1 Introduction

This work is concerned with the field of proof complexity: the measure of efficiency of automated theorem proving methods. Most systems studied in this area are motivated by solving problems in propositional logic, most notably the satisfiability problem [6], however there are also methods such as the Lovász-Shrijver (LS) and Cutting Planes proof systems [1] that are motivated by solving integer linear programming problems, the SA operator can be viewed as one of the latter. Generally, such systems operate by taking a 0/1 programming problem and from it producing a linear programming problem with constraints which increasing resemble those in the original problem. The reason behind using this approach is that linear programming is solvable in polynomial time [3], whilst the original 0/1 programming problem is known to be NP-Complete.

The SA operator was introduced in [5] as a method for generating a hierarchy of relaxations for linear and polynomial 0/1 programming problems, in this paper we discuss the operator as a proof system. The obvious measure of proof complexity with the SA operator is the rank required. There are a number of results on the rank lower bounds of proofs using both cutting planes procedures and LS systems presented in [1] and [2]. It has been shown in [4] that the SA operator is at least a powerful as LS (i.e. every tautology which can be proved with rank k in LS can be proven with a SA proof of rank at most k). In this paper we prove two linear rank lower bounds for the SA operator. We also define a metric which allows us to view the complexity of a SA proof as the size of the proof required, we then show that rank and size are not necessarily comparable by demonstrating that the pigeon hole principle requires a SA proof of linear rank, but, just polynomial size.

2 Preliminaries

The SA operator is a procedure that produces progressively tighter formulations for 0/1 integer linear programs by introducing new variables and constraints. As with many similar systems, one begins by dropping the constraint that the variables must take integer values i.e. they can be fractional. This gives us a linear programming problem which we know to be solvable in polynomial time. By introducing new constraints, we restrict the values the variables can take until eventually, only values bounded by the convex hull of feasible solutions in the original 0/1 integer linear programming problem will satisfy them. It was shown in [5] that we can be certain to reach this state when the SA rank is one less than the number of variables in the original set of inequalities. The SA operator can be viewed as a static proof system since it simply generates a set of linear inequalities which can either be proven to have integer solutions or not, with the use of a polynomial time LP algorithm, there is no need to consider the computational path taken to obtain the contradiction or solution.

To use the SA operator we initially rewrite the given instance of the 0/1 linear programming problem Φ, by replacing each positive occurrence of the variable x_i in each inequality $\varphi \in \Phi$ with the variable $\chi_{[x_i|\emptyset]}$ and each negative occurrence with $1 - \chi_{[\emptyset|x_i]}$, e.g. the inequality $3x_1 - 4x_2 \geq 1$ is replaced with $3\chi_{[x_1|\emptyset]} - 4(1 - \chi_{[\emptyset|x_2]}) \geq 1$, which becomes $3\chi_{[x_1|\emptyset]} + 4\chi_{[\emptyset|x_2]} \geq 5$. Let $k \geq 0$ be an integer. The Sherali-Adams formulation of level-k derived from a set of converted inequalities Φ on $2n$ variables is the following:

It has a variable $\chi_{[P|N]}$ for every pair of disjoint subsets P and N of all variables appearing in Φ, where $|P \cup N| \leq \min\{k+1, n\}$. For any variable $\chi_{[P|N]}$ we will refer to the set P as the positive side of the variable and the set N as the negative side. We add the inequalities $\chi_{[P|N]} \leq 1$ and $\chi_{[P|N]} \geq 0$ for each variable in the system, ensuring the value each can take is bounded between 0 and 1. We also add the following constraints:

$$\chi_{[\emptyset|\emptyset]} = 1.$$

For all variables $\chi_{[P|N]} \in \Phi$ and all variables i in the original instance, where $|P \cup N| \leq k$ and $i \notin P \cup N$,

$$\chi_{[P\cup i|N]} + \chi_{[P|N\cup i]} = \chi_{[P|N]}.$$

Each inequality $\varphi_i \in \Phi$ multiplied by each variable $\chi_{[P|N]}$, where $|P \cup N| \leq k$. Each variable $\chi_{[A|B]} \in \varphi_i$, multiplied by $\chi_{[P|N]}$ becomes $\chi_{[A\cup P|B\cup N]}$. If the sets $A \cup P$ and $B \cup N$ are not disjoint the variable is assigned the value 0. Also, trivially $0 \times \chi_{[P|N]} = 0$ and $\chi_{[P|N]} \times 1 = \chi_{[P|N]}$, for all possible P and N. We will refer to the deriving of a new inequality by multiplying by a variable $\chi_{[P|N]}$ where $|P \cup N| = 1$ in this way as an SA multiplication.

When the SA rank is k there are $1 + \sum_{i=0}^{k} \binom{n}{i+1} \times 2^{i+1}$, different variables in total and at most polynomially times as many inequalities. Since we know linear programming is in P, if a given set of inconsistent inequalities requires at most constant rank, we can be sure that the SA operator can prove them to be inconsistent in polynomial time. It can be observed that HORNSAT and 2-SAT require SA ranks 0 and 1 respectively. If however the SA rank required is linear, we know that any algorithm assigned to solve the generated set of inequalities will take at least exponential time in the size of the original instance.

The two combinatorial principles this paper is concerned with are the Pigeon Hole Principle (PHP) and the Least Number Principle (LNP).

PHP states that for any natural number $n \geq 2$, if you tried to put $n+1$ pigeons into n holes, you would have a hole with at least two pigeons in it. It can be stated as a collection of two sets of inequalities with SA variables, which we will call the *holeset* and the *pigeonset*. When discussing PHP we use ordered pairs to represent a single variable and when the pair (i, j) appears in the positive side of a variable we take this to mean that the pigeon i is assigned to hole j, whilst if it appears in the negative side, this indicates pigeon i does not go to hole j. The *holeset* ensures that any two pigeons are not assigned to the same hole, it can be written as all the inequalities of the form $\chi_{[(i,j)|\emptyset]} + \chi_{[(i',j)|\emptyset]} \leq 1$, where $i \neq i'$, $1 \leq i, i' \leq n+1$ and $1 \leq j \leq n$ for some given n. The *pigeonset* states that each pigeon must go to a hole, as a set of inequalities this is written as $\sum_{j=1}^{n} \chi_{[(i,j)|\emptyset]} \geq 1$, for every i where $1 \leq i \leq n+1$.

LNP says that every set of n natural numbers has a smallest element, where $n \geq 1$. When discussing this principle we use ordered pairs to represent single variables. The pair (i, j) appearing in the positive side of a variable indicates that $i \prec j$ or conversely $j \succ i$, whilst the same pair appearing in the negative side of the variable means $i \succ j$ or $j \prec i$, unless $i = j$ in which case we take it to refer to some trivially true statement. The principle can be written as a system of inequalities consisting of three sets which we will refer to as *trans*, *lower* and *self*. The set *trans*, is so called because it ensures that the transitive property of the set elements is adhered to, namely that if $i \prec j$ and $j \prec k$ then $i \prec k$ must be true, this translates to the set of inequalities $\chi_{[(i,j)|\emptyset]} + \chi_{[(j,k)|\emptyset]} - \chi_{[(i,k)|\emptyset]} \leq 1$ for all i,j and k where $1 \leq i, j, k \leq n$. The set of inequalities *lower* take the form $\sum_{i=1}^{n} \chi_{[(i,j)|\emptyset]} \geq 1$ for all j, where $1 \leq j \leq n$, and state that there must be at least one element of the set smaller than element j. The final set *self* states that an element is not less than itself, this translates to the set of inequalities $\chi_{[(i,i)|\emptyset]} \leq 0$ for all i, where $1 \leq i \leq n$.

3 Rank Lower Bounds

Theorem 1. *The Pigeon Hole Principle requires at least SA level* $\lceil \frac{n+1}{2} \rceil - 2$ *to prove.*

Proof. To show this statement is true we will argue that the set of inequalities produced by SA level $\lfloor \frac{n}{2} \rfloor - 2$ can be satisfied by assigning suitable values to the variables.

The value given to the variable $\chi_{[P|N]}$ is roughly related to the probability of picking a random assignment of $\frac{n}{2}$ pigeons to $\frac{n}{2}$ holes, including all those mentioned in the pairs in the variable, and finding that the information declared within the variable fits the assignment. It is calculated as follows:

1. If P contains two pairs (i, j) and (i', j) then we can see straight away there is a contradiction and so assign the value 0.

2. Otherwise for every j, where $1 \leq j \leq n$ and j is not the second element of any pair in P, we take all pairs in N which have j as a second element, and put them in a new set we denote S_j.

3. We assign any variable not already assigned 0 the value $\frac{2}{n}^{|P|} \times \prod_{j=1}^{n} (1 - \frac{2|S_j|}{n})$.

We can see straight away that when the SA level is at most $\lfloor\frac{n}{2}\rfloor - 1$ and $n \geq 2$ all inequalities of the form $\chi_{[P|N]} \geq 0$ and $\chi_{[P|N]} \leq 1$ are trivially satisfied by the method, as is the constraint that $\chi_{[\emptyset|\emptyset]} = 1$ and that if $\chi_{[P|N]} = 0$, then $\chi_{[Q|W]} = 0$, where $P \subseteq Q$ and $N \subseteq W$.

Lemma 1. *The method provides values which satisfy the set of equations of the form $\chi_{[P \cup (i,j),N]} + \chi_{[P|N \cup (i,j)]} = \chi_{[P|N]}$ for all pigeons i, all holes j and all possible sets P and N where $|P \cup N| \leq \frac{n}{2} - 1$ and $n > 2$.*

Proof. There are three cases to consider with this lemma depending on whether or not the hole j already appears in N or P, these are (1) when j does not appear in either N or P, (2) when j appears in P and finally (3) when j does not appear in P but does appear at least once in N. Note that case two includes instances where j appears in pairs in N, as well as those where it does not. Throughout we will denote the value assigned to the variable $\chi_{[P|N]}$ as ξ.

Case 1. In this instance $\chi_{[P \cup (i,j)|N]} = \xi \times \frac{2}{n}$, since the extra pair does not affect the set S_j and therefore the only difference is that the set P is one larger. We can also see that $\chi_{[P|N \cup (i,j)]} = \xi \times \frac{n-2}{n}$. Given this, it is easy to show that the equations are satisfied for any value of ξ since substituting these values gives us the following:

$$\chi_{[P \cup (i,j)|N]} + \chi_{[P|N \cup (i,j)]} = \chi_{[P|N]}$$
$$(\xi \times \frac{2}{n}) + (\xi \times \frac{n-2}{n}) = \xi.$$

Case 2. In this instance $\chi_{[P \cup (i,j)|N]} = 0$ since we have at least two pairs in P with hole j, however we can be sure that $\chi_{[P|N \cup (i,j)]} = \xi$ as $S_j = \emptyset$ as the hole j appears in P. Using these values, the equation is trivially satisfied.

Case 3. In this situation, $\chi_{[P|N \cup (i,j)]} = \kappa \times (f - \frac{2}{n})$ where $\xi = \kappa \times f$ and f is the factor $\frac{n-(2|S_j|)}{n}$, since the additional pair in N makes S_j one larger. We can also see that $\chi_{[P \cup (i,j)|N]} = \kappa \times \frac{2}{n}$, since the factor f is removed by the addition of the positive assignment of a pigeon to hole j, yet the addition makes P one larger, hence the inclusion of the factor $\frac{2}{n}$. We can see that this gives us the required result as follows:

$$\begin{aligned} \chi_{[P\cup(i,j)|N]} + \chi_{[P|N\cup(i,j)]} &= \chi_{[P|N]} \\ \kappa \times \frac{2}{n} + \kappa \times (f - \frac{2}{n}) &= \kappa \times f \\ f - \frac{2}{n} + \frac{2}{n} &= f \end{aligned}$$

Lemma 2. *The method satisfies the set of inequalities holeset, when the SA level $\lfloor \frac{n}{2} \rfloor - 2$ is used and $n \geq 4$.*

Proof. To prove this lemma, we will split the set of all possible variables by which the inequalities *holeset* maybe multiplied, when using SA level $\lfloor \frac{n}{2} \rfloor - 2$, into 5 distinct cases and discuss each in turn. Throughout this section we will refer to the variable by which the inequalities are multiplied by as δ. The positive side of this variable will be denoted δ_P and the negative side δ_N. We will denote the value assigned by the method to the variable δ as ξ.

Case 1. $(k, j) \notin \delta_P$ and $(k, j) \notin \delta_N$ for all k where $1 \leq k \leq n + 1$.

Both $\chi_{[(i,j)|\emptyset]} \times \delta = \xi \times \frac{2}{n}$ and $\chi_{[(i',j)|\emptyset]} \times \delta = \xi \times \frac{2}{n}$ must be true as the addition of these pairs only affects the size of P, as $S_j = \emptyset$. It is easy to see that the inequalities are satisfied as when $n \geq 4$, for any possible value of ξ as follows:

$$\begin{aligned} \xi \times \frac{2}{n} + \xi \times \frac{2}{n} &\leq \xi \\ \xi \times \frac{4}{n} &\leq \xi \\ \frac{4}{n} &\leq 1 \end{aligned}$$

Case 2. At least one pair $(i'', j) \in \delta_P$, where $i'' \neq i'$ and $i'' \neq i$.

This case is trivial since both $\chi_{[(i,j)|\emptyset]} \times \delta$ and $\chi_{[(i',j)|\emptyset]} \times \delta$, contain at least two pairs with the same hole j in their positive parts P and therefore will be set to 0, giving us $0 + 0 \leq \xi$, which is true for all values of $\xi \geq 0$.

Case 3. At least one pair $(i'', j) \in \delta_N$, where $i'' \neq i'$, $i'' \neq i$ and $(k, j) \notin \delta_P$ for all k where $1 \leq k \leq n + 1$.

The addition of the pairs (i, j) and (i', j) to δ_P ensures $S_j = \emptyset$, making the inclusion of any pair (i'', j) in δ_N irrelevant, but, is sure to make P one larger since no other pair in P contains hole j. We define $\xi = \kappa \times f$ where f is the factor $\frac{n-(2|S_j|)}{n}$, giving us $\chi_{[(i,j)|\emptyset]} \times \delta = \chi_{[(i',j)|\emptyset]} \times \delta = \kappa \times \frac{2}{n}$. Using this, together with the fact that the the limit on the SA level means $|S_j| \leq \lfloor \frac{n}{2} \rfloor - 2$ we can show the inequality holds no matter what that value of ξ as follows:

$$\begin{aligned} \kappa \times \frac{2}{n} + \kappa \times \frac{2}{n} &\leq \kappa \times \frac{n - (2|S_j|)}{n} \\ \kappa \times \frac{4}{n} &\leq \kappa \times \frac{n - (2|S_j|)}{n} \\ \frac{4}{n} &\leq \frac{n - (n-4)}{n} \end{aligned}$$

Case 4. At least one of $(i,j) \in \xi_P$ and $(i',j) \in \xi_P$ is true.

If both pairs were in ξ_P then all the variables would be 0, so the inequality would be satisfied. If only one of the pairs is in ξ_P then the variable which adds pair not in ξ_P will contain two pigeons going to the hole j and hence be assigned 0. Since from lemma 1 we know adding a pair to a set can never increase the value assigned to the variable the inequality must be satisfied.

Case 5. At least one of $(i,j) \in \xi_N$ and $(i',j) \in \xi_N$ is true where $(i,j) \notin \xi_P$ and $(i',j) \notin \xi_P$.

From the definition of the SA operator we know that any variable containing the same pair in both positive and negative sides must have the value of 0 (the method does not account for such variables as their value is trivial). In the case where both statements are true, our inequalities are trivially satisfied since we have $0 + 0 \leq \xi$. In the case that one of the statements is true, we know that one of the variables must be 0, and again from lemma 1 we know the other variable can not be assigned a greater value than ξ and hence the inequality is satisfied.

Lemma 3. *The method satisfies the set of inequalities pigeonset, when the maximum SA level is* $\lfloor \frac{n}{2} \rfloor - 2$.

Proof. Throughout this proof we denote the variable by which the inequalities can be multiplied as δ. The positive side of δ is denoted δ_P, its negative side δ_N and the value assigned to it by the method is denoted as ξ.

The maximum number of pairs in δ is limited to the the maximum SA level and since each pair contains only one hole, this is also the maximum number of holes that can appear in δ. Each inequality in *pigeonset* contains n different j values. This means that we can be sure that at least $n - (\lfloor \frac{n}{2} \rfloor - 2)$ variables in any inequality in *pigeonset* that contain a hole j where $j \notin \delta$. Each of these will be assigned the value $\xi \times \frac{2}{n}$ and therefore see that even in the worst case, where all the variables of the form $\chi_{[(i,j)|\emptyset]} \times \delta$ where $j \in \delta$ are assigned the value of 0, any inequality in *pigeonset* is satisfied as follows:

$$\begin{aligned} (n - (\lfloor \frac{n}{2} \rfloor - 2)) \times (\xi \times \frac{2}{n}) &\geq \xi \\ (\frac{n}{2} + 2) \times \frac{2\xi}{n} &\geq \xi \\ \xi + \frac{4\xi}{n} &\geq \xi \end{aligned}$$

As we have proved in the previous lemmas that all the inequalities and equations produced by the SA operator running the pigeon hole principle will be satisfied by the values given by the method, our proof of theorem 1 is concluded.

Theorem 2. *LNP requires at least SA level $n-2$ to prove.*

Proof. We will argue that the set of equations produced by SA level $n-3$ can be satisfied by assigning suitable values to the variables.

The intuition behind the value we assign a variable is that it is the probability of the information contained within it being true, under the assumption that each element in the set has equal probability of being either smaller or larger than any other element. We calculate the value of the variable $\chi_{[P|N]}$ as follows:

1. Consider a directed acyclic graph (DAG) in which each of the set of n integers is represented by a node and each pair (i,j) in P is represented as an edge going from the ith node to the jth node, if the pair instead appeared in N it would be represented as an edge going from the jth node to the ith node unless $i=j$ in which case the pair is removed from N.
2. If this graph contains a cycle, including one consisting of a single edge, there is inconsistent information so we assign the value 0.
3. Remove all edges in the graph between two nodes i and j where there exists a longer path from i to j via some other nodes. This ensures we remove irrelevant information.
4. If the variable has not been assigned the value 0, we assign it the value of $\frac{1}{2}^E$ where E is the number of edges left in the graph.

For example, the variable $\chi_{[(1,2),(2,3),(1,4)|(4,3),(4,4)]}$ is assigned the same value as the variable $\chi_{[(1,2),(2,3)|(4,3)]}$, which is $\frac{1}{8}$.

It is trivial to show that the method satisfies the constraint $\chi_{[\emptyset|\emptyset]}=1$, as such a variable will generate a graph with no edges and therefore be assigned $\frac{1}{2}^0=1$. Also, that all inequalities of the form $\chi_{[P|N]}\leq 1$ and $\chi_{[P|N]}\geq 0$ for all sets P and N, will be adhered to. The members of $self$ are also trivially satisfied using this method since we assign 0 to any variable containing the pair (i,i) in its positive side therefore giving us:

$$\chi_{[P\cup(i,i)|N]}\leq 0\times\chi_{[P|N]}$$

Throughout the rest of this proof we will refer to the variable by which we multiply the inequalities when using the SA operator as $\chi_{[P|N]}$.

Lemma 4. *The inequalities in lower are satisfied using this method when the SA level is set to $n-3$ or less.*

Proof. As the right hand side of each equation is simply 1, we know that this side will take the value of whichever variable we multiply the inequality by. Therefore, when we multiply by the variable $\chi_{[P|N]}$ we get the inequality $\sum_{i=1}^{n}\chi_{[(i,j)\cup P|N]}\geq\chi_{[P|N]}$. If the graph for the variable $\chi_{[P|N]}$ contains a cycle then so too must each of the variables $\chi_{[(i,j)\cup P|N]}$, as these will be represented by the same graph with perhaps one more edge between the ith and jth nodes, if the fact that $i \prec j$ can not be derived from the other edges. In this case the inequality is trivially satisfied as it reads as follows:

$$\sum_{i=1}^{n} 0 \geq 0$$

If however the graph for the variable $\chi_{[P|N]}$ does not contain a cycle, then adding the pair (i,j) will in the worst case create a cycle, and therefore a variable with the value 0. Apart from the case where $i = j$, which will trivially give you a loop, the only way this loop can be made by adding the pair (i,j) is if there is a path from j to i, this in turn means that at least one edge must finish at i. For such an edge to exist, at least one of the following must be true, $(k,i) \in P$ or $(i,k) \in N$, for some k, where $1 \leq k \leq n$ and $k \neq i$ as otherwise the variable $\chi_{[P|N]}$ would contain a loop. Since all the variables on the left side of any inequality in *lower* contain different values for i, we can see that even in the worst case we still need at least one pair in the variable $\chi_{[P|N]}$ to render a single variable inconsistent. As SA level $n-3$ allows us at most $n-3$ pairs in $P \cup N$ and each equation in $self$ contains $n-1$ variables on the left side which do not trivially contain a cycle (i.e. all those except the one with the pair (i,i)), we can be sure that at least 2 variables will not contain a cycle. The minimum value these variables can be assigned is half the value of $\chi_{[P|N]}$, in which case the inequality can be shown to be satisfied as follows:

$$\begin{aligned} \chi_{[(k,j)\cup P|N]} + \chi_{[(k',j)\cup P|N]} &\geq \chi_{[P|N]} \\ \frac{\chi_{[P|N]}}{2} + \frac{\chi_{[P|N]}}{2} &\geq \chi_{[P|N]} \end{aligned}$$

Clearly, since this is the worst case, in all other cases where $\sum_{i=1}^{n} \chi_{[(i,j)\cup P|N]}$ has a greater value, the inequalities are also satisfied.

Lemma 5. *The inequalities in trans can be shown to be satisfied using this method.*

Proof. To prove this lemma, we consider a number of cases. In all but the first case we assume $i \neq j$, $i \neq k$ and $j \neq k$.

Case 1. When at least two of i, j or k have the same value. We can see straight away that adding an additional pair to the positive side of a variable can never increase the value assigned to the variable and that any variable which contains a pair (g,g) in its positive side, for any value g where $1 \leq g \leq n$, is assigned the value of 0. Using these statements and the fact that the left side of each inequality in *trans* has just two positive variables we can see that if any of i, j or k have the same value, except when $i = k$ and $i \neq j$, then the inequality is satisfied as one of these positive parts will contain a single edge cycle, take the value 0, and leave us with an inequality in which the single positive part remaining on the left can not be larger than the right side as it is the same variable except it has an additional pair in its positive side.

When $i = k$ and $i \neq j$, the inequality multiplied by a variable becomes $\chi_{[(i,j)\cup P|N]} + \chi_{[(j,i)\cup P|N]} \leq \chi_{[P|N]}$, we can remove the negative part of the left side as it will be given the value 0. This is satisfied by the method as if no path exists between nodes i and j or vice versa in the graph for $\chi_{[P|N]}$ then we get:

$$\frac{\chi_{[P|N]}}{2} + \frac{\chi_{[P|N]}}{2} \leq \chi_{[P|N]}$$

If however it contains one or more of these paths then at least one variable on the left side of the inequality must contain a cycle going from i to j and back again and will therefore be assigned the value 0, in which case the inequality must be satisfied as the remainder of the left is the same as the right with one extra pair in its positive side.

Case 2. When the graph for the variable $\chi_{[P|N]}$ contains a cycle. This case is trivial as all the variables on the left must also generate a graph which contains the graph for $\chi_{[P|N]}$ as a subgraph and therefore will also contain a cycle. As all cyclic graphs are assigned the value of 0, the inequality becomes $0 + 0 - 0 \leq 0$, which is trivially true.

Case 3. The graph derived from $\chi_{[P|N]}$ using the method contains a path between nodes i and k. In this case $\chi_{[P\cup(i,k)|N]}$ will generate the same graph, and hence be assigned the same value as the variable $\chi_{[P|N]}$, as the extra pair will be removed as irrelevant. Since adding extra pairs can not increase the value of a variable, even if the other variables are assigned the maximum possible value, the inequality is still satisfied as follows:

$$\begin{aligned} \chi_{[P\cup(i,j)|N]} + \chi_{[P\cup(j,k)|N]} - \chi_{[P|N]} &\leq \chi_{[P|N]} \\ \chi_{[P|N]} + \chi_{[P|N]} - \chi_{[P|N]} &\leq \chi_{[P|N]} \\ 1 + 1 - 1 &\leq 1 \end{aligned}$$

Case 4. The variable $\chi_{[P|N]}$ generates a graph containing neither a path between nodes i and k nor vice versa. In this instance, the variable $\chi_{[P\cup(i,k)|N]}$ will always generate a graph with one more edge than the one from $\chi_{[P|N]}$ and therefore be assigned half the value of $\chi_{[P|N]}$ or 0 if $\chi_{[P|N]}$ contains contradictory information. The latter instance is already covered by case 2, and the former is also covered as at least one of $\chi_{[P\cup(i,j)|N]}$ and $\chi_{[P\cup(j,k)|N]}$ must be assigned no more than half the value of $\chi_{[P|N]}$ otherwise we would have a path from i to k. Even if the other takes the maximum possible value the inequality is still satisfied as follows:

$$\begin{aligned} \chi_{[P\cup(i,j)|N]} + \chi_{[P\cup(j,k)|N]} - \frac{\chi_{[P|N]}}{2} &\leq \chi_{[P|N]} \\ \chi_{[P|N]} + \frac{\chi_{[P|N]}}{2} - \frac{\chi_{[P|N]}}{2} &\leq \chi_{[P|N]} \\ \chi_{[P|N]} &\leq \chi_{[P|N]} \end{aligned}$$

Case 5. The graph for $\chi_{[P|N]}$ contains a path between nodes k and i and does not contain a cycle. When this happens, the variable $\chi_{[P\cup(i,k)|N]}$ will take the value 0. If the graph for the variable $\chi_{[P|N]}$ contains neither a path between nodes i and j nor between nodes j and k then both $\chi_{[P\cup(i,j)|N]}$ and $\chi_{[P\cup(j,k)|N]}$ will take the value of half that of $\chi_{[P|N]}$, thus giving:

$$\frac{\chi_{[P|N]}}{2} + \frac{\chi_{[P|N]}}{2} - 0 \leq \chi_{[P|N]}$$
$$\frac{1}{2} + \frac{1}{2} \leq 1$$

If however $\chi_{[P|N]}$ does contain one of the aforementioned paths, then the variable whose graph contains the path not included in $\chi_{[P|N]}$ will always contain a cycle and therefore be assigned the value of 0. Even though the other variable left will be set to the same value as $\chi_{[P|N]}$ the inequality is clearly satisfied.

Lemma 6. *The method satisfies the equations introduced by the SA operator of the form* $\chi_{[P\cup(i,j)|N]} + \chi_{[P|N\cup(i,j)]} = \chi_{[P|N]}$ *for all sets* P *and* N *and all* i *and* j *where* $1 \leq i, j \leq n$ *and* $(i,j) \notin P \cup N$.

Proof. We can show that the the set of equations are satisfied through the following four cases:

Case 1. When the graph associated with $\chi_{[P|N]}$ contains a cycle. This case is trivial since the cycle will also be contained within the graphs of the other variable giving us $0 + 0 = 0$.

Case 2. When $i = j$ and the graph for $\chi_{[P|N]}$. This is satisfied as we ignore such a pair when it is added to the set N and we set any variable to be 0 when it is added to the set P giving us an equation equivalent to $0 + \chi_{[P|N]} = \chi_{[P|N]}$.

Case 3. When $i \neq j$, the graph for $\chi_{[P|N]}$ is acyclic and does not contain a path between nodes i and j or vice versa. In this case both the variable on the left side of the equation will take the value of $\frac{\chi_{[P|N]}}{2}$ because the graph they produce will always contain the extra edge giving us $\frac{\chi_{[P|N]}}{2} + \frac{\chi_{[P|N]}}{2} = \chi_{[P|N]}$.

Case 4. When $i \neq j$ and the graph for $\chi_{[P|N]}$ is acyclic and contains a path between nodes i and j or vice versa. The graph containing the same path as $\chi_{[P|N]}$ will clearly be assigned the same value as it since the extra edge will be removed. The other variable will add an edge which completes a cycle and hence be assigned 0. The equation is then clearly satisfied.

Since we have proven all the sets of inequalities produced by the SA operator will be satisfied, our proof of theorem 2 is concluded.

4 SA Proof Size

Another measure by which we can judge the lengths of SA proofs, rather than the required rank, is the combined number of SA multiplications required to reach a contradiction. This measure we shall refer to as the size of the proof. We will now show that for the SA operator, the required rank does not reflect the required proof size, specifically that the proof required for PHP is at most polynomial in size, whilst we have already shown the required rank to be linear.

Theorem 3. *The size SA of the proof required for PHP is at most polynomial.*

Proof. We will derive the contradiction by deducing $\sum_{j=1}^{n}\sum_{i=1}^{n+1}\chi_{[(i,j)|\emptyset]} \geq n+1$ and $\sum_{j=1}^{n}\sum_{i=1}^{n+1}\chi_{[(i,j)|\emptyset]} \leq n$. The first inequality can be derived straight away by adding up all equations in the set *pigeonset*, we will prove the other can be proven by induction.

Lemma 7. *With any inequality in holeset,* $\chi_{[(i,j)|\emptyset]} + \chi_{[(i',j)|\emptyset]} \leq 1$*, we can derive that* $\chi_{[(i,j)|(i',j)]} = \chi_{[(i,j)|\emptyset]}$ *with a single SA multiplication.*

Proof. We accomplish this by multiplying the inequality by the variable $\chi_{[(i',j)|\emptyset]}$ then proceeding as follows:

$$\begin{aligned}
\chi_{[(i,j),(i'j)|\emptyset]} + \chi_{[(i',j)|\emptyset]} &\leq \chi_{[(i',j)|\emptyset]}\\
\chi_{[(i,j),(i'j)|\emptyset]} &\leq 0\\
\chi_{[(i,j),(i'j)|\emptyset]} &= 0\\
\chi_{[(i,j),(i'j)|\emptyset]} + \chi_{[(i,j)|(i'j)]} &= \chi_{[(i,j)|\emptyset]}\\
\chi_{[(i,j)|(i'j)]} &= \chi_{[(i,j)|\emptyset]}
\end{aligned}$$

Lemma 8. *We can derive* $\sum_{i=1}^{q+1}\chi_{[(i,j)|\emptyset]} \leq 1$ *from* $\sum_{i=1}^{q}\chi_{[(i,j)|\emptyset]} \leq 1$ *with one SA multiplication.*

Proof. To accomplish this we multiply by the variable $\chi_{[\emptyset|(q+1,j)]}$ and proceed as follows:

$$\begin{aligned}
\sum_{i=1}^{q}\chi_{[(i,j)|(q+1,j)]} &\leq \chi_{[\emptyset|(q+1,j)]}\\
\sum_{i=1}^{q}\chi_{[(i,j)|(q+1,j)]} &\leq 1 - \chi_{[(q+1,j)|\emptyset]}\\
\chi_{[(q+1,j)|\emptyset]} + \sum_{i=1}^{q}\chi_{[(i,j)|(q+1,j)]} &\leq 1\\
\chi_{[(q+1,j)|\emptyset]} + \sum_{i=1}^{q}\chi_{[(i,j)|(q+1,j)]} &\leq 1\\
\sum_{i=1}^{q+1}\chi_{[(i,j)|\emptyset]} &\leq 1
\end{aligned}$$

Note that we make use of lemma 7 for each variable within the summation to accomplish the final step of this derivation.

By continually applying lemma 8 to each of the inequalities of the form $\chi_{[(1,j)|\emptyset]} + \chi_{[(2,j)|\emptyset]} \leq 1$ in *holeset* then we will be able to derive $\sum_{j=1}^{n+1}\chi_{[(i,j)|\emptyset]} \leq 1$ for each value i and hence by adding all such inequalities be able to derive $\sum_{j=1}^{n}\sum_{i=1}^{n+1}\chi_{[(i,j)|\emptyset]} \leq n$. The number of multiplications required to accomplish this was clearly polynomial in n.

5 Further Work

In this paper we have shown the SA operator to require linear rank for both PHP and LNP and that PHP only requires at most a polynomially sized SA proof. Whilst a number of more powerful LS systems have been proposed in [2], no extensions to the SA operator have currently been developed to make it as powerful as these new systems. We will look to add additional rules to enhance the SA operator and prove linear rank lower bounds for these systems. One major challenge with this will be finding candidate tautologies, since it is difficult to argue about randomly generated formulas, but, as with most powerful proof systems, the standard set of combinatorial principles and other well known tautologies are likely to yield constant rank bounds.

References

1. Buresh-Oppenheim, J., Galesi, N., Hoory, S., Magen, A., Pitassi, T.: Rank bounds and integrality gaps for cutting planes procedures. Theory of Computing 2, 65–90 (2006)
2. Grigoriev, D., Hirsch, E.A., Pasechnik, D.V.: Complexity of semi-algebraic proofs. Moscow Mathematical Journal 2(4), 647–679 (2002)
3. Khachian, L.G.: A polynomial time algorithm for linear programming. Doklady Akademii Nauk SSSR, n.s., 244(5), pp. 1063–1096. English translation in Soviet Math. Dokl. 20, pp. 191–194 (1979)
4. Laurent, M.: A comparison of the Sherali-Adams, Lovász-Schrijver and Lasserre relaxations for 0-1 programming. Mathematics of Operations Research 28(3), 470–496 (2003)
5. Sherali, H.D., Adams, W.P.: A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM Journal of Discrete Mathematics 3, 411–430 (1990)
6. Warners, J.P.: Nonlinear approaches to satisfiability problems. PhD thesis, Eindhoven University of Technology, The Netherlands (1999)

Infinite Computations and a Hierarchy in Δ_3

Branislav Rovan and Ľuboš Steskal[⋆]

Department of Computer Science,
Faculty of Mathematics Physics and Informatics,
Comenius University, 84248 Bratislava, Slovakia
{rovan,steskal}@dcs.fmph.uniba.sk

Abstract. We present a hierarchy of families between the Σ_2 and Δ_3 levels of the arithmetic hierarchy. The structure of the top five levels of this hierarchy is in some sense similar to the structure of the Chomsky hierarchy, while the bottom levels are reminiscent of the bounded oracle query hierarchy.

Keywords: Super Turing computation, arithmetical hierarchy, infinite computation, Chomsky hierarchy.

1 Introduction

Several models of computation having "super Turing" computational power have been proposed and studied in the literature (e.g., Relativistic Computation using Malament-Hogarth space-times [EN02] [WvL02], Infinite Time Turing Machines [HL00], Neural Networks with real value weights[SS95], or the Accelerated Turing Machine proposed by Russel, Blake and Weyl). Some of these machines share the capability to perform infinite computations. In this paper we introduce a new model which allows us to separate the finite and infinite parts of computations and to retrieve information about infinite computations. Our model allows us to show that more sophisticated postprocessing of the information provided by infinite computation increases computational power.

2 Notation and Preliminaries

In this part, we introduce a new computational model– the *Display Turing Machine*[1]. We shall assume basic notations of the formal languages theory (see e.g. [HU79]). Let Λ be a finite alphabet. We denote by Λ^* the set of all finite words over Λ, by Λ^ω the set of all infinite words over Λ, and by Λ^∞ the union of Λ^* and Λ^ω. We shall also use the standard notation for the degrees in the arithmetic hierarchy $(\Sigma_i, \Pi_i, \Delta_i)$. Let $\mathcal{C}$ be a family of machines. We shall denote by $\mathcal{L}(\mathcal{C})$ the family of corresponding languages.

[⋆] This research was supported in part by the grant VEGA 1/3106/06.

[1] We shall use the acronym TMD (Turing Machine with Display) to avoid confusion with the Deterministic Turing Machine.

Informally, the TMD is a standard one–tape deterministic Turing Machine equipped with an additional tape called the display tape. During the computation both tapes are used in a standard way. Initially the first tape contains the input word and the second tape, the display, is blank. We shall be interested in the content of the display tape after the computation is performed. We shall use it, e.g., to determine the acceptance (in $TMDC$ – *the Display Turing Machine with Control*). The key feature of the TMD is its ability to display information about infinite computations. The content of the display tape represents "the result" of an infinite computation in the following way. The resulting content of each tape square of the display tape is either a letter of the tape alphabet (in case the content of the square stops changing after finitely many steps of the computation) or a special symbol † (in case the content of the square changes infinitely many times during the computation). We shall also say that the TMD transforms the input word to a display word (the limit contents of the display tape in case of an infinite computation).

Let us now formally define TMD and $TMDC$.

Definition 1. *A* Display Turing Machine *is a* 5*-tuple* $(\Lambda, \Gamma, \delta, K, q_0)$ *where* $\Lambda \subseteq \Gamma$ *are the input and auxiliary alphabets,* B *and* $\$$ *in* $\Gamma - \Lambda$ *are special symbols for a blank space and the beginning of the display tape,* $\dagger \notin \Gamma$ *is the special symbol mentioned above,* K *is a finite set of all states of the machine,* $q_0 \in K$ *is the initial state, and* $\delta\colon K \times \Gamma^2 \to K \times (\{\Gamma - \{B\}\} \times \{-1, 0, 1\})^2$ *is the transition function such, that* $\delta(q, a_0, a_1) = (p, a_0', d_0, \$, d_1)$ *if and only if* $a_1 = \$$*, and* d_1 *never equals* -1 *if* a_1 *is* $\$$.

Definition 2. *A* Configuration of TMD T *is an element of the set* $K \times B(\Gamma - \{B\})^*B \times \mathbb{N} \times \$(\Gamma - \{B, \$\})^*B \times \mathbb{N}$

Configurations are interpreted in the usual way. For example in (p, BwB, i, $\$vB$, j), p denotes the current state of the machine, w content of the first tape, the head on the first tape is on the i–th square, v is the content of the display tape and j gives the position of the display tape head.

Definition 3. A computational step of a TMD T *is the relation* $\vdash_T$ *on configurations of TMD defined by* $(p, Bu_0u_1 \dots u_nB, k, \$v_0v_1 \dots v_mB, l) \vdash_T (q, Bu_0u_1 \dots u_{k-1}au_{k+1} \dots u_nB, k + d_0, \$v_0v_1 \dots v_{l-1}bv_{l+1} \dots v_mB, l + d_1) \iff \delta(p, u_k, v_l) = (q, a, d_0, b, d_1)$*. We omit the index* T *and shall write only* $\vdash$ *if it is obvious what machine we are referring to.*

The result $T(w)$ of the computation of the TMD T on the word w is the "limit" content of the display tape defined formally as follows.

Definition 4. *Let* T *be an* TMD *and* $w \in \Lambda^*$*. Let* $D^{T(w)} = \{d^{T(w)}(n)\}_{n=0}^{\infty}$ *denote the sequence of contents of the display tape of the machine* T *working on the input* w*. Let* $D_i^{T(w)} = \{d_i^{T(w)}(n)\}_{n=0}^{\infty}$ *denote the sequence of letters written on the* i*-th square of the display tape during the computation of* T *on the input* w *(where the beginning of the tape* $\$$ *is considered to be the minus first square).*

Let $\overline{d}_i^{T(w)}$ *be the limit of* $D_i^{T(w)}$ *if it converges,* † *if it does not. If there exists an* l *such that* $l = \min\{i | D_i$ *converges to* $B\}$*, then we say that*

$$T(w) = \overline{d}_0^{T(w)} \overline{d}_1^{T(w)} \dots \overline{d}_{l-1}^{T(w)}$$

is the result of the TMD T *with* w *on input.*

If there is no such l *then* the result of the TMD T *with* w *on input is the infinite word*

$$T(w) = \overline{d}_0^{T(w)} \overline{d}_1^{T(w)} \overline{d}_2^{T(w)} \dots$$

We have defined TMD as a machine transforming an input word by a (possibly infinite) computation to a (possibly infinite) display word. We shall now turn TMD into a language accepting device by using the resulting display word to determine the acceptance of the input word.

Definition 5. *The* Display Turing Machine with Control *is a pair* $A = (T, S)$ *such, that* T *is an TMD and* S *is a set. We shall call* S *the* Control set *of* A *and sometimes refer to it as* $C(A)$*. The language accepted by* A *is the set* $L(A) = \{w \in \Lambda^* | T(w) \in S\}$*.*

We shall now define a few useful functions and notations.

Notation 1. *Let* $f(n)\colon \mathbb{N} \to \mathbb{N}$ *be a nondecreasing function. We denote by* $\mathcal{T}_{f(n)}$ *the family of all* TMD *for which the size of their result on input words of length at most* n *does not exceed* $f(n)$*.*

Definition 6. *Let* T *be a* TMD*.* $\#_\dagger T(w)$ *shall denote the number of* † *symbols in* $T(w)$*.*

Definition 7. $PAR(u)$ *shall denote a predicate that is true if the number of non–*† *symbols in* u *is even*[2]*.*

We shall use the following notation in our study of subfamilies of $TMDC$ based on restrictions on the size of the display tape and the use of † in control languages.

Notation 2

– *Let* $(\mathcal{T}_{f(n)}, \mathcal{L})$ *denote a family of Display Turing Machines with Control such that* $A = (T, S)$ *is in* $(\mathcal{T}_{f(n)}, \mathcal{L})$ *iff* A *is in* $\mathcal{T}_{f(n)}$ *and* S *is in* $\mathcal{L}$ *and there is no* w *in* S *containing the symbol* †*.*
– *Let* $(\mathcal{T}^\dagger_{f(n)}, \mathcal{L})$ *denote a family of Display Turing Machines with Control such that* $A = (T, S)$ *is in* $(\mathcal{T}^\dagger_{f(n)}, \mathcal{L})$ *iff* A *is in* $\mathcal{T}_{f(n)}$ *and* S *is in* $\mathcal{L}$*.*

In what follows, we might omit some indices if they are clearly implied by the context.

[2] We shall use this predicate for $T(w)$, the result of a TMD T on an input w, and write $PAR\ T(w)$ instead of $PAR(T(w))$.

3 Restricted Display Turing Machines with Control

We shall consider $TMDC$ with restrictions on the size of the display tape and on the use of † in control languages. We shall also study the influence of the Chomsky type of control languages on the power of $TMDC$

3.1 $TMDC$ with Single Square Display

To demonstrate some techniques we shall first consider $TMDC$ having the display tape restricted to a single square. We shall see that even this minimal increase in resources over the standard TM substantially increases their power.

Theorem 1. *$\mathcal{L}(\mathcal{T}_1, \mathcal{R}) = \Sigma_2$.*

Proof. Consider $\Sigma_2 \subseteq \mathcal{L}(\mathcal{T}_1, \mathcal{R})$. Let L be in Σ_2 and let M be an oracle machine (with a Σ_1 oracle) accepting L. We shall construct a $(\mathcal{T}_1, \mathcal{R})$ machine $D = (T, \{1\})$ accepting L as follows.

As long as there is no oracle query, T shall simulate the computation of M in a standard way. Once the machine M enters a oracle query state, the computation of T shall fork into two different computations. One computation follows the simulation of M presuming a negative answer to the query, while the other computation actually computes its real outcome. If the simulation of M enters another query state, the computation shall fork again, thus creating more parallel "threads". If the computation of an oracle query eventually halts with an affirmative answer (meaning, that T has made a wrong guess for the oracle outcome), then the entire computation is rolled back to the point where the query was made (and the computation of respective threads is abandoned). Now the simulation continues as described above, only that there is no fork and the query answer is positive.

If the simulated machine M enters an accepting state, the symbol 1 is written onto the output square. Should the simulated computation of M halt but not accept, the symbol 0 is written on the display. Whilst the simulation of M is in process (by this we mean solely the computation of M, not those of the oracle queries), the content of the display oscillates between two contents (say a and b).

Clearly, T shall eventually know the outcome of each query asked by M. Thus T shall write the symbol 1 onto the display tape and not rewrite it from some time on iff M accepts its input. On the other hand, if the computation of M does not halt at all, the content of the display square is altered infinitely many times, thus it will result into the † symbol. Thus $(T, \{1\})$ accepts an input word if and only if M does.

Consider $\mathcal{L}(\mathcal{T}_1, \mathcal{R}) \subseteq \Sigma_2$. Let $A = (T, C)$ be in $(\mathcal{T}_1, \mathcal{R})$[3]. We shall construct an oracle machine M (with a Σ_1 oracle) accepting w if and only if w is in $L(A)$. All we need to do is to determine the result $T(w)$ of the display machine T in finite time assuming, that the output belongs to C. We shall utilize the fact, that if $T(w) \in C$ then $T(w) \neq$ † (in other words the number of changes of the

[3] Since the display size is limited to 1 only some finite languages from $\mathcal{R}$ are usable.

display is finite). So M shall simulate A and ask the oracle, whether the content of the display square shall be changed during the forthcoming computation (it can be seen, that such a query is recursively enumerable, thus in Σ_1). If it does not change, we shall just check if it is in C. If it does, it must happen after some finite time so we can simulate the computation to the point of the change and then ask the oracle again. Thus $w \in L(A) \iff w \in L(M)$. $\square$

We see, that the infinite computation of the Display Turing Machine can be used to answer (arbitrary many, but finite) number of queries to the Halting problem of standard Turing machines. It can be seen that if all the words in the control set are †–free it suffices to consider control sets consisting of words of length one. Thus having display tape of size $k, k > 1$, does not increase the power of $TMDC$ in this case. We shall now consider the question whether $TMDC$ can compute more in case the words in the control language may contain the symbol †.

Theorem 2. $\{L | \exists A = (T, C) \in (\mathcal{T}_1^{\dagger}, \mathcal{R}) : A \textit{ accepts } L \textit{ and } \dagger \in C\} = \Pi_2$

Proof. Consider $\Pi_2 \subseteq \mathcal{L}(\mathcal{T}_1^{\dagger}, \mathcal{R})$ Let $L \in \Pi_2$ then its complement $\overline{L} \in \Sigma_2$. Thus there exists an oracle Turing machine M with a Σ_1 oracle accepting $\overline{L}$. We shall construct an equivalent Display Turing Machine with Control $A = (T, \{\dagger\})$ such that $T(w) = \dagger \iff w \notin L(M)$. Let T' be the display machine used in the proof of Theorem 1. We already know, that $T'(w) = \dagger$ if the computation of M requires infinitely many oracle queries. We shall construct T by a slight modification of T'. The only reason, why T' does not satisfy our needs is that if the computation of M needs only a finite number of queries and then rejects the input, T' does not produce a † on its display. So T shall differ from T' in such case. T shall simulate M just as T' does, but should the simulation come to the point, where M would reject, T shall enter a loop in which it continually rewrites the content of the display square. Of course, the parallel simulation of oracle queries does not stop. This implies, that $w \in L(M) \Rightarrow T(w) = 1$ as in the proof of Theorem 1 and $w \notin L(M) \Rightarrow T(w) = \dagger \Rightarrow w \in L(A)$.

Consider $\{L | \exists A = (T, C) \in (\mathcal{T}_1^{\dagger}, \mathcal{R}) : A$ accepts L and $\dagger \in C\} \subseteq \Pi_2$. Let $A = (T, C)$ be in $(\mathcal{T}_1^{\dagger}, \mathcal{R})$ such that $\dagger \in C(A)$. We shall construct an oracle machine M with a Σ_1 oracle such that $L(M) = \overline{L(A)}$. M shall simulate the computation of T on w and try to determine the display $T(w)$ of T. It shall ask the oracle, whether given a configuration of the display machine T, the content of the display changes after finite number of computational steps. If it does not, M will know the resulting content of the display $\overline{d}_0$ of T and shall accept if $\overline{d}_0 \notin C$ and vice versa.

If the content is to be rewritten, then M can simulate the computation of T to that point and then ask again. If $T(w) = \dagger$ then M never halts and thus does not accept w. Thus $w \in L(M) \iff T(w) \notin C$. $\square$

Corollary 1. $\mathcal{L}(\mathcal{T}_1^{\dagger}, \mathcal{R}) = \Pi_2 \cup \Sigma_2$

The above results show that in one square, we can compute the answer to one oracle call to a Σ_2 oracle. So if there was a larger display tape, we might be able to compute more. Since we know (see [Bie95]), that for an oracle machine $k+1$ queries to Σ_2 are better then k queries, it would be of interest to have an analogous result for our model. This shall be the goal of the next section.

3.2 The Tape–Size Hierarchy

In this section, we shall examine the relation between oracle machines with a Σ_2 oracle and Display Turing Machines with Control. The main result of this section is the proof that, for all k, the display tape of fixed size $k+1$ is stronger than the one of size k. We shall call the hierarchy established in this way the *Tape–size Hierarchy.* Since all machines studied in this section have a constant bound on their display tape, their control languages may be restricted to finite sets and are thus regular.

We shall first formulate an easy to see but useful "concatenation" property of the Display Turing Machines with Control. For any two machines, we can create a third one by "gluing" these two machines together. Conversely, if we have one display machine using a display tape of size k, we can "split" it into two machines with display tapes of sizes a and $k-a$ each computing the respective part of the result of the original machine.

Lemma 1 (Concatenation Lemma)

i. Let (T_1, S_1) and (T_2, S_2) be two $TMDC$. Then there exists a machine (T, S) such that $L(T, S) = L(T_1, S_1) \cdot L(T_2, S_2)$.
ii. Let T be a TMD. Then there exist two machines T_1 and T_2 such that for all w a $T(w) = T_1(w) \cdot T_2(w)$ and if $|T(w)| > 0$ then $|T_1(w)| = 1$.

We might refer to T by writing $T_1 \cdot T_2$. We shall now show how to simulate an oracle machine that uses at most k oracle calls by using an exponentially long display tape.

Lemma 2. *Let M be an oracle machine that is allowed to make at most k oracle calls (to a Σ_2 oracle). Then $L(M) \in \mathcal{L}(\mathcal{T}^{\dagger}_{2^{k+1}-1}, \mathcal{R})$*

Proof. We shall find a $TMDC$ $A = (T, C)$ simulating M. The proof is based on the fact, that if the computation of M is limited to k queries, then there are 2^k possible computations depending on the outcome of the queries.

T shall compute the result of M for each possible computation as well as the result of each possible query. This obviously requires a display tape of length not exceeding $2^{k+1}-1$. From this information, A can easily determine the proper computation of M on w. □

Since our computational model with fixed size display tape is similar to the model of a Turing machine making parallel queries presented by Richard Biegel in [Bie95] [BGH89], we shall try to use similar proof techniques.

We shall start by examining the properties of the predicate $PAR\,T(w)$ which is true if the number of non–† symbols in $T(w)$ is even.

Lemma 3. *If* $(\forall m \in \mathbb{N})(\exists n \in \mathbb{N})(\exists T \in \mathcal{T}_n) : (PAR(T) \notin \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$ *then*[4] $(\forall i \in \mathbb{N}) : \mathcal{L}(\mathcal{T}_i^\dagger, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}_{i+1}^\dagger, \mathcal{R})$.

Proof. Let n' be the maximum, for which all the machines in $\mathcal{T}_{n'}$ have their PAR language acceptable by a machine from $\mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R})$. Thus there exists a $T' \in \mathcal{T}_{n'+1}$ such, that $PAR(T') \notin \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R})$. Using the Concatenation Lemma (Lemma 1) we know, that $T' = T_n \cdot T_1$ where T_n has its display tape of size n and T_1 has its display tape of size 1.

Since $PAR(T_n) \in \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}) \wedge PAR(T_1) \in \mathcal{L}(\mathcal{T}_1^\dagger, \mathcal{R})$ we can find a machine in $(\mathcal{T}_{m+1}^\dagger, \mathcal{R})$ accepting $PAR(T')$. Thus $\mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}_{m+1}^\dagger, \mathcal{R})$. □

We shall now show, that the assumption of the previous Lemma is true and thus that its consequence is true as well. We shall use the following known fact.

Lemma 4. *Let* $n \in \mathbb{N}$ *and let* $bin_i(n)$ *denote the* i*-th bit in the binary encoding of* n. *Then* $bin_i(n) = bin_0\binom{n}{2^i}$.

Lemma 5. *If* $(\exists m \in \mathbb{N})(\forall n \in \mathbb{N})(\forall T \in \mathcal{T}_n) : (PAR(T) \in \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$ *then* $(\forall k \in N)(\forall T \in \mathcal{T}_k)(\exists T' \in \mathcal{T}_{m \cdot \lceil \lg k \rceil}$ *and there exists a recursive function* f *such, that* $f(T'(w)) = \#_\dagger T(w)$.

Proof. T' shall use $PAR(T)$ to compute $\#_\dagger T(w)$ bit by bit. It is obvious that knowing $PAR\,T(w)$ is sufficient to determine the value of the last bit of $\#_\dagger T(w)$. Let n denote the number of non-† symbols in $T(w)$. Obviously $n = k - \#_\dagger T\,(w)$. Let T_i denote the display machine computing the content of the i-th output square of T. Let S be a subset of $\{1, 2, \cdots, k\}$. Let T_S be a display machine returning 1 if $(\forall i \in S)(T_i(w) \neq \dagger$ and returning † otherwise.

Let $\mathcal{S}_{2^j}$ be the set of all 2^j element subsets of $\{1, \cdots, k\}$ and let $\mathbb{T}_{2^j}$ be the machine $\mathbb{T}_{2^j} = \bigodot_{S \in \mathcal{S}_{2^j}} T_S$ (where $\bigodot$ denotes the concatenation operator).

Then $PAR\,\mathbb{T}_{2^j}(w) = bin_j\,(n)$. This follows from the fact, that $PAR\,\mathbb{T}_{2^j}(w) = bin_0\binom{n}{2^j} = bin_j(n)$.

Let $PAR(\mathbb{T}_l) = (T_{PAR(\mathbb{T}_l)}, C)$. Let us presume, that $T_{PAR(\mathbb{T}_l)}$ always produces an output of size m. Let J_k be the set of all powers of 2 smaller or equal to k. Then $T' = \bigodot_{l \in J_k} \left(T_{PAR(\mathbb{T}_l)}\right)$. Since each PAR needs only m squares, T' needs $m \cdot \lceil \lg k \rceil$ squares. Now, f can transform each block of length m to a 0 or 1 depending on its correspondence to A and thus compute $\#_\dagger T(w)$. It can be easily seen that f is recursive. □

Lemma 6. $(\forall m \in \mathbb{N})(\exists n \in \mathbb{N})(\exists T \in \mathcal{T}_n) : (PAR(T) \notin \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$

Proof. By contradiction. Let $(\exists m \in \mathbb{N})(\forall n \in \mathbb{N})(\forall T \in \mathcal{T}_n) : (PAR(T) \in \mathcal{L}(\mathcal{T}_m^\dagger, \mathcal{R}))$ be true. Let M be an oracle Turing machine with a Σ_2 oracle using no more then k oracle queries such, that there is no oracle Turing machine accepting the same language using less then k queries. Then by Lemma 2 there is $TMDC\ D = (T, C) \in (\mathcal{T}_{2^{k+1}-1}^\dagger, \mathcal{R})$ accepting $L(M)$. Then by Lemma 5 there is T' and f' with $T' \in \mathcal{T}_{m \cdot \lceil \lg 2^{k+1} - 1 \rceil}$ computing $\#_\dagger T$ and a T'' and f'' with

[4] By $PAR(T)$ we mean the language of all words w, for which $PAR\,T(w)$ is true.

$T'' \in \mathcal{T}_{m \cdot \lceil \lg m + \lg (k+1) \rceil}$ computing $\#_\dagger T'$. We shall construct a Σ_2–oracle machine M' simulating M with only $m \cdot \lceil \lg m + \lg (k+1) \rceil + 1$ queries.

First of all M' shall compute the value $\#_\dagger T'$. Due to Corollary 1 and the recursiveness of f'' this will require only $m \cdot \lceil \lg m + \lg (k+1) \rceil$ queries. With the value $\#_\dagger T'$ given, there exists a Σ_1–oracle Turing machine M'' computing $\#_\dagger T$. This follows from Theorem 1 and from the fact that M'' "knows" when to halt. Analogously, there is another Σ_1–oracle Turing machine M''' computing T and accepting if the result of T is in C. Thus M' needs only one more query to determine the acceptance of M''' and thus the acceptance of D and M.

This clearly contradicts $m \cdot \lceil \lg m + \lg (k+1) \rceil + 1 < k$ for k big enough. □

Thus Lemmas 3 and 6 imply that

Theorem 3. $k < l \Rightarrow \mathcal{L}(\mathcal{T}_k^\dagger, \mathcal{L}) \subsetneq \mathcal{L}(\mathcal{T}_l^\dagger, \mathcal{L})$

We can see, that there is an infinite hierarchy of machines with constant display tape sizes.

3.3 The Extended Chomsky Hierarchy

In this section we shall examine the impact of placing Chomsky like constrains on the Control language on the $TMDC$ computational power. We shall also impose a finiteness restriction on the display machine output.

Notation 3. *By $\mathcal{T}_{<\omega}^\dagger$ we denote the set of all TMD having for each v on input an output on the display tape of finite size.*

We shall often refer to the machines accepting control languages as Control Machines. We shall show (for Chomsky hierarchy[5]), that using stronger Control we can accept more languages. In our proofs, we shall use the diagonalization argument.

Lemma 7 (diagonalization). *Let $\mathcal{C}$ be a family of machines and let there exist a code for each machine from this family. Then no machine in $\mathcal{C}$ can accept the language consisting of codes of machines rejecting their own codes. We shall denote this language by $D_\mathcal{C}$ and call it the diagonal language for $\mathcal{C}$.*

We shall now show, that regular Control sets are weaker then context–free Control sets. This result shall be achieved by providing a machine using a context–free Control accepting the diagonal language $D_{\mathcal{L}(\mathcal{T}_{<\omega}^\dagger, \mathcal{R})}$.

Theorem 4. $\mathcal{L}(\mathcal{T}_{<\omega}^\dagger, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}_{<\omega}^\dagger, \mathcal{L}_{CF})$

Proof. Let A be a finite automaton and let $M = (T, L(A)) \in (\mathcal{T}_{<\omega}^\dagger, \mathcal{R})$. Let $\langle T \rangle \langle A \rangle$ be a (proper) code of M. The machine $M_D = (T_D, L(A_D))$ (where A_D

[5] It is the hierarchy $\mathcal{R} \subsetneq \mathcal{L}_{CF} \subsetneq \mathcal{L}_{ECS} \subsetneq \mathcal{L}_{REC} \subsetneq \mathcal{L}_{RE}$ of regular (accepted by finite automata), context-free (accepted by push-down automata), extended context-sensitive (accepted by lineary bounded automata), recursive and recursively enumerable languages respectively.

is a push–down automaton) accepting the diagonal language shall work in the following way. Given $\langle T\rangle\langle A\rangle$ on input, T_D shall output the output of T concatenated with all words of size $|T(\langle T\rangle\langle A\rangle)|$ (we shall denote this number by P) and one bit holding the information, whether A accepts it (as shown below)

$T(\langle T\rangle\langle A\rangle)$	w_1^R	$A(w_1)$	w_2^R	$A(w_2)$	$\cdots$	w_P^R	$A(w_P)$

A_D shall work in the following way. At first, it shall read the output $T(\langle T\rangle\langle A\rangle)$ into its push-down stack and continue to move along the display tape. If A_D is about to read $T(\langle T\rangle\langle A\rangle)^R$ (A_D can nondeterministically guess this) it shall compare it letter by letter with the content of its push-down stack. Then A_D reads the next letter. It is the symbol 1 if $\langle T\rangle\langle A\rangle \in L(A)$ and 0 otherwise. If it was the symbol 0, A_D accepts, otherwise rejects.

Thus we created a machine $T_D \in (\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{CF})$ accepting $D_{\mathcal{L}(\mathcal{T}^\dagger_{<\omega},\mathcal{R})}$. □

Theorem 5. $\mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{CF}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{ECS})$

Proof. Let the code of a machine $M = (T, L(A)) \in (\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{CF})$ be the string $\langle T\rangle\langle A\rangle$ where $\langle T\rangle$ is the code of the display machine T and $\langle A\rangle$ is the code for the push-down automaton accepting $L(A)$. Then the machine $M_D = (T_D, L(A_D)) \in (\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{ECS})$ for accepting the diagonal language operates similarly as in the previous proof, only the reverses are not necessary any more and T_D simulates the computation of a push–down automaton. □

Proofs showing that $\mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{ECS}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{REC})$ as well as $\mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{REC}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{RE})$ differ from the previous proof only in small technical details.

Thus, we have proved the existence of a Chomsky like hierarchy. We shall call it the Extended Chomsky hierarchy.

$$\mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{R}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{CF}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{ECS}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{REC}) \subsetneq \mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{RE})$$

4 Conclusion

To conclude our study we shall prove, that by increasing the power of the Control as shown in the previous section, the models remain in the domain of the Δ_3 level of the arithmetic hierarchy.

Theorem 6. $\mathcal{L}(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{RE}) \subseteq \Delta_3$

Proof. Let $A = (T, C)$ be a machine in $(\mathcal{T}^\dagger_{<\omega}, \mathcal{L}_{RE})$. As we have seen in the Section 3, an oracle machine M with a Σ_2 oracle needs only one query to determine, if the content of the i-th output square of T is the † symbol. If T is working over the alphabet Γ it can be easily seen, that M needs at most $|\Gamma| + 1$ queries to determine the exact output of the i-th query. Since $T \in \mathcal{T}_{<\omega}$, the output $T(w)$ has finite size for each input. Thus, the machine M needs at most

$$|T(w)| \cdot |\Gamma| + 1$$

queries to determine the output of T.

Since there is a standard Turing machine accepting C, M needs only one more query to find out, if $T(w)$ is in C. This implies, that M is an always halting machine, thus $\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE}) \subseteq \Delta_3$. □

We can now summarize all our results. In Section 3.2, we established existence of a Tape–size hierarchy. Since all the control languages in the Tape–size hierarchy are regular, one can easily see that all degrees of this hierarchy are in the lowest level of the Extended Chomsky hierarchy. By combining all our results we obtain the following structure of the Σ_3 level of the Arithmetical hierarchy.

Σ_3
Δ_3
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{RE})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{REC})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{ECS})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{L}_{CF})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{<\omega}, \mathcal{R})$
$\vdots$
$\mathcal{L}(\mathcal{T}^{\dagger}_{3}, \mathcal{R})$
$\mathcal{L}(\mathcal{T}^{\dagger}_{2}, \mathcal{R})$
$\Pi_2 \cup \Sigma_2 = \mathcal{L}(\mathcal{T}^{\dagger}_{1}, \mathcal{R})$
$\Sigma_2 = \mathcal{L}(\mathcal{T}_{1}, \mathcal{R})$

Some open questions. Our study could be continued by examining the properties of more machines coupled together, i.e., using one $TMDC$ as control in another $TMDC$. It would also be interesting to show exact relation of the degrees in our Tape–size hierarchy to the bounded query hierarchy and to show, that each degree of the arithmetical hierarchy contains its own Extended–Chomsky hierarchy.

References

[BGH89] Biegel, R., Gasarch, W.I., Hay, L.: Bounded query classes and the difference hierarchy. Archive for Mathematical Logic, 29(2) (1989)

[Bie95] Biegel, R.: Query-limited reducibilities. Dissertation at Stanford University (1995)

[EN02] Etesi, G., Németi, I.: Non-turing computations via malament-hogarth space-times. Int. J. Theor. Phys, 41, 2002. see also: http://arXiv.org/abs/gr-qc/0104023.

[HL00] Hamkins, J.D., Lewis, A.: Infinite time turing machines. Journal of Symbolic Logic, 65(2) (2000)

[HU79] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, London (1979)

[SS95] Siegelmann, H.T., Sontag, E.D.: On the computational power of neural nets. Journal of Computer and System Sciences 50(1), 132–150 (1995)

[WvL02] Wiedermann, J., van Leeuwen, J.: Relativistic computers and non-uniform complexity theory (2002)

Natural Computing: A Natural and Timely Trend for Natural Sciences and Science of Computation

Grzegorz Rozenberg

Leiden Institute of Advanced Computer Science, Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
Department of Computer Science, University of Colorado at Boulder,
Boulder, CO 80309, USA

Natural computing refers to computation taking place in nature and to human-designed computation inspired by nature. When complex phenomena going on in nature are viewed as computational processes, our understanding of these phenomena and of the essence of computation is enhanced. On the other hand human-designed computing inspired by nature has had already a big impact on the development of computer science (think, e.g., about neural computing, evolutionary computing, molecular computing and quantum computing).

The currently accepted/used notion of computation is actually the result of formalizing the concept of a calculation. For example, the Turing machine model explicitly formalizes mental activities of a person performing a calculation (following a finite set of rules).

To a great majority of the contemporary society computer science associates only with the Information and Communication Technology (ICT). Indeed, computer science is largely responsible for the continuous impressive progress of ICT, as it designs both “hard and soft” instrumentarium needed for this progress. However, from the scientific point of view there is much more to computer science than ICT. Most importantly, computer science has developed into the science of information processing, and as such it is a fundamental science for other scientific disciplines.

This development goes hand in hand with the development of the “information trend” in a number of scientific disciplines which adopted Information and Information Processing as their central notions and thinking habits. Biology and physics are prime examples of such disciplines - here computer science provides not only the instrumentarium (such as powerful computers and software), but also a way of thinking.

Also, the only common denominator for scientific research done in so many and so diverse areas of computer science is the study of various aspects of information processing. Clearly, in this context, the term “informatics” widely used in Europe is much better than the term “computer science”, with the latter stipulating that a particular instrument, viz. a computer, is the main research topic of our discipline.

Research in computer science is genuinely interdisciplinary and in this way natural computing forms a bridge between informatics and natural science. Natural computing had already a big and lasting impact on the development of informatics, and in particular it contributed to our understanding of the notion of computation.

One of the central questions of Natural Computing is: “How does nature compute?”. It is a very challenging question, as the term “compute” in the context of nature is difficult to define, and certainly very difficult to formalize. One will have to redefine the notion of computation so that it accommodates also information processing taking place in nature. This will lead to a grand interdisciplinary science of computation that shows up already now on the horizon. How far away are we from this horizon? Nobody knows, but we have to be patient: it took (at least) 300 years to understand, formalize and utilize the notion of human-designed computation/calculation !!!

References

Cardelli, L.: Abstract machines of systems biology. In: Priami, C., Merelli, E., Gonzalez, P., Omicini, A. (eds.) Transactions on Computational Systems Biology III. LNCS (LNBI), vol. 3737, pp. 145–168. Springer, Heidelberg (2005)

Chen, J., Jonoska, N., Rozenberg, G. (eds.): Nanotechnology: Science and Computation. Springer, Berlin (2006)

Davidson, E.H.: The Regulatory Genome: Gene Regulatory Networks in Development and Evolution, Academic Press/Elsevier, Burlington (2006)

Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta Informaticae 76, 1–18 (2006)

Kitano, H.: Systems biology: a brief overview. Science 295, 1662–1664 (2002)

Lindenmayer, A.: Mathematical nodels for cellular interaction in development, I and II. Journal of Theoretical Biology 18, 280–315 (1968)

Lindenmayer, A., Rozenberg, G.: Introduction, In: Lindenmayer, A., Rozenberg, G. (eds.) Automata, Languages, Development, v–vi, North Holland, Amsterdam (1976)

Lloyd, S.: Programming the Universe: A quantum computer scientist takes on the cosmos. Jonathan Cape, London (2006)

McCulloch, W.S., Pitts, W.H.: A logical calculus of the ideas immanent in neural nets. Bulletin of Mathematical Biophysics 5, 115–133 (1943)

Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, New York (1980)

Turing, A.M.: The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London, B 237, 37–72 (1952)

Biochemical Reactions as Computations

Andrzej Ehrenfeucht[1] and Grzegorz Rozenberg[1,2]

[1] Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309, USA
[2] Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

Two main mechanisms behind the functioning of biochemical reactions are facilitation and inhibition; these mechanisms are also central for the interaction between biochemical reactions. This observation underlies the theory of reaction systems which is a formal framework for the investigation of biochemical reactions, and especially interactions between them.

More specifically, a biochemical reaction is formalized as follows. A (formal) *reaction* is a triplet $a = (R_a, I_a, P_a)$, where R_a is the set of *reactants*, I_a is the set of *inhibitors*, and P_a is the set of *products*. The intuition behind this formal notion of a reaction corresponds in a straightforward way to the functioning of a biochemical reaction: reaction a converts/transforms the set of reactants R_a into the product set P_a providing that it is not inhibited by (one or more) inhibitors from I_a. Therefore, more formally, the result of applying reaction a to a set T, denoted by $res_a(T)$, is conditional: if T separates R_a from I_a (i.e., if R_a is included in T and I_a is disjoint with T), then a is enabled on (applicable to) T, otherwise a is not enabled on (not applicable to) T. If a is enabled on T, then a transforms the set of reactants into the product set, and so $res_a(T) = P_a$; otherwise $res_a(T)$ is the empty set.

Then, the notion of transformation by a single reaction is extended to sets of reactions, and the dynamics of *reaction systems* (which are essentially sets of reactions) is investigated through the notion of an interactive process. Such a process is a sequence of states, where each state is a set (a subset of the background set fixed for a given reaction system) which is a union of two sets: the result of transforming the previous state in the sequence, and a context set (which may, e.g., represent an interaction with "the rest of the system" or "the environment").

In developing the theory of reaction systems we adhere to a number of assumptions (axioms) that hold for a great number of biochemical reactions. First of all, in our approach reactions are primary while structures are secondary: reactions create states rather than transform states as is the case in traditional models in theoretical computer science. Another way to express this is to say that reactions create the environment rather than they work in an environment. We assume the "threshold supply" of elements (molecules): either an element is present, and then there is "enough" of it, or an element is not present.

Thus we do not have counting here (we work with sets rather than multisets), and therefore we present here a qualitative rather than quantitative analysis of interactions between reactions. Another important assumption we make is that there is no permanency of elements: if "nothing" happens to an element (it is not a reactant for any active reaction), then it ceases to exist. The only way to keep an element present is to sustain it by a suitable reaction – sustaining an element requires an effort, it is not for free ("life/existence must be sustained!").

As we argue in the lecture, the axioms/assumptions underlying the functioning of biochemical reactions, and hence the underlying assumptions of our model, are very different (often ortogonal to) the underlying axioms of the vast majority of models in theoretical computer science. This lecture introduces reaction systems to the audience of scientists interested in natural computation. It provides the basic definitions, illustrates them by providing biology (genetic regulatory networks) as well as computer science (counters) oriented examples, relates this model to a couple of traditional models of computation (elementary net systems and boolean functions), and proves some basic properties of reaction systems. We will also outline a theory of formation of "ordered" biochemical substructures (modules).

References

Ehrenfeucht, A., Rozenberg, G.: Basic notions of reaction systems. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 27–29. Springer, Heidelberg (2004)

Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta Informaticae 76, 1–18 (2006)

Ehrenfeucht, A., Rozenberg, G.: Events and modules in reaction systems, Theoretical Computer Science (To appear)

Doing Without Turing Machines: Constructivism and Formal Topology

Giovanni Sambin

Dipartimento di Matematica Pura ed Applicata
via Trieste 63, 35121 Padova, Italy
sambin@math.unipd.it

Abstract. We add some new insights, and thus hopefully contribute to give new impetus, to an old theme: constructive mathematics, and topology in particular, can be thought of as an abstract way to deal with computation.

Keywords: constructive mathematics, minimalist foundation, pointfree topology, formal topology.

With recent technological developments, computability theory is growing into the science of information processing. Also in this new field mathematics can still play a crucial role. A first necessary step is to abandon a static view of mathematics, as in the classical foundation which sees mathematics as the science of absolute truths. More fruitful is a dynamic view, in which a foundation is seen as the choice of which kind of information is considered relevant, and thus must be kept in the process of abstraction. For example, the choice for intuitionistic logic means that one cares of proofs of propositions, and not only of their truth; the proofs-as-programs principle reminds of the impact of this on computability. Similarly, the set theoretic postulates characterizing the various brands of constructive mathematics correspond to different choices in the management of information.

Constructivism as conceived by E. Bishop [1] and P. Martin-Löf [4] is characterized by a computational view: any statement of mathematics is meaningful only if it has a clear computational (or even numerical) content. Topology permeates the book [4], which deals mostly with four key examples of topological spaces. Since they are introduced by means of basic neighbourhoods given primitively, [4] has rightly been seen as a precursor of pointfree topology. The link with computation is also evident, since a point is defined as a *recursively enumerable* subset of intervals, satisfying certain closure conditions (and the author's intention is witnessed also by a complete treatment of computability given in the first chapter).

With the advent of formal topology in the 80s [5], the predicative and pointfree approach has been extended to topological spaces in general. The link with computability is however less visible, since one speaks of sets, subsets, relations, etc. with no explicit mention of computations. However, the strictly predicative treatment allows one to say that formal topology *respects* computational content;

roughly speaking, any amount of effectiveness one puts in, is fully preserved by all definitions and theorems. This is guaranteed by realizability models.

A more recent global project, of which the first published act is [3], seems to shed new light on the link between topology and computation. As argued in [3], one needs a foundation with two distinct levels of abstraction. First, a ground type theory, which deals with computations and actually acts as an abstract programming language. Then a properly extensional theory, in which sets are closed under quotients, functions are graphs, etc. Integration between these two levels is due to the fact that the latter is obtained from the former by abstraction in a controlled way, so that implementation remains possible. Roughly speaking, the extensional level is obtained by forgetting only that information which can be restored (see [2]).

As a consequence, the treatment of some effectively presented structures (like frames or domains) can be done avoiding any explicit mention of recursive functions and their indices, that is, one can do without Turing machines. In fact, this information can be restored whenever wished.

The foundation introduced in [3] was called minimalist since, contrary to Martin-Löf's type theory, it does not validate the axiom of choice AC!. This brings to two additional advantages. Firstly, the absence of AC! and the presence of two levels makes the minimalist foundation compatible also with an impredicative foundation, like topos theory. This means that communication is open with an important tradition in constructive mathematics, that is the theory of locales. Secondly, the absence of AC! allows for a better integration between the computational and the geometric, or infinitary, aspect of mathematics. In fact, while the pointfree part remains effective, the notion of formal point is openly infinitary. A predicative foundation takes care of not confusing these two aspects. However, if AC! holds, a formal point (in Baire space, for example) is reduced to a law-like sequence. The absence of AC! means that formal points can be identified with choice sequences, and so also the geometric aspect is respected.

References

1. Bishop, E.: Foundations of constructive analysis. McGraw-Hill Book Co, New York (1967)
2. Maietti, M.E.: Quotients over minimal type theory. this volume
3. Maietti, M.E., Sambin, G.: Toward a miminalist foundation for constructive mathematics, in From Sets and Types to Topology and Analysis. In: Crosilla, L., Schuster, P. (eds.) Towards practicable foundations for constructive mathematics, Oxford Logic Guides, pp. 91–114. Clarendon Press, Oxford (2005)
4. Martin-Löf, P.: Notes on Constructive Mathematics, Almqvist & Wiksell (1970)
5. Sambin, G.: Intuitionistic formal spaces – a first communication, in Mathematical Logic and its Applications, Skordev, D. (ed.), Plenum, pp. 187–204 (1987)

Problems as Solutions

Peter Schuster

Mathematisches Institut, Universität München
Theresienstr. 39, 80333 München, Germany
pschust@math.lmu.de

Abstract. If a continuous function on a complete metric space has approximate roots and in a uniform manner at most one root, then it actually has a root. We validate this heuristic principle in Bishop–style constructive mathematics without countable choice, following Richman's way of defining the completion of a metric space as the set of all locations.

MSC (2000): Primary 03F60; Secondary 03E25, 26E40, 54E50.

Keywords: Metric spaces, completeness, uniform continuity, unique existence, constructive mathematics, countable choice.

1 Introduction

This note is conceived in the realm of Bishop's constructive mathematics [6,7,9]. As compared with the—then dubbed classical—customary way of doing mathematics, the principal characteristic of the framework created by Bishop is the exclusive use of intuitionistic logic. According to Richman [21], this allows to view Bishop's setting as a generalisation of classical mathematics.

Moreover, we follow Richman's proposal, first put forward in [22], to perform constructive mathematics à la Bishop without countable choice. For a discussion of this approach with references we refer to [25]; see also [24] for early observations in the context of the present paper.

Doing without countable choice is further indispensable because we want our work to be expressible in the constructive Zermelo–Fraenkel set theory (CZF) begun by Aczel with [1]. Countable choice, namely, does not belong to CZF. Details on this and on CZF in general can be found in [3,20].

In the sequel, however, we will repeatedly use the principle of unique choice.[1] Put in set–theoretic terms, this principle says that, for arbitrary sets A and B, if R is a subset of $A \times B$ such that for each $a \in A$ there is a uniquely determined $b \in B$ with $(a, b) \in R$, then there is a function $f : A \to B$ with $(a, f(a)) \in R$ for every $a \in A$. Now unique choice is trivial also in CZF where, as common in set theory, the function–as–graphs paradigm is assumed: the given relation $R \subseteq A \times B$ is already the desired function $f : A \to B$.

To recollect the necessary prerequisites from [26], let (S, d) be a metric space with at least one point. We often suppose that S is complete—or even that it is

[1] Or, as it is sometimes called, the principle of "non–choice".

compact in the sense of Bishop: that is, totally bounded and complete. (Until we adopt the different definition proposed by Richman, completeness means that every Cauchy sequence converges.) Also, let δ, ε and x, x', y, y' always denote positive rational numbers and points of S, respectively. Note that $x \neq y$ stands for $d(x, y) > 0$, which is to say that $d(x, y) \geqslant \delta$ for some δ. Furthermore, let $F : S \to \mathbb{R}$ denote a non–negative continuous function. It will be specified as occasion demands whether F is supposed to be sequentially, pointwise, or uniformly continuous. The way in which these notions are related to each other is recalled with references in [26].

The conditional existence problem

If $\inf_{x \in S} F(x) = 0$*, is there any* $\xi \in S$ *with* $F(\xi) = 0$ *?*

subsumes, with $F(x) = d(y, x) - \operatorname{dist}(y, S)$, the search for the best approximations in S to a point y of a metric space T comprising S as a subspace. A best approximation, namely, is nothing but a point of S at which the uniformly continuous function $d(y, \cdot) : S \to \mathbb{R}$ attains its infimum $\operatorname{dist}(y, S)$. Note that if $S \subseteq T$ is totally bounded, then it is located in T: that is, $\operatorname{dist}(y, S) = \inf_{x \in S} d(y, x)$ can be computed for each $y \in T$. This, however, does not mean that a best approximation can always be constructed even if S is compact—see below.

An alternative formulation of the existence problem we are looking at is

If $F(x) = 0$ *has approximate solutions in* S*, does it have an exact solution in* S *?*

In more logical terms, the problem consists in the validity of the uniformity principle

$$\forall \varepsilon > 0 \, \exists x \in S \ \ (F(x) < \varepsilon) \ \ \Rightarrow \ \ \exists \xi \in S \, \forall \varepsilon > 0 \ \ (F(\xi) < \varepsilon) \ .$$

Given the antecedent of this implication, a natural first attempt to obtain its consequent is to choose—by countable choice—a sequence (x_n) in S with

$$\forall n \ (F(x_n) < 1/n) \ . \tag{1}$$

In classical analysis, one can proceed by saying that if S is compact, then (x_n) has a cluster point ξ in S, for which $F(\xi) = 0$ if, in addition, F is sequentially continuous. To do so, one needs to invoke the Bolzano–Weierstrass theorem (BWT) that every sequence in a compact metric space has a cluster point.

By a consequence of BWT, the minimum theorem (MIN) which says that every continuous function on a compact metric space has a minimum, the problem considered above would anyway have been solved with a single stroke of the pen. However, MIN and thus BWT are essentially non–constructive. We refer to [26] for an outline of this, including references.

The non–constructive character of MIN notwithstanding, there is a constructive proof [9, Chapter 2, Theorem 4.5] that $\inf_{x \in S} F(x)$ can be computed whenever F is uniformly continuous, and S is totally bounded. So the question remains under which circumstances our problem allows for a constructive solution. In other words, when does a uniformly continuous function on a compact metric space have a minimum: that is, attain its infimum?

2 With Countable Choice

A function $F : S \to \mathbb{R}$ with $\inf F = 0$ has *uniformly at most one* minimum [26] if

$$\forall\delta\,\exists\varepsilon\,\forall x, y\ (F(x) < \varepsilon \wedge F(y) < \varepsilon\ \Rightarrow\ d(x, y) < \delta)$$

or, equivalently,

$$\forall\delta\,\exists\varepsilon\,\forall x, y\ (d(x, y) \geqslant \delta \Rightarrow F(x) \geqslant \varepsilon \vee F(y) \geqslant \varepsilon)\ .$$

(To see this equivalence, consider the function $(x, y) \mapsto F(x) + F(y)$.) If F has uniformly at most one minimum, then F has *at most one* minimum [4]: that is,

$$\forall x, y\ (x \neq y\ \Rightarrow\ F(x) > 0 \vee F(y) > 0)\ . \tag{2}$$

By substituting $F - c$ for F, one can adapt the definition of "F has (uniformly) at most one minimum" to the case of a function $F : S \to \mathbb{R}$ with arbitrary infimum $c \in \mathbb{R}$.

If F has at most one minimum, then the point—if it exists—at which F attains its infimum 0 is uniquely determined:

$$\forall x, y\ (F(x) = 0 \wedge F(y) = 0\ \Rightarrow\ d(x, y) = 0)\ . \tag{3}$$

If F has uniformly at most one minimum, then the unique $\xi \in S$ with $F(\xi) = 0$ (if there is any) is even a *strong* minimum [4]: that is,

$$\forall\delta\,\exists\varepsilon\,\forall x\ (F(x) < \varepsilon \Rightarrow d(x, \xi) < \delta)\ .$$

Conversely, if F has a strong minimum, then F has uniformly at most one minimum.

As the author has shown in [26], Brouwer's fan theorem for decidable bars is equivalent to the statement that, for uniformly continuous functions on a compact metric space, the condition that F has uniformly at most one minimum follows from its non–uniform counterpart that F has at most one minimum. (One half of this equivalence corresponds to the rule from [14, Theorem 4.3].) This classification in the spirit of the constructive reverse mathematics—as propagated by Ishihara [12,13] and others—sharpens an earlier result obtained jointly with Berger and Bridges [4] (see also [5]).

Let S be a complete metric space, and $F : S \to \mathbb{R}$ a sequentially continuous function with $\inf F = 0$. As recalled in Proposition 2.2 of [26], if F has uniformly at most one minimum, then F has a minimum: that is, attains its infimum 0. (This is analogous to the rule from [14, Theorem 4.4].) To see this, one starts as in the classical argument discussed before: by $\inf F = 0$ one can choose—notably by countable choice—a sequence (x_n) in S with (1). Now one can profit from the additional hypothesis that F has uniformly at most one minimum, which ensures that (x_n) is a Cauchy sequence. Since S is complete, the sequence (x_n) has a limit ξ in S, for which $F(\xi) = 0$ by the sequential continuity of F.

Being essentially folklore, this argument has some history. The presumably first printed occurrences with a uniform uniqueness condition are Theorem 4 of

Lifschitz's [17],[2] and Problem 10 in Chapter 2 of Bridges and Richman's [9].[3] (The work Bridges did in the 1980s was centred around non–uniform uniqueness; see [8] for an overview and for references.) The very concept of uniform uniqueness was then used intensively by Kohlenbach from the early 1990s [14], first in the context of best approximations; see also the survey [15].

To be more precise on the latter, with the rules from [14, Section 4] a formal classical proof of the non–uniform uniqueness of the desired solution is converted into a formal constructive proof first of its uniform uniqueness (actually of the presence of a so–called modulus of uniqueness), and next of its existence as a point, which, finally, varies continuously with the parameters. In each of those rules, moreover, a choice principle for quantifier–free formulas is eliminated.

By the argument sketched above, one obtains a constructive solution, which *a fortiori* is unique, of the aforementioned problem under the additional hypothesis that F has uniformly at most one minimum—and if countable choice is taken for granted. Even if S fails to be complete, the given data are converted—by a simple invocation of countable choice—into an element of the completion of S: namely, into the Cauchy sequence (x_n) in S.

3 Without Countable Choice

To solve our problem without countable choice, we first recall from [22] Richman's particular way to define the completion of S as the set $\hat{S}$ of all locations on S. (In [23] he has used the wider concept of a locater.) Similar definitions were given before by Burden–Mulvey [10], Mulvey [18], and Stolzenberg [27]; recent occurrences of analogous concepts in point–free topology include the work by Vickers [28,29] and Palmgren [19].

Let $\mathbb{R}$ denote the set of Dedekind reals understood—as in [22]—as (located) lower cuts in $\mathbb{Q}$. A *location* on a metric space S is a function $f : S \to \mathbb{R}$ with $\inf f = 0$ and

$$|f(x) - f(y)| \leqslant d(x,y) \leqslant f(x) + f(y) \ .$$

The set $\hat{S}$ of all locations on S is a metric space whose metric is given by

$$d(f,g) = \sup |f - g| = \inf (f + g) \ .$$

We note in passing that $\mathbb{R}$ is a set in CZF [3]; more generally, if S is a set in CZF, then so is $\hat{S}$, for it can be separated from the set $\mathbb{R}^S$. To prove this one only needs, in addition to exponentiation and (restricted) separation, a binary form of the principle of fullness [11] (see also [2]).

Again according to [22], there is the isometric embedding

$$S \hookrightarrow \hat{S} \, , \ z \mapsto d(z, \cdot) \ ,$$

[2] There is no proof of Lifschitz's Theorem 4 in the English translation of the Russian original from 1971. In the context of unique existence Kreinovich [16] also refers to a metatheorem by M.G. Gelfond from 1972.

[3] This was communicated to the authors by P. Aczel.

along which we identify each point of S with its image in $\hat{S}$, and the whole of S with the corresponding subspace of $\hat{S}$. For all $f \in \hat{S}$ and $z \in S$ we have

$$d(f, z) = f(z) \;;$$

whence each location on S measures the distance between itself and the points of S. Moreover, S is dense in $\hat{S}$, and if S equals $\hat{S}$, then S is said to be *complete.* Needless to say, $\hat{S}$ is complete; in particular, $\mathbb{R}$ is complete, for $\mathbb{R} \cong \hat{\mathbb{Q}}$.

The following is hidden in the proof of [22, Theorem 4]. Let $h : S \to \mathbb{R}$ and $a \in \mathbb{R}$ with

$$\inf_{x \in S} |h(x) - a| = 0 \,. \tag{4}$$

For every $g : S \to \mathbb{R}$ and $b \in \mathbb{R}$, we use

$$\lim_{h(x) \to a} g(x) = b \tag{5}$$

as a shorthand for

$$\forall \varepsilon\, \exists \delta\, \forall x \; (|h(x) - a| < \delta \Rightarrow |g(x) - b| < \varepsilon) \;. \tag{6}$$

(In the specific case of $S = \mathbb{R}$ and $h(x) = x$ this is nothing but $\lim_{x \to a} g(x) = b$.) In view of (4) the notation (5) is well–defined by (6), because the limit—if it exists—is uniquely determined. In fact, if there is any $b \in \mathbb{R}$ with (6), then $b = c$ for every $c \in \mathbb{R}$ satisfying (6) with c in place of b.

However, when does $\lim_{h(x) \to a} g(x)$ exist in $\mathbb{R}$ for any given $g : S \to \mathbb{R}$? A sufficient condition for the existence of the limit is that g satisfies

$$\forall \varepsilon\, \exists \delta\, \forall x, y \; (|h(x) - a| < \delta \wedge |h(y) - a| < \delta \Rightarrow |g(x) - g(y)| < \varepsilon) \;. \tag{7}$$

In fact, if (7) holds, then the subset L of $\mathbb{Q}$ characterised by

$$r \in L \Leftrightarrow \exists s \in \mathbb{Q} \; [r < s \wedge \exists \delta\, \forall x \; (|h(x) - a| < \delta \Rightarrow s < g(x))]$$

is a lower cut in $\mathbb{Q}$ defining $b \in \mathbb{R}$ with $\lim_{h(x) \to a} g(x) = b$. Note that condition (7) is also necessary for the existence of the limit.

Back to [22], every $f \in \hat{S}$ is uniformly continuous (in fact, it is Lipschitz continuous with Lipschitz constant 1), has uniformly at most one minimum, and satisfies

$$f(y) = \lim_{f(x) \to 0} d(x, y) \;.$$

Each $\varphi : S \to T$ that is uniformly continuous on bounded subsets extends uniquely to a mapping $\hat{\varphi} : \hat{S} \to \hat{T}$ with

$$\hat{\varphi}(f)(z) = \lim_{f(x) \to 0} d(\varphi(x), z)$$

for every $z \in T$, which is uniformly continuous on bounded subsets. If S and T are complete, then $\hat{\varphi} = \varphi$.

Having at hand all this material from [22], we can solve our problem without countable choice.

Lemma 1. *Let $F : S \to \mathbb{R}$ be a function on a metric space S with* $\inf F = 0$. *If F is uniformly continuous and has uniformly at most one minimum, then*

$$f(x) = \lim_{F(y)\to 0} d(x,y)$$

defines $f \in \hat{S}$ with $\hat{F}(f) = 0$.

Proof. For every x the limit exists and is uniquely determined by x; whence unique choice suffices to obtain the function f. It is routine to verify that f is a location on S. Since

$$d\left(\hat{F}(f), 0\right) = \hat{F}(f)(0) = \lim_{f(x)\to 0} \overbrace{d(F(x), 0)}^{F(x)} \overset{(\dagger)}{=} 0 ,$$

this f is also a root of $\hat{F}$. As for $(\dagger)$, it is straightforward to give a rigorous version of the following argument: if $f(x)$ is small, then x is close—by the definition of f—to some y with $F(y)$ small, so that $F(x)$ is small by the continuity of F.

Example 1. If $S = \mathbb{R}$ and $F(x) = |x - \xi|^k$ with $k \geqslant 1$ and $\xi \in \mathbb{R}$, then

$$f(x) = \lim_{F(y)\to 0} d(x,y) = \lim_{y\to\xi} d(x,y) = d(x,\xi) = |x-\xi| .$$

In particular, $f = F$ precisely when $k = 1$, which is the only k for which F is a location.

Remark 1. Let F and f be as in Lemma 1 above. If F is already a location, then $f = F$.

Theorem 1. *Let $F : S \to \mathbb{R}$ be a uniformly continuous function on a complete metric space S. If* $\inf F = 0$ *and F has uniformly at most one minimum, then F has a minimum: that is, attains its infimum* 0.

Literally as in [26], Theorem 1 can be extended to the more general case of an equation $F(x,u) = 0$ depending on a parameter u varying over yet another metric space T. For the sake of a complete presentation we include this result, and its short proof. The step from Theorem 1 to Corollary 1 is analogous to the one from Theorem 4.4 to Corollary 4.5 of [14].

Let $F : S \times T \to \mathbb{R}$ be a function. If for each $u \in T$ there is exactly one $\xi_u \in S$ with $F(\xi_u, u) = 0$ (for instance, if $F(\,\cdot\,, u) : S \to \mathbb{R}$ satisfies the hypotheses of Theorem 1 for every $u \in T$), then—by unique choice—there is a mapping $h : T \to S$ with $F(h(u), u) = 0$ for all $u \in T$.

Corollary 1. *Let S and T be metric spaces with S complete, and $F : S \times T \to \mathbb{R}$ such that $F(\,\cdot\,, u) : S \to \mathbb{R}$ is uniformly continuous with* $\inf_{x\in S} F(x,u) = 0$ *for each $u \in T$. If*

$$\forall \delta\, \forall u\, \exists \varepsilon\, \forall v\, \forall x,y\, \left(d(u,v) < \varepsilon \wedge F(x,u) < \varepsilon \wedge F(y,v) < \varepsilon \Rightarrow d(x,y) < \delta\right) , \tag{8}$$

then there is a pointwise continuous mapping $h : T \to S$ *with* $F(h(u), u) = 0$ *for all* $u \in T$*. If even*

$$\forall \delta\, \exists \varepsilon\, \forall u, v\, \forall x, y\ (d(u, v) < \varepsilon \wedge F(x, u) < \varepsilon \wedge F(y, v) < \varepsilon \Rightarrow d(x, y) < \delta)\ , \tag{9}$$

then h *is uniformly continuous.*

Proof. Note first that (9) implies (8).[4] The case $u = v$ of (8) says that $F(\cdot, u)$ has uniformly at most one minimum, and thus satisfies the hypotheses of Theorem 1. Arguing as before, we obtain a mapping $h : T \to S$ with $F(h(u), u) = 0$ for all $u \in T$. The case $x = h(u)$ and $y = h(v)$ of (8) finally says that h is pointwise continuous, or uniformly continuous if F even satisfies (9).

4 Discussion

In view of Lemma 1, in Theorem 1—and thus also in Corollary 1—a given problem has prompted its desired solution in a fairly direct way. Example 1 and Remark 1 raise the questions how far a problem is from its solution, to what extent they can be identified with one another, and how many problems have the same solution. Questions of practical interest are which common types of equations satisfy the (uniform) uniqueness precondition, and whether any kind of local uniqueness suffices. Future work in this area will also include an adaptation of the techniques we have developed above from Richman's concept of a completion to the road followed by Vickers [28,29] and Palmgren [19] in point–free topology.

Acknowledgements. Vladimir Lifschitz provided the reference of his article, whose existence was first indicated by Grigori Mints. The present paper was worked out at the Lorentz Center of Leiden University during the workshop *Mathematics: Algorithms and Proofs* organised in January 2007 by Herman Geuvers, Bas Spitters (chair), and Freek Wiedijk. Suggestions by Douglas Bridges helped to bring this article into its final form. Last but not least, the author owes special thanks to Ulrich Berger.

References

1. Aczel, P.: The type theoretic interpretation of constructive set theory. In: Macintyre, A., Pacholski, L., Paris, J. (eds.) Logic Colloquium '77, pp. 55–66. North–Holland, Amsterdam (1978)
2. Aczel, P., Crosilla, L., Ishihara, H., Palmgren, E., Schuster, P.: Binary refinement implies discrete exponentiation. Studia Logica 84, 361–368 (2006)
3. Aczel, P., Rathjen, M.: Notes on Constructive Set Theory. Institut Mittag–Leffler Preprint No. 40 (2000/01)

[4] They differ from one another only by the order of the quantifiers $\exists\varepsilon$ and $\forall u$.

4. Berger, J., Bridges, D., Schuster, P.: The fan theorem and unique existence of maxima. J. Symbolic Logic 71, 713–720 (2006)
5. Berger, J., Ishihara, H.: Brouwer's fan theorem and unique existence in constructive analysis. Math. Log. Quart. 51, 369–373 (2005)
6. Bishop, E.: Foundations of Constructive Analysis. McGraw–Hill, New York (1967)
7. Bishop, E., Bridges, D.: Constructive Analysis. Springer, Berlin (1985)
8. Bridges, D.: Recent progress in constructive approximation theory. In: Troelstra, A.S., van Dalen, D. (eds.) The L.E.J. Brouwer Centenary Symposium, pp. 41–50. North–Holland, Amsterdam (1982)
9. Bridges, D., Richman, F.: Varieties of Constructive Mathematics. Cambridge University Press, Cambridge (1987)
10. Burden, C.W., Mulvey, C.J.: Banach spaces in categories of sheaves. In: Fourman, M., Mulvey, C., Scott, D. (eds.) Applications of Sheaves. Proceedings, Durham, 1977. Lecture Notes in Math, vol. 753, pp. 169–196. Springer, Berlin and Heidelberg (1979)
11. Crosilla, L., Ishihara, H., Schuster, P.: On constructing completions. J. Symbolic Logic 70, 969–978 (2005)
12. Ishihara, H.: Informal constructive reverse mathematics. Sūrikaisekikenkyūsho Kōkyūroku 1381, 108–117 (2004)
13. Ishihara, H.: Constructive reverse mathematics: compactness properties. In: Crosilla, L., Schuster, P. (eds.) From Sets and Types to Topology and Analysis. Oxford Logic Guides, vol. 48, pp. 245–267. Oxford University Press, Oxford (2005)
14. Kohlenbach, U.: Effective moduli from ineffective uniqueness proofs. An unwinding of de La Vallée Poussin's proof for Chebycheff approximation. Ann. Pure Appl. Logic 64, 27–94 (1993)
15. Kohlenbach, U., Oliva, P.: Proof mining: a systematic way of analysing proofs in mathematics. Proc. Steklov Inst. Math. 242(3), 136–164 (2003)
16. Kreĭnovič, V.Ja.: Review of Constructive Functional Analysis. MR0521982 (82k:03094)
17. Lifshits, V.A.: Investigation of constructive functions by the method of fillings. J. Soviet Math. 1, 41–47 (1973)
18. Mulvey, C.J.: Banach spaces over a compact space. In: Herrlich, H., Preuss, G. (eds.) Categorical Topology. Proceedings, Berlin, 1978. Lecture Notes in Math, vol. 719, pp. 243–249. Springer, Berlin and Heidelberg (1979)
19. Palmgren, E.: A constructive and functorial embedding of locally compact metric spaces into locales. Department of Mathematics, Uppsala University, Report 25 (2006)
20. Rathjen, M.: Choice principles in constructive and classical set theories. In: Chatzidakis, Z., Koepke, P., Pohlers, W. (eds.) Logic Colloquium '02. Proceedings, Münster, 2002. Lect. Notes Logic 27, Assoc. Symbol. Logic, La Jolla pp. 299–326 (2006)
21. Richman, F.: Intuitionism as generalization. Philos. Math (3) 5, 124–128 (1990)
22. Richman, F.: The fundamental theorem of algebra: a constructive development without choice. Pacific J. Math. 196, 213–230 (2000)
23. Richman, F.: Spreads and choice in constructive mathematics. Indag. Math. (N.S.) 13, 259–267 (2002)
24. Schuster, P.: Unique existence, approximate solutions, and countable choice. Theoret. Comput. Sci. 305, 433–455 (2003)
25. Schuster, P.: Countable choice as a questionable uniformity principle. Philos. Math (3) 12, 106–134 (2004)

26. Schuster, P.: Unique solutions. Math. Log. Quart. 52 (2006), pp. 534–539. Corrigendum: Math. Log. Quart. 53, 214 (2007)
27. Stolzenberg, G.: Sets as limits yellow. Typescript (1988)
28. Vickers, S.: Localic completion of generalised metric spaces. I. Theor. Appl. Categ. (15) (electronic) 14, 328–356 (2005)
29. Vickers, S.: Localic completion of generalised metric spaces. II. Power locales. Preprint, School of Computer Science, University of Birmingham (2004)

A Useful Undecidable Theory

Victor L. Selivanov*

A.P. Ershov Institute of Informatics Systems
Siberian Division Russian Academy of Sciences
and
Theoretische Informatik, Universität Würzburg
selivanov@informatik.uni-wuerzburg.de

Abstract. We show that many so called discrete weak semilattices considered earlier in a series of author's publications have hereditary undecidable first-order theories. Since such structures appear naturally in some parts of computability theory, we obtain several new undecidability results. This applies e.g. to the structures of complete numberings, of m-degrees of index sets and of the Wadge degrees of partitions in the Baire space and ω-algebraic domains.

Keywords: Semilattice, discrete weak semilattice, partition, reducibility, undecidability, theory.

1 Introduction

In a series of papers [Se79, Se82, Se04, Se05, Se06] we discovered that in several parts of computability theory and hierarchy theory structures of certain kind (called discrete weak semilattices or dws for short) appear frequently. In this paper we show that any dws with an antichain property (to be defined in the next section) has hereditary undecidable first-order theory. Recall that a theory of signature σ is *hereditary undecidable* if any of its subtheories of the same signature σ is undecidable. This general undecidability result applies to the dws's considered earlier because they turn out to have the antichain property. In this way, we obtain several new undecidability results which maybe of interest to the corresponding parts of computability theory.

We shall use standard set-theoretic notation. Let us fix some terminology concerning partitions. Let M be a set, $P(M)$ the class of subsets of M, and for each $k \geq 2$ let k^M be the set of all functions $A : M \to k$ (we identify a natural number $k < \omega$ with the set $\{0, \ldots, k-1\}$). We call maps $A \in k^M$ *k-partitions* of M because they are in a natural bijective correspondence with the tuples $(A_0, \ldots, A_{k-1})$ of pairwise disjoint sets satisfying $A_0 \cup \cdots \cup A_{k-1} = M$. For any class $\mathcal{C} \subseteq P(M)$, let $\mathcal{C}_k$ denote the set of $\mathcal{C}$-partitions, i.e. partitions $A \in k^M$ such that $A^{-1}(i) \in \mathcal{C}$ for each $i < k$. For any class $\mathcal{C}$ of subsets of M, $BC(\mathcal{C})$ denotes the Boolean closure of $\mathcal{C}$.

* Supported by RFBR Grant 4413.2006.1, by DFG Mercator program and by DFG-RFBR Grant 06-01-04002.

In some parts of the text we assume the reader to be acquainted with basic notions of computability theory and hierarchy theory like computable functions, computably enumerable sets, m-reducibility, hyperarithmetical hierarchy with levels denoted $\Sigma^0_n, \Pi^0_n, \Delta^0_n$ or Borel hierarchy in topological spaces with levels denoted $\mathbf{\Sigma}^0_n, \mathbf{\Pi}^0_n, \mathbf{\Delta}^0_n$, for $n > 0$. By $\mathbf{B}$ we denote the class of Borel sets. By $\varkappa$ and π we denote the standard numberings of the class of computable partial functions and of the class of computably enumerable sets, respectively. Following A.I. Mal'cev, we call $\varkappa$ and π Kleene and Post numberings, respectively. For more information on this, see e.g. [Od99, Ke94].

We use some standard notation and terminology on posets which may be found e.g. in [DP94]. We will not be very cautious when applying notions about posets also to preorders; in such cases we mean the corresponding quotient-poset of the preorder. A poset $(P; \leq)$ will be often shorter denoted just by P. Any subset of P may be considered as a poset with the induced partial ordering. In particular, this applies to the "lower cones" $\hat{x} = \{y \in P \mid y \leq x\}$, for any $x \in P$. A *semilattice* is a structure $(P; \leq, \cup)$ consisting of a preorder $(P; \leq)$ and a binary operation $\cup$ of supremum in $(P; \leq)$ (thus, we consider only upper semilattices). With a slight abuse of notation, we apply the operation $\cup$ also to finite non-empty subsets of P. This causes no problem because the supremum of any non-empty finite set is unique ap to equivalence relation $\equiv$ induced by $\leq$.

In Section 2 we introduce main notions and establish some necessary facts about dws's. In Section 3 we prove the undecidability result. In Section 4 we describe the above-mentioned applications of the undecidability result, and we conclude in Section 5.

2 The Theory

We start with a definition which is a slight modification of the corresponding notions introduced in [Se79, Se82].

Definition 1. *Let I be a non-empty set. By I-discrete weak semilattice (dws, for short) we mean a structure $(P; \leq, \{P_i\}_{i \in I})$ with $P_i \subseteq P$ such that:*

(i) $(P; \leq)$ is a preorder;

(ii) for all $n < \omega$, $x_0, \ldots, x_n \in P$ and $i \in I$ there exists $u_i = u_i(x_0, \ldots, x_n) \in P_i$ which is a supremum for $x_0, \ldots, x_n$ in P_i, i.e. $\forall j \leq n(x_j \leq u_i)$ and for any $y \in P_i$ with $\forall j \leq n(x_j \leq y)$ it holds $u_i \leq y$;

(iii) for all $n < \omega$, $x_0, \ldots, x_n \in P$, $i \neq i' \in I$ and $y \in P_{i'}$, if $y \leq u_i(x_0, \ldots, x_n)$ then $y \leq x_j$ for some $j \leq n$.

Throughout the paper, we consider only the case when I is a non-empty finite set with at least two elements. Usually we consider the case when $I = k$ for some integer $k \geq 2$; in this case we write the dws also in the form $(P; \leq, P_0, \ldots, P_{k-1})$. Note that the operations u_i above maybe considered as n-ary operations of P for each $n > 0$. These operations are associative and commutative, so we can apply them also to finite nonempty subsets of P. It is clear that for any dws the structure $(P_i; \leq, u_i)$ is a semilattice for each $i \in I$, and that $(P; \leq, \{P_j\}_{j \in J})$ is a dws for any nonempty subset J of I. The following properties of dws's are also immediate (see [Se79, Se82]).

Proposition 1. *Let $(P; \leq, P_0, \ldots, P_{k-1})$ be a dws and $y, x_0, \ldots, x_n \in P_0 \cup \cdots \cup P_{k-1}$.*

(i) If $x_j \leq y$ for all $j \leq n$ then $u_i(x_0, \ldots, x_n) \leq y$ for some $i < k$.
(ii) If $y \leq u_i(x_0, x \ldots, x_n)$ for all $i < k$ then $y \leq x_j$ for some $j \leq n$.
(iii) If $\{x_0, \ldots, x_n\}$ has no greatest element then it has no supremum in $P_0 \cup \cdots \cup P_{k-1}$.

The next easy assertion shows that considering of only binary operations u_i is sufficient to recover the structure of a dws.

Proposition 2
(i) Let $(P; \leq, P_0, \ldots, P_{k-1})$ be a dws. Then the binary operations $u_0, \ldots, u_{k-1}$ on P have for all $x, y, z, t \in P$ and distinct $i, j < k$ the following properties: $x, y \leq u_i(x, y)$; $x, y \leq u_i(z, t) \to u_i(x, y) \leq u_i(z, t)$; $u_j(z, t) \leq u_i(x, y) \to (u_j(z, t) \leq y \vee u_j(z, t) \leq z)$.
(ii) Let $(P; \leq)$ be a preorder and $u_0, \ldots, u_{k-1}$ binary operations on P satisfying the properties in (i). Then $(P; \leq, P_0, \ldots, P_{k-1})$, where $P_i = \{u_i(x, y) \mid x, y \in P\}$, is a dws.
(iii) The maps $(P; \leq, P_0, \ldots, P_{k-1}) \mapsto (P; \leq, u_0, \ldots, u_{k-1})$ and back are inverses of each other, up to isomorphism of the quotient-structures.

By the last proposition, we may apply the term "dws" also to structures $(P; \leq, u_0, \ldots, u_{k-1})$ satisfying the three properties in (i). Note that the class of dws's written in this form is universally axiomatizable, so any substructure of a dws $(P; \leq, u_0, \ldots, u_{k-1})$ is also a dws.

Note that for any dws the unary operations u_i are closure operators on $(P; \leq)$, i.e. they satisfy $\forall x(x \leq u_i(x))$, $\forall x \forall y(x \leq y \to u_i(x) \leq u_i(y))$ and $\forall x(u_i(u_i(x)) \leq u_i(x))$. They have also the discreteness property: $\forall x \forall y(u_i(x) \leq u_j(y) \to u_i(x) \leq y)$, for all $i \neq j$. This shows a close relation of dws's to semilattices with discrete closures (*dc*-semilattices, for short) introduced in [Se82]. By a *dc-semilattice* we mean a structure $(P; \leq, \cup, p_0, \ldots, p_{k-1})$ satisfying the following properties: $(P; \leq, \cup)$ is a semilattice; the unary operations p_i are closure operators on $(P; \leq)$ with the discreteness property; every element $p_i(x)$ is join-irreducible, i.e. $p_i(x) \leq y \cup z \to (p_i(x) \leq y \vee p_i(x) \leq z))$ for all $y, z \in P$. The next easy assertion shows that dws's that are semilattices essentially coincide with *dc*-semilattices.

Proposition 3
(i) Let $(P; \leq, P_0, \ldots, P_{k-1})$ be a dws and $(P; \leq, \cup)$ is a semilattice. Then the structure $(P; \leq, \cup, u_0, \ldots, u_{k-1})$ with the unary operations u_i on P is a dc-semilattice.
(ii) If $(P; \leq, \cup, p_0, \ldots, p_{k-1})$ is a dc-semilattice then $(P; \leq, P_0, \ldots, P_{k-1})$, where $P_i = \{p_i(x) \mid x \in P\}$ is a dws.
(iii) The maps $(P; \leq, \cup, P_0, \ldots, P_{k-1}) \mapsto (P; \leq, \cup, u_0, \ldots, u_{k-1}))$ and back are inverses of each other, up to isomorphism of the quotient-structures.

In [Se82] we considered also some variations of dws's and dc-semilattices. By a *2-dws* we mean a structure $(P;\leq,\{P_i^j\}_{i,j\in I})$ with the properties similar to those of dws's with the only exception: this time the property (iii) states that for all $n<\omega$, $x_0,\ldots,x_n\in P$, $i\neq i'$, $j\neq j'$ and $y\in P_{i'}^{j'}$, if $y\leq u_i^j(x_0,\ldots,x_n)$ then $y\leq x_l$ for some $l\leq n$. By a *2-dc-semilattice* we mean a structure $(P;\leq,\cup,\{r_i^j\}_{i,j\in I})$ satisfying the same properties as *dc*-semilattices with a similar modification of the discreteness property: for all $x,y\in P$, $i\neq i'$, $j\neq j'$, $r_i^j(x)\leq r_{i'}^{j'}(y)\rightarrow r_i^j(x)\leq y$. Analogs of Propositions 1—3 are easily seen to be true also for 2-dws's and 2-*dc*-semilattices. We state also the following evident relationship between the introduced notions.

Proposition 4
(i) If $(P;\leq,\{P_i^j\}_{i,j\in I})$ is a 2-dws then $(P;\leq,\{P_i^i\}_{i\in I})$ is a dws.
(ii) If $(P;\leq,\cup,\{r_i^j\}_{i,j\in I})$ is a 2-dc-semilattice then $(P;\leq,\cup,\{r_i^i\}_{i\in I})$ is a dc-semilattice.

We conclude this section with discussing a subclass of dws's for which the undecidability result holds true. Recall that a subset F of a preorder is an *antichain* if every two elements of F are incomparable. By a *strong antichain* in a dws $(P;\leq,P_0,\ldots,P_{k-1})$ we mean a finite set $F\subseteq P$ that has at least two elements and $x\not\leq u_i(F\setminus\{x\})$ for all $x\in F$ and $i<k$. We say that a dws $(P;\leq,P_0,\ldots,P_{k-1})$ has *unbounded width* if P_i has antichains with any finite number of elements, for some $i<k$.

Proposition 5. *If $(P;\leq,P_0,\ldots,P_{k-1})$ is a dws of unbounded width then for each $j<k$ the set P_j has strong antichains with any finite number of elements.*

Proof. Suppose w.l.o.g. that P_0 has antichains with any finite number of elements. For all $n>0$ and $j<k$ we have to find a strong antichain $\{a_0,\ldots,a_n\}$ with $n+1$ elements in P_j. First we consider the case $j\neq 0$. Let $\{b_0,\ldots,b_{2n+1}\}$ be an antichain in P_0 with $2n+2$ elements. Set $a_i=u_j(b_{2i},b_{2i+1})$ for all $i\leq n$. We have to check that $a_i\not\leq u_l(a_0,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$ for all $i\leq n$ and $l<k$. Suppose the contrary. If $j\neq l$ then, since $a_i\in P_j$, $a_i\leq a_m=u_j(b_{2m},b_{2m+1})$ for some $m\leq n$, $m\neq i$. Then $b_{2i}\leq u_j(b_{2m},b_{2m+1})$. Since $b_{2i}\in P_0$ and $j\neq 0$, $b_{2i}\leq b_{2m}$ or $b_{2i}\leq b_{2m+1}$, a contradiction. Now let $l=j$. Then $b_{2i}\leq u_l(a_0,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$. Since $b_{2i}\in P_0$ and $j\neq 0$, $b_{2i}\leq a_m$ for some $m\leq n$, $m\neq i$, and we obtain the same contradiction.

It remains to consider the case $j=0$, i.e. to construct a strong antichain with $n+1$ elements in P_0. By the preceding paragraph, P_1 has strong antichains, hence also antichains with any finite number of elements. Taking in the preceding argument 1 in place of 0 and $j=0$ we obtain the desired strong antichain in P_0. This completes the proof.

Finally, we show that many dws's automatically have unbounded width.

Proposition 6. *Let $k\geq 3$ and $(P;\leq,P_0,\ldots,P_{k-1})$ be a dws that is not linearly ordered. Then it has unbounded width.*

Proof. Let x,y be incomparable elements in P. Let k^+ be the set of finite nonempty words over alphabet k. For any $w\in k^+$, let $|w|$ be the length of w, and for $i < |w|$ let $w(i)$ be the i-th letter of w, so $w = w(0)\cdots w(n-1)$ where $n = |w|$. We say that w is *repetition-free* if $w(i) \neq w(i+1)$ for any $i < |w| - 1$. Relate to any $w \in k^+$ the element $u_w = u_{w(0)} \cdots u_{w(n-1)}(x, y) \in P$ where $n = |w|$. E.g., $u_{021} = u_0(u_2(u_1(x, y)))$.

We claim that for all repetition-free words $w, v \in k^+$, $u_w \leq u_v$ iff $w \subseteq v$, i.e. $v = v_0 w(0) v_1 \cdots w(n-1) v_n$ for some (possibly, empty) words $v_0, \ldots, v_n$, where again $n = |w|$. From right to left, the assertion follows from the fact that any unary operation u_i is a closure operator. Conversely, let $u_w \leq u_v$; we argue by induction on (n, m) in the lexicographic ordering, where $m = |v|$. Assume first that $w(0) \neq v(0)$. Since $x, y < u_w$, from the discreteness property we obtain $m > 1$. Again by discreteness, $u_w \leq u_{v'}$ where $v' = v(1) \cdots v(m-1)$. By induction, $w \subseteq v'$, hence $w \subseteq v$. Now assume $w(0) = v(0)$. If $n = 1$, we are done. Otherwise, $u_{w'} \leq u_v$ (because $u_{w'} \leq u_w$). Since $w'(0) \neq v(0)$, by the first case we have $m > 1$ and $u_{w'} \leq u_{v'}$. By induction, $w' \subseteq v'$ and therefore $w \subseteq v$.

For any $n > 0$, let A_n be the set of repetition-free words of length n from k^+ with the first letter 0. By induction on n, A_n has exactly $(k-1)^n$ elements. Since $k \geq 3$, the number of elements in A_n is unbounded when n is growing. By the property of u_w, $\{u_w \mid w \in A_n\}$ is an antichain in $(P_0; \leq)$. This completes the proof.

3 The Undecidability

In this section we present the undecidability result

Theorem 1. *Let $k \geq 2$ and $(P; \leq, P_0, \ldots, P_{k-1})$ be a dws of unbounded width. Then the first-order theory $FO(P; \leq)$ is hereditary undecidable.*

Proof is a slight modification of a proof in [KS06]. As is well-known [E+65], it suffices to find first-order formulas $\phi_0(x, \bar{p})$, $\phi_1(x, y, \bar{p})$ and $\phi_2(x, y, \bar{p})$ of signature $\{\leq\}$ (where x, y are variables and $\bar{p}$ is a string of variables called parameters) with the following property:

(*) for every $n < \omega$ and for all equivalence relations ξ, η on $\{0, \ldots, n\}$ there are values of parameters $\bar{p} \in P$ such that the structure $(\{0, \ldots, n\}; \xi, \eta)$ is isomorphic to the structure $(\phi_0(P, \bar{p}); \phi_1(P, \bar{p}), \phi_2(P, \bar{p}))$.

Here $\phi_0(P, \bar{p}) = \{a \in P | (P; \leq) \models \phi_0(a, \bar{p})\}$, $\phi_1(P, \bar{p}) = \{(a, b) \in P | (P; \leq) \models \phi_1(a, b, \bar{p})\}$ and similarly for ϕ_2. In other words, for all n, ξ, η as above there are parameter values $\bar{p} \in P$ such that the set $\{0, \ldots, n\}$ and the relations ξ, η are first-order definable in $(P; \leq)$ with parameters $\bar{p}$.

We will use short notation $\bar{v} = (v_0, \ldots, v_{k-1})$, $\bar{w} = (w_0, \ldots, w_{k-1})$ and $\bar{z} = (z_0, \ldots, z_{k-1})$ for strings of different variables. By Proposition 5, for any $n < \omega$ there is a strong antichain $\{c_0, \ldots, c_n\}$ in P_0. Let $\tau(x, \bar{v})$ be the formula

$$x \in P_0 \cup \cdots \cup P_{k-1} \wedge \bigwedge_{i<k} (x \leq v_i) \wedge \neg \exists y > x (y \in P_0 \cup \cdots \cup P_{k-1} \wedge \bigwedge_{i<k} (y \leq v_i))$$

which means that x is a maximal lower bound for $\{v_0, \ldots, v_{k-1}\}$ in $P_0 \cup \cdots \cup P_{k-1}$. If we fix the values

$$v_j = u_j(c_0, \ldots, c_n) \in P_j,\ j < k, \tag{1}$$

of parameters $\bar{v}$ then, by Proposition 1,

$$\tau(P,\bar{v}) = \{c_0,\dots,c_n\}. \tag{2}$$

Let $\psi(x,y,\bar{v},\bar{w})$ be the formula $\tau(x,\bar{v}) \wedge \tau(y,\bar{v}) \wedge \exists t(\tau(t,\bar{w}) \wedge x \leq t \wedge y \leq t)$. Let us fix values $\bar{v}$ as in (1) and values of $\bar{w}$ as follows:

$$w_j = u_j(u_0(\{c_i \mid i \in \xi_0\}),\dots,u_0(\{c_i \mid i \in \xi_m\})) \in P_j,\ j < k, \tag{3}$$

where ξ is an equivalence relation on $n+1$ and $(\xi_0,\dots,\xi_m)$ is the partition of $n+1$ to ξ-equivalence classes. From the strong antichain property of $c_0,\dots,c_n$ and Proposition 1 it follows that for these values we have

$$\psi(P,\bar{v},\bar{w}) = \{(c_i,c_j)|(i,j) \in \xi\}. \tag{4}$$

Now let $\bar{p}$ be the string of $3k$ variables $(\bar{v},\bar{w},\bar{z})$, $\phi_0(x,\bar{p})$ be $\tau(x,\bar{v})$, $\phi_1(x,y,\bar{p})$ be $\psi(x,y,\bar{v},\bar{w})$ and $\phi_2(x,y,\bar{p})$ be $\psi(x,y,\bar{v},\bar{z})$ (the last formula is obtained from $\psi(x,y,\bar{v},\bar{w})$ by substituting $\bar{z}$ in place of $\bar{w}$). We claim that formulas ϕ_0,ϕ_1,ϕ_2 satisfy the condition (*). Let equivalence relations ξ,η on $n+1$ be given. Specify values of parameters $\bar{v},\bar{w}$ as in (1), (3), and values of parameters $\bar{z}$ as

$$z_j = u_j(u_0(\{c_i \mid i \in \eta_0\}),\dots,u_0(\{c_i \mid i \in \eta_l\})) \in P_j,\ j < k,$$

where $(\eta_0,\dots,\eta_l)$ is the partition of $n+1$ to η-equivalence classes. From (2) and (4) we obtain $\phi_0(P,\bar{p}) = \{c_0,\dots,c_n\}$, $\phi_1(P,\bar{p}) = \{(c_i,c_j)|(i,j) \in \xi\}$, and $\phi_2(P,\bar{p}) = \{(c_i,c_j)|(i,j) \in \eta\}$. This means that $i \mapsto c_i$ is an isomorphism of $(n+1;\xi,\eta)$ onto $(\phi_0(P,\bar{p});\phi_1(P,\bar{p}),\phi_2(P,\bar{p}))$. This completes the proof.

4 The Usefulness

In this section we present some applications of the undecidability result from the previous section.

4.1 The Homomorphic Preorder

Here we consider the so called homomorphic preorder introduced and studied in [KW00, Se04, Se06] in connection with the Boolean hierarchy of partitions. We briefly recall some definitions in which all posets are assumed to be (at most) countable and without infinite chains. By *a forest* we mean a poset in which every lower cone $\hat{x}$ is a chain. *A tree* is a forest having the least element (called *the root* of the tree).

A *k-poset* is a triple $(P;\leq,c)$ consisting of a poset $(P;\leq)$ and a labeling $c : P \to k$. A *morphism* $f : (P;\leq,c) \to (P';\leq',c')$ between k-posets is a monotone function $f : (P;\leq) \to (P';\leq')$ respecting the labelings, i.e. satisfying $c = c' \circ f$. Let $\widetilde{\mathcal{P}}_k$, $\widetilde{\mathcal{F}}_k$, $\widetilde{\mathcal{T}}_k$ and $\widetilde{\mathcal{T}}_k^i$ denote the classes of all countable k-posets, countable k-forests, countable k-trees and countable i-rooted k-trees without infinite chains, respectively. The homomorphic preorder $\leq$ on $\widetilde{\mathcal{P}}_k$ is defined as follows: $(P,\leq,c) \leq (P',\leq',c')$, if there is a morphism from $(P,\leq,c)$ to $(P',\leq',c')$. Let $\mathcal{P}_k$, $\mathcal{F}_k$, $\mathcal{T}_k$ and $\mathcal{T}_k^i$ be the subsets of the corresponding tilde-sets formed by finite posets only. The next result extends a result in [KS06].

Theorem 2. *For all $k \geq 3$ and $i < k$, the first-order theories of $(\widetilde{\mathcal{P}}_k; \leq)$, as well as of the other seven sets from the preceding paragraph with the homomorphic preorder, are hereditary undecidable.*

Proof. Let $\sqcup$ be the join operation on $(\widetilde{\mathcal{P}}_k; \leq)$. For all $P \in \widetilde{\mathcal{P}}_k$ and $j < k$, let $p_j(P)$ be the k-poset obtained from P by adjoining a new smallest element and assigning the label j to this element. By [Se06], $(\widetilde{\mathcal{P}}_k; \leq, \sqcup, p_0, \ldots, p_{k-1})$, as well as the corresponding structures on $\widetilde{\mathcal{F}}_k$, $\mathcal{P}_k$ and $\mathcal{F}_k$, are *dc*-semilattices. By Propositions 3, 6 and Theorem 1, first-order theories of $(\widetilde{\mathcal{P}}_k; \leq)$ and the other three preorders are hereditary undecidable.

Again by [Se06], there are operations $\cup, q_1, \ldots, q_{k-1}$ such that the structure $(\widetilde{\mathcal{T}}_k^i; \leq, \cup, q_1, \ldots, q_{k-1})$ is a *dc*-semilattice, and the same applies to $\mathcal{T}_k^i$. This proves the assertion for these two structures for $k > 3$. For $k = 3$, it is easy to show that the corresponding dws's have unbounded width. Hence, the first-order theories are hereditary undecidable also in this case.

Since $p_j(\widetilde{\mathcal{F}}_k) = \widetilde{\mathcal{T}}_k^j$ for any $j < k$, $(\widetilde{\mathcal{T}}_k; \leq, \widetilde{\mathcal{T}}_k^0, \ldots, \widetilde{\mathcal{T}}_k^{k-1})$ is a dws that has, by the preceding paragraph, unbounded width. Hence, $FO(\widetilde{\mathcal{T}}_k; \leq)$ is hereditary undecidable. The same argument applies to $\mathcal{T}_k$. This completes the proof.

4.2 Complete Numberings

Here we discuss some parts of the numbering theory [Er77]. For definition of the well-known notions of a numbering, reducibility, complete and 2-complete numbering see e.g. [Er77, Se82].

For any $\mathcal{C} \subseteq P(\omega)$, let $\mathcal{C}_k$ be the corresponding subset of k^ω (see Section 1). Let $\mathcal{C}_k^1$ ($\mathcal{C}_k^2$) be the set of complete (respectively, 2-complete) numberings from $\mathcal{C}_k$. E.g., $P(\omega)_k = k^\omega$, $P(\omega)_k^1$ is the set of complete numberings in k^ω, and $(\Delta_2^0)_k^2$ is the set of 2-complete Δ_2^0-partitions in k^ω.

Theorem 3. *Let $k \geq 2$ and $\mathcal{C}$ be one of $P(\omega), \Sigma_n^0, \Pi_n^0, BC(\Sigma_n^0), \Delta_{n+1}^0$, where $n > 0$. Then $FO(\mathcal{C}_k; \leq)$, $FO(\mathcal{C}_k^1; \leq)$ and $FO(\mathcal{C}_k^2; \leq)$ are hereditary undecidable.*

Proof. Let $\mu \oplus \nu$ be the join of numberings μ and ν. For $\nu \in k^\omega$ and $i < k$, let $p_i(\nu) \in k^\omega$ be the completion of ν w.r.t. i [Er77, Se82]. By [Se82], $(k^\omega; \leq, \oplus, p_0, \ldots, p_{k-1})$ is a *dc*-semilattice. It is clear that $\mathcal{C}_k$ is closed under $\oplus, p_0, \ldots, p_{k-1}$ for all the classes $\mathcal{C}$. Hence, $(\mathcal{C}_k; \leq, \oplus, p_0, \ldots, p_{k-1})$ is a *dc*-semilattice as well. The corresponding dws has unbounded width (for $k \geq 3$ by Proposition 6, and for $k = 2$ by a direct easy construction). By Proposition 3 and Theorem 1, $FO(\mathcal{C}_k; \leq)$, is hereditary undecidable.

By [Se82], $\mathcal{C}_k^1 = \bigcup_{i<k} p_i(\mathcal{C}_k)$ for any $\mathcal{C}$ from the formulation. Hence, $(\mathcal{C}_k^1; \leq, p_0(\mathcal{C}_k), \ldots, p_{k-1}(\mathcal{C}_k))$ is a dws of unbounded width (for $k = 2$ this follows from [Moh83]) and therefore $FO(\mathcal{C}_k^1; \leq)$ is hereditary undecidable.

For $\nu \in k^\omega$ and $i, j < k$, let $r_i^j(\nu) \in k^\omega$ be the 2-completion of ν w.r.t. i, j. By [Se82], $(k^\omega; \leq, \oplus, r_i^j)$ is a 2-*dc*-semilattice. It is clear that $\mathcal{C}_k$ is closed under r_i^j for all $i, j < k$ and the classes $\mathcal{C}$ in the formulation. Hence, $(\mathcal{C}_k; \leq, \oplus, r_i^j)$ is a 2-*dc*-semilattice as well. The corresponding dws has unbounded width (for $k \geq 3$

by Propositions 6, and for $k = 2$ this follows from [Se78]). Again by [Se82], $\mathcal{C}_k^2 = \bigcup_{i,j<k} r_i^j(\mathcal{C}_k)$ for any $\mathcal{C}$ from the formulation. Hence, $(\mathcal{C}_k^2; \leq, r_i^j(\mathcal{C}_k))$ is a 2-dws of unbounded width and therefore $FO(\mathcal{C}_k^2; \leq)$ is hereditary undecidable. This completes the proof of the theorem.

4.3 Index Sets and Partitions

Let ν be a numbering of S. A *ν-index set* of a set $A \subseteq S$ is the preimage $\nu^{-1}(A)$. Let I_ν be the class of all ν-index sets. Investigation of m-degrees of index sets of important numberings (especially of the Kleene and Post numberings) is a popular topic in computability theory (see e.g. [Hay72, Se79, Se82, Kuz81, Moh83]). Similar questions are also interesting for the more general case of ν-index k-partitions which are partitions of the form $c \circ \nu$ where $c : S \to k$ (in [Se82] they are called generalized index sets). For any $\mathcal{C} \subseteq P(\omega)$, let $\mathcal{C}_k^\nu$ be the set of ν-index partitions in $\mathcal{C}_k$. E.g., $P(\omega)_2^\varkappa$ is the class of all Kleene index sets while $(\Delta_2^0)_3^\pi$ is the class of Post 3-partitions in $(\Delta_2^0)_3$.

Theorem 4. *Let $k \geq 2$ and $\mathcal{C}$ be one of $P(\omega), \Sigma_n^0, \Pi_n^0, BC(\Sigma_n^0), \Delta_{n+1}^0$, where $n > 0$. Then $FO(\mathcal{C}_k^\varkappa; \leq)$ and $FO(\mathcal{C}_k^\pi; \leq)$ are hereditary undecidable.*

Proof. From [Se82] it follows that the quotient-structures of $(\mathcal{C}_k^\varkappa; \leq)$ and of $(\mathcal{C}_k^1; \leq)$ from the preceding subsection, and also those of $(\mathcal{C}_k^\pi; \leq)$ and $(\mathcal{C}_k^2; \leq)$, are isomorphic. Hence, the assertion follows from Theorem 3.

Remark. The reason for isomorphisms in the proof above is that the Kleene numbering $\varkappa$ is universal complete [Er77, Se82] while the Post numbering π is universal 2-complete [Se82]. By [Se82], Theorem 4 is true actually for all universal complete and 2-complete numberings, and among those are many numberings of interest for computability theory.

4.4 Wadge Reducibility in the Baire Space

Here we discuss the Wadge reducibility of partitions. Let X be a topological space and $A, B : X \to k$ be k-partitions of X. We say that A is Wadge reducible to B (in symbols, $A \leq_W B$) if $A = B \circ f$ for some continuous function f on X. Study of the Wadge reducibility on subsets of the Baire space ω^ω is a central topic in descriptive set theory [Ke94]. Study of the Wadge reducibility of partitions was initiated in [H96, Se04, Se06]. Analog of the next result holds true also for the Cantor space.

Theorem 5. *Let $k \geq 3$ and $\mathcal{C}$ be one of the classes $P(\omega^\omega), \mathbf{B}, BC(\mathbf{\Sigma}_1^0), \mathbf{\Sigma}_n^0, \mathbf{\Pi}_n^0$, $BC(\mathbf{\Sigma}_n^0)$, $\mathbf{\Delta}_{n+1}^0$ in the Baire space, where $n > 1$. Then $FO(\mathcal{C}_k; \leq_W)$ is hereditary undecidable.*

Proof. Define an operation $A \oplus B$ on k-partitions of ω^ω by $(A \oplus B)(0\xi) = A(\xi)$ and $(A \oplus B)(i\xi) = B(\xi)$ for all $0 < i < \omega$ and $\xi \in \omega^\omega$. For all $i < k$ and k-partition A of ω^ω, define a k-partition $p_i(A)$ of ω^ω as follows: $[p_i(A)](\xi) = i$, if

$\xi \notin 0^*1\omega^\omega$, and $[p_i(A)](\xi) = A(\eta)$, if $\xi = 0^n1\eta$. In [Se04, Se06] it was shown that $(k^{\omega^\omega}; \leq_W, \oplus, p_0, \ldots, p_{k-1})$ is a *dc*-semilattice. The classes $\mathcal{C}_k$ are closed under $\oplus, p_0, \ldots, p_{k-1}$, hence $(\mathcal{C}_k; \leq_{CA}, \oplus, p_0, \ldots, p_{k-1})$ are *dc*-semilattices as well. The assertion follows by Propositions 3, 6 and Theorem 1.

4.5 Wadge Reducibility in ω-Algebraic Domains

Here we discuss the Wadge reducibility in the ω-algebraic domains that are central objects of the domain theory (for definitions and general properties of these objects see e.g. [AJ94]). Since ω-algebraic domains are topological spaces, it is possible to define the Borel hierarchy for them with the usual properties [Se05]. We will consider two classes of ω-algebraic domains introduced and studied in [Se05].

By a *reflective domain* we mean an ω-algebraic domain X with a bottom element $\perp$ such that there exist continuous functions $q_0, e_0, q_1, e_1 : X \to X$ such that $q_0e_0 = q_1e_1 = id_X$ and $e_0(X), e_1(X)$ are disjoint open sets. Examples of reflective domains are the domain $\omega^{\leq\omega}$ of finite and infinite sequences of natural numbers, the domain $\omega_\perp^\omega$ of partial functions $g : \omega \rightharpoonup \omega$, and many other natural (in particular, functional) domains, see [Se05].

By a *2-reflective domain* we mean an ω-algebraic domain X with a bottom element $\perp$ and a top element $\top$ such that there exist continuous functions $q_0, e_0, q_1, e_1 : X \to X$ and open sets B_0, C_0, B_1, C_1 with the following properties: $q_0e_0 = q_1e_1 = id_X$; $B_0 \supseteq C_0$ and $B_1 \supseteq C_1$; $e_0(X) = B_0 \setminus C_0$ and $e_1(X) = B_1 \setminus C_1$; $B_0 \cap B_1 = C_0 \cap C_1$. Examples of 2-reflective domains are the domain $P\omega$, and many other natural (in particular, functional) domains, see [Se05].

Theorem 6. *Let X be a reflective or a 2-reflective domain, $k \geq 3$ and $\mathcal{C}$ is one of the classes $P(X), \mathbf{B}$, $BC(\mathbf{\Sigma}_1^0), \mathbf{\Sigma}_n^0$, $\mathbf{\Pi}_n^0, BC(\mathbf{\Sigma}_n^0), \mathbf{\Delta}_{n+1}^0$ in X, where $n > 1$. Then $FO(\mathcal{C}_k; \leq_W)$ is hereditary undecidable.*

Proof. Let X be reflective. For $i < k$, let $\mathcal{P}_i = \{A \in k^X | A(\perp) = i\}$. By [Se05], $(k^X; \leq, \mathcal{P}_0, \ldots, \mathcal{P}_{k-1})$ is a dws. The set $\mathcal{C}_k$ is easily seen to be closed under the corresponding operations u_i, hence the assertion follows.

Now let X be 2-reflective. For all $i, j < k$, let $\mathcal{P}_i^j = \{A \in k^X | A(\perp) = i \wedge A(\top) = j\}$. By [Se05], $(k^X; \leq, \{\mathcal{P}_i^j\}_{i,j<k})$ is a 2-dws. Again, the set $\mathcal{C}_k$ is easily seen to be closed under the corresponding operations u_i^j. The assertion follows from Propositions 4, 6 and Theorem 1.

5 Conclusion

When the notions of dws and *dc*-semilattice were introduced in [Se79, Se82] for the study of index sets they could look rather ad hoc. This paper shows that these notions appear naturally in several areas of computability theory and hierarchy theory. Moreover, it is possible to prove a general undecidability result that applies to those areas. Note that for some of the theories in Section 4 exact complexity estimations are known (see e.g. [KS06]). But for most of them such estimations are still open.

References

[AJ94] Abramsky, S., Jung, A.: Domain theory. In: Handbook of Logic in Computer Science, v. 3, Oxford, pp. 1–168 (1994)

[DP94] Davey, B.A., Pristley, H.A.: Introduction to Lattices and Order. Cambridge (1994)

[Er77] Ershov, Yu.L.: Theory of numberings, Moscow, Nauka (Russian) (1977)

[E+65] Ershov, Yu.L., Lavrov, I.A., Taimanov, A.D., Taitslin, M.A.: Elementary theories. Uspechi Mat. Nauk (Russian) 20(4), 37–108 (1965)

[Hay72] Hay, L.: A discrete chain of degrees of index sets. J. Symbolic Logic 37, 139–149 (1972)

[H96] Hertling, P.: Unstetigkeitsgrade von Funktionen in der effectiven Analysis. PhD thesis, FernUniversität Hagen, Informatik-Berichte, pp. 208–211 (1996)

[Ke94] Kechris, A.S.: Classical Descriptive Set Theory. Springer, New York (1994)

[KS06] Kudinov, O.V., Selivanov, V.L.: Undecidability in the homomorphic quasiorder of finite labeled forests. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 289–296. Springer, Heidelberg (2006) (full version submitted to Journal of Logic and Computation.)

[Kuz81] Kuzmina, T.M.: Structure of m-degrees of index sets of families of partial recursive functions. Algebra and Logic (Russian, there is an English translation) 20, 55–68 (1981)

[KW00] Kosub, S., Wagner, K.: The Boolean hierarchy of NP-partitions. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 157–168. Springer, Heidelberg (2000)

[Moh83] Mohrherr, J.: Kleene index sets and functional m-degrees. J. Symbolic Logic 48(4), 829–840 (1983)

[Od99] Odifreddi, P.G.: Classical Recursion Theory. Elsevier, Amsterdam (1999)

[Se78] Selivanov, V.L.: On the index sets of computable classes of finite sets. In: Algorithms and Automata, Kazan (Russian) pp. 95–99 (1978)

[Se79] Selivanov, V.L.: On the structure of degrees of index sets. Algebra and Logic (Russian, there is an English translation) 18(4), 463–480 (1979)

[Se82] Selivanov, V.L.: On the structure of degrees of generalized index sets. Algebra and Logic (Russian, there is an English translation) 21(4), 472–491 (1982)

[Se04] Selivanov, V.L.: Boolean hierarchy of partitions over reducible bases. Algebra and Logic (Russian, there is an English translation) 43(1), 77–109 (2004)

[Se05] Selivanov, V.L.: Variations on the Wadge reducibility. Siberian Advances in Math. 5(3), 44–80 (2005)

[Se06] Selivanov, V.L.: The algebra of labeled forests modulo homomorphic equivalence. Conf. Computability in Europe-2006 Beckman, A. et.al. (eds.) University of Swansea Report Series #CSR 7-2006, pp. 241–250 (full version to appear in Algebra and Logic) (2006)

On the Computational Power of Flip-Flop Proteins on Membranes

Shankara Narayanan Krishna

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay,
Powai, Mumbai, India 400 076
krishnas@cse.iitb.ac.in

Abstract. P Systems with proteins on membranes were introduced recently by A.Paun and B. Popa, in an effort to bridge the gap between membrane computing and brane calculi. In this variant, one considers multisets of objects inside the membranes as well as proteins on the membranes. The action of the proteins on the objects is classified broadly into 5 categories. In this paper, we study the computational power of these actions and come up with upper and lower bounds in terms of computational power for some of them.

Keywords: Membrane Computing, Universality, Register Machines.

1 Introduction

Membrane computing [7] and brane calculi [1] start from the same reality, viz., the living cell, but they develop in different directions and have different objectives. Membrane computing tries to abstract the computing power of biologically inspired models in the Turing sense, whereas brane calculi work in the framework of process algebra. Various operations on membranes appear in both areas. We continue the study of P systems with proteins on membranes [6], a variant of P systems introduced to understand and bridge the gap between membrane computing and brane calculi. A few related attempts in this direction include [2], [3].

Five operations of proteins acting on membranes, inspired from brane calculi, have been considered in [6]. The power of some of the operations individually ($3ffp$) and in combination ($2res$ and $4cpp$, $2res$ and $1cpp$, $1res$ and $2ffp$) has been looked at in [6] and universality results obtained with arbitrary number of proteins (in case of $3ffp$, $1res$ and $2ffp$), and with 2 proteins for the other two combinations. However, no comparisons between the operations and no characterizations of classes less than RE were made.

In this paper, we obtain a universality result with 6 proteins for the operation $3ffp$, and also show that universality cannot be obtained with (i) one protein even when having an unbounded environment, and (ii)having a finite number of proteins $k, k \geq 1$, and a finite environment. We also compare the operation $3ffp$ with two pairs of operations $(1ffp, 2ffp)$ and $(2ffp, 5ffp)$, and study the power of the single operation $4ffp$. The full hierarchy of these operations and their expressive power under various conditions remains to be studied. The next section is devoted towards pre-requisites; section 3 introduces the variant we study here and also analyzes the computational power; an Appendix contains proofs for some of the Theorems.

2 Some Pre-requisites

We refer to [9] for the elements of formal language theory we use here. We list a few notions and notations: $\mathbf{N}$ denotes the set of natural numbers; V denotes a finite alphabet; V^* is the free monoid generated by V under the operation of concatenation and the empty string denoted by λ, as unit element; by $\mathbf{N}FIN, \mathbf{N}REG, \mathbf{N}CF, \mathbf{N}CS$ and $\mathbf{N}RE$ we denote the family of finite sets, regular sets, context-free sets, context-sensitive sets and recursively enumerable sets of natural numbers, respectively. These can also be looked at as the family of sets of numbers recognized by these languages. For $k \geq 1$ and a family of languages FL, by $\mathbf{N}_kFL$ we denote the length sets of FL excluding the initial segment upto $k-1$. Equivalently, $\mathbf{N}_kFL = \{k + L \mid L \in \mathbf{N}FL\}$, where $k + L = \{k + n \mid n \in L\}$. A multiset over an alphabet V is represented by a string over V (and by all its permutations) and each string precisely identifies a multiset. It is known that $\mathbf{N}FIN \subset \mathbf{N}REG = \mathbf{N}CF \subset \mathbf{N}CS \subset \mathbf{N}RE$.

For basic elements of membrane computing we refer to [8]; for the state-of-the art of the domain, the reader may consult the bibliography from the web address `http://psystems.disco.unimib.it`.

For proving computational universality, we use the concept of Minsky's register machine [5]. Such a machine runs a program consisting of numbered instructions of several simple types. Several variants of register machines with different number of registers and different instruction sets were shown to be computationally universal (e.g., see [4], [5]).

An *n-register machine* is a construct $M = (n, P, i, h)$, where: (i) n is the number of registers, (ii) P is a set of labeled instructions of the form $j : (op\,(r), k, l)$, where $op\,(r)$ is an operation on register r of M, j, k, l are labels from the set $Lab\,(M)$ (which numbers the instructions in a one-to-one manner), (iii) i is the initial label, and (iv) h is the final label.

The machine is capable of the following instructions:

$(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variants usually considered in the literature we demand $k = l$.
$(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.
halt: Stop the machine. This additional instruction can only be assigned to the final label h.

In their *deterministic variant*, such n-register machines can be used to compute any partial recursive function $f : \mathbf{N}^\alpha \rightarrow \mathbf{N}^\beta$; starting with $(n_1, \ldots, n_\alpha) \in$

$\mathbf{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label E with registers 1 to β containing r_1 to r_β. If the final label cannot be reached, then $f(n_1, \ldots, n_\alpha)$ remains undefined. A deterministic m-register machine can also analyze an input $(n_1, \ldots, n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to α, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, the analysis was not successful.

In their *non-deterministic variant*, n-register machines can compute any recursively enumerable set of non-negative integers (or of vectors of non-negative integers). Starting with all registers being empty, we consider a computation of the n-register machine to be successful, if it halts with the result being contained in the first (β) register(s) and with all other registers being empty. In fact, [5] has shown that 3 registers are enough for computing any recursively enumerable set of numbers, such that the input is in register 1, register 3 is never decremented, and the machine, when it halts, has the output in register 3. Some classic results on register machines are mentioned below:

Proposition 1. *For any partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$-register machine M computing f in such a way that, when starting with $(n_1, \ldots, n_\alpha) \in \mathbf{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label h with registers 1 to β containing r_1 to r_β (and with all other registers being empty); if the final label cannot be reached, $f(n_1, \ldots, n_\alpha)$ remains undefined.*

Proposition 2. *For any recursively enumerable set $L \subseteq \mathbf{N}^\beta$ of vectors of non-negative integers there exists a non-deterministic $(\beta + 2)$-register machine M generating L in such a way that, when starting with all registers 1 to $\beta + 2$ being empty, M non-deterministically halts with n_i in registers i, $1 \leq i \leq \beta$, and registers $\beta + 1$ and $\beta + 2$ being empty if and only if $(n_1, \ldots, n_\beta) \in L$.*

A register machine can also be used for defining a language, in the following way. If $V = \{a_1, \ldots, a_k\}$, then each string $w \in V^*$ can be interpreted as a number in base $k + 1$. Specifically, if $w = a_{i_1} a_{i_2} \ldots a_{i_n}$, $1 \leq i_j \leq k, 1 \leq j \leq n$, then $val(w) = i_1(k + 1)^{n-1} + \ldots + i_{n-1}(k + 1) + i_n$. Then, we have:

Proposition 3. *If $L \subseteq V^*$, $card(V) = k, L \in RE$, then a 3-register machine M exists such that for every $w \in V^*$ we have $w \in L$ if and only if M halts when starting with $val_{k+1}(w)$ in its first register; in the halting step, all registers of the machine are empty.*

3 P Systems with Proteins on Membranes

The systems we work with are of the form $\Pi = (O, P, \mu, w_1/z_1, \ldots, w_m/z_m, E, R_1, \ldots, R_m, i_o)$, where (i) O is a *finite* alphabet, (ii) P is the *set* of flip-flop proteins (with $O \cap P = \emptyset$), (iii) μ is the membrane structure, consisting of m membranes, (iv) $w_i, 1 \leq i \leq m$ are the *finite multisets* of objects over O present in region i of μ, (v) $z_i \subseteq P, 1 \leq i \leq m$ are the *sets* of proteins present in region i of μ, (vi) E is the *multiset* of objects from some set $J \subseteq O$ present in the environment (in an *arbitrarily large number of copies/ only finitely many copies* each), (vii) $R_i, 1 \leq i \leq m$ are *finite sets* of rules associated with region i of μ, of the types $1ffp$ to $5ffp$ given below, and (viii) i_o is the output membrane, a membrane from μ.

1. $1ffp : [_i p|a \rightarrow [_i p'|b, a[_i p| \rightarrow b[_i p'|$ or $[_i p'|a \rightarrow [_i p|b, a[_i p'| \rightarrow b[_i p|$
   (protein p (p') modifies an object, changes state to p' (p))
2. $2ffp : [_i p|a \rightarrow a[_i p'|, a[_i p| \rightarrow [_i p'|a$ or $[_i p'|a \rightarrow a[_i p|, a[_i p'| \rightarrow [_i p|a$
   (protein p (p') moves one object unmodified, changes state to p' (p))
3. $3ffp : [_i p|a \rightarrow b[_i p'|, a[_i p| \rightarrow [_i p'|b$ or $[_i p'|a \rightarrow b[_i p|, a[_i p'| \rightarrow [_i p|b$
   (protein p (p') modifies, moves one object, changes state to p' (p))
4. $4ffp : a[_i p|b \rightarrow b[_i p'|a$ or $a[_i p'|b \rightarrow b[_i p|a$
   (protein p (p') interchanges two objects, changes state to p' (p))
5. $5ffp : a[_i p|b \rightarrow c[_i p'|d$ or $a[_i p'|b \rightarrow c[_i p|d$
   (protein p (p') interchanges, modifies two objects, changes state to p' (p))

In all the above rules, a protein can alternate in the two states p or p' upon application of a rule. (ffp stands for flip-flop protein).

The rules are used in the maximally parallel way: in each step, a maximal multiset of rules is used, that is, no *other* rule can be applied to the objects and the proteins which remain unused by the chosen multiset. As usual, each object and each protein can be involved in the application of only one rule, but the membranes are not considered as involved in the rule applications, hence the same membrane can appear in any number of rules at the same time. Only halting computations are considered successful, thus a non-halting computation will yield no result. With a halting computation we associate a result, in the form of the multiplicity of objects present in region i_o in the halting configuration.

The family of sets of numbers $N(\Pi)$ generated by systems Π with at most m membranes, using rules as specified in the list-of-types-of-rules, and with at most r flip-flop proteins present on a membrane is denoted by $\mathbf{N}OP_m^x(pro_r;$ list-of-types-of-rules). If E has arbitrarily large number of copies of objects, $x = \infty$. If there is an upper bound on the number of copies for each object in E, then $x = b$. If there is no upper bound on the number of proteins (membranes), then we replace r (m) by $*$.

Theorem 1. $\mathbf{N}OP_1^\infty(pro_6; 3ffp) = \mathbf{N}RE.$

Proof. We only prove the assertion $\mathbf{N}RE \subseteq \mathbf{N}OP_1^\infty(pro_6; 3ffp)$, and infer the other inclusion from the Church-Turing thesis. We simulate a non-deterministic register machine with 3 registers.

In order to prove this assertion, we consider a register machine with 3 registers, the last one being a special output register which is never decremented. Let there be a program consisting of instructions $l_0, l_1, \ldots, l_h$ which computes f. Let l_h correspond to the instruction HALT and l_0 be the first instruction. The input

value x is expected to be in register 1 and the output value in register 3. Without loss of generality, we can assume that all registers other than the first one are empty at the beginning of a computation. We can also assume that in the halting configuration all registers except the third, where the result of the computation is stored, are empty.
Construct a P system $\Pi = (O, P, [_1\,]_1, A/P, E, R_1, 1)$ with

$$O = \{o_1, o_2, o_3\} \cup \{L_j, L_j', l_j, l_j', l_j'', l_j''' \mid 0 \le j \le h\} \cup \{\hat{l}_0, \bar{l}_0, A, H, \dagger\},$$
$$P = \{p_1, p_2, p_3, s_1, s_2, p\},$$
$$E = \{o_1, o_2, o_3\}.$$

The proteins p_i are used to simulate ADD instructions of register i, and proteins s_i are used to simulate SUB instructions of register i. p is a protein which is used in both simulations. *Creation of $o_1^x, x \ge 0$ in membrane 1, representing input x in register 1*

1. $[_1 p_1|A \to A[_1 p_1'|,\ o_1[_1 p_1'| \to [_1 p_1|o_1, A[_1 p_1| \to [_1 p_1'|\alpha, \alpha \in \{A, \hat{l}_0\},$
   $[_1 p_1'|\hat{l}_0 \to \bar{l}_0[_1 p_1|, \bar{l}_0[_1 p| \to [_1 p'|l_0, [_1 p_1|\hat{l}_0 \to \bar{l}_0[_1 p_1'|$

Simulation of instructions l_i, where $i = (add(r), j, k), 0 \le i, j, k \le h, 1 \le r \le 3$

2. $[_1 p_r \mid l_i \to l_i'[_1 p_r' \mid,$
3. $o_r[_1 p_r' \mid \to [_1 p_r \mid o_r,\ l_i'[_1 p \mid \to [_1 p' \mid \alpha,\ l_i'[_1 p' \to [_1 p \mid \alpha, \alpha \in \{l_j, l_k\}$

An add instruction l_i is simulated when object l_i in membrane 1. Rule 2 shows that the object l_i is transformed into l_i' and is sent out, while the protein becomes p_r'. Next, rule 3 gets a copy of o_r from the environment, changing p_r' into p_r, and in parallel p (or p') replaces l_i' by l_j or l_k, the next instruction to be simulated.

Table 1. Rules for decrementing registers 1,2

4. $[_1 s_r \mid l_i \to L_i[_1 s_r' \mid$			
5. $[_1 s_r' \mid o_r \to o_r[_1 s_r \mid$, $L_i[_1 p \mid \to [_1 p' \mid L_i'$, $L_i[_1 p' \mid \to [_1 p \mid L_i'$			
6. $[_1 p \mid L_i' \to L_i''[_1 p' \mid$ or $[_1 p' \mid L_i' \to L_i''[_1 p \mid$			
Register r is non-zero		Register r is zero	
7.	$L_i''[_1 s_r \to [_1 s_r' \mid l_j''$	7.	$L_i''[_1 s_r' \mid \to [_1 s_r \mid l_k$
8a.	$[_1 s_r' \mid l_j'' \to l_j'''[_1 s_r \mid$		
9a.	$l_j'''[_1 p \mid \to [_1 p' \mid l_j$ or $l_j'''[_1 p' \mid \to [_1 p \mid l_j$		
8b.	$[_1 s_r' \mid o_r \to o_r[_1 s_r \mid$ and $[_1 p \mid l_j'' \to \dagger[_1 p' \mid$ or $[_1 p' \mid l_j'' \to \dagger[_1 p \mid$		
9b.	$\dagger[_1 p \mid \to [_1 p' \mid \dagger$ or $\dagger[_1 p' \mid \to [_1 p \mid \dagger$		

Simulation of instructions l_i, where $i=(sub(r),j,k), 0 \leq i,j,k \leq h, r\in\{1,2\}$ A sub instruction (rules in Table 1) l_i is simulated by s_r (rule 4) when object l_i is inside the membrane. l_i is sent out as L_i. Next, s'_r sends out a copy of o_r (if any present) in the membrane, while in parallel, p (or p') interacts with L_i, and brings it in as L'_i, and subsequently, takes it out as L''_i (rules 5,6).

Case 1: Register r is non-zero: Rule 7 is used, and L''_i is brought inside as l''_j by s_r. Next, s'_r, interacts with l''_j, and sends it out as l'''_j, which in the next step, is brought in as l_j by p (or p') (rules 8a, 9a). Another possibility is that the s'_r again send out a copy of o_r, in which case, the object l''_j is sent out as the trap symbol † by p or (p') leading to a never ending computation (rules 8b, 9b).

Case 2: Register r is zero: In this case, rule 5 for s'_r is not used, so we apply rule 7, by which s'_r acts on L''_i and brings it in as l_k.

Halting: The system should halt when the halt instruction l_h is introduced in membrane 1. Since the output is supposed to consist of objects of register 3 alone, we remove l_h when it is introduced. Since there are no instructions to be simulated after l_h, we can add the rule $[_1 p \mid l_h \rightarrow H[_1 p'|$ or $[_1 p' \mid l_h \rightarrow H[_1 p\,|$, and the system halts.

Theorem 2. *$\mathbf{N}OP_1^b(pro_k; 3ffp) \subseteq \mathbf{N}FIN$, for any finite k.*

Proof. Observe that the system has a finite number of possible configurations, in particular the initial number of symbols in the system cannot be increased. Hence, it is clear, that corresponding family is (possibly strictly) included in NFIN.

Theorem 3. *$\mathbf{N}OP_1^\infty(pro_1; 3ffp) \subseteq \mathbf{N}FIN$.*

Proof. It is enough to observe that the number of objects in the system may only increase if a rule $a[\ p| \rightarrow [\ p'|c$ is used, with a being a symbol present in an infinite number of copies. However, in this case no rule $b[\ p'| \rightarrow [\ p|x$, with b being in infinite number of copies, may be used, otherwise the system never stops. Hence, only one of p (or p') may be used to increase the number of objects inside. But in order to reach p (or p'), another object must be sent outside the membrane. This means that for any halting computation, at most $n+1$ may be generated, with n being the number of objects initially present in the membrane. Hence, the generated set of numbers is (probably strictly) included in $\mathbf{N}FIN$.

Corollary 1. *$\mathbf{N}OP_*^\infty(pro_*; 3ffp) \subseteq \mathbf{N}FIN$ provided there is only one protein in the skin membrane.*

Next, we analyze the operations $2ffp, 1ffp$ and $2ffp, 5ffp$ in combination. We show that in principle, these two pairs of operations can simulate $3ffp$.

Theorem 4. *1. $\mathbf{N}OP_1^\infty(pro_k; 3ffp) \subseteq \mathbf{N}OP_1^\infty(pro_{k+1}; 1ffp, 2ffp)$,*
2. $\mathbf{N}_1OP_1^\infty(pro_k; 3ffp) \subseteq \mathbf{N}OP_1^\infty(pro_{2k+1}; 2ffp, 5ffp)$.

Proof. We give only the construction here.

1. Consider a system $\Pi = (O, P, [_1]_1, w_1/z_1, E, R_1, 1)$ with rules $3ffp$, and let $P = \{p_i \mid 1 \le i \le k\}$. Construct $\Pi' = (O', P', [_1]_1, w_1/z_1', E, R_1', 1)$ where $O' = O \cup \{\hat{a}_p, \bar{a}_p \mid a \in O, p \in P\}$, $P' = P \cup \{Z\}$ and $z_1' = z_1 \cup \{Z\}$. Construction of R_1' based on R_1 is as follows:
   I. Simulation of a rule $[_1p|a \to b[_1p'|$ $([_1p'|a \to b[_1p|)$ is done in 3 steps:
   (a) $[_1p|a \to [_1p'|\hat{b}_p$ $([_1p'|a \to [_1p|\hat{b}_p)$
   (b) $[_1p'|\hat{b}_p \to \hat{b}_p[_1p|$ $([_1p|\hat{b}_p \to \hat{b}_p[_1p'|)$, $[_1Z(\text{or } Z')|\hat{b}_p \to [_1Z'(\text{or } Z)|\dagger$
   (c) $\hat{b}_p[_1p| \to b[_1p'|$ $(\hat{b}_p[_1p'| \to b[_1p|)$, $\hat{b}_p[_1Z(\text{or } Z')| \to \dagger[_1Z'(\text{or } Z)|$
   I(a), (c) are $1ffp$, I(b) are $2ffp, 1ffp$.

   II. Simulation of a rule $a[_1p| \to [_1p'|b$ $(a[_1p'| \to [_1p|b)$ is done in 3 steps:
   (a) $a[_1p| \to \bar{b}_p[_1p'|$ $(a[_1p'| \to \bar{b}_p[_1p|)$
   (b) $\bar{b}_p[_1p'| \to [_1p|\bar{b}_p$ $(\bar{b}_p[_1p| \to [_1p'|\bar{b}_p\ )$, $\bar{b}_p[_1Z(\text{or } Z')| \to \dagger[_1Z'(\text{or } Z)|$
   (c) $[_1p|\bar{b}_p \to [_1p'|b$ $([_1p'|\bar{b}_p \to [_1p|b)$, $[_1Z(\text{or } Z')|\bar{b}_p \to [_1Z'(\text{or } Z)|\dagger$

   Common rule for I, II: For any $q \in P$,
   $\dagger[_1q \to [_1q'|\dagger, [_1q|\dagger \to \dagger[_1q'|, \dagger[_1q' \to [_1q|\dagger, [_1q'|\dagger \to \dagger[_1q|$.
   Let us examine I. Protein p interacts with a inside and modifies it as $\hat{b}_p$. In the next step, p' must necessarily process $\hat{b}_p$, otherwise, a never ending computation will be induced by Z. Note that indexing the object $\hat{b}_p$ by p makes it impossible for any other protein (other than p, p') to process it, even if there are similar rules (say for instance, we have rules $a[_1p| \to [_1p'|b, a[_1q| \to [_1q'|b$ in Π). In the last step again $\hat{b}_p$ is changed to b, and protein p becomes p', thereby producing the same scenario as the $3ffp$ rule. Replacing p by p' and p' by p in I, we obtain the simulation of a rule $[_1p'|a \to b[_1p|$. In short, I says that in the presence of an object $\hat{b}_p$ inside/outside the membrane, the protein p (or p') must necessarily act on it for correct results. Note that since we always have only one copy of either p or p' inside, this will work.

   Now, look at II. Initially, p acts on a, changing it to $\bar{b}_p$. Similar to I, this new object $\bar{b}_p$ needs immediate processing by p (or p') to obtain the desired results : the object b inside, and protein in state p' (p) at the end. Note again, that changing p to p' and p' to p in II will simulate $a[_1p'| \to [_1p|b$ as in I. Finally, since both kinds of rules require 3 steps to be simulated, the $3k$th configuration (object contents inside the membrane) of Π' will be the same as the kth configuration of Π, $k \ge 0$. Thus, Π' halts when Π halts with the same result.
2. Let $\Pi{=}(O, P, [_1]_1, w_1/z_1, E, R_1, 1)$ be a system with rules $3ffp$, with P being equal to $\{p_1, \ldots, p_k\}$. Construct $\Pi' = (O', P', [_1]_1, w_1'/z_1', E', R_1', 1)$ where $O' = \{\eta, U_a, V_a, X_a, Z_a\} \cup \{\bar{a}_p, \hat{a}_p, a_p', Y_{ap} \mid a \in O, p \in P\} \cup O$, $P' = P \cup \{n_p \mid p \in P\} \cup \{Z\}$, $E' = \{\hat{a}_p \mid a \in E, p \in P\} \cup \{\kappa, X_a, U_a \mid a \in O\}$, $z_1' = z_1 \cup \{n_p \mid p \in z_1\} \cup \{Z\}$ and $w_1' = w_1\eta$. Construction of R_1' based on R_1 is as follows:

(a) $X_a[_1 p|a \to \bar{b}_p[_1 p'|Y_{bp}$ $(X_a[_1 p'|a \to \bar{b}_p[_1 p|Y_{bp})$
(b) $[_1 p'|Y_{bp} \to Y_{bp}[_1 p|$ $([_1 p|Y_{bp} \to Y_{bp}[_1 p'|)$, $\bar{b}_p[_1 n_p(\text{or } n'_p)| \to [_1 n'_p(\text{or } n_p)|\bar{b}_p$
$\kappa[_1 Z|Y_{bp} \to \dagger[_1 Z'|\dagger$ or $\kappa[_1 Z'|Y_{bp} \to \dagger[_1 Z|\dagger$
(c) $Y_{bp}[_1 p|\bar{b}_p \to \hat{b}_q[_1 p'|Z_b$, $(Y_{bp}[_1 p'|\bar{b}_p \to \hat{b}_q[_1 p|Z_b)$, $q \in P$,
$Y_{bp}[_1 Z(\text{or } Z')|\bar{b}_p \to \dagger[_1 Z'(\text{or } Z)|\dagger$,
(d) $[_1 n_p|Z_b \to Z_b[_1 n'_p|$ or $[_1 n'_p|Z_b \to Z_b[_1 n_p|$

II. Simulation of a rule $a[_1 p| \to [_1 p'|b$ ($a[_1 p'| \to [_1 p|b)$
(a) $\hat{a}_p[_1 p| \to [_1 p'|\hat{a}_p$ $(\hat{a}_p[_1 p'| \to [_1 p|\hat{a}_p)$,
$\hat{a}_q[_1 p(\text{or } p')|\eta \to \dagger[_1 p'(\text{or } p)|\dagger$, $q \neq p$,
(b) $U_a[_1 p'|\hat{a}_p \to b'_p[_1 p|V_b$ $(U_a[_1 p|\hat{a}_p \to b'_p[_1 p'|V_b)$, $\kappa[_1 Z|\hat{a}_p \to \dagger[_1 Z'|\dagger$
(c) $b'_p[_1 p|V_b \to V_b[_1 p'|b$, $\kappa[_1 Z|V_b \to \dagger[_1 Z'|\dagger$

Common rule for I, II: For any $q \in P$,
$\dagger[_1 q \to [_1 q'|\dagger, [_1 q|\dagger \to \dagger[_1 q'|$, $\dagger[_1 q' \to [_1 q|\dagger, [_1 q'|\dagger \to \dagger[_1 q|$.
I(a), I(c), II(b) and II(c) are $5ffp$; I(b), II(a) are $2ffp$ and $5ffp$, while I(d) is $2ffp$.

Corollary 2. *$\mathbf{N}OP_1^{\infty}(pro_{13}; 2ffp, 5ffp) = \mathbf{N}_1RE$, $\mathbf{N}OP_1^{\infty}(pro_7; 2ffp, 1ffp) = \mathbf{N}RE$.*

However, by independent construction, the above result can be improved:

Theorem 5. *$\mathbf{N}OP_1^{\infty}(pro_7; 2ffp, 5ffp) = \mathbf{N}OP_1^{\infty}(pro_6; 1ffp, 2ffp) = \mathbf{N}RE$.*

Proof. We prove only $\mathbf{N}OP_1(pro_6; 1ffp, 2ffp) = \mathbf{N}RE$. As in Theorem 1, we start with a register machine with 3 registers, the input being in register 1, and the output in register 3, register 3 is never decremented, having instructions $l_0, \ldots, l_h$, l_0 being the initial instruction and l_h being the HALT instruction.

Construct the P system $\Pi = (O, P, [_1]_1, Z/P, E, R_1, 1)$ where

$$O = \{a_1, a_2, a_3\} \cup \{l_i, l'_i, l''_i, l'''_i \mid 0 \leq i \leq h\} \cup \{Z, \dagger\},$$
$$P = \{p_1, p_2, p_3, s_1, s_2, p\}, E = \{a_1, a_2, a_3\}.$$

In the initial configuration, we have the single object Z in the membrane. The proteins p_i are used in the simulation of an ADD instruction to register i. The proteins s_i, p assist in simulation of a SUB instruction of register i. First we generate contents a_1^x in the membrane representing the initial contents of register 1 as follows:

$$[_1 p_1|Z \to [_1 p'_1|Z,\ a_1[_1 p'_1| \to [_1 p_1|a_1,\ [_1 p'_1|Z \to [_1 p_1|l_0$$

1. Simulation of an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k), 1 \leq r \leq 3$:

$$[_1 p_r \mid l_i \to [_1 p'_r \mid l_j \text{ or } [_1 p_r \mid l_i \to [_1 p'_r \mid l_k,\ a_r[_1 p'_r \mid \ \to [_1 p_r \mid a_r.$$

To start with, we have the instruction l_i in membrane one. Protein p_r interacts with l_i, replacing it with either l_j or l_k. In the process, p_r enters into the state p'_r. In the next step, this state of the protein is used to bring an object a_r from the environment, finishing a simulation correctly.

2. Simulation of a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k), 1 \le r \le 2$

Step	Rules	Type
1	$[_1 s_r \mid l_i \to [_1 s'_r \mid l'_i$	$1ffp$
2	$[_1 s'_r \mid a_r \to a_r [_1 s_r \mid$ and $[_1 p \mid l'_i \to [_1 p' \mid l''_i$ or $[_1 s'_r \mid a_r \to a_r [_1 s_r \mid$ and $[_1 p' \mid l'_i \to [_1 p \mid l''_i$	$2ffp, 1ffp$
	Register r is zero	
3	$[_1 s'_r \mid l''_i \to [_1 s_r \mid l_k$	$1ffp$
	Register r is non-zero	
3	$[_1 s_r \mid l''_i \to [_1 s'_r \mid l'''_j$	$1ffp$
4	$[_1 s'_r \mid l'''_j \to [_1 s_r \mid l_j$	$1ffp$
3	$[_1 s_r \mid l''_i \to [_1 s'_r \mid l'''_j$	$1ffp$
4	$[_1 s'_r \mid a_r \to a_r [_1 s_r \mid$ and $[_1 p \mid l'''_j \to [_1 p' \mid \dagger$ or $[_1 s'_r \mid a_r \to a_r [_1 s_r \mid$ and $[_1 p' \mid l'''_j \to [_1 p \mid \dagger$	$2ffp, 1ffp$
5	$[_1 p \mid \dagger \to [_1 p' \mid \dagger$ or $[_1 p' \mid \dagger \to [_1 p \mid \dagger$	$1ffp$

We start with l_i in membrane one. s_r acts upon l_i, replacing it by l'_i. s_r enters s'_r in the process. We use this state of the protein to check if there are any a_r's in the membrane, and if yes, send one out. In parallel, l'_i is replaced by l''_i by the protein p (or p'). We now look at the cases where register r was zero or not.

Case 1: Register r is zero

In this case, we have the protein s'_r at the end of step 2, and hence, we can apply the rule as given in step 3, replacing l''_i by l_k, thereby, finishing the simulation.

Case 2: Register r is non-zero

This means we have s_r at the end of step 2. We have to use the rule where s_r interacts with l''_i, and converts it to l'''_j. After this, s_r becomes s'_r. Now we have two possibilities, a correct one, and a wrong one, leading to an infinite computation. These are depicted by subtables 2,3. If we follow subtable 2, s'_r acts on l'''_j, and replaces it by l_j, thereby, completing the simulation. However, if the s'_r does not act on l'''_j, but instead acts on a_r, then p (or p') acts on l'''_j and introduces the trap symbol $\dagger$, which can never be removed.

3. Halting : As in the previous theorem, we remove l_h from the membrane at the end of a halting configuration by $[_1 p \mid l_h \to l_h [_1 p' \mid$ or $[_1 p' \mid l_h \to l_h [_1 p \mid$.

Next, we look into the operation $4ffp$. We first observe that for $4ffp$, we do not require E to contain arbitrarily many objects.

Proposition 4. *Let $\Pi = (O, P, [_1]_1, w/z, E, R, 1)$ be a P system with rules $4ffp$, $|w| = n$ and let z contain arbitrarily many proteins. Then, among the symbols initially present in the environment, atmost $2n$ symbols are involved in rules.*

Proof. We give only a proof sketch. Note that since all rules are of type $4ffp$, the number of objects inside the membrane will always remain n. Since we have not assumed anything about the number of proteins, it is possible that in any step, all n objects have applicable rules. Assume that k objects inside the membrane have applicable rules in step 1. Assume further that all of them, bring inside the same object a from the environment. This means, we have k copies of a inside the membrane at the end of step 1. In the next step, it is possible that each of these a's, and l of the remaining $n-k$ objects bring in a from the environment. In the worst case, we may have n copies of a inside the membrane, and considering a rule which exchanges a for a, we need a total of $2n$ copies of a.

Based on the above, we conjecture the following (to be proved like Theorem 2):

Theorem 6. $\mathbf{N}OP_*^{\infty}(pro_k; 4ffp) \subseteq \mathbf{N}FIN$.

4 Conclusion

In this paper, we have examined the power of some of the operations introduced in [6]. The power of the other operations as well the optimality of the results proved here remains open.

We summarize the results in this paper below:

Number of proteins	Rules	Number of objects in E	Power
6	$3ffp$	Arbitrary	$\mathbf{N}RE$
$k \geq 0$	$3ffp$	Bounded	$\mathbf{N}FIN$
1	$3ffp$	Arbitrary	$\mathbf{N}FIN$
*	$4ffp$	Arbitrary	$\mathbf{N}FIN$
6	$1ffp, 2ffp$	Arbitrary	$\mathbf{N}RE$
7	$5ffp, 2ffp$	Arbitrary	$\mathbf{N}RE$

References

1. Cardelli, L., Calculi, B.: Interactions of biological membranes, In: Proceedings of Computational Methods in Systems Biology (2004)
2. Cardelli, L., Paun, G.: An universality result for a (mem)brane calculus based on mate/drip operations. Int. J. Found. Comput. Sci. 17(1), 49–68 (2006)
3. Krishna, S.N.: Universality Results for P Systems based on Brane Calculi Operations, Theoretical Computer Science (to appear)
4. Lambek, J.: How to program an infinite abacus. Canad. Math. Bull. 4, 295–302 (1961)
5. Minsky, M.L.: Finite and Infinite Machines. Prentice Hall, EngleWood Cliffs, New Jersey (1967)
6. Paun, A., Popa, B.: P Systems with Proteins on Membranes. Fundamenta Informaticae 72(4), 467–483 (2006)
7. Paun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
8. Păun, G.: Computing with Membranes. An Introduction. Springer, Heidelberg (2002)
9. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Heidelberg (1997)
10. Salomaa, A.: Formal Languages. Academic Press, San Diego (1973)

Computability and Incomputability

Robert I. Soare

Department of Mathematics
University of Chicago
Chicago, Illinois 60637-1546
soare@uchicago.edu

Abstract. The conventional wisdom presented in most computability books and historical papers is that there were several researchers in the early 1930's working on various precise definitions and demonstrations of a function specified by a finite procedure and that they should all share approximately equal credit. This is incorrect. It was Turing *alone* who achieved the characterization, in the opinion of Gödel. We also explore Turing's oracle machine and its analogous properties in analysis.

Keywords: Turing *a*-machine, computability, Church-Turing Thesis, Kurt Gödel, Alan Turing, Turing *o*-machine, computable approximations, effectively continuous functions on reals, computability in analysis, strong reducibilities reexamined.

1 The Modern Era of Computability Theory

Mathematicians have studied algorithms and calculation at least since the time of the Babylonians and later Euclid (c. 330 B.C.). However, it was only in the modern period which began in the 1930's that mathematicians were able to give precise formal models which characterized the informal notion of a finite procedure, and harness these models in what evolved into modern computers. The modern period of the theory of computability can be split into three periods.

1. **λ-Definability Era: 1931–1935**
2. **Recursion Theory Era: 1935–1995**
3. **Computability Era: 1996-present**

1.1 Transition from λ-Definable to Recursive in 1935

Kleene arrived as a graduate student of Church in Princeton in 1935 and worked on showing that a large class of number theoretic functions were λ-definable. Kleene gave lectures to audiences of mathematicians but was disappointed that they were unfamiliar with the λ-calculus definitions and failed to appreciate Kleene's work.

In the spring of 1934 Gödel moved to Princeton and gave his lectures on the "general recursive functions." Both Kleene and Church immediately switched from the λ-definable formalism to the Herbrand-Gödel general recursive functions. Church introduced the use of "recursive" as an adverb to mean "computable," e.g. "recursively enumerable," and Kleene later introduced the term "recursive function theory." The term "recursive" had previously meant "inductive," but in 1935 under Church and Kleene, it acquired the meaning "computable" for the sake of explaining the results to mathematicians unfamiliar with the λ-definable terminology.

1.2 Transition from Recursive to Computable in 1996

By 1995 the computer revolution had brought computers and their concepts and terminology into everyday lives. One of the biggest influences was the introduction in 1981 of the IBM personal computer (PC) which meant that more and more ordinary people had a PC on their desks by 1995. Not only scientists but the general population recognized the basic concepts and terminology of computing, but very few recognized the terminology of "recursive." Those who did associated it with "inductive" not "computable."

Soare wrote a paper *Computability and Recursion* [1996] whose content was delivered in an invited address at the International Congress for Logic, Methodology and Philosophy of Science in Florence in August, 1995.

He pointed out that in the 1930's the principal founders of Computability Theory, Turing and Gödel, *never* used the term "recursive" to mean computable and explicitly rejected such suggestions. The field was ripe for a change of terminology to make itself better understood by the public, just as Kleene and Church had changed terminology. Soare's paper suggested that using the term "computable" would not only be more recognizable by the public, but would be more scientifically and historically accurate. By the time the American Mathematical Society international meeting on the subject was held in Boulder, Colorado in 1999 (see Soare [2000]) the majority of the researchers had changed to the computability terminology, and the title of the conference was now *Computability Theory and its Applications: Current Trends and Open Problems.*

1.3 Three Main Points in [1996]

Soare's paper [1996] on computability was not by itself responsible for the change which took place from 1996 to 1999 with "recursive" replaced by "computable." The seeds of change were already there. However, Soare's paper made three main points which have been subsequently confirmed.

1. The subject is about *computability*, not recursion, not λ-definability, not Post canonical systems. The subject is about studying functions which can be "computed by a finite procedure," to use the words of Gödel [1934].
2. It was *Turing*, not Kleene, not Church, not Post, not even Gödel. It was Turing *alone* who: (1) gave the first convincing formal definition of a computable function (Turing *a*-machine); (2) proved that the informal notion coincided with this formal one; (3) defined the *universal* Turing machine;

and (4) defined the Turing oracle machine (*o*-machine), the central concept in computability.
3. Finally, if we use terminology and notation of computability as Turing did, and indicate our interest in wider questions beyond narrow technical ones as Turing did, then we can form connections with other diverse researchers also interested in other aspects of computability.

This has indeed happened since 1996. An organization called *Computability and Complexity in Analysis (CCA)* has held several meetings, the most recent in November, 2006, in Gainesville, Florida. A related organization, *Computability in Europe (CIE),* is sponsoring this conference and this paper. Both have attracted a large, diverse collection of members. The lectures reflect a much wider range of interests in computability than in recursion theory a decade or so ago. Could either organization have attracted so many members today with the term *"recursion"* in place of *"computability"* in the title?

2 Defining Computability in the 1930's

2.1 Several Formalisms Emerge

Gödel's Incompleteness Theorem [1931] not only solved the first Hilbert question on whether a formal system could prove its own consistency, but it generated a lot of interest in Hilbert's second question, the *Entscheidungsproblem*, decision problem, described in Hilbert and Ackermann [1928] for first order logic. Hilbert had characterized this as the fundamental problem of mathematical logic. In Princeton Alonzo Church had introduced a precise formal system called the λ-*calculus* (now used as a programming language), but by 1931 he knew only that the successor function and addition were λ-definable. Stephen Kleene arrived in Princeton in 1931 as a student of Church, and Church put him to work adding to the knowledge of λ-definable functions. The first version of Church's λ-notation was shown to be inconsistent by Rosser and Kleene, but Church and Kleene focused on a more restricted version now known as the λ-*calculus.*. By 1934 Kleene had shown that virtually all common functions in number theory algebra were λ-definable.

Gödel, by then the most famous and respected mathematical logician in the world, moved to Princeton in 1934. In [1931] he had introduced two elements which would play an important role in computability: the use of primitive recursive functions to code configurations; and the use of sequences to code a sequence of syntactical objects such as a proof. However, Gödel knew that the primitive recursive functions did not exhaust all the computable functions. In the spring of 1934 Gödel [1934] gave a series of lectures, recorded by Kleene and Rosser, modifying an idea of Herbrand to produce what Gödel called a *general recursive function* to distinguish it from the 1931 functions he had called "recursive function" (*"rekursiv"* in German). By 1936 Gödel's terminology was changed by Church and Kleene so that "recursive function" came to mean "general recursive function" and the adjective "primitive" was added to the earlier notion.

2.2 Towards a Computability Thesis

The history of the development of a thesis characterizing the computable functions is fascinating. Gödel [1934] wrote:

Gödel Considers a Thesis in [1934]

> "[Primitive] recursive functions have the important property that, for each given set of values for the arguments, the value of the function can be computed by a finite procedure[3]."
>
> *Footnote 3.*
> "The converse seems to be true, if, besides recursion according to scheme (V) [primitive recursion], recursions of other forms (e.g., with respect to two variables simultaneously) are admitted. This cannot be proved, since the notion of finite computation is not defined, but it serves as a heuristic principle."

We shall return to this quote, especially the second paragraph (his footnote 3), which gives us crucial insight into Gödel's thinking about the computability thesis and his later pronouncements about the achievements of Turing versus others.

Church's Thesis "Thoroughly Unsatisfactory". Largely on the basis of the evidence of the large class of number theoretic functions shown to be λ-definable by Kleene, and based also on his own intuition, Church proposed privately to Gödel the first version of Church's Thesis. Around March, 1934, Church suggested to Gödel that the notion of "λ-definable" be identified with "effectively calculable" (which was Church's term for what we now call "intuitively computable.")

Gödel strongly rejected this suggestion of Church which he called "thoroughly unsatisfactory." Undeterred by this encounter with Gödel, Church changed formal definitions from "λ-definable" to the (Herbrand-Gödel) (general) *recursive* functions just introduced by Gödel in his lectures in 1934. This time without consulting Gödel, Church presented on to the American Mathematical Society on April 19, 1935, his famous proposition published in *1936* and known since Kleene [1952] as *Church's Thesis.*

Thesis 1 (Church's Thesis) *[1935] and [1936]. "In this paper a definition of* recursive function of positive integers *which is essentially Gödel's is adopted. It is maintained that the notion of an effectively calculable function of positive integers should be should be identified with that of a recursive function, ..."*

Church always presented this as a *definition* of an effectively calculable function, not as a thesis. It was Kleene who much later called it a *Thesis* [1943] and finally [1952] called it "Church's Thesis." Using the identification of solvable problems with those solved by a recursive function, Church went on to exhibit an unsolvable problem in mathematics, and hence a negative solution to Hilbert's *Entscheidungsproblem.*

2.3 Church and Kleene Collect Evidence

By the beginning of 1936 Church had become ever more confident of his formal definition of effectively calculable function and its application to unsolvable problems. He had announced his work [1935] to the American Mathematical Society. Church's major paper [1936] had been submitted and would soon appear on recursive functions and the definition of effectively calculable functions. In an abstract received by the American Mathematical Society in July, 1935, which would soon appear as Kleene [1936b], Kleene had announced the equivalence of the formal definitions of λ-definable and recursive, which Kleene and Church had shown.

Kleene had introduced a third formalism, the *μ-recursive functions,* the least class of functions closed under the former schemes (I)–(V) for the primitive recursive functions and also scheme (VI) the least number operator, $\psi(x) = (\mu y)[\theta(x, y) = 0]$. This definition was derived (as Kleene pointed out) from Gödel's [1931] paper with its use of coding of syntax using primitive recursive functions. Also received on July 1, 1935, was an abstract of the paper to appear as Kleene [1936] in which Kleene stated that every (general) recursive function is μ-recursive. This gives a useful and succinct formalism, but it is entirely derivative of Gödel [1931] because its two key ingredients are the primitive recursive functions used for coding as in Gödel and the idea of coding of syntax (or here coding steps in the computation).

The coincidence of these formal definitions of effectively calculable function gave Church more confidence that he had indeed correctly captured the informal notion of finite procedure. By January, 1936 Church had already written this for his major paper Church [1936] to appear shortly.

> "The fact, however, that two such widely different and (in the opinion of the author) equally natural definitions of effective calculability turn out to be equivalent adds to the strength of the reasons addressed below for believing that they constitute as general a characterization of this notion as is consistent with the usual intuitive understanding of it."

2.4 Stalemate at Princeton in Early 1936

Church had accumulated more and more evidence for his case, but he still did not have the approval of one man who counted most to him and to the community in mathematical logic. Gödel continued to reject Church's Thesis. He was not convinced by the confluence argument above even though he himself had supplied virtually all the essential mathematical ingredients in [1931] and [1934] for the work by Church and Kleene including the (general) recursive functions, and the use of primitive recursive functions and coding of sequences of syntactical objects later used by Kleene in the μ-recursive functions and T predicate and normal form. Why did Gödel reject Church's Thesis by early 1936, and why did he not put forward a thesis himself in 1934?

Gödel's Doubts. First, Gödel had hinted in footnote 3 of [1934] quoted above in S2.2 that if other recursions were added they might comprise all mechanically calculable functions. However, in a letter to Martin Davis dated February 15, 1965, Gödel wrote as follows.

> "...It is *not true* that footnote 3 is a statement of Church's Thesis. The conjecture stated there refers to the equivalence of "finite (computation) procedure" and "recursive procedure." However, I was, at the time of these lectures, not at all convinced that my concept of recursion comprises all possible recursions ..."
> -*Gödel 1965, letter to Martin Davis*

Second, Gödel had written in footnote 3,

> "This cannot be proved, since the notion of finite computation is not defined, but it serves as a heuristic principle."

For Gödel a crucial ingredient was to analyze the *intrinsic nature* of the notion of "finite procedure" not just prove the confluence of various formal definitions as Church and Kleene had done. In January, 1936 Gödel not only believed that this had not *yet* been done, but he also expressed some doubt as to whether "finite procedure" *could* be formally analyzed at all or whether it must serve only "as a heuristic principle."

3 Turing Breaks the Stalemate

Those gathered at Princeton, Gödel, Church, Kleene, Rosser, and Post nearby, constituted the most distinguished and powerful group of scholars in the world working on the notion of a computable function, and yet as the year 1936 began they could not agree. At that moment stepped forward a twenty-two year old youth. Well, not just any youth. Alan Turing had already proved the Central Limit Theorem in probability theory (not knowing it had been previously proved, see Zabell [1995]), and as a result Turing had been elected a Fellow of King's College, Cambridge.

The work of Hilbert and Gödel had already attracted interest at Cambridge University where topologist Professor M.H.A. (Max) Newman gave lectures on Hilbert's *Entscheidungsproblem* in 1935. Alan Turing attended. Turing's mother had a typewriter which fascinated him as a boy. He designed his *automatic machine (a-machine)* as a kind of typewriter with an infinite carriage over which the reading head passes with the ability to read, write, and erase one square at a time. Equally important with this Turing machine was Turing's analysis of the intuitive conception of a "function produced by a mechanical procedure." In a masterful demonstration, which Robin Gandy considered as precise as most mathematical proofs, Turing analyzed the informal nature of functions computable by a finite procedure, and demonstrated that they coincide with those computable by an *a*-machine. Also Turing [1936, p. 243] introduced the universal Turing machine which has great theoretical and practical importance.

Thesis 2 (Turing's Thesis [1936]). *A function on the integers is computable by a finite procedure if and only if it is computable by a Turing a-machine.*

Church [1936] had tried to give a similar informal argument for Church's Thesis but Gandy [1988, p. 79] and especially Sieg [1994, pp. 80, 87] in their excellent analyses brought out the weakness in Church's argument. In 1936 Turing's was the only convincing demonstration of any thesis that a formal definition captured the informal notion of computable.

4 Gödel's Opinion of Turing's Work

Gödel's reaction was swift and emphatic. He never accepted Church's Thesis, but he accepted Turing's Thesis at once.[1]

> "That this really is the correct definition of mechanical computability was established beyond any doubt by Turing."
> -*Gödel* 193? *Notes in Nachlass [*1935*]*

> " But I was completely convinced only by Turing's paper."
> -*Gödel: letter to Kreisel of May 1, 1968 [Sieg, 1994, p. 88].*

> " . . . one [Turing] has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, *i.e.*, one not depending on the formalism chosen."
> -*Gödel, Princeton Bicentennial, [1946, p. 84].*

> ". . . For the concept of computability, however, although it is merely a special kind of demonstrability or decidability, the situation is different. By a kind of miracle it is not necessary to distinguish orders, and the diagonal procedure does not lead outside the defined notion."
> —*Gödel: [1946, p. 84], Princeton Bicentennial*

> "The greatest improvement was made possible through the precise definition of the concept of finite procedure, . . . This concept, . . . is equivalent to the concept of a 'computable function of integers' . . . The most satisfactory way, in my opinion, is that of reducing the concept of finite procedure to that of a machine with a finite number of parts, as has been done by the British mathematician Turing."
> —-*Gödel [1951, pp. 304–305], Gibbs lecture*

[1] Gödel was interested in the *intensional* analysis of finite procedure. He never believed the arguments and confluence evidence which Church presented to justify his Thesis. On the other hand Gödel accepted immediately not only Turing machines, but more importantly the analysis Turing gave of a finite procedure. The fact that Turing machines were later proved *extensionally* equivalent to general recursive functions did not convince Gödel of the intrinsic merit of the other definitions.

"...due to A.M. Turing's work a precise and unquestionably adequate definition of the general concept of formal system can now be given, the existence of undecidable arithmetical propositions and the non-demonstrability of the consistency of a system in the same system can now be proved rigorously for *every* consistent formal system containing a certain amount of finitary number theory."
-Godel's Postscriptum to Gödel [1934], see Davis, [1965].

4.1 Gödel on Church's Thesis

In June, 1964, Gödel remarked (see Davis [1965, p. 72]).

"See A. Turing [1936] and the almost simultaneous paper by E.L. Post [1936]. As for previous equivalent definitions of computability, which, however, are much less suitable for our purpose, see A. Church [1936]."

4.2 Kleene Said About Turing

"Turing's computability is intrinsically persuasive" but "λ-definability is not intrinsically persuasive" and "general recursiveness scarcely so (its author Gödel being at the time not at all persuaded)."
-*Stephen Cole Kleene [1981b, p. 49]*

"Turing's machine concept arises from a direct effort to analyze computation procedures as we know them intuitively into elementary operations. Turing argued that repetitions of his elementary operations would suffice or any possible computation. For this reason, Turing computability suggests the thesis more immediately than the other equivalent notions and so we choose it for our exposition."
-*Stephen Cole Kleene, second book [1967, p. 233]*

4.3 Church Said About Turing

Computability by a Turing machine, " has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately—*i.e.*, without the necessity of proving preliminary theorems."
-*Alonzo Church, [1937], Review of Turing [1936]*

5 Turing Defines Oracle Machines

After Turing's discovery in April, 1936, Professor Newman suggested that he go to Princeton to take his Ph.D., which he did under Church. Turing's thesis in 1939 was on ordinal logics, an attempt to get around Gödel's incompleteness theorem by adding new axioms. In an obscure part of his paper Turing [1939, §4] wrote,

> "Let us suppose we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were. . . . this oracle . . . cannot be a machine.
>
> With the help of the oracle we could form a new kind of machine (call them *o*-machines), having as one of its fundamental processes that of solving a given number-theoretic problem."

There are several ways that a Turing machine with oracle may be defined. We prefer the definition in Soare [1987, p. 46] of a machine with a head which reads the work tape and oracle tape simultaneously. Any of these definitions gives the definition of a Turing functional $\Phi_e^A(x) = y$. The crucial point is that any definition must produce a computably enumerable (c.e.) set V_e as the *graph*[2] of Φ_e,

$$V_e \ = \ \{\langle \sigma, x, y \ : \ \Phi_e^\sigma(x) = y\}.$$

These Turing computable functionals are exactly like the more general continuous functions on Cantor space 2^ω but relativized to an oracle set $X \subset om$. A continuous function on Cantor space is one with graph V_e^X for some $x \subset \omega$. Is a a researcher in analysis wants to work on the slightly different space of the real numbers with the usual topology in analysis given by open balls B_i with rational center and rational radius as the basic open sets, then the description is exactly analogous. If the function is continuous, then the inverse of a basic open set B_i is set of balls c.e. in X for some read $X \subset \omega$.

In September, 1939, Turing entered the world of British crypotography and did not develop the notion of oracle machine further. Post [1944] began to explore the notion of oracle machines and their associated definition of Turing reducibility $B \leq_{\mathrm{T}} A$, and Post studied a number of stronger less general reducibilities, such as 1-reducible, m-reducible, *btt*-reducible, *tt*-reducible (truth table reducible). The full Turing reduciblity was not well understood until at least a decade later with the Kleene Post [1954] paper finding Turing incomparable sets below $\emptyset'$.

Many problems in the real world are not explicitly computable but are limit computable. The function f may be represented as $f(x) = \lim_s \widehat{f}(x,s)$ for some computable function $\widehat{f}(x,s)$. These problems in turn can be looked at via oracle machines. Consider the halting problem $K = \ \{ \ e : e \in W_e \ \}$.

Lemma 1. [Limit Lemma] *A is limit computable iff* $A \leq_{\mathrm{T}} K$.

Many processes can be looked at as having a fixed finite control, but an oracle which may be changed as external conditions warrant, *i.e.*, an oracle K_s at stage s which goes to a limit $K = \cup_s K_s$. Thus, the concept of Turing's oracle machine (*o*-machine) is the most important in the subject of computability at both the theoretical and practical level.

A number of textbooks on computability theory follow Post's lead by defining a Turing *a*-machine in chapter 1 and not defining a Turing *o*-machine and Turing

[2] The term "graph" for this set is becoming standard by analogy with the graph of partial computable (p.c.) function ϕ_e which is the same but with out the σ.

functionals until a much later chapter, spending the intermediate chapters on the intermediate reducibilities or other smaller themes. This is analogous to having a calculus textbook delay the definition of continuous or differentiable function until chapter 10 spending the time on derivates of special functions like polynomials. We should proceed as quickly to the full definition of a Turing functional Φ_e^A and its graph V_e defined above.

References

[Church, 1936] Church, A.: An unsolvable problem of elementary number theory. American J. of Math. 58, 345–363 (1936)

[Church, 1937] Church, A.: Review of Turing 1936. J. Symbolic Logic 2(1), 42–43 (1937)

[Davis, 1965] Davis, M.: The Undecidable. Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions. Raven Press, Hewlett, New York (1965)

[Davis, 1982] Davis, M.: Why Gödel did not have Church's Thesis. Information and Control 54, 3–24 (1982)

[Gandy, 1980] Gandy, R.: Church's thesis and principles for mechanisms, In: The Kleene Symposium, North-Holland, pp. 123–148 (1980)

[Gandy, 1988] Gandy, R.: The confluence of ideas in 1936, In: Herken, pp. 55–111 (1988)

[Godel, 1931] Gödel, K.: Über formal unentscheidbare sätze der Principia Mathematica und verwandter systeme. I, Monatsch. Math. Phys. vol. 38 pp. 173–178 (1931) (English trans. in Davis 1965, pp. 4–38, and in van Heijenoort, pp. 592–616 (1967)

[Godel, 1934] Gödel, K.: On undecidable propositions of formal mathematical systems, Notes by Kleene, S.C., Rosser, J.B. (eds.) on lectures at the Institute for Advanced Study, Princeton, New Jersey, 30 pp (Reprinted in Davis 1965 [3, 39–74] (1934)

[Godel, 193?] Gödel, K.: Undecidable diophantine propositions, In: Gödel, pp. 156–175 (1995)

[Godel, 1946] Gödel, K.: Remarks before the Princeton bicentennial conference of problems in mathematics, Reprinted in: Davis 1965 [3], pp. 84–88 (1946)

[Godel, 1951] Gödel, K.: Some basic theorems on the foundations of mathematics and their implications, In: Gödel pp. 304–323 (This was the Gibbs Lecture delivered by Gödel on December 26, 1951 to the Amer. Math. Soc.) (1995)

[Godel, 1964] Gödel, K.: Postscriptum to Gödel 1931, written in 1946, printed in Davis pp. 71–73 (1965)

[Hilbert, Ackermann] Hilbert, D., Ackermann, W.: Grundzüge der theoretischen Logik. In (English translation of 1938 edition, Chelsea, New York, 1950), Springer, Berlin (1928)

[Hodges, 1983] Hodges, A.: Alan Turing: The Enigma, Burnett Books and Hutchinson, London, and Simon and Schuster, New York (1983)

[Kleene, 1936] Kleene, S.C.: General recursive functions of natural numbers. Math. Ann. 112, 727–742 (1936)

[Kleene, 1943] Kleene, S.C.: Recursive predicates and quantifiers. Trans. A.M.S. 53, 41–73 (1943)

[Kleene, 1952] Kleene, S.C.: Introduction to Metamathematics, Van Nostrand, New York. Ninth reprint 1988, Walters-Noordhoff Publishing Co., Groningën and North-Holland, Amsterdam (1952)

[Kleene, 1967] Kleene, S.C.: Mathematical Logic. John Wiley and Sons, Inc, New York, London, Sydney (1967)

[Kleene, 1981] Kleene, S.C.: Origins of recursive function theory. Annals of the History of Computing 3, 52–67 (1981)

[Kleene, 1987] Kleene, S.C.: Reflections on Church's Thesis. Notre Dame. Journal of Formal Logic 28, 490–498 (1987)

[Kleene, 1988] Kleene, S.C.: Turing's analysis of computability, and major applications of it, In: Herken, pp. 17–54 (1988)

[Kleene, Post, 1954] Kleene, S.C., Post, E.L.: The upper semi-lattice of degrees of recursive unsolvability. Ann. of Math. 59, 379–407 (1954)

[Post, 1936] Post, E.L.: Finite combinatory processes–formulation, J. Symbolic Logic vol. 1 pp. 103–105 (1936). Reprinted in Davis, pp. 288–291 (1965)

[Post, 1944] Post, E.L.: Recursively enumerable sets of positive integers and their decision problems, Bull. Amer. Math. Soc. vol. 50, pp. 284–316 (1944). Reprinted in Davis, pp. 304–337 (1965)

[Sieg, 1994] Sieg, W.: Mechanical procedures and mathematical experience. In: George, A. (ed.) Mathematics and Mind, Oxford Univ. Press, Oxford (1994)

[Soare, 1987] Soare, R.I.: Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets. Springer, Heidelberg (1987)

[Soare, 1996] Soare, R.I.: Computability and recursion. Bulletin of Symbolic Logic 2, 284–321 (1996)

[Soare, 2000] Soare, R.I.: Extensions, Automorphisms, and Definability, In: Cholak, P., Lempp, S., Lerman, M., Shore, R. (eds.) Computability Theory and its Applications: Current Trends and Open Problems, American Mathematical Society, Contemporary Math. #257, American Mathematical Society, Providence, RI, pps. 279–307 (2000)

[Soare, cta] Soare, R.I.: Computability Theory and Applications, Springer-Verlag, Heidelberg (To appear)

[Turing, 1936] Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. In: Proc. London Math. Soc. ser. 2 vol. 42 (Parts 3 and 4) pp. 230–265 (1936) [Turing, *1937*] A correction, ibid. vol. 43, pp. 544–546 (1937)

[Turing, 1939] Turing, A.M.: Systems of logic based on ordinals. In: Proc. London Math. Soc. vol. 45 Part 3 pp. 161–228 (1939) reprinted in Davis, pp. 154–222 (1965)

[Zabell, 1995] Zabell, S.L.: Alan Turing and the Central Limit Theorem. American Mathematical Monthly 102(6), 483–494 (1995)

A Jump Inversion Theorem for the Degree Spectra

Alexandra A. Soskova*

Faculty of Mathematics and Computer Science,
Sofia University,
5 James Bourchier Blvd.,
1164 Sofia, Bulgaria
asoskova@fmi.uni-sofia.bg

Abstract. A jump inversion theorem for the degree spectra is presented. For a structure $\mathfrak{A}$ which degree spectrum is a subset of the jump spectrum of a structure $\mathfrak{B}$, a structure $\mathfrak{C}$ is constructed as a Marker's extension of $\mathfrak{A}$ such that the jump spectrum of $\mathfrak{C}$ is exactly the degree spectrum of $\mathfrak{A}$ and the degree spectrum of $\mathfrak{C}$ is a subset of the degree spectrum of $\mathfrak{B}$.

Keywords: enumeration degrees, degree spectra, Marker's extensions, enumerations.

1 Introduction

The notion of a degree spectrum of a countable structure is introduced by Richter [9] as the set of all Turing degrees generated by all one-to-one enumerations of the structure. It is studied by Ash, Downey, Jockush and Knight [1,4,7]. It is a kind of a measure of complexity of the structure. Soskov [11] represented the notion of a degree spectrum of a structure from the point of view of enumeration degrees.

Let $\mathfrak{A}$ be a countable structure. *The degree spectrum* of the structure $\mathfrak{A}$ is the set $\mathrm{DS}(\mathfrak{A})$ of all enumeration degrees generated by all enumerations of $\mathfrak{A}$. The main benefit of considering not only one-to-one but all enumerations of the structure is that the degree spectrum is always closed upwards with respect to total degrees [11], i.e. if $\mathbf{a} \in \mathrm{DS}(\mathfrak{A})$ then each total enumeration degree $\mathbf{b}$ greater than $\mathbf{a}$ is in $\mathrm{DS}(\mathfrak{A})$. If $\mathbf{a}$ is the least element of $\mathrm{DS}(\mathfrak{A})$ then $\mathbf{a}$ is called the *degree of* $\mathfrak{A}$.

The jump spectrum of $\mathfrak{A}$ is the set $\mathrm{DS}_1(\mathfrak{A})$ of all enumeration jumps of the elements of $\mathrm{DS}(\mathfrak{A})$. If $\mathbf{a}$ is the least element of $\mathrm{DS}_1(\mathfrak{A})$ then $\mathbf{a}$ is called *the first jump degree of* $\mathfrak{A}$.

For any countable structures $\mathfrak{A}$ and $\mathfrak{B}$ define the relation

$$\mathfrak{B} \preceq \mathfrak{A} \iff \mathrm{DS}(\mathfrak{A}) \subseteq \mathrm{DS}(\mathfrak{B}) \ .$$

And let $\mathfrak{A} \equiv \mathfrak{B}$ if $\mathfrak{A} \preceq \mathfrak{B}$ and $\mathfrak{B} \preceq \mathfrak{A}$.

* This work was partially supported by Sofia University Science Fund. The author thanks the anonymous referees for helpful comments.

Let $\mathfrak{B}' \preceq \mathfrak{A}$ if $\mathrm{DS}(\mathfrak{A}) \subseteq \mathrm{DS}_1(\mathfrak{B})$ and $\mathfrak{A} \preceq \mathfrak{B}'$ if $\mathrm{DS}_1(\mathfrak{B}) \subseteq \mathrm{DS}(\mathfrak{A})$. We say that $\mathfrak{A} \equiv \mathfrak{B}'$ if $\mathfrak{A} \preceq \mathfrak{B}'$ and $\mathfrak{B}' \preceq \mathfrak{A}$.

Soskov [12] showed that each jump spectrum is a degree spectrum of a structure. So, for every structure $\mathfrak{B}$ there is a structure $\mathfrak{A}$ such that $\mathfrak{B}' \equiv \mathfrak{A}$, i.e. $\mathrm{DS}(\mathfrak{A}) = \mathrm{DS}_1(\mathfrak{B})$.

In this paper we shall show that if $\mathfrak{A}$ and $\mathfrak{B}$ are structures and $\mathfrak{B}' \preceq \mathfrak{A}$ then there exists a structure $\mathfrak{C}$ such that $\mathfrak{B} \preceq \mathfrak{C}$ and $\mathfrak{C}' \equiv \mathfrak{A}$.

The structure $\mathfrak{C}$ we shall construct as a Marker's extension of $\mathfrak{A}$. In [6] two model-theoretic extension operators were introduced based on the ideas of Marker's construction from [8]. These extensions are called Marker's $\exists$ and $\forall$-extensions and are studied in [5,6]. In our construction we will use also the relativized representation lemma for Σ^0_2 sets proved by Goncharov and Khoussainov [6].

As an application we shall show that if a structure $\mathfrak{A}$ has a degree and $\mathfrak{B}' \preceq \mathfrak{A}$ for some structure $\mathfrak{B}$ then there is a torsion free abelian group $\mathfrak{G}$ of rank 1 such that $\mathfrak{B} \preceq \mathfrak{G}$, $\mathfrak{G}' \equiv \mathfrak{A}$ and $\mathfrak{G}$ has a degree as well.

As a corollary of the main result we receive an analogue of the jump inversion theorem for the joint spectra of finitely many structures considered in [13,15]. Let $\mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n$ be countable structures. *The joint spectrum of* $\mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n$ is the set of all enumeration degrees $\mathbf{a} \in \mathrm{DS}(\mathfrak{A})$ such that $\mathbf{a}' \in \mathrm{DS}(\mathfrak{A}_1), \ldots, \mathbf{a}^{(n)} \in \mathrm{DS}(\mathfrak{A}_n)$.

We will prove that if there is a structure $\mathfrak{B}$ such that $\mathfrak{B}' \preceq \mathfrak{A}$ then there exists a structure $\mathfrak{C} \succeq \mathfrak{B}$ such that the joint spectrum of $\mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n$ is exactly the jump joint spectrum of $\mathfrak{C}, \mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n$.

Next application is a similar result for another relativized version of the notion of a degree spectrum of a structure with respect to finitely many abstract structures studied in [14]. It is shown [13,14,15] that both generalized notions of degree spectra have all general properties of the degree spectra of a structure such as minimal pair theorem and the existence of quasi-minimal degrees.

The relative spectrum of the structure $\mathfrak{A}$ *with respect to* $\mathfrak{A}_1, \ldots, \mathfrak{A}_n$ is the set of all enumeration degrees generated by those enumerations of $\mathfrak{A}$ which "assume" that each $\mathfrak{A}_i$ is relatively Σ^0_{i+1} on $\mathfrak{A}$ for $i = 1, \ldots k$. We will show that if there is a structure $\mathfrak{B}$ such that $\mathfrak{B}' \preceq \mathfrak{A}$ then there exists a structure $\mathfrak{C} \succeq \mathfrak{B}$ such that the relative spectrum of $\mathfrak{A}$ with respect to $\mathfrak{A}_1, \ldots, \mathfrak{A}_n$ coincide with the jump relative spectrum of $\mathfrak{C}$ with respect to $\mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n$.

2 Preliminaries

2.1 Enumeration Degrees

For any sets of natural numbers A and B the set A is enumeration reducible to B ($A \leq_{\mathrm{e}} B$) if there is an enumeration operator Γ_z such that $A = \Gamma_z(B)$. In other words:

$$A \leq_{\mathrm{e}} B \iff (\exists z)(\forall x)(x \in A \iff (\exists v)(\langle v, x\rangle \in W_z \;\&\; D_v \subseteq B))$$

where D_v is the finite set with the canonical code v and $\{W_z\}_{z<\omega}$ is a Gödel enumeration of the c.e. sets.

The relation $\leq_{\mathrm{e}}$ is reflexive and transitive and induces an equivalence relation $\equiv_{\mathrm{e}}$ on all sets of natural numbers. The respective equivalence classes are called enumeration degrees.

By $d_{\mathrm{e}}(A)$ we denote the enumeration degree of the set A and by $\mathcal{D}_{\mathrm{e}}$ the set of all enumeration degrees. Let $A^+ = A \oplus (\mathbb{N} \setminus A)$. The set A is total if $A \equiv_{\mathrm{e}} A^+$. An enumeration degree $\mathbf{a}$ is total if $\mathbf{a}$ contains the e-degree of a total set. The jump operation "$'$" denotes here the enumeration jump introduced by Cooper [3].

Definition 1. Let $L_A = \{\langle x, z \rangle \mid x \in \mathit{\Gamma}_z(A)\}$.
The *e-jump* A' of A is the set $(L_A)^+$.

In fact, the set A is Σ^0_2 relatively the set B ($A \in \Sigma^0_2(B)$) if and only if $A \leq_{\mathrm{e}} (B^+)'$. This follows from the observation that $K_B^+ \equiv_{\mathrm{e}} (B^+)'$ where $K_B = \{\langle e, x \rangle \mid x \in W_e^B\}$.

$$A \in \Sigma^0_2(B) \iff A \text{ is c.e. in } K_B \iff A \leq_{\mathrm{e}} K_B^+ \iff A \leq_{\mathrm{e}} (B^+)' \ .$$

So, if the set B is total then $B \equiv_{\mathrm{e}} B^+$ and hence $A \in \Sigma^0_2(B) \iff A \leq_{\mathrm{e}} B'$.

2.2 Degree Spectra

Let $\mathfrak{A} = (A; R_1, \ldots, R_s)$ be a countable structure such that $=$ is among the predicates $R_1, \ldots, R_s$.

An enumeration f of $\mathfrak{A}$ is a total mapping of $\mathbb{N}$ onto A.

For $B \subseteq A^a$ define $f^{-1}(B) = \{\langle x_1, \ldots, x_a \rangle \mid (f(x_1), \ldots, f(x_a)) \in B\}$.

For each predicate R of $\mathfrak{A}$ of arity r the pullback R^f of R is defined by $R^f(x_1, \ldots, x_r) \iff R(f(x_1), \ldots, f(x_r))$. Let

$$\begin{aligned} f^{-1}(R) = \{\langle x_1, \ldots, x_r, 0 \rangle \mid R^f(x_1, \ldots, x_r)\} \cup \\ \{\langle x_1, \ldots, x_r, 1 \rangle \mid \neg R^f(x_1, \ldots, x_r)\} \ . \end{aligned}$$

Denote by $f^{-1}(\mathfrak{A}) = f^{-1}(R_1) \oplus \ldots \oplus f^{-1}(R_s)$.

Definition 2. *The degree spectrum of* $\mathfrak{A}$ is the set

$$\mathrm{DS}(\mathfrak{A}) = \{d_{\mathrm{e}}(f^{-1}(\mathfrak{A})) \mid f \text{ is an enumeration of } \mathfrak{A}\} \ .$$

Our definition of degree spectra is equivalent to Soskov's from [11]. The structure $\mathfrak{A}$ is total if the predicates $R_1, \ldots, R_s$ are totally defined on A. We consider here only total structures. It is easy to see that if the structure $\mathfrak{A}$ is total then all elements of the degree spectra of $\mathfrak{A}$ are total enumeration degrees. Let ι be the Roger's embedding of the Turing degrees into the enumeration degrees. Then

$$\mathrm{DS}(\mathfrak{A}) = \{\iota(d_{\mathrm{T}}(f^{-1}(\mathfrak{A}))) \mid f \text{ is an enumeration of } \mathfrak{A}\} \ .$$

Richter [9] and Knight [7] defined the degree spectra by taking into account only the bijective enumerations, while we allow as in [11] arbitrary surjective enumerations.

Proposition 3. *[11] Let f be an arbitrary enumeration of $\mathfrak{A}$. There exists a bijective enumeration g of $\mathfrak{A}$ such that $g^{-1}(\mathfrak{A}) \leq_e f^{-1}(\mathfrak{A})$.*

The above proposition shows that almost all of the known results about Turing degree spectra remain valid also for enumeration degree spectra.

Proposition 4. *[11] Let g be an enumeration of $\mathfrak{A}$. Suppose that F is a total set and $g^{-1}(\mathfrak{A}) \leq_e F$. There exists an enumeration f of $\mathfrak{A}$ such that $f^{-1}(\mathfrak{A}) \equiv_e F$.*

From the last proposition it follows that the degree spectrum $\mathrm{DS}(\mathfrak{A})$ is closed upwards with respect to the total enumeration degrees.

The jump spectrum of $\mathfrak{A}$ is the set $\mathrm{DS}_1(\mathfrak{A}) = \{\mathbf{a}' \mid \mathbf{a} \in \mathrm{DS}(\mathfrak{A})\}$.

Since by [12] every jump spectrum is a degree spectrum of a structure it follows that $\mathrm{DS}_1(\mathfrak{A})$ is also closed upwards with respect to total enumeration degrees. One can see this fact directly using the jump inversion theorem from [10].

3 Marker's Extensions

Marker [8] presented a method of constructing for any $n \geq 1$ a $\aleph_0$-categorical almost strongly minimal theory which is not Σ_n-axiomatizable. Further Goncharov and Khoussainov [6] adapted the construction to the general case in order to find for any $n \geq 1$ examples of $\aleph_1$-categorical computable models as well as $\aleph_0$-categorical computable models whose theories are Turing equivalent to $\emptyset^{(n)}$. We shall give the definition of Marker's $\exists$ and $\forall$ extensions following [6].

Let $\mathfrak{A} = (A; R_1, \ldots, R_s, =)$ be a countable total structure and for each i the predicate R_i has arity r_i.

Marker's $\exists$-extension of R_i, denoted by $R_i^{\exists}$, is defined as follows. Consider a set X_i with new elements such that $X_i = \{x^i_{\langle a_1,\ldots,a_{r_i}\rangle} \mid R_i(a_1, \ldots, a_{r_i})\}$. The set X_i we shall call a $\exists$-fellow for R_i. We suppose that all sets A, X_1,..., X_s are pairwise disjoint.

The predicate $R_i^{\exists}$ is a predicate of arity r_i+1 such that $R_i^{\exists}(a_1, \ldots, a_{r_i}, x) \iff a_1, \ldots, a_{r_i} \in A \ \& \ x \in X_i \ \& \ x = x^i_{\langle a_1,\ldots,a_{r_i}\rangle}$ (and so $R_i(a_1, \ldots, a_{r_i})$).

From the definition of $R_i^{\exists}$ it follows that if $a_1, \ldots, a_{r_i} \in A$ then $(\exists x \in X_i) R_i^{\exists}(a_1, \ldots, a_{r_i}, x) \iff R_i(a_1, \ldots, a_{r_i})$.

Definition 5. The structure $\mathfrak{A}^{\exists}$ is defined as follows:

$$(A \cup \bigcup_{i=1}^{s} X_i, R_1^{\exists}, \ldots, R_s^{\exists}, \bar{X}_1, \ldots, \bar{X}_s, =),$$

where each $R_i^{\exists}$ is a Marker's $\exists$-extension of R_i with $\exists$-fellow X_i and $\bar{X}_i$ is a unary predicate true over the elements of the $\exists$-fellow for R_i.

Marker's $\forall$-extension of R_i, denoted by $R_i^{\forall}$, is defined as follows. Consider an infinite set Y_i of new elements such that $Y_i = \{y^i_{\langle a_1,\ldots,a_{r_i}\rangle} \mid \neg R_i(a_1, \ldots, a_{r_i})\}$. The set Y_i we shall call a $\forall$-fellow for R_i.

The predicate $R_i^{\forall}$ is a predicate of arity $r_i + 1$ such that

1. If $R_i^{\forall}(a_1, \ldots, a_{r_i}, y)$ then $a_1, \ldots, a_{r_i} \in A$ and $y \in Y_i$;
2. If $a_1, \ldots, a_{r_i} \in A$ & $y \in Y_i$ then $\neg R_i^{\forall}(a_1, \ldots, a_{r_i}, y) \iff y = y^i_{\langle a_1, \ldots, a_{r_i} \rangle}$.

Note that from the definition of $R_i^{\forall}$ it follows that if $a_1, \ldots, a_{r_i} \in A$ then $(\forall y \in Y_i) R_i^{\forall}(a_1, \ldots, a_{r_i}, y) \iff R_i(a_1, \ldots, a_{r_i})$.

Definition 6. The structure $\mathfrak{A}^{\forall}$ is defined as follows:

$$(A \cup \bigcup_{i=1}^{s} Y_i, R_1^{\forall}, \ldots, R_s^{\forall}, \bar{Y}_1, \ldots, \bar{Y}_s, =),$$

where each $R_i^{\forall}$ is a Marker's $\forall$-extension of R_i with $\forall$-fellow Y_i and $\bar{Y}_i$ is a unary predicate true over the elements of the $\forall$-fellow for R_i. The $\forall$-fellows of the distinct predicates and the set A are pairwise disjoint.

Definition 7. The structure $\mathfrak{A}^{\exists\forall}$ is obtained from $\mathfrak{A}$ as $(\mathfrak{A}^{\exists})^{\forall}$, i.e.

$$(A \cup \bigcup_{i=1}^{s} X_i \cup \bigcup_{i=1}^{s} Y_i, R_1^{\exists\forall}, \ldots, R_s^{\exists\forall}, \bar{X}_1, \ldots, \bar{X}_s, \bar{Y}_1, \ldots, \bar{Y}_s, =),$$

where X_i is a $\exists$-fellow for R_i and Y_i is a $\forall$-fellow for $R_i^{\exists}$.

The structure $\mathfrak{A}^{\exists\forall}$ has the following properties:

Proposition 8. *Let $a_1, \ldots, a_{r_i} \in A$. Then:*

1. $R_i(a_1, \ldots, a_{r_i}) \iff (\exists x \in X_i)(\forall y \in Y_i) R_i^{\exists\forall}(a_1, \ldots, a_{r_i}, x, y)$;

2. For each $y \in Y_i$ there exists a unique sequence $a_1, \ldots, a_{r_i} \in A$ and $x \in X_i$ such that $\neg R_i^{\exists\forall}(a_1, \ldots, a_{r_i}, x, y)$;

3. For each $x \in X_i$ there exists a unique sequence $a_1, \ldots, a_{r_i} \in A$ such that for all $y \in Y_i$ it holds that $R_i^{\exists\forall}(a_1, \ldots, a_{r_i}, x, y)$.

Proof. 1.($\Rightarrow$) Let $R_i(a_1, \ldots, a_{r_i})$. Then there exists $x \in X_i$ such that $R_i^{\exists}(a_1, \ldots, a_{r_i}, x)$. From the definition of Y_i it follows that for any $y \in Y_i$ $R_i^{\exists\forall}(a_1, \ldots, a_{r_i}, x, y)$.

($\Leftarrow$) Let $x \in X_i$ and $R_i^{\exists\forall}(a_1, \ldots, a_{r_i}, x, y)$ for all $y \in Y_i$. Then $R_i^{\exists}(a_1, \ldots, a_{r_i}, x)$ and hence $R_i(a_1, \ldots, a_{r_i})$.

2. Follows from the definition of Y_i.

3. Let $x \in X_i$ then $x = x^i_{\langle a_1, \ldots, a_{r_i} \rangle}$ and $R_i(a_1, \ldots, a_{r_i})$. Hence $R_i^{\exists}(a_1, \ldots, a_{r_i}, x)$. Then for any $y \in Y_i$ it is not possible that $\neg R_i^{\exists\forall}(a_1, \ldots, a_{r_i}, x, y)$.

4 Join of Two Structures

Let $\mathfrak{A} = (A; R_1, \ldots, R_s, =)$ and $\mathfrak{B} = (B; P_1, \ldots, P_t, =)$ be countable structures in the language $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Suppose that $\mathcal{L}_1 \cap \mathcal{L}_2 = \{=\}$ and $A \cap B = \emptyset$. Let $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \{\bar{A}, \bar{B}\}$, where $\bar{A}$ and $\bar{B}$ are unary predicates.

Definition 9. The join of the structures $\mathfrak{A}$ and $\mathfrak{B}$ is the structure $\mathfrak{A} \oplus \mathfrak{B} = (A \cup B; R_1, \dots, R_s, P_1, \dots, P_t, \bar{A}, \bar{B}, =)$ in the language $\mathcal{L}$, where

(a) the predicate $\bar{A}$ is true only over the elements of A and similarly $\bar{B}$ is true only over the elements of B;

(b) the predicate R_i is defined on the elements of A as in the structure $\mathfrak{A}$ and false on all elements not in A and the predicate P_j is defined similarly.

Lemma 10. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be countable total structures and $\mathfrak{C} = \mathfrak{A} \oplus \mathfrak{B}$. Then $\mathfrak{A} \preceq \mathfrak{C}$ and $\mathfrak{B} \preceq \mathfrak{C}$.*

Proof. We have to prove that $\mathrm{DS}(\mathfrak{C}) \subseteq \mathrm{DS}(\mathfrak{A})$ and $\mathrm{DS}(\mathfrak{C}) \subseteq \mathrm{DS}(\mathfrak{B})$.

Let f be an enumeration of $\mathfrak{C}$. Fix $x_0 \in f^{-1}(A)$. Define
$m(0) = x_0$; $m(i+1) = \mu z \in f^{-1}(A)[(\forall k \leq i)(\langle m(k), z\rangle \notin f^{-1}(=))]$.

Set $h = \lambda x. f(m(x))$. Note that $m \leq_{\mathrm{e}} f^{-1}(\mathfrak{C})$ since $z \in f^{-1}(A) \iff \langle z, 0\rangle \in f^{-1}(\bar{A})$.

Define $h^{-1}(R_i) = \{\langle x_1, \dots, x_{r_i}, e\rangle \mid \langle m(x_1), \dots, m(x_{r_i}), e\rangle \in f^{-1}(R_i)\}$. And $h^{-1}(=) = \{\langle x, y, e\rangle \mid \langle m(x), m(y), e\rangle \in f^{-1}(=)\}$.

Then h is an enumeration of $\mathfrak{A}$ and $h^{-1}(\mathfrak{A}) \leq_{\mathrm{e}} f^{-1}(\mathfrak{C})$. Since $\mathfrak{C}$ is a total structure and $\mathrm{DS}(\mathfrak{A})$ is closed upwards then $\mathrm{d}_e(f^{-1}(\mathfrak{C})) \in \mathrm{DS}(\mathfrak{A})$.

5 Representation of $\Sigma^0_2(D)$ Sets

Let $D \subseteq \mathbb{N}$. A set $M \subseteq \mathbb{N}$ is in $\Sigma^0_2(D)$ if there exists a computable in D predicate Q such that

$$n \in M \iff \exists a \forall b Q(n, a, b) \ .$$

Definition 11. [6] If $M \in \Sigma^0_2(D)$ then M is one-to-one representable if there is a computable in D predicate Q with the following properties:

1. $n \in M \iff$ there exists a unique a such that $\forall b Q(n, a, b)$;
2. for every b there is a unique pair $\langle n, a\rangle$ such that $\neg Q(n, a, b)$;
3. for every a there exists a unique n such that $\forall b Q(n, a, b)$.

The predicate Q from the above definition is called an one-to-one representation of M. Goncharov and Khoussainov [6] proved the following lemma:

Lemma 12. *[6] If M is a coinfinite $\Sigma^0_2(D)$ subset of $\mathbb{N}$ which has an infinite computable in D subset S such that $M \setminus S$ is infinite then M has an one-to-one representation.*

Remark 13. We will use this lemma in the next section in our proof of Theorem 14. In order to satisfy the conditions of the lemma we need the following technical explanations.

Let $\mathfrak{A} = (A; R_1, \dots, R_s)$. Suppose that each R_i is true over infinitely many elements and it is false over infinitely many elements also.

We can add to the domain A of the structure $\mathfrak{A}$ two new elements say "T" and "F". Define the predicate R_i^* as follows:

1. Let $r_i \geq 2$. Then $R_i^*(a_1, \dots, a_{r_i})$ is defined as $R_i(a_1, \dots, a_{r_i})$ if F and T are not among the arguments $\{a_1, \dots, a_{r_i}\}$. If $T \in \{a_1 \dots a_{r_i}\}$ then $R_i^*(a_1, \dots, a_{r_i})$ and if $F \in \{a_1, \dots, a_{r_i}\}$ and $T \notin \{a_1, \dots, a_{r_i}\}$ then $\neg R_i^*(a_1, \dots, a_{r_i})$.

2. Let the predicate R_i be unary. Then we define the binary predicate R_i^* as follows: $R_i^*(a, a) \iff R_i(a)$ if $a \notin \{T, F\}$. If $T \in \{a, b\}$ then $R_i^*(a, b)$ is true and if $F \in \{a, b\}$ and $T \notin \{a, b\}$ then $\neg R_i^*(a, b)$.

Let $\mathfrak{A}^*$ be the obtained structure with domain $A \cup \{T, F\}$ and predicates R_i^* for $i = 1, \dots, s$. Then one can easily see using Proposition 4 and Proposition 3 that $\mathrm{DS}(\mathfrak{A}) = \mathrm{DS}(\mathfrak{A}^*)$. Indeed, note that if an enumeration of the structure $\mathfrak{A}$ is bijective then the pullback of the equality is computable. Let f be an enumeration of $\mathfrak{A}$ and $\mathrm{d}_e(f^{-1}(\mathfrak{A})) \in \mathrm{DS}(\mathfrak{A})$. By Proposition 3 there is a bijective enumeration g of $\mathfrak{A}$ such that $g^{-1}(\mathfrak{A}) \leq_{\mathrm{e}} f^{-1}(\mathfrak{A})$. Then there is a bijective enumeration h of $\mathfrak{A}^*$ such that $h^{-1}(\mathfrak{A}^*) \equiv_{\mathrm{e}} g^{-1}(\mathfrak{A})$. Moreover in $h^{-1}(\mathfrak{A}^*)$ each $h^{-1}(R_i^*)$ is infinite and posses a computable subset S such that $h^{-1}(R_i^*) \setminus S$ is infinite. The set S is formed by all tuples containing the number $h^{-1}(T)$. Since $h^{-1}(\mathfrak{A}^*) \leq_{\mathrm{e}} f^{-1}(\mathfrak{A})$ and $\mathfrak{A}$ is total then $\mathrm{d}_e(f^{-1}(\mathfrak{A})) \in \mathrm{DS}(\mathfrak{A}^*)$ by Proposition 4. The proof of $\mathrm{DS}(\mathfrak{A}^*) \subseteq \mathrm{DS}(\mathfrak{A})$ is similar.

6 Jump Inversion Theorem for the Degree Spectra

Theorem 14. *Let $\mathfrak{A}$ and B be total structures such that $\mathfrak{B}' \preceq \mathfrak{A}$. Then there exists a structure $\mathfrak{C}$ such that $\mathfrak{B} \preceq \mathfrak{C}$ and $\mathfrak{C}' \equiv \mathfrak{A}$.*

Proof (Sketch). Without loss of generality we may suppose that the structures $\mathfrak{B}$ and $\mathfrak{A}^{\exists\forall}$ are disjoint. Let $\mathfrak{C} = \mathfrak{B} \oplus \mathfrak{A}^{\exists\forall}$. By Lemma 10 $\mathfrak{B} \preceq \mathfrak{C}$. We shall prove that $\mathfrak{C}' \equiv \mathfrak{A}$, i.e. $\mathrm{DS}(\mathfrak{A}) = \mathrm{DS}_1(\mathfrak{C})$.

1. $\Longrightarrow$ $[\mathrm{DS}_1(\mathfrak{C}) \subseteq \mathrm{DS}(\mathfrak{A})]$.

Let $\mathbf{c} \in \mathrm{DS}_1(\mathfrak{C})$ and let h be an enumeration of $\mathfrak{C}$ such that $\mathbf{c} = \mathrm{d}_e(h^{-1}(\mathfrak{C}))'$. We shall construct an enumeration f of $\mathfrak{A}$ such that $f^{-1}(\mathfrak{A}) \leq_{\mathrm{e}} h^{-1}(\mathfrak{C})'$. Since $h^{-1}(\mathfrak{C})'$ is a total set, by Proposition 4 it will follow that $\mathbf{c} \in \mathrm{DS}(\mathfrak{A})$.

Fix $x_0 \in h^{-1}(A)$. Define

$m(0) = x_0$; $m(i+1) = \mu z \in h^{-1}(A)[(\forall k \leq i)(\langle m(k), z\rangle \notin h^{-1}(=))]$.

Set $f = \lambda a.h(m(a))$. We have $m \leq_{\mathrm{e}} h^{-1}(\mathfrak{A}^{\exists\forall})$ since $z \in h^{-1}(A) \iff (\forall i \leq s)(\langle z, 1\rangle \in h^{-1}(\bar{X}_i) \cap h^{-1}(\bar{Y}_i) \cap h^{-1}(\bar{B}))$. Define:

$$R_i^{\exists\forall,h} = \{\langle a_1, \dots, a_{r_i}, x, y, e\rangle \mid \langle m(a_1), \dots, m(a_{r_i}), x, y, e\rangle \in h^{-1}(R_i^{\exists\forall}) \ \& \ \langle x, 0\rangle \in h^{-1}(\bar{X}_i) \ \& \ \langle y, 0\rangle \in h^{-1}(\bar{Y}_i)\} \ .$$

$$R_i^f(a_1, \dots, a_{r_i}) \iff (\exists x)(\forall y)(\ \langle a_1, \dots, a_{r_i}, x, y, 0\rangle \in R_i^{\exists\forall,h} \ \& \ \langle x, 0\rangle \in h^{-1}(\bar{X}_i) \ \& \ \langle y, 0\rangle \in h^{-1}(\bar{Y}_i)) \ .$$

Then it is clear that f is an enumeration of $\mathfrak{A}$ and $f^{-1}(\mathfrak{A}) \in \Sigma_2^0(h^{-1}(\mathfrak{A}^{\exists\forall}))$. Then $f^{-1}(\mathfrak{A}) \leq_{\mathrm{e}} h^{-1}(\mathfrak{A}^{\exists\forall})' \leq_{\mathrm{e}} h^{-1}(\mathfrak{C})'$ by the monotonicity of the e-jump.

2. $\Longrightarrow$ [$\mathrm{DS}(\mathfrak{A}) \subseteq \mathrm{DS}_1(\mathfrak{C})$].

Let $\mathbf{a} \in \mathrm{DS}(\mathfrak{A})$ and $\bar{f}$ be an enumeration of $\mathfrak{A}$ such that $\mathbf{a} = \mathrm{d}_e(\bar{f}^{-1}(\mathfrak{A}))$. By Proposition 3 there is a bijective enumeration f of $\mathfrak{A}$ such that $f^{-1}(\mathfrak{A}) \leq_e \bar{f}^{-1}(\mathfrak{A})$. We are going to construct an enumeration h of $\mathfrak{C}$ such that $h^{-1}(\mathfrak{C})' \leq_{\mathrm{e}} f^{-1}(\mathfrak{A})$. Then since $\mathfrak{A}$ is a total structure and the $\mathrm{DS}_1(\mathfrak{C})$ is upwards closed with respect to total degrees then $\mathbf{a} \in \mathrm{DS}_1(\mathfrak{C})$.

Since $\mathfrak{B}' \preceq \mathfrak{A}$, i.e. $\mathrm{DS}(\mathfrak{A}) \subseteq \mathrm{DS}_1(\mathfrak{B})$ there is an enumeration g of $\mathfrak{B}$ such that $f^{-1}(\mathfrak{A}) \equiv_{\mathrm{e}} (g^{-1}(\mathfrak{B}))'$. Denote by $D = g^{-1}(\mathfrak{B})$ and note that D is a total set since the structure $\mathfrak{B}$ is total. So for each predicate R_i of the structure $\mathfrak{A}$ we have that $f^{-1}(R_i) \leq_{\mathrm{e}} D'$. Then $f^{-1}(R_i) \in \Sigma^0_2(D)$. Denote by $M_i = f^{-1}(R_i)$. If the positive part or the negative part of $f^{-1}(R_i)$ is finite then $f^{-1}(R_i)$ is computable. Otherwise by Remark 13 we can suppose that M_i satisfies all conditions from Lemma 12. Then by Lemma 12 for each $i \leq s$ there exists a computable in D predicate Q_i which is an one-to-one representation of M_i. Then

— $\bar{n} \in M_i \iff$ there exists a unique a such that $(\forall b)Q_i(\bar{n}, a, b)$;
— for every b let $r(b) = \langle \bar{n}, a\rangle$ be the unique pair such that $\neg Q_i(\bar{n}, a, b)$;
— for every a let $l(a) = \bar{n}$ be the unique $\bar{n}$ such that $\forall b Q_i(\bar{n}, a, b)$.

Denote by $\mathbb{N}_1 = \{\langle 1, n\rangle \mid n \in \mathbb{N}\}$, $\mathbb{N}_2 = \{\langle 2, i, a\rangle \mid i \leq s \;\&\; a \in \mathbb{N}\}$ and $\mathbb{N}_3 = \{\langle 3, i, b\rangle \mid i \leq s \;\&\; b \in \mathbb{N}\}$. Let $\mathbb{N}_0 = \mathbb{N} \setminus (\bigcup_{i=1}^3 \mathbb{N}_i)$. Consider a computable bijection m of $\mathbb{N}_0$ onto $\mathbb{N}$ and denote by $\langle 0, n\rangle = m(n)$.

The definition of the enumeration h of $\mathfrak{C}$ is the following:

$h(\langle 0, n\rangle) = g(n)$;
$h(\langle 1, n\rangle) = f(n)$;
$h(\langle 2, i, a\rangle) = x^i_{\langle f(n_1),\ldots,f(n_{r_i})\rangle}$, if $l(a) = \langle n_1, \ldots, n_{r_i}\rangle$;
$h(\langle 3, i, b\rangle) = y^i_{\langle f(n_1),\ldots,f(n_{r_i}),h(\langle 2,i,a\rangle)\rangle}$, if $r(b) = \langle\langle n_1, \ldots, n_{r_i}\rangle, a\rangle$.

Here $X_i = \{x^i_{\langle a_1,\ldots,a_{r_i}\rangle} \mid R_i(a_1, \ldots, a_{r_i})\}$ is the $\exists$-fellow for R_i and $Y_i = \{y^i_{\langle a_1,\ldots,a_{r_i},x\rangle} \mid \neg R^{\exists}_i(a_1, \ldots, a_{r_i}, x)\}$ is the $\forall$-fellow for $R^{\exists}_i$. Define

$$R^{\exists\forall,h}_i(\langle 1, n_1\rangle, \ldots, \langle 1, n_{r_i}\rangle, \langle 2, i, a\rangle, \langle 3, i, b\rangle) \iff Q_i(\langle n_1, \ldots, n_{r_i}\rangle, a, b) \ .$$

Let $h^{-1}(A) = \mathbb{N}_1$, $h^{-1}(X_i) = \mathbb{N}_2$, $h^{-1}(Y_i) = \mathbb{N}_3$.

It follows that

$$\begin{aligned} R_i(f(n_1)\ldots f(n_{r_i})) &\iff \langle n_1, \ldots, n_{r_i}, 0\rangle \in f^{-1}(R_i) \\ &\iff (\exists a)(\forall b)Q_i(\langle n_1, \ldots, n_{r_i}\rangle, a, b) \\ &\iff (\exists a)(\forall b)R^{\exists\forall,h}_i(\langle 1, n_1\rangle, \ldots, \langle 1, n_{r_i}\rangle, \langle 2, i, a\rangle, \langle 3, i, b\rangle) \\ &\iff (\exists x)(\forall y)R^{\exists\forall}_i(f(n_1)\ldots f(n_{r_i}), x, y) \;\& x \in X_i \& y \in Y_i. \end{aligned}$$

From the definition of h it follows that h is an enumeration of $\mathfrak{A}^{\exists\forall}$. It is clear that $h^{-1}(\mathfrak{A}^{\exists\forall}) \leq_{\mathrm{e}} D$.

Let $\mathfrak{B} = (B; P_1, \ldots, P_t, =)$ then for each $j \leq t$

$h^{-1}(P_j) = \{\langle\langle 0, n_1\rangle, \ldots, \langle 0, n_{p_j}\rangle, e\rangle \mid \langle n_1, \ldots, n_{p_j}, e\rangle \in g^{-1}(P_j)\}$ and $h^{-1}(B) = \mathbb{N}_0$. It is obvious that $h^{-1}(\mathfrak{B}) \leq_{\mathrm{e}} D$.

The pullback of the equality is defined naturally over the elements which are pullbacks of elements of A as $f^{-1}(=)$ and over the elements which are pullbacks of elements of B as $g^{-1}(=)$. Over the elements which are the pullbacks of X_i and

Y_i is a normal equality, since the special form of the Marker's $\exists$ and $\forall$ extensions. So, $h^{-1}(=) \leq_{\mathrm{e}} D$ since $f^{-1}(=)$ is computable.

Thus h is an enumeration of $\mathfrak{C} = \mathfrak{B} \oplus \mathfrak{A}^{\exists\forall}$. Moreover $h^{-1}(\mathfrak{C}) \leq_{\mathrm{e}} D$. Hence $h^{-1}(\mathfrak{C})' \leq_{\mathrm{e}} f^{-1}(\mathfrak{A})$ as $D' \equiv_{\mathrm{e}} f^{-1}(\mathfrak{A})$.

7 Some Applications

The degree of the structure $\mathfrak{A}$, if it exists, is the least element of the degree spectrum of $\mathfrak{A}$. The results of Richter [9] show that there exist structures, e.g. linear orders, which do not have degrees. Richter proved that if the degree spectrum of a linear order has a degree then it is $\mathbf{0}$.

If the jump spectrum $\mathrm{DS}_1(\mathfrak{A})$ has a least element then it is called *the first jump degree of* $\mathfrak{A}$. For example Knight [7] shows that if a linear order has a first jump degree then it is $\mathbf{0}'$. There are examples of structures [1,4] which have a first jump degree but do not posses a degree. In [2,11] it is shown that every torsion free abelian group $\mathfrak{G}$ of rank 1, i.e. $\mathfrak{G}$ is a subgroup of the group of the rational numbers Q, has a first jump degree.

Let $\mathfrak{G}$ be a nontrivial subgroup of the additive group of the rational numbers. Fix $a \neq 0$ an element of $\mathfrak{G}$. For every prime number p set

$$h_p(a) = \begin{cases} k & \text{if } k \text{ is the greatest number such that } p^k|a \text{ in } \mathfrak{G}, \\ \infty & \text{if } p^k|a \text{ in } \mathfrak{G} \text{ for all } k. \end{cases}$$

Let $p_0, p_1, \dots$ be the standard enumeration of the prime numbers and set

$$S_a(\mathfrak{G}) = \{\langle i, j\rangle : j \leq h_{p_i}(a)\}.$$

If a and b are non-zero elements of $\mathfrak{G}$ then $S_a(\mathfrak{G}) \equiv_e S_b(\mathfrak{G})$. Let $\mathbf{d}_{\mathfrak{G}} = \mathrm{d}_e(S_a(\mathfrak{G}))$, where a is some non-zero element of $\mathfrak{G}$.

In [11] it is proved that for every total enumeration degree $\mathbf{d}$, there exists a bijective enumeration f of $\mathfrak{G}$ such that $f^{-1}(\mathfrak{G}) \in \mathbf{d}$ if and only if $\mathbf{d}_{\mathfrak{G}} \leq \mathbf{d}$. Since for every enumeration f we have that $f^{-1}(\mathfrak{G})$ is a total set and $\mathbf{d}_{\mathfrak{G}} \leq d_e(f^{-1}(\mathfrak{G}))$, $\mathrm{DS}(\mathfrak{G}) = \{\mathbf{a} : \mathbf{a} \text{ is total } \& \ \mathbf{a} \geq \mathbf{d}_{\mathfrak{G}}\}$.

It turns out that for any total structures $\mathfrak{A}$ and $\mathfrak{C}$ such that $\mathfrak{C}' \equiv \mathfrak{A}$ if $\mathfrak{C}$ has a degree $\mathbf{a}$ then $\mathbf{a}'$ is the first jump degree of $\mathfrak{C}$ and clearly $\mathbf{a}'$ is the degree of $\mathfrak{A}$ since $\mathrm{DS}(\mathfrak{A}) = \mathrm{DS}_1(\mathfrak{C})$.

Proposition 15. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be total structures such that $\mathfrak{B}' \preceq \mathfrak{A}$. Then if the structure $\mathfrak{A}$ has a degree then there exists a torsion free abelian group $\mathfrak{G}$ of rank 1 which has a degree such that $\mathfrak{B} \preceq \mathfrak{G}$ and $\mathfrak{G}' \equiv \mathfrak{A}$.*

Proof. Let $\mathfrak{C} = \mathfrak{B} \oplus \mathfrak{A}^{\exists\forall}$ be the structure constructed in Theorem 14 such that $\mathfrak{B} \preceq \mathfrak{C}$ and $\mathfrak{C}' \equiv \mathfrak{A}$.

Suppose now that $\mathbf{a}$ is the degree of $\mathfrak{A}$. Then there is a total degree $\mathbf{c} \in \mathrm{DS}(\mathfrak{C})$ such that $\mathbf{c}' = \mathbf{a}$. Then by [11] since $\mathbf{c}$ is a total degree there exists a subgroup $\mathfrak{G}$ of Q such that $\mathbf{d}_{\mathfrak{G}} = \mathbf{c}$. So, $\mathrm{DS}(\mathfrak{G}) = \{\mathbf{e} : \mathbf{e} \text{ is total and } \mathbf{e} \geq \mathbf{d}_{\mathfrak{G}}\}$. And hence $\mathrm{DS}_1(\mathfrak{G}) = \{\mathbf{e}' : \mathbf{e} \text{ is total } \& \ \mathbf{e}' \geq \mathbf{a}\}$. It is clear that $\mathrm{DS}_1(\mathfrak{G}) \subseteq \mathrm{DS}(\mathfrak{A})$. If

$\mathbf{d} \in \mathrm{DS}(\mathfrak{A})$ then $\mathbf{d} \geq \mathbf{a}$. Since the structure $\mathfrak{A}$ is total $\mathbf{d}$ is total. By the jump inversion theorem from [10] there is a total enumeration degree $\mathbf{e}$ such that $\mathbf{e}' = \mathbf{d}$ and $\mathbf{e} \geq \mathbf{c}$. Then $\mathbf{e}' \in \mathrm{DS}_1(\mathfrak{G})$ and thus $\mathbf{d} \in \mathrm{DS}_1(\mathfrak{G})$. Hence $\mathrm{DS}(\mathfrak{A}) = \mathrm{DS}_1(\mathfrak{G})$. Clearly $\mathrm{DS}(\mathfrak{G}) \subseteq \mathrm{DS}(\mathfrak{B})$ since $\mathbf{d}_{\mathfrak{G}} = \mathbf{c} \in \mathrm{DS}(\mathfrak{G}) \subseteq \mathrm{DS}(\mathfrak{B})$.

The next application concerns a generalization of the notion of degree spectra considered in [13,15]. Let $\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n$ be countable structures.

Definition 16. *The joint spectrum of* $\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n$ is the set

$$\mathrm{DS}(\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n) = \{\mathbf{a} \mid \mathbf{a} \in \mathrm{DS}(\mathfrak{A}), \mathbf{a}' \in \mathrm{DS}(\mathfrak{A}_1), \dots, \mathbf{a}^{(n)} \in \mathrm{DS}(\mathfrak{A}_n)\} \ .$$

The next proposition follows directly from Theorem 14.

Proposition 17. *Let* $\mathfrak{A}$ *and* $\mathfrak{B}$ *be total structures such that* $\mathfrak{B}' \preceq \mathfrak{A}$. *Then there exists a structure* $\mathfrak{C} \succeq \mathfrak{B}$ *such that* $\mathrm{DS}(\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n) = \mathrm{DS}_1(\mathfrak{C}, \mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n)$.

We can show a similar result for the relativized spectra from [14].

Definition 18. An enumeration f of $\mathfrak{A}$ is *n-acceptable with respect to the structures* $\mathfrak{A}_1, \dots, \mathfrak{A}_n$, if $f^{-1}(\mathfrak{A}_i) \leq_{\mathrm{e}} (f^{-1}(\mathfrak{A}))^{(i)}$ for each $i \leq n$.

The relative spectrum of the structure $\mathfrak{A}$ with respect to $\mathfrak{A}_1, \dots, \mathfrak{A}_n$ is the set

$$\mathrm{RS}(\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n) = \{d_{\mathrm{e}}(f^{-1}(\mathfrak{A})) \mid f \text{ is a } n\text{-acceptable enumeration of } \mathfrak{A}\} \ .$$

Proposition 19. *Let* $\mathfrak{A}$ *and* $\mathfrak{B}$ *be total structures such that* $\mathfrak{B}' \preceq \mathfrak{A}$. *Then there exists a structure* $\mathfrak{C} \succeq \mathfrak{B}$ *such that* $\mathrm{RS}(\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n) = \mathrm{RS}_1(\mathfrak{C}, \mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n)$.

Proof (sketch). Let $\mathfrak{C} = \mathfrak{B} \oplus \mathfrak{A}^{\exists\forall}$. Suppose that h is a $(n+1)$-acceptable enumeration of $\mathfrak{C}$ and $\mathrm{d}_e(h^{-1}(\mathfrak{C}))' \in \mathrm{RS}_1(\mathfrak{C}, \mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n)$. Let $F = h^{-1}(\mathfrak{C})$. Consider a computable in F function m with range $h^{-1}(A)$. Let $s \neq t \in A$. Define an enumeration of $\mathfrak{A}$:

$$f(x) \simeq \begin{cases} h(m(x/2)) & \text{if } x \text{ is even,} \\ s & \text{if } x = 2z+1 \text{ and } z \in F', \\ t & \text{if } x = 2z+1 \text{ and } z \notin F'. \end{cases}$$

Then $f^{-1}(\mathfrak{A}) \equiv_{\mathrm{e}} F'$ and $f^{-1}(\mathfrak{A}_i) \leq_{\mathrm{e}} h^{-1}(\mathfrak{A}_i) \oplus F' \leq_{\mathrm{e}} h^{-1}(\mathfrak{C})^{(i+1)} \oplus F' \equiv_{\mathrm{e}} F^{(i+1)} \equiv_{\mathrm{e}} f^{-1}(\mathfrak{A})^{(i)}$ for every $i \leq n$. So, $\mathrm{d}_e(h^{-1}(\mathfrak{C}))' \in \mathrm{RS}(\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n)$.

Let f be a n-acceptable enumeration of $\mathfrak{A}$ such that $\mathrm{d}_e(f^{-1}(\mathfrak{A})) \in \mathrm{RS}(\mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n)$. Then as in Theorem 14 one can construct an enumeration h of $\mathfrak{C}$ such that $h^{-1}(\mathfrak{C})' \equiv_{\mathrm{e}} f^{-1}(\mathfrak{A})$ and additionally $h^{-1}(\mathfrak{A}_i) \leq_{\mathrm{e}} f^{-1}(\mathfrak{A}_i)$ for each $i \leq n$. Then $h^{-1}(\mathfrak{A}_i) \leq_{\mathrm{e}} f^{-1}(\mathfrak{A})^{(i)} \leq_{\mathrm{e}} h^{-1}(\mathfrak{C})^{(i+1)}$. Then $\mathrm{d}_e(f^{-1}(\mathfrak{A})) \in \mathrm{RS}_1(\mathfrak{C}, \mathfrak{A}, \mathfrak{A}_1, \dots, \mathfrak{A}_n)$.

References

1. Ash, C.J., Jockush, C., Knight, J.F.: Jumps of orderings. Trans. Amer. Math. Soc. 319, 573–599 (1990)
2. Coles, R., Downey, R., Slaman, T.: Every set has a least jump enumeration. Bulletin London Math. Soc 62, 641–649 (2000)
3. Cooper, S.B.: Partial degrees and the density problem. Part 2: The enumeration degrees of the Σ_2 sets are dense. J. Symbolic Logic 49, 503–513 (1984)
4. Downey, R.G., Knight, J.F.: Orderings with αth jump degree $\mathbf{0}^{(\alpha)}$. Proc. Amer. Math. Soc. 114, 545–552 (1992)
5. Gavryushkin, A.N.: On complexity of Ehrenfeucht Theories with computable model. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J. (eds.) Logical Approaches to Computational barriers CiE2006, University of Wales Swansea, Report Series, No. CSR 7-2006 pp. 105–108 (2006)
6. Goncharov, S., Khoussainov, B.: Complexity of categorical theories with computable models. Algbra and Logic 43(6), 365–373 (2004)
7. Knight, J.F.: Degrees coded in jumps of orderings. J. Symbolic Logic 51, 1034–1042 (1986)
8. Marker, D.: Non Σ_n-axiomatizable almost strongly minimal theories. J. Symbolic Logic 54(3), 921–927 (1989)
9. Richter, L.J.: Degrees of structures. J. Symbolic Logic 46, 723–731 (1981)
10. Soskov, I.N.: A jump inversion theorem for the enumeration jump. Arch. Math. Logic 39, 417–437 (2000)
11. Soskov, I.N.: Degree spectra and co-spectra of structures. Ann. Univ. Sofia 96, 45–68 (2004)
12. Soskov, I.N.: The Jump Spectra are Spectra. in preparation
13. Soskova, A.A.: Minimal pairs and quasi-minimal degrees for the joint spectra of structures. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 451–460. Springer, Heidelberg (2005)
14. Soskova, A.A.: Relativized degree spectra. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 546–555. Springer, Heidelberg (2006)
15. Soskova, A.A., Soskov, I.N.: Co-spectra of joint spectra of structures. Ann. Univ. Sofia 96, 35–44 (2004)

Cupping Δ_2^0 Enumeration Degrees to $\mathbf{0}_e'$

Mariya Ivanova Soskova[1,⋆] and Guohua Wu[2,⋆⋆]

[1] Department of Pure Mathematics
University of Leeds, Leeds LS2 9JT, U.K.
mariya@maths.leeds.ac.uk

[2] School of Physical and Mathematical Sciences
Nanyang Technological University,
Singapore 639798
guohua@ntu.edu.sg

Abstract. In this paper we prove that every nonzero Δ_2^0 e-degree is cuppable to $\mathbf{0}_e'$ by a 1-generic Δ_2^0 e-degree (so low and nontotal) and that every nonzero ω-c.e. e-degree is cuppable to $\mathbf{0}_e'$ by an incomplete 3-c.e. e-degree.

1 Introduction

Intuitively, we say that a set A is *enumeration reducible* to a set B, denoted as $A \leq_e B$, if there is an effective procedure to enumerate A, given any enumeration of B. More formally, $A \leq_e B$ if there is a computably enumerable set W such that

$$A = \{x : (\exists u)[\langle x, u\rangle \in W \;\&\; D_u \subseteq B]\}$$

where D_u is the finite set with canonical index u.

Let $\equiv_e$ denote the equivalence relation generated by $\leq_e$ and let $[A]_e$ be the equivalence class of A — the *enumeration degree* (*e-degree*) of A. The degree structure $\langle \mathcal{D}_e, \leq\rangle$ is defined by setting $\mathcal{D}_e = \{[A]_e : A \subseteq \omega\}$ and setting $[A]_e \leq [B]_e$ if and only if $A \leq_e B$. The operation of least upper bound is given by $[A]_e \vee [B]_e = [A \oplus B]_e$ where $A \oplus B = \{2x : x \in A\} \cup \{2x+1 : x \in B\}$. The structure $\mathcal{D}_e$ is an upper semilattice with least element $\mathbf{0}_e$, the collection of computably enumerable sets. Gutteridge [9] proved that $\mathcal{D}_e$ does not have minimal degrees (see Cooper [1]).

An important substructure of $\mathcal{D}_e$ is given by the Σ_2^0 e-degrees i.e. the e-degrees of Σ_2^0 sets. Cooper [2] proved that Σ_2^0 e-degrees are the e-degrees below $\mathbf{0}_e'$, the e-degree of $\overline{K}$. An e-degree is Δ_2^0 if it contains a Δ_2^0 set, a set A with a computable approximation f such that for every element x, $f(x,0) = 0$ and $\lim_s f(x,s)$ exists and equals to $A(x)$. Cooper and Copestake [5] proved that

⋆ Soskova is supported by the Marie Curie Early Training grant MATHLOGAPS (MEST-CT-2004-504029).

⋆⋆ Wu is partially supported by a a start-up grant No. M48110008 and a research grant No. RG58/06 from NTU.

below $\mathbf{0}'_e$ there are e-degrees that are not Δ^0_2. These e-degrees are called *properly* Σ^0_2 *e*-degrees.

In this paper we are mainly concerned with the cupping property of Δ^0_2 e-degrees. An e-degree $\mathbf{a}$ is cuppable if there is an incomplete e-degree $\mathbf{c}$ such that $\mathbf{a} \cup \mathbf{c} = \mathbf{0}'_e$. In [6], Cooper, Sorbi and Yi proved that all nonzero Δ^0_2 e-degrees are cuppable and that there are noncuppable Σ^0_2 e-degrees.

Theorem 1. *(Cooper, Sorbi and Yi [6]) Given a nonzero Δ^0_2 e-degree* $\mathbf{a}$*, there is a total Δ^0_2 e-degree* $\mathbf{c}$ *such that* $\mathbf{a} \cup \mathbf{c} = \mathbf{0}'_e$*, where an e-degree is total if it contains the graph of a total function. Meanwhile, noncuppable e-degrees exist.*

In this paper we first prove that each nonzero Δ^0_2 e-degree $\mathbf{a}$ is cuppable to $\mathbf{0}'_e$ by a non-total Δ^0_2 e-degree.

Theorem 2. *Given a nonzero Δ^0_2 e-degree* $\mathbf{a}$*, there is a 1-generic Δ^0_2 e-degree* $\mathbf{b}$ *such that* $\mathbf{a} \cup \mathbf{b} = \mathbf{0}'_e$*. Since 1-generic e-degrees are quasi-minimal and 1-generic Δ^0_2 e-degrees are low,* $\mathbf{b}$ *is nontotal and low.*

Here a set A is *1-generic* if for every computably enumerable set S of $\{0,1\}$-valued strings there is some initial segment σ of A such that either S contains σ or S contains no extension of σ. An enumeration degree is 1-generic if it contains a 1-generic set. Obviously, no nonzero e-degree below a 1-generic e-degree contains a total function and hence 1-generic e-degrees are quasi-minimal. Copestake proved that a 1-generic e-degree is low if and only if it is Δ^0_2 (see [7]).

Our second result is concerned with cupping ω-c.e. e-degrees to $\mathbf{0}'_e$. A set A is n-c.e. if there is an effective function f such that for each x, $f(x,0) = 0$, $|\{s+1 \mid f(x,s) \neq f(x,s+1)\}| \leq n$ and $A(x) = \lim_s f(x,s)$. A is is ω-c.e. if there are two computable functions $f(x,s), g(x)$ such that for all x, $f(x,0) = 0$, $|\{s+1 \mid f(x,s) \neq f(x,s+1)\}| \leq g(x)$ and $\lim_s f(x,s) \downarrow = A(x)$.

An enumeration degree is n-c.e. (ω-c.e.) if it contains an n-c.e. (ω-c.e.) set. It's easy to see that the 2-c.e. e-degrees are all total and coincide with the Π_1 e-degrees, see [3]. Cooper also proved the existence of a 3-c.e. nontotal e-degree. As the construction presented in [6] actually proves that any nonzero n-c.e. e-degree can be cupped to $\mathbf{0}'_e$ by an $(n+1)$-c.e. e-degree, we will prove that any nonzero ω-c.e. e-degree is cuppable to $\mathbf{0}'_e$ by a 3-c.e. e-degree.

Theorem 3. *Given a nonzero ω-c.e. e-degree* $\mathbf{a}$*, there is a 3-c.e. e-degree* $\mathbf{b}$ *such that* $\mathbf{a} \cup \mathbf{b} = \mathbf{0}'_e$*.*

This is the strongest possible result. We explain it as follows. Consider the standard embedding ι of $\mathcal{D}_T$ to $\mathcal{D}_e$ given by: $\iota(deg_T(A)) = deg_e(\chi_A)$ where χ_A denotes the graph of the characteristic function of A. It is well-known that ι is an order-preserving mapping and that the Π_1 enumeration degrees are exactly the images of the Turing c.e degrees under ι. Consider a noncuppable c.e. degree $\mathbf{a}$. $\iota(\mathbf{a})$ is Π_1, hence ω-c.e., and $\iota(\mathbf{a})$ is not cuppable by any Π_1 e-degree, as ι preserves the least upper bounds. Therefore, no 2-c.e. e-degree cups $\iota(\mathbf{a})$ to $\mathbf{0}'_e$.

We use standard notation, see [4] and [10].

2 Basic Ideas of Cooper-Sorbi-Yi's Cupping

In this section we describe the basic ideas of Cooper-Sorbi-Yi's construction given in [6]. Let $\{A_s\}_{s<\omega}$ be a Δ_2^0 approximation of the given Δ_2^0 set A which is assumed to be not computably enumerable. We will construct two Δ_2^0 sets B and E (auxiliary) and an enumeration operator Γ such that the following requirements are satisfied:

$$S : \Gamma^{A,B} = \overline{K}$$
$$N_\Phi : E \neq \Phi^B$$

The first requirement is the global cupping requirement and it guarantees that the least upper bound of the degrees of A and B is $\mathbf{0}'_e$. Here $\Gamma^{A,B}$ denotes an enumeration operation relative to the enumerations of A and B.

The second group of requirements N_Φ, where Φ ranges over all enumeration operators, guarantees that the degree of B is not complete. Indeed, we have a witness — the degree of E is not below that of B.

To satisfy the global requirement S we will construct by stages an enumeration operator Γ such that $\overline{K} = \Gamma^{A,B}$. That is, at stage s we find all $x < s$ such that $x \in \overline{K}_s$ but $x \notin \Gamma^{A,B}[s]$, the approximation of $\Gamma^{A,B}$ at stage s, we define two markers a_x (bound of the A-part) and b_x (bound of the B-part and $b_x \in B$) and enumerate x into $\Gamma^{A,B}$ via the axiom $\langle x, A_s \upharpoonright a_x + 1, B_s \upharpoonright b_x + 1\rangle$. If x leaves $\overline{K}$ later, we can make this axiom invalid by extracting b_x from B or by a change (from 1 to 0) of A on $A_s \upharpoonright a_x + 1$. Intuitively we must use A-changes in the definition of Γ since otherwise B would be complete, contradicting the N-requirements. Since A is not in our control, if A does not provide such changes then we have to extract b_x out of B. We call this process the rectification of Γ at x.

Note that after stage s, at stage $t > s$ say, if $x \in \overline{K}_t$ but $A_t \upharpoonright a_x + 1 \not\subseteq A_t$ or $B_s \upharpoonright b_x + 1 \not\subseteq B_t$ then we need to put x into $\Gamma^{A,B}$ by enumerating a new axiom into Γ. If this happens infinitely often then x is not in $\Gamma^{A,B}$ and we cannot ensure that $\Gamma^{A,B}(x) = \overline{K}(x)$. To avoid this at stage t, when we re-enumerate x into $\Gamma^{A,B}$, we keep a_x the same as before, but let b_x be a bigger number. We put $b_x[t]$ into B and extract $b_x[s]$ from B (we want only one valid axiom enumerating x into $\Gamma^{A,B}$). Assuming that the G-strategies also do not change a_x after a certain stage, as A is Δ_2^0 there can be only finitely many changes in $A \upharpoonright a_x$ and hence we will eventually stop enumerating axioms for x in Γ.

Now we consider how to satisfy a N_Φ-requirement. We use variant of the Friedberg-Muchnik strategy. Namely, we select x as a witness, enumerate it into E and wait for $x \in \Phi^B$. If x never enters Φ^B then N_Φ is satisfied. Otherwise we will extract x from E, preserving $B \upharpoonright \phi(x)$ where $\phi(x)$ denotes the use function of the computation $\Phi^B(x) = 1$.

The need to preserve $B \upharpoonright \phi(x)$ conflicts with the need to rectify Γ. To avoid this before choosing x the N_Φ-strategy will first choose a (big) number k as its threshold and try to achieve $b_n > \phi(x)$ for all $n \geq k$. For elements $n < k$, S will be allowed to rectify Γ at its will. Whenever $\overline{K}$ changes below $k+1$ we reset this N_Φ-strategy by cancelling all associated parameters except for this k. Since k is fixed such a *resetting* process can happen at most $k + 1$ many times, so we can assume that after a stage large enough this N_{Φ}-strategy will never be reset anymore.

If k enters K, the threshold is moved automatically to the next number in $\overline{K}$. Since $\overline{K}$ is infinite, eventually, the threshold will stop changing its value. This threshold will be the real threshold of the corresponding N_{Φ}-strategy.

In order to be able to preserve some initial segment of B for the diagonalization, N_{Φ} will first try to move all markers b_n for elements $n \geq k$ above the restraint. A useful A-change will facilitate this. In the event that no such useful change appears we will be able to argue that A is c.e. contrary to hypothesis. To do this we will have an extra parameter U, aimed to construct a c.e. set approximating A.

The N_{Φ}-strategy works as follows at stage s:

Setup: Define a threshold k to be a big number. Choose a witness $x > k$ and enumerate it in E.

K**-Check:** If a marker b_n for an element $n \leq k$ has been extracted from B during Γ-rectification then restart the attack.

Attack:

1. If $x \in \Phi^B$ go to step 2. Otherwise return to step 1 at the next stage.
2. Approximate A by $A_s \upharpoonright a_k$ at stage s. Extract $b_k[s]$ from B. Cancel all markers a_n and b_n for $n \geq k$. Define a_k new, bigger than any element seen so far in the construction. Go to step 3.
3. Initialize all strategies of lower priority. If a previous approximation of A defined at stage $t < s$ is not true then enumerate $b_k[t]$ back in B, extract x from E and go to step 4, otherwise go back to step 1.
4. While the observed change in A is still apparent, do nothing. Otherwise enumerate x back in E and extract $b_k[t]$ from B, go back to step 3.

If after a large enough stage the strategy waits at 1 or 4 forever then the N_{Φ}-requirement is obviously satisfied. In the latter case $\Phi^B(x) = 1 \neq 0 = E(x)$ and the construction of Γ will never change the enumeration of $\Phi^B(x) = 1$ since all γ-markers are lifted to bigger values by the changes of A below $a_k[s]+1$. This strategy will not go from 1 or 4 back to 3 infinitely often and hence the N_{Φ}-requirement is satisfied. Otherwise as A is Δ^0_2 it would pass through 2 infinitely often. Let $t_1 < t_2 < \cdots < t_n < \cdots$ be the stages at which this strategy passes through 2. Then for each i, $A_{t_i} \upharpoonright a_k[t_i] + 1 \subset A$. By this property we argue that A is computably enumerable as follows: for each x, x is in A if and only if x is in A_{t_i} for some i, or

$$x \in A \quad \Leftrightarrow \quad \exists i (x \in A_{t_i}).$$

This contradicts our assumption on A.

3 Cupping by 1-Generic Degrees

In this section we give a proof of Theorem 2. That is, given an non-c.e. Δ^0_2 set A, we will construct a Δ^0_2 1-generic B satisfying the following requirements:

$S : \Gamma^{A,B} = \overline{K}$;
$G_i : (\exists\lambda \subset B)[\lambda \in W_i \vee (\forall\mu \supseteq \lambda)[\mu \notin W_i]]$.

If all requirements G_i together with the global requirement S are satisfied then B will have the intended properties. It is well known that the degree of a 1-generic set can not be complete.

Definition 1. *The tree of outcomes will be a perfect binary tree T. Each node $\alpha \in T$ of length i will be labelled by the requirement G_i. We will say that α is a G_i-strategy.*

At stage 0 $B = \emptyset$, $\Gamma = \emptyset$, $U_\alpha = \emptyset$ for all α and all thresholds and witnesses will be undefined.

At stage s we start by rectifying Γ and then construct a path through the tree δ_s of length s visiting all nodes $\alpha \subset \delta_s$ and performing actions as stated in the construction.

The Γ-rectification module for satisfying the global S requirement is as follows:

***Γ*-rectification module.** Scan all elements $n < s$ and perform the following actions for the elements n such that $\Gamma^{A,B}(n) \neq \overline{K}(n)$:

– $n \in \overline{K}$.

1. If $a_n \uparrow$, define $a_n = a_{n-1} + 1$(if n= 0, define $a_n = 1$). Note that this is the only case when the Γ-module changes the value of a_n. Once defined a_n can only be redefined due to a G-strategy. The idea is that eventually G-strategies will stop cancelling a_n, so that we can approximate $A \upharpoonright a_n$ correctly and obtain a true axiom for n.
2. If $b_n \downarrow$ then extract it from B and cancel all markers $b_{n'}$ for $n' > n$.
3. Define b_n to be big, i.e a number greater than any number mentioned in the construction so far, and enumerate it in B.
4. Enumerate in Γ the axiom $\langle n, A \upharpoonright a_n + 1, \{b_m | m \leq n\}\rangle$.

– $n \notin \overline{K}$
Then find all valid axioms in Γ for n – $\langle n, A \upharpoonright a + 1, M_n\rangle$ and extract the greatest element of M_n from B.

Construction of $\boldsymbol{\delta_s}$. We will define $\delta_s(n)$ for all $n < s$ by induction on n. Suppose we have already defined $\delta_s \upharpoonright i = \alpha$ working on requirement G_W. We will perform the actions assigned to α and choose its outcome $o \in \{0, 1\}$. Then $\delta_s(i) = o$.

α will be equipped with a threshold k and a witness λ, a finite binary string. When α is visited for the first time after initialization it starts from *Setup*. At further stages it always performs *Check* first. If the *Check* does not empty U_α then it continues with the *Attack* module from where it was directed to at the previous α-true stage. Otherwise it continues with the *Setup* to define λ again and then proceeds to step 1 of *Attack*.

Setup: If a threshold has not been defined or is cancelled then define k to be big – bigger than any element appeared so far in the construction. If a witness has not yet been defined choose a binary string λ of length $b_k + 1$ so that $\lambda = B \upharpoonright b_k + 1$.

Check: If a marker b_n for an element $n \leq k$ has been extracted from B during Γ-rectification at a stage t such that $s- < t \leq s$ where $s-$ is the previous α-true stage then initialize the subtree below α, empty U.

If $k \notin \overline{K}$ then define k to be the least $k' > k$ such that $k' \in \overline{K}$. I nitialize the subtree below α, empty U.

If b_k has changed since the last α-true stage and $\lambda \not\subseteq B$ then define λ to be $B \upharpoonright b_k$. Do not empty U.

Attack:

1. Check if there is a finite binary string $\mu \supseteq \lambda$ in W . If not then the outcome is $o = 1$. Return to step 1 at the next stage. If there is such a μ then remember the least one and go to step 2.
2. Enumerate in the guess list U a new entry $\langle A_s \upharpoonright a_k, \mu, b_k \rangle$. Extract b_k from B. Let $\hat{\mu}$ be the string μ but with position $b_k = 0$. For all elements $n > |\lambda|$ such that $\hat{\mu}(n)$ is defined let $B(n) = \hat{\mu}(n)$. Cancel all markers a_n and b_n for $n \geq k$. Define a_k to be bigger. Note that $\hat{\mu} \subset B$ and at the next stage Check will define a new value of λ to be $B \upharpoonright b_k + 1$ so that $\lambda \supseteq \hat{\mu}$. Go to step 3.
3. Initialize all strategies below α. Scan the guess list U for errors. The entries in the guess list will be of the following form $\langle U_t, \mu_t, b_t \rangle$ where U_t is a guess of A and b_t is the marker that was extracted from B when this guess was made at stage t. Note that to make $\mu_t \subset B$ we only need to enumerate b_t in B. If there is an error in the guess list, i.e. some $U_t \not\subseteq A_s$, then enumerate b_t in B and go to step 4 with current guess $G = \langle U_t, \mu_t, b_t \rangle$ where t is the least index of an error in U. If all elements are scanned and no errors are found go back to step 1.
4. If the current guess $G = \langle U_t, \mu, b_t \rangle$ has the property $U_t \not\subseteq A_s$ then let the outcome be $o = 0$. Come back to step 4 at the next stage. Otherwise extract b_t from B. If the Γ-rectification module has extracted a marker m for an axiom that includes b_t in its $B-$part since the last stage on which this strategy was visited then enumerate m back in B. Go back to step 3.

The Proof. Define the true path $f \subset T$ to be the leftmost path through the tree that is visited infinitely many times, i.e. $\forall n \exists^\infty t(f \upharpoonright n \subseteq \delta_t)$ and $\forall n \exists t_n \forall t > t_n(\delta_t \not<_L f \upharpoonright n)$.

Lemma 1. *For each strategy $f \upharpoonright n$ the following is true:*

1. *There is a stage $t_1(n) > t_n$ such that at all $f \upharpoonright n$-true stages $t > t_1(n)$ Check does not empty U.*
2. *There is a stage $t_2(n) > t_1(n)$ such that at all $f \upharpoonright n$-true stages $t > t_2(n)$ the Attack module never passes through step 3 and hence the strategies below $f \upharpoonright n$ are not initialized anymore, B is not modified by $f \upharpoonright n$, and the markers a_n for any elements n are not moved by $f \upharpoonright n$*

Proof. Suppose the two conditions are true for $m < n$. Let $f \upharpoonright n = \alpha$. Let t_0 be an α-true stage bigger than $t_2(m)$ for all $m < n$ and t_n.

Then after stage t_0 α will not be initialized anymore.

After stage t_0 all elements $n < k$ have permanent markers a_n. Indeed none of the strategies above α modify them anymore according to the induction hypothesis, strategies to the left are not accessible anymore and strategies to the right are initialized on stage t_0, hence the next time they are accessed they will have new thresholds greater than k.

The threshold k will stop shifting its value as $\overline{K}$ is infinite and we will eventually find the true threshold $k \in \overline{K}$.

As A is Δ_2^0, eventually all $A \upharpoonright a_n$ for element $n < k$ will have their final value and so will $\overline{K} \upharpoonright k$. Hence there is a stage $t_1(n) > t_0$ after which no markers b_n for elements $n \leq k$ will be extracted from B by the Γ-rectification and the *Check* module at α will never empty U again.

To prove the second clause suppose that the module passes through step 3 infinitely many times and consider the set $V = \bigcup L(U)$ where $L(U)$ denotes the left part of entries in the guess list U, that is the actual guesses at the approximation of A. By assumption A is not c.e. hence $A \neq V$.

If $V \nsubseteq A$ then there is a least stage t' and element p such that $p \in U_{t'} \backslash A$ and all U_t for $t < t'$ are subsets of A. Let $t_p > t_2$ be a stage such that the Δ_2^0 approximation of A settles down on p, i.e. for all $t > t_p$, $A_t(p) = A(p) = 0$. Then when we pass through step 3 after stage t_p we will spot this error, go to step 4 and never again return to step 3.

If $V \subset A$, let p be the least element such that $p \in A \backslash V$. Every guess in U is eventually correct and returns to step 1. To access step 3 again we pass through step 2, i.e. we pass through step 2 infinitely often. As a result a_k grows unboundedly and will eventually reach a value greater than p. As on all but finitely many stages t, $p \in A_t$, p will enter V. □

Corollary 1. *Every G_i-requirement is satisfied.*

Proof. Consider the G_i-strategy $\alpha = f \upharpoonright i$. Choose a stage $t_3 > t_2(i)$ from Lemma 1, after which the Attack module is stuck at step 1 or step 4, we have a permanent value for a_k and $A \upharpoonright a_k$ remains unchanged. Then so will the marker b_k and we will never modify λ again and $\lambda \subseteq B_t$ at all $t > t_3$.

If the module is stuck at step 1 we have found a string λ such that $\lambda \subset B$ and no string $\mu \supset \lambda$ is in the set W_i.

If the module is stuck at step 4 we have found a string μ from the guess $G = \langle U_t, \mu, b_t \rangle$ which is in W_i. It follows from the construction that $\mu \subset B$. The current markers b_n, for $n \geq k$ at stage t were cancelled and $b_k[t] = b_t$ was extracted from B. Any axiom defined after stage t has b-marker greater than $|\mu|$. Hence the Γ-rectifying procedure will not extract any element below the restraint $B \upharpoonright |\mu|$ from B. It does not extract markers of elements $n < k$. If $n \geq k$ and $n \in \overline{K}$ then its current marker is greater than $|\mu|$. If $n > k$ and $n \notin \overline{K}$ then any axiom defined before stage t is invalid, because its b-marker is already extracted from B at a previous stage $t_0 < t$ or else it has an A-component that contains as a subset $U_t \nsubseteq A$. □

Lemma 2. *The S-requirement is satisfied.*

Proof. At each stage s we make sure that Γ is rectified. For elements $n < s$, we have $\Gamma^{A,B}(n)[s] = \overline{K}(n)[s]$. This is enough to prove that $n \notin \overline{K} \Rightarrow n \notin \Gamma^{A,B}$. Indeed if we assume that $n \in \Gamma^{A,B}$ then there is an axiom $\langle n, A_n, M_n\rangle \in \Gamma$ and $A_n \subseteq A$, $M_n \subset B$. Hence this axiom is valid on all but finitely many stages. But according to our construction we will ensure $M_n \nsubseteq B$ on infinitely many stages, a contradiction.

To prove the other direction, $n \in \overline{K} \Rightarrow n \in \Gamma^{A,B}$, we have to establish that the N-strategies will stop modifying the markers a_n and b_n eventually. Indeed the markers can be modified only by N-strategies with thresholds $k < n$. The way we choose each threshold guarantees that there will be only finitely many nodes on the tree with this property. The nodes to the left of the true path will eventually not be accessible anymore and the nodes to the right will be cancelled and will choose new thresholds, bigger then n. Lemma 1 proves that every node along the true path will eventually stop moving a_n and b_n by property 2.

Suppose the markers are not modified after stage t_1. After stage t_1, a_n has a constant value. As A is Δ^0_2 there will be a stage $t_2 > t_1$ such that for all $t > t_2$ $A \restriction a_n[t] = A \restriction a_n$. At stage $t_2 + 1$ we rectify Γ. If $n \in \Gamma^{A,B}$ then there is an axiom $\langle n, A_n, M_n\rangle$ in Γ such that $A_n \subset A \restriction a_n$ and at all further stages this axiom will remain valid, so the Γ-rectifying procedure will not modify it again. Otherwise it will extract a b-marker for the last time and enumerate an axiom $\langle n, A \restriction a_n, M'_n\rangle$ that will be valid at all further stages. In both cases we have found an axiom for n that is valid on all but finitely many stages, hence $n \in \Gamma^{A,B}$. □

Lemma 3. *B is Δ^0_2.*

Proof. We need to show that for each n, n can be put in and moved out from B at most finitely times. To see this fix n and consider the G_i-strategy along the true path that has a threshold $k_i > n$. As we have already established in Corollary 1 there is a stage $t_3 > t_2(i)$, after which we will never modify λ_i again and $\lambda \subseteq B_t$ on all $t > t_3$. As $n < |\lambda_i|$ then $B_t(n)$ will remain constant on all stages $t > t_3$. This means that $B(n)$ changes at most t_3 many times. □

4 Cupping the ω-c.e. Degrees

In this section we give a proof of Theorem 3. Suppose we are given an ω-c.e. set A with bounding function g. We will modify the construction of the set B so that it will turn out to be $3 - c.e.$. The requirements are:

$$\begin{aligned} S &: \Gamma^{A,B} = \overline{K} \\ N_\Phi &: E \neq \Phi^B \end{aligned}$$

The structure of the axioms enumerated in Γ will be more complex. Again we will have an a-marker a_n for each element n, but instead of just one marker b_n we will have a set of b-markers B_n of size g_n+1 where $g_n = \sum_{x<a_n} g(x)$ together

with a counter c_n that will tell us which element we should extract if we need to. Every time $A \upharpoonright a_n$ changes we will extract from B a different element – the c_n-th element $b_n \in B_n$ and then add 1 to c_n to ensure that each element in B will be extracted only once. If we need to restore a computation due to the N-strategies we will enumerate the extracted marker back in B, hence B is $3-c.e.$. Note that if a restored computation has to be destroyed again, we will need to extract a different marker from B. This could destroy further computations. That is why will always try to restore the last computation $\Phi^B(x)$.

***Γ*-rectification module.** Scan all elements $n < s$ and perform the following actions for the elements n such that $\Gamma^{A,B}(n) \neq \overline{K}(n)$:

- $n \in \overline{K}$.
  1. If $a_n \uparrow$ then define $a_n = a_{n-1} + 1$(if n = 0, define $a_n = 1$).
  2. If $B_n \downarrow$. Extract the c_n-th member of B_n. Move the counter c_n to the next position $c_n + 1$. Cancel all $B_{n'}$ for $n' > n$.
  3. If $B_n \uparrow$ then define a set of new markers B_n of size $g_n + 1$ where $g_n = \sum_{x<a_n} g(x)$ and a new counter $c_n = 1$ and enumerate B_n in B.
  4. Enumerate in Γ the axiom $\langle n, A_s \upharpoonright a_n + 1, \bigcup\{B_{n'}(c_{n'}) | n' \leq n\}\rangle$ where $B_{n'}(c_{n'})$ is the set of all elements in $B_{n'}$ with positions greater than or equal to $c_{n'}$.
- $n \notin \overline{K}$
  Then find all valid axioms in Γ for n – $\langle n, A_t \upharpoonright a + 1, M_n\rangle$ where $M_n = \bigcup\{B_{n'} | n' \leq n\}$ and extract the least member of B_n that has not yet been extracted from B. Increment the counter c_n that corresponds to the set of markers B_n.

Construction of $\boldsymbol{\delta_s}$. Setup: If a threshold has not been defined or is cancelled then define k to be big, bigger than any element appeared so far in the construction. If a witness has not yet been defined choose $x > k$ and enumerate it in E.

Check: If a marker from B_n for an element $n < k$ has been extracted from B during Γ-rectification at a stage t, $s- < t \leq s$ where $s-$ is the previous α-true stage, then initialize the subtree below α, empty U.

If $k \notin \overline{K}$ then shift it to the next possible value and redefine x to be bigger. Again initialize the subtree below α and empty U.

Attack:

1. Check if $x \in \Phi^B$. If not then the outcome is $o = 1$, return to step 1 at the next stage. If $x \in \Phi^B$ go to step 2.
2. Initialize all strategies below α. Scan the guess list U for errors. If there is an error then take the last entry in the guess list, say the one with index t: $\langle U_t, B_t, c_t\rangle \in U$ and $U_t \nsubseteq A_s$. Enumerate the $(c_t - 1)$-th member of B_t back in B. Extract x from E and go to step 4 with current guess $G = \langle U_t, B_t, c_t\rangle$. If all elements are scanned and no errors are found go to step 3.

3. Enumerate in the guess list U a new entry $\langle A_s \restriction a_k, B_k, c_k \rangle$. Extract the c_k-th member of B_k from B and move c_k to the next position $c_k + 1$. Cancel all markers a_n and B_n for $n \geq k$. Define a_k new, bigger than any element seen so far in the construction. Go to back to step 1.
   Note that this ensures that our guesses at the approximation of A are monotone. Hence if there is an error in the approximation, this error will be apparent in the last guess. This allows us to always use the computation corresponding to the last guess. We will always be able to restore it.
4. If the current guess $G = \langle U_t, B_t, c_t \rangle$ has the property $U_t \not\subseteq A_s$ then let the outcome be $o = 0$. Come back to step 4 at the next stage. Otherwise enumerate x back in E and extract the c_t-th member of B_t from B and move the value of the counter to $c_t + 1$. If at this stage during the Γ-rectification procedure a different marker m for an axiom that contains B_t was extracted then enumerate m back in B. Go back to step 1.

The Proof. The construction ensures that for any n, at any stage t, at most one axiom in Γ defines $\Gamma^{A,B}(n)$. Generally, we extract a number from B_n to drive n out of $\Gamma^{A,B}$. When an N-strategy α acts at step 3 of the Attack module, at stage s say, α needs to preserve $\Phi^B(x_\alpha)$. All lower priority strategies are initialized and an element b_1 in B_{k_α} is extracted from B to prevent the S-strategy from changing B on $\phi(x_\alpha)$. Note that all axioms for elements $n \geq k_\alpha$ contain B_{k_α}. So at stage s, when we extract b_1 from B, n is driven out of $\Gamma^{A,B}$. As in the remainder of the construction, at any stage, we will have either that A has changed below a_n or B has changed on B_n, these axioms will never be active again. As the Γ-module acts first, it may still extract a marker m from an axiom for $n > k_\alpha$ if $A \restriction a_n$ has changed back and thereby injure $B \restriction \phi(x_\alpha)$. But when α is visited it will correct this by enumerating m back in B and extracting a further element $b_2 \in B_{k_\alpha}$ from B to keep Γ true. This makes our N-strategies and the S-strategy consistent. We comment here that such a feature is also true in the proof of Theorem 2, but there we do not worry about this as we are constructing a Δ^0_2 set. In the proof of Theorem 3, this becomes quite crucial, as we are constructing B as a 3-c.e. set, and we have less freedom to extract numbers out from B.

The construction ensures that B is a $3 - c.e.$ set. First we prove that the counter c_n never exceeds the size of its corresponding set B_n and therefore we will always have an available marker to extract from B if it is necessary.

Lemma 4. *For every set of markers B_n and corresponding counter c_n at all stages of the construction $c_n < |B_n|$ and the c_n-th member of B_n is in B.*

Proof. For each set of markers B_n only one node along the true path can enumerate its elements back into B. Indeed if B_n enters the guess list U_t at some node α on the tree then at stage t, B_n is the current set of markers for n and n is the threshold for α. When α enumerates B_n in its U_t, it cancels the current markers for the element n. Hence B_n does not belong to any $U^\beta_{t'}$ for $t' \leq t$ and any node β or else B_n will not be current and B_n will not enter $U^\beta_{t''}$ at any stage $t'' \geq t$ and any node β as it is not current anymore.

We ensure that n being a threshold is in $\overline{K}$, hence after stage t the Γ-rectification procedure will not modify $B \restriction B_n$. Before stage t while the markers were current the counter c_n was moved only when the Γ-rectification procedure observed a change in $A \restriction a_n$, i.e some element that was in $A \restriction a_n$ at the previous stage is not there anymore. After stage t α will move the marker c_n once at entry in U_t and then only when it observes a change in $A \restriction a_n$, i.e $U_n = A \restriction a_n[t]$ was a subset of A at a previous step but is not currently. Altogether c_n will be moved at most $g_n + 1 < |B_n|$ times.

Otherwise B_n belongs to an axiom which contains the set B_k for a particular threshold k and $n \notin \overline{K}$. Then again its members are enumerated back in B only in reaction to a change in $A \restriction a_n$. □

We will now prove that Lemma 1 is valid for this construction as well. Note that this construction is a bit different, therefore we will need a new proof. The true path f is defined in the same way.

Lemma 5. *For each strategy $f \restriction n$ the following is true:*

1. *There is a stage $t_1(n) > t_n$ such that at all $f \restriction n$-true stages $t > t_1(n)$ Check does not empty U.*
2. *There is a stage $t_2(n) > t_1(n)$ such that at all $f \restriction n$-true stages $t > t_2(n)$ the Attack module never passes through step 2 and hence the strategies below α are not initialized anymore, B is not modified by $f \restriction n$, and the markers a_n for any elements n are not moved by $f \restriction n$*

Proof. Suppose the two conditions are true for $m < n$. Let $f \restriction n = \alpha$. Let t_0 be an α-true stage bigger than $t_2(m)$ for all $m < n$ and t_n.

Then after stage t_0 α will not be initialized anymore. The proof of the the existence of stage $t_1(n)$ satisfying the first property is the same as in Lemma 1.

To prove the second clause suppose that the module passes through step 2 infinitely many times and consider the set $V = \bigcup L(U)$ where $L(U)$ denotes the left part of entries in the guess list U. By assumption A is not c.e. hence $A \neq V$.

If $V \nsubseteq A$ then there is element p such that $p \in V \backslash A$. Let $t_p > t_2$ be a stage such that the approximation of A settles down on p, i.e. for all $t > t_p$, $A_t(p) = A(p) = 0$. Then when we pass through step 2 after stage t_p we will spot this error, go to step 4 and never again return to step 1.

If $V \subset A$, let p be the least element such that $p \in A \backslash V$. Every guess in U is eventually correct and allows us to move to step 3, i.e. we pass through step 3 infinitely often. As a result a_k grows unboundedly and will eventually reach a value greater than p. As on all but finitely many stages t, $p \in A_t$, p will enter V. □

Corollary 2. *Every N_i-requirement is satisfied.*

Proof. Let $\alpha \subset f$ be an N_i-strategy. As a corollary of Lemma 5 there is a stage $t_3 > t_2(i)$ after which the *Attack* module is stuck at step 1, and hence $x \notin \Phi^B$, but $x \in E$. Or else the module is stuck at step 4, in which case $x \in \Phi^B$ and $x \notin E$. Indeed step 4 was accessed with $G = \langle U_t, B_t, c_t \rangle$, belonging to the last entry in the guess list $\langle U_t, B_t, c_t \rangle$. At stage t we had $x \in \Phi^B[t]$. The current markers b_n , for $n \geq k$ were cancelled and $b_k[t]$ was extracted from B. Hence the Γ-rectifying procedure will not extract any element below the restraint $B \upharpoonright \phi(x)$ from B. It does not extract markers of elements $n < k$. If $n \geq k$ and $n \in \overline{K}$ then its current marker is greater than $\phi(x)$. If $n > k$ and $n \notin \overline{K}$ then any axiom defined before stage t is invalid, because one of its b-markers is extracted from B at a previous stage or else it has an A-component $U_t \not\subseteq A$. Any axiom defined after stage t has b-markers greater than $\phi(x)$.

After stage t, if α modifies B it will be in the set of markers B_t, and when step 4 is accessed we have $B_t \subset B$. □

Lemma 2 is now valid for Theorem 3 as well, hence all requirements are satisfied and this concludes the proof of Theorem 3.

Acknowledgments. We thank an anonymous reviewer for pointing out an inaccuracy in the previous version of this paper.

References

1. Cooper, S.B.: Partial degrees and the density problem. J. Symb. Log. 47, 854–859 (1982)
2. Cooper, S.B.: Partial Degrees and the density problem. part 2: the enumeration degrees of the Σ_2 sets are dense. J. Symb. Log. 49, 503–513 (1984)
3. Cooper, S.B.: Enumeration reducibility, nondeterminitsic computations and relative computability of partial functions. In: Ambos-Spies, K., Müller, G., Sacks, G.E (eds.) Recursion Theory Week, Oberwolfach 1989. Lecture Notes in Mathematics, vol. 1432, pp. 57–110. Springer, Heidelberg (1990)
4. Cooper, S.B.: Computability Theory, Chapman & Hall/CRC Mathematics, Boca Raton, FL, New York, London (2004)
5. Cooper, S.B., Copestake, C.S.: Properly Σ_2 enumeration degrees. Zeits. f. Math. Logik. u. Grundl. der Math. 34, 491–522 (1988)
6. Cooper, S.B., Sorbi, A., Yi, X.: Cupping and noncupping in the enumeration degrees of Σ^0_2 sets. Ann. Pure Appl. Logic 82, 317–342 (1996)
7. Copestake, K.: 1-Genericity in the enumeration degrees below $0'_e$. In: Petkov, P.P. (ed.) Mathemcatical Logic, pp. 257–265. Plenum Press, New York (1990)
8. Copestake, K.: 1-Genericity enumeration Degrees. J. Symb. Log. 53, 878–887 (1988)
9. Gutteridge, L.: Some Results on Enumeration Reducibility, PhD thesis, Simon Fraser University (1971)
10. Soare, R.I.: Recursively enumerable sets and degrees. Springer, Berlin, Heidelberg, London, New York, Paris, Tokyo (1987)

What Is the Lesson of Quantum Computing?
(Extended Abstract)

Christopher G. Timpson

Division of History and Philosophy of Science, Department of Philosophy,
University of Leeds, Leeds, LS2 9JT, UK
c.g.timpson@leeds.ac.uk

It would be a mistake to seek for a *single* lesson that the advent of quantum computing has provided: the theory is rich in both physics and computer science terms. But my quarry is *a*, or perhaps *the*, central *conceptual* point that we should draw; and it concerns putative shifts in our understanding of the Church-Turing hypothesis inspired by reflection on quantum computation.

Deutsch [1,2] argues forcefully that a principle called the Turing Principle ought to replace the familiar Church-Turing hypothesis [3,4] as the centrepiece of the theory of computation:

> **Turing Principle:** *Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means.*
>
> **Church-Turing hypothesis:** *The class of effectively calculable functions is the class of Turing machine computable functions.*

He takes his Turing Principle to support and be supported by the notion of quantum computers; and finds it noteworthy that the Principle is satisfied in a quantum world, but not in a classical one.

The point here really concerns liberation: The classical Turing machine, abstractly characterised, has dominated theorising since its conception. What the development of quantum computers has shown is that *just* focusing on abstract computational models, in isolation from the consideration of the physical laws governing the objects that might eventually have to implement them, can be to miss a lot.

What the progenitors of quantum computation (Benioff, Feynman, Deutsch) realised was that the question of *what computational processes fundamental physics might allow* was a very important one; and one which had typically been neglected in the purely mathematical development of computer science. One can argue that Turing's model of computing involved implicit classical assumptions about the kinds of physical computational processes there could be; hence his model was not the most general, hence Feynman's tongue-in-cheek remark *a propos* Turing: 'He thought he understood paper'[1]. This is the inspiration for Deutsch's move to the Turing Principle, for that Principle is intended to be *minimally committed* physically.

[1] Cited by [2, p.252].

But while the point about liberation is very important indeed, I shall suggest that we ought not to buy in to Deutsch's version of it by endorsing the Turing Principle. Instead we ought to think rather harder about what propositions like the Church-Turing hypothesis and the Turing Principle are for; and to be aware of salient differences between some of these propositions.

As many commentators have urged (e.g., [5,6,7]) it is essential to distinguish between the Church-Turing hypothesis proper (as above), which is a stipulation or definition of how the rough intuitive notion of effective calculability is to be understood and concerns primarily what can be computed by *human agents*; and what are sometimes called *physical* versions of the thesis, which come in various flavours, but are roughly of the form:

> **Physical Church-Turing thesis:** *The class of functions that can be computed by any physical system is co-extensive with the Turing computable functions.*

Notice that the kind of evidence that might be cited in support of these respective theses is quite different. In fact, since the first is a stipulation, it wouldn't make sense to offer evidence in support of its truth. All one can do is offer reasons for or against it as a good definition. The facts that are typically cited to explain its entrenchment are precisely of this form: one points to all the different attempts at capturing the notion of algorithm or of the effectively calculable: they all return the same class of functions. This tells us that Church and Turing did succeed in capturing the intuitive notion exceedingly well: we have no conflict with our pre-theoretic notions.

By contrast, the physical thesis is an empirical claim and consequently requires inductive support. It's truth depends on what you can get physical systems to do for you. The physical possibility of, for example, Malament-Hogarth spacetimes [8,9] would prove it wrong. It's not clear how much direct or (more likely) indirect inductive support it actually possesses, certainly it should not be thought as deservedly entrenched as the Church-Turing hypothesis, although many are inclined to believe it. (See Pitowsky, this volume, for further discussion.)

Now Deutsch's Turing Principle has more in common with physical versions of the Church-Turing thesis: it is the 'liberated' analogue: it is concerned to describe the ultimate limits of physical computation, computation by machines; and is notionally unconstrained by parochial reference to any particular physical theory. But suppose it were true. Would that mean that there is no place for something like the original Church-Turing hypothesis? I shall argue not. Not only because their subject matter is different (characterising the effectively calculable vs. delimiting the ultimate abilities of mechanical computers) but more significantly because there is a further dimension to consider: the question of where physical evolutions get their mathematical meaning from.

What can be computed in physical reality has two sorts of determinant, mathematical and physical. The mathematical determines what the evolution of given physical states into others in a certain way would mean, what would have been computed by such a process; and the physical determines whether such a process can occur. Identifying the 'naturally computable' functions with those that can be computed by any physical system (as in Deutsch's Turing Principle) we emphasize the physical determinant to the exclusion of the mathematical one—we say that what can be computed is *whatever* can be computed by *any* physical system, but we have not said what, if anything, these various physical processes amount to in mathematical terms [10].

Thus I shall argue that while postulates like the Turing Principle and physical versions of the Church-Turing thesis might well have an important role to play in characterising the limits of machine computation, they do not stand on their own as potential foundations for the theory of computation. We require in addition definitions along the lines of the original Church-Turing hypothesis, which are apt to endow mathematical meaning on physical evolutions. In making this argument I will therefore endorse the line that whether or not a given physical process counts as a computation is not an intrinsic property of that process, but is rather an *extrinsic* one. Physical processes are only adventitiously computational: they gain their status because we mark them so; or so I shall argue.

References

1. Deutsch, D.: Quantum theory, the Church-Turing Principle and the universal quantum computer. Proceedings of the Royal Society of London A 400, 97–117 (1985)
2. Deutsch, D.: The Fabric of Reality. Penguin Books, London (1997)
3. Church, A.: An unsolvable problem of elementary number theory. American Journal of Mathematics. 58: pp. 345–365, repr. in [11] pp.89-107 (1936)
4. Turing, A.: On Computable Numbers, with an application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society, 42: pp. 230–265, repr. in [11] pp.116-51 (1936)
5. Jack Copeland, B.: Narrow versus wide mechanism: Including a re-examination of Turing's views on the mind-machine issue. The Journal of Philosophy, XCVI(1) (2000)
6. Jack Copeland, B.: The Church-Turing thesis, The Stanford Encyclopedia of Philosophy (2002), http://plato.stanford.edu/archives/fall2002/entries/church-turing/
7. Pitowsky, I.: Quantum speed-up of computations. Proceedings of PSA 2000, Symposia papers 69(3), 168–177 (2002)
8. Hogarth, M.: Non-Turing computers and non-Turing computability. Philosophy of Science Supplementary, I: pp. 126–138 (1994)
9. Shagrir, O., Pitowsky, I.: Physical hypercomputation and the Church-Turing thesis. Minds and Machines 18, 87–101 (2003)
10. Timpson, C.G.: Quantum computers: The Church-Turing hypothesis versus the Turing Principle. In: Teuscher, C. (ed.) Alan Turing: Life and Legacy of a Great Thinker, pp. 213–240. Springer, Berlin Heidelberg (2004)
11. Davis, M. (ed.): The Undecidable. Raven Press, Hewlett, New York (1965)

Does the Cell Compute?

Giuseppe Trautteur

Dipartimento di Scienze fisiche, Università di Napoli Federico II

Abstract. We propose a tentative parallel between computational features and cellular operations in order to explore the possibility that the cell itself is intrinsically and not metaphorically a symbol-processing, computing entity. Central to this possible identification is the notion of virtuality, as understood in computational theory.

Keywords: Systems biology, Computability, Virtuality.

1 Introduction

Natural Computing works bidirectionally. In one direction, biological inspiration drives the search for computational systems with desirable characteristics of natural systems, such as robustness or self-repair. Notable examples include neural networks, evolutionary algorithms and genetic programs, p-systems or membrane computing, ant-computing, etc. In the opposite direction, computation is applied to natural systems. This is done in an instrumental or heuristic way (e.g. data mining of accrued results, plain data processing in experiments, modeling), but also by looking at natural systems *qua* computational systems.

Investigations of natural systems *qua* computational ones – and here 'natural' is practically synonymous with 'biological' – are mainly concerned with nervous systems and cellular machinery. But is there actual computing outside the human mind and the computing machinery created by the same human mind? Does the brain – as distinguished from mind – compute? Is the cellular machinery intrinsically computational?

We address here the latter question, which is an instance of the more general problem whether computing is a natural kind. It is worth noting that this question is not settled by showing that the cell can be computationally simulated: it has indeed now become a commonplace to remark that in the computing room of a hurricane simulation it does not rain.

The real issue at stake is whether the cell itself is – intrinsically, and not metaphorically – a symbol-processing, computing entity.

These questions give rise to epistemologically intriguing problems and might be the origin of useful insights for the development of the understanding of cellular machinery. As examples one could look at formal methods developed in Computer science to cope with parallelism, cooperation, and concurrency in programs with the aim of carrying over those methods on regulatory networks of the cell, or at model checking techniques for assessing properties of (cell) programs, for instance sustainability of an internal trophic path in a cell. On a more abstract plane, concepts of self-maintenance and reproduction, basilar to the endeavour of establishing the concept of minimal gene set [2], might be usefully connected with von Neumann's self reproducing machines [4], or the intriguing [3], and the underlying recursion theorem [6].

Therefore, we now turn to consider whether distinguishing features of computing entities are present in cell machinery. More specifically, we ask whether virtuality – which is, in our view, one of the key properties of computing entities [10] – is present in cell machinery.

2 Virtuality in Computational Systems

By virtuality in computational systems we understand the capability of interpreting, transforming, and running machines which are present, as it were, only under their code (program) aspect. The theoretical underpinning of virtuality is, of course, the normal form theorem of Partial recursive functions theory and the closely related notion of Universal Turing Machine (UTM) [7].

Distinctive features of the notion of virtuality that directly take their origin in computability theory are:

- the capability for *creating* or *modifying* machine code, and
- the capability for *simulating* or rather *realizing* the behaviour of any member of some given class of computing machines by *interpreting* the *stored* code of the machine to simulate.

In a recent paper [10] it was proposed – in connection with brain activity – to consider virtuality, rather than discreteness and finiteness, as the hallmark of computation, insofar virtuality allows the physical entities so endowed with some sort of "immaterial" or "as if" capability consisting in the substitution of actual machinery by its (code) description. Is there anything like virtuality in biological cell operation?

3 Metabolic Pathways and the Coding of Programs

In Systems biology [5] it is now customary to treat metabolic pathways as if they were algorithms or programs, albeit under the name of regulatory networks, genic expression networks, etc.

Closed and intertwined plain metabolic pathways can be seen as instances of multiply nested and looped programs except that the program appears to be written nowhere. It is as if it were implicit in the physical structure of the loop.

Pathways of the same kind, but which distinctively involve steps including actual gene expression, do possess a measure of explicit representation of the program, insofar as the enzymatic proteins which take part in the pathway's execution are assembled out of a read-only memory (ROM), which is formed by strings of genetic material. However, those strings of genetic material are not complete and explicit programs for the pathway. There seem to be two components in them, according to an intuition originally formulated by Gunther Stent[8]:

- the sequence of bases of the operon or set of interrelated genes, which can be looked upon as a text specifying the primary structure of the proteins and, by way of the actual transcription process, its ternary and active structure;
- the unique placement of the genes so that their appearance and subsequent presence in the cytoplasm identifies and enacts the actual path of reactions.

An important step forward in the algorithmic interpretation of pathways would ensue from the discovery of biochemical constraints on the abstract shape (loops, nesting, interconnection between pathways as co-routines, etc.) of a pathway due to the concentration, possibly timed appearance, diffusion and brownian motion, etc. of the chemical species coded for by genic material. The active elements (proteins), because of their chemical affinities, could not but interact between themselves and the extant (intra and extra cellular) environment in the way "designed" in the ROM. The existence of such constraints would make more explicit the coding of the entire biochemical program (the pathway) in terms of the (sequences) of genic texts.

4 The Search for Full-Fledged Virtuality in the Genic-Metabolic Complex

An important feature of virtuality as applied to those systems that are called universal in computability theory, is the belonging of the interpreting program to the system itself: e.g. the UTM, the interpreter of a language written in the language itself, etc.

Now there might exist different kinds of (sub-)universalities in which a given system does include an interpreter for some important although incomplete class of programs. In the case of the cell machinery, the gene expression/metabolic pathways complex may not be universal in the sense of being able to realize any generic chemical process for the production of any chemical species, but still capable, within itself, of coding the processing of at least those pathways and related chemical species which allow its own successful self-maintenance and reproduction in some range of environments.

In order to deepen the analysis of the relations between fixed machinery and simulated behaviour we will consider three different time scales observable in the biological world:

- the time scale appropriate to the description of behavior (units of milliseconds to minutes);
- ditto for development and learning (except for one-shot learning, units of hours to years);
- evolutionary time scale (up to hundreds of millions of years).

The reason for introducing these time scales is that, making use of the notion of adiabatic learning hypothesis introduced in [1], we want to consider the structures of the lower levels as fixed or frozen with respect to the activities (behaviour) of the upper level. More precisely the adiabatic hypothesis maintains that over durations short with respect to the lower level time scale the phenomena on the upper level can be studied as if the structures of the lower scale were constant in time.

For procariotes, with which we are primarily concerned here, it is hard to consider development or learning of single cells. We therefore consider only two time scales and orders of activities:

– metabolic and reproductive actions of a single cell (minutes);
– processing of the genome (many generations: hours, days).

The actual cell machinery on the level of metabolic and reproductive actions could be taken to be some sort of fixed "universal machine" for assembling – out of the inert, but potentialy reactive nutrient material – a very large number of special purpose machines (the biochemical processing plants) of varying degrees of complexity, and up to that "universal machine" which is the cell itself.

It can be therefore surmised that at least the second of the virtuality features, i.e. the capability for simulating or realizing the behaviour by *interpreting* some stored code, is present in nature and confirmed by genetic engineering, recombinant DNA technology and reproductive cloning. However, because of the validity of the Central Dogma, the only active processing of the explicit program, the DNA which acts as a ROM, consists in its duplication – possibly with crossingover in eucariotes. Further processing of the genome does not exist and therefore full virtuality, including the capability for *creating* or *modifying* machine code, does not appear to subsist. Indeed the modification or creation of program text is evolutionary in nature and resorts uniquely to the slowest, evolutionary time scale. The methods used in genetic engineering, which do process the genome, are entirely outside the actual operation of cell machinery.

Further steps in search of virtuality in cell machinery require a detailed examination of specific cellular pathways, with the aim of isolating plausible candidates for biological "realizers" of algorithmic processes that give rise to virtual behaviour. Accordingly, we conclude this extended abstract by proposing a tentative table of correspondences between algorithmic precursors of virtual behaviour and their candidate realizers in the cell.

Features of computability	**Features of the genic-metabolic complex**
Program interpretation and execution.	Transcription of gene or operon + ribosome translation. Execution is the combined running of chemical reactions dictated by the genome.
Program creation and modification.	The genome has unknown origins and cannot be altered except by agencies such as radiation or a chemically aggressive, mutagenic, environment. Spontaneous or internal mutations also occur, mainly because of deamination or flipping of tautomeric equilibria. However, the frame of analysis we are concerned with is situated at the fast metabolic and reproductive time scale level

	and, according to the adiabatic hypothesis, the genic (symbolic, DNA) structure is fixed. In such a frame of analysis the behaviour of the cell is stationary: i.e. the rate of variation of concentration of its metabolites is on the average zero. This implies, as is well known, that the vector of the metabolites fluxes belongs to the (right) null space of the S-matrix of the cell [5]. We surmise that the fluxes of the mutagenic factors are kinetically negligible, and so excluded from the S-matrix, also on account of the strong defenses of the genome against mutation. Were the mutagenic agencies included in the overall S-matrix, including the external ones, we might gain true virtuality (a program acting on programs including itself) and have a theory also at the evolutionary time scale level. But would it be computational? Its essential stochastic nature poses formidable problems.
Conditionals.	Fluxes of enzimes.
Addressing. This universal feature of concrete programming systems is not theoretically necessary for full fledged computational capability [9]. Rewriting systems only need a unidimensional proximity relation, rather akin to enzymatic site recognition.	Via specific affinities of chemical species and diffusion or brownian motion processes in cytoplasm.
Locations and values.	Locations are chemical species, values their concentrations or activities.
Procedure calls, constructs both hardware and software for concurrency, cooperation and distributed processing.	The activation of a gene expression path by intertwined and nested chains of reactions, possibly arising from the environment. Allosteric activation.
Memory. Stored programs.	The genome: ROM; transient storage in RNA.
Physical machinery, usually electronic. CPU chips are UTMs.	Purine-pyrimidine bases chains and the 20 amino acids.

Acknowledgements. The author acknowledges gratefully the many discussions and material help generously proffered by Carmine Garzillo and Guglielmo Tamburrini. Francesco Donnarumma and Paolo Coragggio are gratefully thanked for some very helpful editing. Two anonymous referees' comments and consequent recommendations have been conducive to a number of revisions and expansions. They are hereby thanked.

References

1. Caianiello, E.R: Outline of a theory of thought-processes and thinking machines. J Theor. Biol. 1, 204–235 (1961)
2. Gil, R., Silva, F.J., Peretó, J., Moya, A.: Determination of the Core of a Minimal Bacterial Gene Set. Microbiology and Molecular Biology Reviews, 68 pp. 518–537 (2004) Koonin, E.V.: How Many Genes Can Make a Cell: The Minimal-Gene-Set Concept. Annual Review of Genomics and Human Genetics 1 pp. 99–116 (2000)
3. Myhill, J.: The abstract Theory of Self-Reproduction. In: Mesarovic, M.D. (ed.): Views on General Systems Theory, Proceedings of the Second Systems Symposium at Case Institute of Technology. John Wiley and Sons pp. 106–118 (1964) also reprinted in Burks, A.W. (ed.): Essays on Cellular Automata. University of Illinois Press pp.206–218 (1970)
4. von Neumann, J.: Theory of Self-Reproducing Automata. Edited and completed by Burks, A.W. Univ. of Illinois Press, Urbana (1966)
5. Palsson, B.Ø.: Systems Biology. Cambridge University Press, Cambridge 2006.
6. Rogers Jr, H.: Theory of partial recursive functions and effective computability, pp. 188–190. McGraw-Hill, New York (1967)
7. Rogers Jr, H.: Theory of partial recursive functions and effective computability. McGraw-Hill, Chap. 1, Theorems IV and X pp. 22–23, 29–30 (1967)
8. Stent, G.: Explicit and Implicit Semantic Content of the Genetic Information. In: The Centrality of Science and Absolute Values, vol. 1, Fourth International Conference on the Unity of the Science, New York. International Cultural Foundation, pp. 261–277 (1975)
9. Trautteur, G.: On Addresses. Proc. of the 1988 IEEE International Conf. on Systems, Man and Cybernetics, Beijing II, 1267–1269 (1988)
10. Trautteur, G., Tamburrini, G.: A note on discreteness and virtuality in analog computing. Theoretical Computer Science 371, 106–114 (2007)

Computational Complexity of Constraint Satisfaction

Heribert Vollmer

Theoretische Informatik, Universität Hannover, Appelstr. 4, D-30167 Hannover
vollmer@thi.uni-hannover.de

Abstract. The input to a constraint satisfaction problem (CSP) consists of a set of variables, each with a domain, and constraints between these variables formulated by relations over the appropriate domains; the question is if there is an assignment of values to the variables that satisfies all constraints. Different algorithmic tasks for CSPs (checking satisfiability, counting the number of solutions, enumerating all solutions) can be used to model many problems in areas such as computational logic, artificial intelligence, circuit design, etc. We will survey results on the complexity of these computational tasks as a function of properties of the allowed constraint relations. Particular attention is paid to the special case of Boolean constraint relations.

Keywords: satisfiability problems, constraint satisfaction, computational complexity, polymorphism, clone, Galois connection.

1 Satisfiability Problems

The propositional satisfiability problem SAT, i.e., the problem to decide, given a propositional formula ϕ (without loss of generality in conjunctive normal form CNF), if there is an assignment to the variables in ϕ that satisfies ϕ, is the first and standard NP-complete problem [Coo71]. This means that it belongs to a class of decision problems for which we do not know if an efficient (that is: running in polynomial time) solution algorithm exists; in fact many researchers strongly believe there is none. However, there are well-known syntactic restrictions for which satisfiability is efficiently decidable, for example if every clause in the CNF formula has at most two literals (2CNF formulas) or if every clause has at most one positive literal (Horn formulas) or at most one negative literal (dual Horn formulas), see [KL99].

To study this phenomenon more generally, let us look at "clauses" of arbitrary shapes. A *logical relation* (or *constraint relation*) of arity k is a relation $R \subseteq \{0,1\}^k$. A *constraint* (or *constraint application*) is a formula $R(x_1,\ldots,k_k)$, where R is a logical relation of arity k and the $x_1,\ldots,x_k$ are (not necessarily distinct) variables. An assignment I of truth values to the variables *satisfies* the constraint if $\big(I(x_1),\ldots,I(x_k)\big) \in R$. A *constraint language* Γ is a finite set of logical relations. A *Γ-formula* is a conjunction of constraint applications using only logical relations from Γ. Such a formula ϕ is satisfied by an assignment I if I satisfies all constraints in ϕ simultaneously.

The for us central family of algorithmic problems, parameterized by a constraint language Γ, now is the following:

Problem:	CSP(Γ)
Input:	a Γ formula ϕ
Question:	Is ϕ satisfiable, i.e., is there an assignment that satisfies ϕ?

The NP-complete problem 3SAT, the satisfiability problem for CNF formulas with exactly three literals per clause, now is the problem $\mathrm{CSP}(\Gamma_{\mathrm{3SAT}})$, where $\Gamma_{\mathrm{3SAT}} = \{x \vee y \vee z, x \vee y \vee \neg z, x \vee \neg y \vee \neg z, \neg x \vee \neg y \vee \neg z\}$; here and in the sequel we do not distinguish between a formula ϕ and the logical relation R_ϕ it defines, i.e., the relation consisting of all satisfying assignments of ϕ. If every relation in Γ is definable by a Horn formula, then CSP(Γ) is polynomial-time decidable, also if every relation in Γ is definable by a 2-CNF formula. Hence we see that the family of problems CSP(Γ) has NP-complete members as well as easily solvable members.

A question attacked by Thomas Schaefer [Sch78] is the following: Can we determine for each constraint language Γ the complexity of CSP(Γ)? Is there even a simple algorithm that, given Γ, determines the complexity of CSP(Γ)? Are there more cases than NP-complete and polynomial-time solvable? Before we can answer these questions we have to recall some basic notions and results from universal algebra.

2 Universal Algebra

When we want to determine the complexity of all CSP-problems, we will certainly need a way to compare the complexity of CSP(Γ) and CSP(Γ') for different constraint languages Γ and Γ'. For example, to show that some CSP(Γ) is NP-complete we might show that using Γ we can "simulate" or "implement" all relations in Γ_{3SAT}, and to show that CSP(Γ) is polynomial-time decidable we might implement all relations in Γ using Horn-formulas. As it turns out, a useful notion of implementation comes from universal algebra, from clone theory.

For a constraint language Γ, we define $\langle\Gamma\rangle$ to be the *relational clone* (or *co-clone*) generated by Γ, i.e., $\langle\Gamma\rangle$ is the smallest set of relations such that

– $\langle\Gamma\rangle$ contains the equality relation and all relations in Γ, and
– $\langle\Gamma\rangle$ is closed under primitive positive definitions, i.e., if ϕ is a $\langle\Gamma\rangle$-formula and $R(x_1,\dots,x_n) \equiv \exists y_1 \dots y_\ell\ \phi(x_1,\dots,x_n,y_1,\dots,y_\ell)$, then $R \in \langle\Gamma\rangle$. (Such formulas are sometimes also called *conjunctive queries* over $\langle\Gamma\rangle$.)

Intuitively, $\langle\Gamma\rangle$ contains all relations that can be implemented by Γ and is thus called the *expressive power* of Γ, as justified by the following observation:

$$\text{If } \Gamma \subseteq \langle\Gamma'\rangle \text{ then } \mathrm{CSP}(\Gamma) \leq_m^{\log} \mathrm{CSP}(\Gamma'). \tag{1}$$

To see this, let F be a Γ-formula. We construct a formula F' by performing the following steps:

- Replace every constraint from Γ by its defining existentially quantified $\big(\Gamma' \cup \{=\}\big)$-formula.
- Delete existential quantifiers.
- Delete equality clauses and replace all variables that are connected via a chain of equality constraints by a common new variable.

Then, obviously, F' is a Γ'-formula, and moreover, F is satisfiable iff F' is satisfiable. The complexity of the above transformation is dominated by the last step, which is essentially an instance of the undirected graph reachability problem, which is solvable in logarithmic space [Rei05]. Hence we conclude that $\mathrm{CSP}(\Gamma)$ is reducible to $\mathrm{CSP}(\Gamma')$ under logspace reductions, q.e.d.

In particular, we thus have shown that

$$\text{if } \langle\Gamma\rangle = \langle\Gamma'\rangle, \text{ then } \mathrm{CSP}(\Gamma) \equiv_m^{\log} \mathrm{CSP}(\Gamma'), \tag{2}$$

i.e., the complexity of $\mathrm{CSP}(\Gamma)$ depends only on $\langle\Gamma\rangle$. Thus, we only have to study co-clones in order to obtain a full classification, and the question arises what co-clones there are.

Astonishingly, all co-clones, each with a “simple” basis, are known. The key to obtain this list is to study closure properties of relations. For this , let $f\colon \{0,1\}^m \to \{0,1\}$ and $R \subseteq \{0,1\}^n$. We say that f *preserves* R, $f \approx R$, if for all $x_1, \ldots, x_m \in R$, where $x_i = (x_i[1], x_i[2], \ldots, x_i[n])$, we have

$$\Big(f\big(x_1[1], \cdots, x_m[1]\big), f\big(x_1[2], \cdots, x_m[2]\big), \ldots, f\big(x_1[n], \cdots, x_m[n]\big)\Big) \in R.$$

In other words, if we apply f coordinatewise to a sequence of m vectors in R then the resulting vector must again be in R. Then we also say that R is *invariant* under f or that f is a *polymorphism* of R, and for a set of relations Γ we write $\mathrm{Pol}(\Gamma)$ to denote the set of all polymorphisms of Γ, i.e., the set of all Boolean functions that preserve every relation in Γ.

It is now straightforward to verify that for every Γ, $\mathrm{Pol}(\Gamma)$ is a *clone*, i.e., a set of Boolean functions that contains all projections (all functions $\mathrm{I}_k^n(x_1, \ldots, x_n) = x_k$ for $1 \leq k \leq n$) and is closed under composition; the smallest clone containing a set B of Boolean functions will be denoted by $[B]$ in the sequel. In fact, the connection between clones and relational clones is much tighter. For a set B of Boolean functions, let $\mathrm{Inv}(B)$ denote the set of all *invariants* of B, i.e., the set of all Boolean relations that are preserved by every function in B. It can be observed that each $\mathrm{Inv}(B)$ is a relational clone. In fact, as shown first in [Gei68, BKKR69] (see also [Lau06, Sect. 2.9]), the operators Pol-Inv constitute a Galois correspondence between the lattice of sets of Boolean relations and the lattice of sets of Boolean functions. In particular, for every set Γ of Boolean relations and every set B be of Boolean functions,

- $\mathrm{Inv}\big(\mathrm{Pol}(\Gamma)\big) = \langle\Gamma\rangle$,
- $\mathrm{Pol}\big(\mathrm{Inv}(B)\big) = [B]$.

Thus, there is a one-one correspondence between clones and co-clones and we may compile a full list of relational clones from the list of clones obtained by Emil Post in [Pos20, Pos41]. In these papers, Post presented a complete list of Boolean clones, the inclusion structure among them, and a finite basis for each of them. We do not have enough space here to describe *Post's lattice*, as the structure became known, in more detail, but we refer the interested reader to [Pip97] for a gentle introduction to clones, co-clones, the Galois connection, and Post's results. A rigorous comprehensive study is [Lau06]. Complexity-theoretic applications of Post's lattice in the constraint context but also the Boolean circuit context are surveyed in [BCRV03, BCRV04]. A compilation of all co-clones with simple bases is given in [BRSV05].

We will give a brief description of that part of the lattice that will be important here. The clone generated by the logical AND function is denoted by E_2. A relation is preserved by AND iff it is Horn, that is, definable by a Horn-formula, i.e., $\mathrm{Inv}(\mathrm{E}_2)$ is the set of all Horn relations. Similarly, $\mathrm{V}_2 = [\{\mathrm{OR}\}]$, and $\mathrm{Inv}(\mathrm{V}_2)$ is the set of all dual Horn relations. Relations definable by 2CNF formulas, the so called *bijunctive* relations, are exactly those in $\mathrm{Inv}(\mathrm{D}_2)$, where D_2 is the clone generated by the 3-ary majority function. Finally, the clone L_2 is generated by the 3-ary exclusive-or $x \oplus y \oplus z$ (the 3-ary addition in GF[2]), and $\mathrm{Inv}(\mathrm{L}_2)$ is the set of all *affine* formulas, i.e., conjunctions of XOR-clauses (consisting of an XOR of some variables plus maybe the constant 1)—these formulas may also be seen as systems of linear equations over GF[2].

Let us say that a constraint language is *Schaefer*, if it belongs to one of the above four types, i.e., Γ is Horn (i.e., every relation in Γ is Horn), dual Horn, bijunctive, or affine. If Γ is Schaefer then $\mathrm{CSP}(\Gamma)$ is polynomial-time solvable, as already noted above for the cases Horn, dual Horn, and bijunctive; for the remaining case of affine relations we remark that we use the interpretation as equations over GF[2] and thus may check satisfiability efficiently using the Gaussian algorithm. (A detailed exposition can be found in [CKS01].)

There is a unique minimal relational clone that is non-Schaefer: this is the co-clone $\mathrm{Inv}(\mathrm{N})$, where the clone N is generated by the negation function NOT plus the Boolean constans 0,1. This relational clone consists of all relations that are at the same time *complementive* (negating all entries of a tuple in the relations leads again to a tuple in the relation), *1-valid* (the all-1 tuple is in the relation), and *0-valid* (the all-0 tuple is in the relation). Because of these latter two properties, satisfiability for CSPs build using only relations from $\mathrm{Inv}(\mathrm{N})$ is again efficiently decidable (in fact, they are all satisfiable). If we drop the requirement 1-valid and 0-valid we arrive at the relational clone $\mathrm{Inv}(\mathrm{N}_2)$ consisting of all complementive relations ($\mathrm{N}_2 = [\{\mathrm{NOT}\}]$). Obviously, $\mathrm{Inv}(\mathrm{N}) \subseteq \mathrm{Inv}(\mathrm{N}_2)$, and from Post's lattice it can be seen that there is no relational clone in between. The only super-co-clone of $\mathrm{Inv}(\mathrm{N}_2)$ is the co-clone $\mathrm{Inv}(\mathrm{I}_2)$ of all relations ($\mathrm{I}_2 = [\emptyset]$).

3 Complexity Results

We have seen that if Γ is Schaefer or $\Gamma \subseteq \mathrm{Inv}(\mathrm{N})$ then $\mathrm{CSP}(\Gamma)$ is decidable in polynomial time. If Γ is not of this form, it follows from Post's lattice that

$\Gamma \supseteq \mathrm{Inv}(\mathrm{N}_2)$, the co-clone of all complementive relations. A particular example here is the relation

$$\mathrm{R}_{\mathrm{NAE}} = \big\{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0)\big\}.$$

The language $\mathrm{CSP}(\{\mathrm{R}_{\mathrm{NAE}}\})$ thus consists of 3CNF formulas with only positive literals where we require that in every clause not all literals obtain the same truth value. This is the so-called NOT-ALL-EQUAL-SAT problem, known to be NP-complete (see, e.g., [Pap94]). Thus, we have proved *Schaefer's Theorem*:

$$\begin{array}{l}\text{If } \langle \Gamma \rangle \supseteq \mathrm{Inv}(\mathrm{N}_2) \text{ then } \mathrm{CSP}(\Gamma) \text{ is NP-complete.}\\ \text{In all other cases, } \mathrm{CSP}(\Gamma) \text{ is polynomial-time decidable.}\end{array} \tag{3}$$

Because each member of the infinite family of the CSP-problems falls in two complexity cases and avoids the (under the assumption $\mathrm{P} \neq \mathrm{NP}$) infinitely many intermediate degrees, this theorem is also known as Schaefer's *Dichotomy Theorem.*

In [ABI$^+$05] the polynomial-time cases in the above classification have been studied more closely using first-order reductions, and it turned out that under this strict approach, seven cases for the complexity of the CSP-problem arise.

Recently there has been growing interest in quantified constraints, and we want to survey some of the developments here. The $\mathrm{CSP}(\Gamma)$ problem is equivalent to asking if a Γ-formula with all variables existentially quantified evaluates to true. In the quantified CSP problem one allows also universal quantifiers.

Let us first go one step back and look at usual propositional formulas again. The problem QBF of deciding, whether a given closed quantified propositional formula is true, is PSPACE-complete [SM73], even if the formula is restricted to 3CNF. If the number of quantifier alternations is bounded, the problem is complete in the *polynomial-time hierarchy*, which was defined by Meyer and Stockmeyer [MS72]. Following the notation of [Pap94], $\Sigma_0\mathrm{P} = \Pi_0\mathrm{P} = \mathrm{P}$ and for all $i \geq 0$, $\Sigma_{i+1}\mathrm{P} = \mathrm{NP}^{\Sigma_i\mathrm{P}}$ and $\Pi_{i+1}\mathrm{P} = \mathrm{coNP}^{\Sigma_i\mathrm{P}}$. The set QBF_k of all closed, true quantified Boolean formulas with $k-1$ quantifier alternations starting with an $\exists$-quantifier, is complete for $\Sigma_k\mathrm{P}$ for all $k \geq 1$ [SM73]. This problem remains $\Sigma_k\mathrm{P}$-complete if we restrict the Boolean formula to be 3CNF for k odd, and 3DNF for k even [Wra77]. Since disjunctive normal forms cannot be naturally modeled in a constraint satisfaction context, in order to generalize QBF_k to arbitrary set of constraints Γ in the same way we generalized SAT to $\mathrm{CSP}(\Gamma)$, we consider the unsatisfiability problem for these cases and we adopt the following definition for $\mathrm{QCSP}_k(\Gamma)$ from [Hem04].

Let Γ be a constraint language and $k \geq 1$. For k odd, a $\mathrm{QCSP}_k(\Gamma)$ formula is a closed formula of the form $\phi = \exists X_1 \forall X_2 \ldots \exists X_k \psi$, and for k even, a $\mathrm{QCSP}_k(\Gamma)$ formula is a closed formula of the form $\phi = \forall X_1 \exists X_2 \ldots \exists X_k \psi$, where the X_j, $j = 1, \ldots, k$, are disjoint sets of variables and ψ is a quantifier-free Γ-formula with variables from $\bigcup_j X_j$.

Problem: $\mathrm{QCSP}_k(\Gamma)$
Input: a $\mathrm{QCSP}_k(\Gamma)$-formula ϕ

Question: If k is odd: Is ϕ true?
If k is even: Is ϕ false?

As in the case of simple CSPs above, we note that the Galois connection still helps to study the complexity of QCSP:

$$\text{If } \Gamma \subseteq \langle \Gamma' \rangle \text{ then } \mathrm{QCSP}_k(\Gamma) \leq_m^{\log} \mathrm{QCSP}_k(\Gamma') \text{ for all } k \geq 1. \tag{4}$$

The proof of this is very similar to the one for (1): Given a Γ-formula F, we construct a formula F' by replacing every constraint from Γ by its defining existentially quantified $\big(\Gamma' \cup \{=\}\big)$-formula. The newly introduced quantified variables will be quantified in the final quantifier block which is by definition of $\mathrm{QCSP}_k(\Gamma)$-formulas always existential. All that remains to do now is to delete equality clauses as above.

Certainly, for every constraint language Γ and every $k \geq 1$, $\mathrm{QCSP}_k(\Gamma) \in \Sigma_k\mathrm{P}$ and $\mathrm{QCSP}_k(\Gamma_{3\mathrm{SAT}})$ is $\Sigma_k\mathrm{P}$-complete. In fact, even for the single constraint relation

$$\mathrm{R}_{\text{1-IN-3}} = \{(1,0,0),(0,1,0),(0,0,1)\}$$

we have that $\mathrm{QCSP}_k(\{\mathrm{R}_{\text{1-IN-3}}\})$ is $\Sigma_k\mathrm{P}$-complete. This follows since $\mathrm{R}_{\text{1-IN-3}}$ is only closed under projections and, thus, $\mathrm{Pol}(\mathrm{R}_{\text{1-IN-3}})$ is the minimal clone I_2 in Post's lattice and $\langle \mathrm{R}_{\text{1-IN-3}} \rangle$ is the co-clone $\mathrm{Inv}(\mathrm{I}_2)$ of all Boolean relations. Thus, from (4) we conclude $\mathrm{QCSP}_k(\Gamma_{3\mathrm{SAT}}) \leq_m^{\log} \mathrm{QCSP}_k(\{\mathrm{R}_{\text{1-IN-3}}\})$.

In the case of Schaefer's theorem for CSP, already the constraint language consisting of the relation $\mathrm{R}_{\mathrm{NAE}}$ is hard. We want to show an analogous result for QCSP next. To show this, we will reduce $\mathrm{QCSP}_k(\{\mathrm{R}_{\text{1-IN-3}}\})$ to $\mathrm{QCSP}_k(\{\mathrm{R}_{\mathrm{NAE}}\})$:

Let φ be a $\mathrm{QCSP}_k(\{\mathrm{R}_{\text{1-IN-3}}\})$-formula,

$$\varphi = QX_1 \ldots \exists X_k \bigwedge_{j=1}^{p} \mathrm{R}_{\text{1-IN-3}}(x_{j_1}, x_{j_2}, x_{j_3}),$$

where Q is existential if k is odd and universal if k is even. We now replace each constraint $\mathrm{R}_{\text{1-IN-3}}(x_{j_1}, x_{j_2}, x_{j_3})$ by the following conjunction:

$$\bigwedge_{j \neq k \in \{j_1, j_2, j_3\}} \mathrm{R}_{\mathrm{NAE}}(x_j, x_k, t) \wedge \mathrm{R}_{\mathrm{NAE}}(x_{j_1}, x_{j_2}, x_{j_3}).$$

It can be checked that this conjunction is true iff exactly 2 of the 4 variables $x_{j_1}, x_{j_2}, x_{j_3}, t$ are true, hence we will abbreviate it by $\mathrm{R}_{\text{2-IN-4}}(x_{j_1}, x_{j_2}, x_{j_3}, t)$. Now define $\varphi' = QtQX_1 \ldots \exists X_k \bigwedge_{j=1}^{p} \mathrm{R}_{\text{2-IN-4}}(x_{j_1}, x_{j_2}, x_{j_3}, t)$. Since $\mathrm{R}_{\text{1-IN-3}}(x,y,z) = \mathrm{R}_{\text{2-IN-4}}(x,y,z,1)$, the formula $\varphi'[t=1]$ (every occurrence of t in φ is replaced by 1) is true iff φ is true. Since $\mathrm{R}_{\text{1-IN-3}}(\bar{x},\bar{y},\bar{z}) = \mathrm{R}_{\text{2-IN-4}}(x,y,z,0)$, the formula $\varphi'[t/0]$ is true iff $\mathrm{Ren}(\varphi)$ is true, where $\mathrm{Ren}(\varphi)$ is obtained from φ by renaming all variables x by their negation $\bar{x}$. Finally, since $\mathrm{Ren}(\varphi)$ is true iff φ is true, we proved that φ is true if and only if φ' is true. Thus, $\mathrm{QCSP}_k(\{\mathrm{R}_{\text{1-IN-3}}\}) \leq_m^{\log} \mathrm{QCSP}_k(\{\mathrm{R}_{\mathrm{NAE}}\})$.

Hence we now know that if $\langle \Gamma \rangle \supseteq \mathrm{Inv}(\mathrm{N}_2)$, then $\mathrm{QCSP}_k(\Gamma)$ is complete for $\Sigma_k\mathrm{P}$ for every $k \geq 1$. What about the next lower relational clone $\mathrm{Inv}(\mathrm{N})$? In the case of $\mathrm{CSP}(\Gamma)$ (i.e., $\mathrm{QCSP}_1(\Gamma)$), satisfiability is trivial for all $\Gamma \subseteq \mathrm{Inv}(\mathrm{N})$, since every formula is satisfied by the constant-0 or constant-1 assignment. However, this tells us nothing about $\mathrm{QCSP}_p(\Gamma)$ for $k \geq 2$. Let us look at the relation

$$R_0 = \bigl\{ (u, v, x_1, x_2, x_3) \bigm| u = v \text{ or } \mathrm{R}_{\mathrm{NAE}}(x_1, x_2, x_3) \bigr\}.$$

It is easy to see that R_0 is complementive, 0-valid, and 1-valid. We will show that $\mathrm{QCSP}_k(\{\mathrm{R}_{\mathrm{NAE}}\})$ reduces to $\mathrm{QCSP}_k(\{R_0\})$. Let

$$\varphi = QX_1 \ldots \exists X_k \bigwedge_{j=1}^{p} \mathrm{R}_{\mathrm{NAE}}(x_{j_1}, x_{j_2}, x_{j_3}),$$

where Q is existential if k is odd and universal if k is even, be an instance of $\mathrm{QCSP}_k(\{\mathrm{R}_{\mathrm{NAE}}\})$. We define

$$\varphi' = QX_1 \ldots \forall X_{i-1} \forall u \forall v \exists X_k \bigwedge_{j=1}^{p} R_0(u, v, x_{j_1}, x_{j_2}, x_{j_3}).$$

Clearly φ is true iff φ' is true, thus $\mathrm{QCSP}_k(\{\mathrm{R}_{\mathrm{NAE}}\}) \leq_m^{\log} \mathrm{QCSP}_k(\{R_0\})$ for all $k \geq 2$.

We conclude that if $\langle \Gamma \rangle \supseteq \mathrm{Inv}(\mathrm{N})$, then $\mathrm{QCSP}_k(\Gamma)$ is complete for $\Sigma_k\mathrm{P}$ for every $k \geq 2$. If we drop the bound on the number of quantifier alternations and denote the resulting problem by $\mathrm{QCSP}(\Gamma)$, we know from [SM73] that $\mathrm{QCSP}(\Gamma_{3\mathrm{SAT}})$ is PSPACE-complete. The just given reductions thus also show that if $\langle \Gamma \rangle \supseteq \mathrm{Inv}(\mathrm{N})$, then $\mathrm{QCSP}(\Gamma)$ is PSPACE-complete.

If Γ does not include $\mathrm{Inv}(\mathrm{N})$, we know from the structure of Post's lattice that it must be Schaefer. However, it is known that in all four cases (Horn, dual Horn, 2CNF, and affine), the evaluation of quantified formulas is computable in polynomial time (the algorithms for the first three cases rely on Q-resolution, a variant of resolution for quantified propositional formulas, see [KL99]; the algorithm for the affine case is a refinement of the Gaussian algorithm [CKS01]). Thus we have proved the following classification:

If Γ is Schaefer then $\mathrm{QCSP}(\Gamma)$ is polynomial-time decidable.
In all other cases, $\mathrm{QCSP}(\Gamma)$ is PSPACE-complete. (5)

This result was stated without proof and only for constraint languages that include the constants in Schaefer's paper [Sch78]. In its full form it was stated and proven for the first time in [Dal97] and later published in [CKS01].

Looking at QCSPs with bounded quantifier alternations we obtain *Hemaspaandra's Theorem* [Hem04]. For all $k \geq 2$ (the case $k = 1$ is given by Schaefer's Theorem) the following holds:

If Γ is Schaefer then $\mathrm{QCSP}_k(\Gamma)$ is polynomial-time decidable.
In all other cases, $\mathrm{QCSP}(\Gamma)$ is $\Sigma_k\mathrm{P}$-complete. (6)

Both (3), (5), and (6) were originally proven in a different much more involved way. The above simple proofs using Galois theory appeared later. The proof of (3) is implicit in [JCG97, Dal00]. The proofs of (5) and (6) are from [BBC+05]. A different proof is given in [Che06].

Many further results, classifying the computational complexity of different algorithmic tasks such as counting the number of satisfying assignments [CH96, BCC$^+$05, BBC$^+$05], enumeration of all satisfying assignments [CH97], equivalence and isomorphism of CSPs [BHRV02, BHRV04], and many more (cf. also [CKS01, BCRV04]) have been obtained in the past decades. Some of these rely on the algebraic approach explained in this paper, for others this approach does not seem to be useful.

4 Non-boolean Domains

A very active research area, which—alas!—we can only address very briefly in the end, is the study of the questions from the above for larger (non-Boolean) domains. Feder and Vardi [FV98] have conjectured that over arbitrary finite domains, the satisfiability for CSPs is, analogously to Schaefer's dichotomy, either NP-complete or polynomial-time solvable. This is the famous *Dichotomy conjecture*, which is still open. What is known is that the conjecture holds for the case of a 3-element universe [Bul06].

Note that the Galois connection described above holds in the non-Boolean context as well, however, already for three elements the lattice of clones is uncountable (see [Pip97]).

Several approaches to identify tractable cases of the satisfiability problem have been intensively developed, one is a logical approach where definability of CSPs in Datalog leads to efficient algorithms (see, e.g., [KV07]), one is the algebraic approach where structural properties of the clones of polymorphisms are exploited (see, e.g., [JCG97, BJK05]). Further computational goals besides satisfiability have also been studied, we only mention evaluating quantified CSPs [BBJK03, BBC$^+$05, Che06], counting the number of solutions [BD03, BBC$^+$05], and enumerating all solutions [SS07].

In the context of non-Boolean domains, a version of the CSP where *both* the formula and the constraint language Γ are part of the problem instance has been studied. This is the so called *uniform CSP*, in contrast to the non-uniform CSP from the above where the constraint language is always fixed (see, e.g., [KV07]).

More on the topics we could only mention here can be found in the different articles in [CKV07] or the recent surveys [Che06, KV07].

References

[ABI$^+$05] Allender, E., Bauland, M., Immerman, N., Schnoor, H., Vollmer, H.: The complexity of satisfiability problems: Refining Schaefer's theorem. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 71–82. Springer, Heidelberg (2005)

[BBC$^+$05] Bauland, M., Böhler, E., Creignou, N., Reith, S., Schnoor, H., Vollmer, H.: Quantified constraints: The complexity of decision and counting for bounded alternation. Technical Report 05-025, Electronic Colloqium on Computational Complexity (Submitted for publication 2005)

[BBJK03] Börner, F., Bulatov, A., Jeavons, P., Krokhin, A.: Quantified constraints: algorithms and complexity. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, Springer, Berlin Heidelberg (2003)

[BCC$^+$05] Bauland, M., Chapdelaine, P., Creignou, N., Hermann, M., Vollmer, H.: An algebraic approach to the complexity of generalized conjunctive queries. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 30–45. Springer, Heidelberg (2005)

[BCRV03] Böhler, E., Creignou, N., Reith, S., Vollmer, H.: Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. ACM-SIGACT Newsletter 34(4), 38–52 (2003)

[BCRV04] Böhler, E., Creignou, N., Reith, S., Vollmer, H.: Playing with Boolean blocks, part II: Constraint satisfaction problems. ACM-SIGACT Newsletter 35(1), 22–35 (2004)

[BD03] Bulatov, A., Dalmau, V.: Towards a dichotomy theorem for the counting constraint satisfaction problem. In: Proceedings Foundations of Computer Science, pp. 562–572. ACM Press, New York (2003)

[BHRV02] Böhler, E., Hemaspaandra, E., Reith, S., Vollmer, H.: Equivalence and isomorphism for Boolean constraint satisfaction. In: Bradfield, J.C. (ed.) CSL 2002 and EACSL 2002. LNCS, vol. 2471, pp. 412–426. Springer, Berlin Heidelberg (2002)

[BHRV04] Böhler, E., Hemaspaandra, E., Reith, S., Vollmer, H.: The complexity of Boolean constraint isomorphism. In: Dunin-Keplicz, B., Nawarecki, E. (eds.) CEEMAS 2001. LNCS (LNAI), vol. 2296, pp. 164–175. Springer, Berlin Heidelberg (2002)

[BJK05] Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the complexity of constraints using finite algebras. SIAM Journal on Computing 34(3), 720–742 (2005)

[BKKR69] Bodnarchuk, V.G., Kalužnin, L.A., Kotov, V.N., Romov, B.A.: Galois theory for Post algebras. I, II. Cybernetics, 5 pp. 243–252, pp. 531–539 (1969)

[BRSV05] Böhler, E., Reith, S., Schnoor, H., Vollmer, H.: Bases for Boolean co-clones. Information Processing Letters 96, 59–66 (2005)

[Bul06] Bulatov, A.A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. Journal of the ACM 53(1), 66–120 (2006)

[CH96] Creignou, N., Hermann, M.: Complexity of generalized satisfiability counting problems. Information and Computation 125, 1–12 (1996)

[CH97] Creignou, N., Hébrard, J.-J.: On generating all solutions of generalized satisfiability problems. Informatique Théorique et Applications/Theoretical Informatics and Applications 31(6), 499–511 (1997)

[Che06] Chen, H.: A rendezvous of logic, complexity, and algebra. ACM-SIGACT Newsletter 37(4), 85–114 (2006)

[CKS01] Creignou, N., Khanna, S., Sudan, M.: Complexity Classifications of Boolean Constraint Satisfaction Problems. Monographs on Discrete Applied Mathematics. SIAM (2001)

[CKV07] Creignou, N., Kolaitis, Ph., Vollmer, H. (eds.): Complexity of Constraints. Springer, Berlin Heidelberg (2007)

[Coo71] Cook, S.A.: The complexity of theorem proving procedures. In: Proceedings 3rd Symposium on Theory of Computing, pp. 151–158. ACM Press, New York (1971)

[Dal97] Dalmau, V.: Some dichotomy theorems on constant-free quantified boolean formulas. Technical Report LSI-97-43-R, Department de Llenguatges i Sistemes Informàtica, Universitat Politécnica de Catalunya (1997)

[Dal00] Dalmau, V.: Computational complexity of problems over generalized formulas. PhD thesis, Department de Llenguatges i Sistemes Informàtica, Universitat Politécnica de Catalunya (2000)

[FV98] Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. SIAM Journal on Computing 28(1), 57–104 (1998)

[Gei68] Geiger, D.: Closed systems of functions and predicates. Pac. J. Math 27(2), 228–250 (1968)

[Hem04] Hemaspaandra, E.: Dichotomy theorems for alternation-bounded quantified boolean formulas. CoRR, cs.CC/0406006 (2004)

[JCG97] Jeavons, P.G., Cohen, D.A., Gyssens, M.: Closure properties of constraints. Journal of the ACM 44(4), 527–548 (1997)

[KL99] Kleine Büning, H., Lettmann, T.: Propositional Logic: Deduction and Algorithms. In: Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge (1999)

[KV07] Kolaitis, P., Vardi, M.: A logical approach to constraint satisfaction. In: Finite Model Theory and its Applications, Texts in Theoretical Computer Science, Springer, Berlin Heidelberg (2007)

[Lau06] Lau, D.: Function Algebras on Finite Sets. In: Monographs in Mathematics, Springer, Berlin Heidelberg (2006)

[MS72] Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential time. In: Proceedings 13th Symposium on Switching and Automata Theory, pp. 125–129. IEEE Computer Society Press, Washington (1972)

[Pap94] Papadimitriou, C.H.: Computational Complexity, Reading. Addison-Wesley, MA (1994)

[Pip97] Pippenger, N.: Theories of Computability. Cambridge University Press, Cambridge (1997)

[Pos20] Post, E.L.: Determination of all closed systems of truth tables. Bulletin of the AMS 26, 437 (1920)

[Pos41] Post, E.L.: The two-valued iterative systems of mathematical logic. Annals of Mathematical Studies 5, 1–122 (1941)

[Rei05] Reingold, O.: Undirected st-connectivity in log-space. In: Proceedings of the 37th Symposium on Theory of Computing, pp. 376–385. ACM Press, New York (2005)

[Sch78] Schaefer, T.J.: The complexity of satisfiability problems. In: Proccedings 10th Symposium on Theory of Computing, pp. 216–226. ACM Press, New York (1978)

[SM73] Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proceedings 5th ACM Symposium on the Theory of Computing, pp. 1–9. ACM Press, New York (1973)

[SS07] Schnoor, H., Schnoor, I.: Enumerating all solutions for constraint satisfaction problems. In: 24nd Symposium on Theoretical Aspects of Computer Science, pp. 694–705 (2007)

[Wra77] Wrathall, C.: Complete sets and the polynomial-time hierarchy. Theoretical Computer Science 3, 23–33 (1977)

Finding Most Likely Solutions

Osamu Watanabe and Mikael Onsjö

[1] Dept. of Math. and Comput. Sci., Tokyo Inst. of Technology, Japan
[2] Dept. of Computer Sci. and Eng., Chalmers Univ. of Technology, Sweden
watanabe@is.titech.ac.jp

Abstract. As one simple type of statistical inference problems we consider *Most Likely Solution problem*, a task of finding a most likely solution (MLS in short) for a given problem instance under some given probability model. Although many MLS problems are NP-hard, we propose, for these problems, to study their average-case complexity under their assumed probability models. We show three examples of MLS problems, and explain that "message passing algorithms" (e.g., belief propagation) work reasonably well for these problems. Some of the technical results of this paper are from the author's recent joint work with his colleagues [WST, WY06, OW06].

1 Introduction

We discuss here the following general statistical inference problem. Consider any function $g(s, r)$ on a pair of binary strings that can be efficiently computable, say, polynomial-time computable w.r.t. $|s|$ by some deterministic algorithm. Let $n = |s|$; on the other hand, we assume that both $m = |r|$ and $|g(s, r)|$ are determined from n and that they are polynomially bounded by n. Consider also some parameterized distribution $\mathcal{D}(m)$ on $\{0, 1\}^m$, and assume that some randomized algorithm generates $r \in \{0, 1\}^m$ with probability following $\mathcal{D}(m)$ efficiently, e.g., in polynomial-time w.r.t. m. By "$r : \mathcal{D}(m)$" we mean that r is generated by this algorithm. Then our statistical problem is defined as follows.

Most Likely Solution Problem
Instance: A string x such that $x = g(s, r)$ for some s and r.
Task: Find s that maximizes probability $\Pr_{r:\mathcal{D}(m)}[\, g(s, r) = x\,]$.

For a given instance x, we regard a string s such that $x = g(s, r)$ holds for some r as a *solution*; then the above probability is the *likelihood* of s being a solution of x. That is, this problem is to find a *most likely solution* (MLS, in short) for a given problem instance x. The function g generating problem instances from solutions is called a (*instance*) *generating function*. Problems like this can be found in various contexts including computational learning theory. Intuitively speaking, an instance x is some observed event or outcome, and s can be regarded as its reasoning; we would like to find out one of the most likely reasonings for the observed event.

As we will see below, actual MLS problems are in many cases NP-hard. But for each MLS problem, since we assume some probability model for input instances, we propose here to study its average-case complexity under this probability model. More specifically, we may consider the following three average-case scenarios for one MLS problem X.

I. Define $\mathcal{I}(s)$ to be the distribution of instances generated as $g(s,r)$ by using r generated randomly following $\mathcal{D}(m)$; study the average-case complexity of X under $\mathcal{I}(s)$, for some typical or all solutions s.
II. Assume also some distribution $\mathcal{S}(n)$ for solutions in $\{0,1\}^n$, and study the average-case complexity of X under $\mathcal{I}(s : \mathcal{S}(n))$.
III. Assume further some distribution $\mathcal{F}(n)$ for the instance generating function f, and study the average-case complexity of X under $\mathcal{I}(f : \mathcal{F}(n), s : \mathcal{S}(n))$.

In these scenarios, a solution s used for creating instances for the problem X is called a *planted solution*. Note that there is a case where a planted solution is not the most likely solution; for example, most of instances x generated by the distribution $\mathcal{I}(s)$ may have some other s_x as their most likely solution. On the other hand, by choosing some appropriate parameters for defining $\mathcal{D}(m)$, we may usually define $\mathcal{I}(s)$ so that, with high probability, the planted solution s is the *unique* most likely solution. Here we consider distributions satisfying this property.

For such average-case scenarios, we can sometimes find algorithms that perform well *on average*; furthermore, some of such performance can be formally proved. We will show three such examples from:- Linear Code Decoding problem, MAX-2SAT problem, and Graph Bisection problem. Interestingly, message passing algorithms work well for these problems, including those derived from Pearl's belief propagation [Pea88] (which we call simply *BP-algorithms*). Although it is unbelievable that message passing algorithms always perform well, we may expect that there is some class of MLS problems that can be solved well on average by message passing algorithms. Characterizing such a set of problems is an interesting and important open question.

Related Work

Three examples shown below are taken from the investigations of average-case performance of message passing algorithms. For studying such average-case performance, probability models called *planted solution models* have been often used (see, e.g., [JS98]), which are essentially the same as the above average-case scenario I. Here we simply look at these algorithms from a different view point, i.e., from the view point of statistical inference.

Note also that our MLS problem includes the problem of inverting a one-way function. More specifically, for a given randomized *one-to-one* one-way function f, inverting it is regarded as a special type of MLS problem that computes the *unique* s from $f(s,r)$, where s is a message and r is a random seed; for discussing its basic average-case complexity, we may assume that s and r are taken uniformly at random, which is our average-case scenario II. Thus, if we believe in the existence of one-way functions, we should also expect that some MLS problem is hard even on average.

2 Example 1: Linear Code Decoding Problem

We recall basics on linear codes. A linear code is determined by a $m \times n$ 0,1-matrix H, which we call a *parity check matrix*. For a parity check matrix H, a vector $\boldsymbol{a} = (a_1, ..., a_n)$ satisfying $H\boldsymbol{a} = \mathbf{0}$ is called a *code word* (w.r.t. H), which is supposed to be sent for communication. We consider a *binary symmetric channel* for our noise model; when a code word is transmitted, some of its bits are flipped independently at random with *noise probability* p. That is, when sending a code word $\boldsymbol{a}$, a message $\boldsymbol{b}$ received through the channel is computed as $\boldsymbol{b} = \boldsymbol{a} + \boldsymbol{n}$, where $\boldsymbol{n}$ is a 0,1-vector whose each element takes 1 with probability p (which we may assume small, e.g., $p < 0.2$). We call $\boldsymbol{n}$ a *noise vector*. (Here by $+$ we mean the bit wise addition under modulo 2.) Now our task is to obtain the transmitted code word $\boldsymbol{a}$ from $\boldsymbol{b}$. Obviously, all code words can be a candidate for the solution, but what we want is the one that is most likely for a given $\boldsymbol{c}$.

Let us state this requirement formally. First we define the following generating function:

$$g_H(\boldsymbol{x}, \boldsymbol{v}) \;=\; \begin{cases} \boldsymbol{x} + \boldsymbol{v}, \text{ if } H\boldsymbol{x} = \mathbf{0}, \text{ and} \\ \bot, \qquad \text{otherwise.} \end{cases}$$

Let $\mathcal{B}(n,p)$ denote a distribution on 0,1-vectors of size n defined by the standard binomial distribution; that is, a random vector following $\mathcal{B}(n,p)$ is generated by independently choosing, for each element, 1 with probability p and 0 with probability $1-p$. Now our task is formally stated as follows.

Most Likely Solution Problem (Linear Code Decoding)
Instance: A received message $\boldsymbol{b}$.
Task: Find a code word $\boldsymbol{a}$ that maximizes the following probability:

$$\Pr_{\boldsymbol{v}:\mathcal{B}(n,p)}[\, g_H(\boldsymbol{a}, \boldsymbol{v}) = \boldsymbol{b}\,].$$

Note that for any noise vector $\boldsymbol{n}$ having 1 at k coordinates, we have

$$\Pr_{\boldsymbol{v}:\mathcal{B}(n,p)}[\, \boldsymbol{n} = \boldsymbol{v}\,] \;=\; p^k(1-p)^k,$$

which takes larger value for smaller k (provided that $p < 1/2$). Thus, our task is to obtain a code word $\boldsymbol{a}$ such that $\boldsymbol{a} + \boldsymbol{n} = \boldsymbol{b}$ with a noise vector with the smallest number of 1's. Note further that

$$\boldsymbol{c} \;=\; H\boldsymbol{b} \;=\; H(\boldsymbol{a} + \boldsymbol{n}) \;=\; H\boldsymbol{a} + H\boldsymbol{n} \;=\; H\boldsymbol{n}.$$

Therefore, our task is essentially to obtain $\boldsymbol{n}$ satisfying the above with the smallest number of 1's. Although it is polynomial-time to compute *some* $\boldsymbol{n}$ from $\boldsymbol{c}$ satisfying the above, computing the one with the smallest number of 1's is known to be NP-hard; see, e.g., [GJ79]. Thus, our MLS problem (or, Linear Code Decoding problem) is NP-hard.

Gallager [Gal62] proposed to use very sparse parity check matrices, i.e., parity check matrices with quite a small number of 1 entries. Linear codes using very sparse parity check matrices are called *Low Density Parity Check Codes* (in short, LDPC). For simplicity, we consider parity check matrices with c 1's in each row d 1's in each column; let $\mathcal{H}(c,d,n)$ denote the set of these matrices. (We will use $\mathcal{H}(c,d,n)$ also for denoting a distribution of choosing a matrix uniformly at random from the set $\mathcal{H}(c,d,n)$.) Gallager proposed[1] to use a random parity check matrix from, e.g., $\mathcal{H}(3,5,n)$. In order to solve the decoding problem for such parity check matrices, he proposed a message passing type algorithm, which is essentially the same as the one derived from the belief propagation, i.e., the BP algorithm [Mac99]. It has been shown (see, e.g., [Mac99]) that this algorithm works well for randomly chosen matrices and random noise up to some noise probability level.

For a more specific discussion, we consider here the following average-case scenario: Use $\mathcal{B}(n,p)$ for the noise vector distribution defined by noise probability p and $\mathcal{H}(3,5,n)$ for the matrix distribution, and assume that, for any fixed code word $\boldsymbol{a} \in \{0,1\}^n$ for $H : \mathcal{H}(3,5,n)$, a received message $\boldsymbol{b}$ is generated $\boldsymbol{b} = \boldsymbol{a} + \boldsymbol{n}$ by $\boldsymbol{n} : \mathcal{B}(n,p)$. This is the distribution of instances for the (3,5)-LDPC Decoding problem. Note that this is a variant of type III scenario, where no distribution is assumed (i.e., the worst-case is considered) for solutions, i.e., the choice of code words $\boldsymbol{a}$. For the decoding problem, it is natural to expect that the original code word $\boldsymbol{a}$ for generating an instance $\boldsymbol{b}$ is the *unique* most likely solution for $\boldsymbol{b}$. We can easily check that this property indeed holds with high probability provided the noise probability p is smaller than a certain threshold. Some detailed experiment shows that, if $p \le p_*$ for some threshold p_*, Gallager's algorithm solves the problem with high probability for sufficiently large n.

Although the theoretical justification of Gallager's algorithm or the BP-algorithm is still open, Luby et al. [LMSS01] gave a theoretical justification to a message passing algorithm that is a similar (but much simpler) variant of Gallager's BP algorithm, which is called *Gallager's hard-decision algorithm.* Roughly speaking, they show that Gallager's heuristic formula for analyzing the performance of the hard-decision algorithm is sufficiently accurate within any constant parallel message exchanging rounds; on the other hand, Gallager's formula shows that the algorithm can get an almost correct code word within a certain constant parallel message exchanging rounds. Thus, they could show that with high probability (under our distribution) Gallager's hard-decision algorithm yields an output $\boldsymbol{a}'$ from a given $\boldsymbol{b}$ that is close to the original code word $\boldsymbol{a}$. (See [LMSS01] for the details.)

3 Example 2: MAX-2SAT Problem

Next consider MAX-2SAT problem, one of the well studied NP-hard optimization problems. Here we use n and m to denote respectively the number of

[1] Although researchers have shown (e.g., [LMSS01]) that irregular random matrices have a better error correcting performance, we consider a simple case for our analysis.

variables and clauses of a given input Boolean formula. We use $x_1, \ldots, x_n$ for denoting Boolean variables. A *CNF formula* is a conjunction of clauses, a *clause* is a disjunction of literals, and a *literal* is either a Boolean variable or its negation. In particular, a *2CNF formula* is a formula defined as a conjunction of clauses with two literals, where each clause is specified as $(x_i \vee x_j)$, $(x_i \vee \overline{x}_j)$, $(\overline{x}_i \vee x_j)$, or $(\overline{x}_i \vee \overline{x}_j)$, for some $1 \le i \le j \le n$. For simplicity, we assume that clauses are syntactically one of the above four types; e.g., there is no clause like $(x_j \vee \overline{x}_i)$ for some $i < j$. Note that it is possible that a formula has a clause like $(x_i \vee x_i)$, $(x_i \vee \overline{x}_i)$, or $(\overline{x}_i \vee \overline{x}_i)$. (Since $(\overline{x}_i \vee x_i)$ is semantically the same as $(x_i \vee \overline{x}_i)$, we do not allow clauses of this type. Thus, there are altogether $\binom{n}{2} \times 4 + 3n = 2n^2 + n$ clauses.) We use ℓ_i to denote either x_i or $\overline{x}_i$.

An *assignment* is a function t mapping $\{x_1, \ldots, x_n\}$ to $\{-1, +1\}$; $t(x_i) = +1$ (resp., $t(x_i) = -1$) means to assign true (resp., false) to a Boolean variable x_i. An assignment is also regarded as a sequence $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ of ± 1's, where $a_i = t(x_i)$ for each i, $1 \le i \le n$. For a given CNF formula F, its *optimal assignment* is an assignment satisfying the largest number of clauses in F. Now the MAX-2SAT problem is defined as follows.

MAX-2SAT problem

Instance: A 2CNF formula $F = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$.

Task: Find an optimal assignment of F.

We give a probability model and explain that this problem can be regarded as a MLS problem for this probability model. Again consider a model generating MAX-2SAT instances from some (fixed) solution. We first fix one assignment $\boldsymbol{a} = (a_1, \ldots, a_n)$, or generate it $\boldsymbol{a} = (a_1, \ldots, a_n)$ uniformly at random. Let $\boldsymbol{a}'$ be its *complement assignment* $(-a_1, \ldots, -a_n)$, i.e., an assignment obtained by flipping the sign of all individual assignments. This pair of $\boldsymbol{a}$ and $\boldsymbol{a}'$ is called a *planted solution pair.* A formula is constructed by adding all possible clauses independently by the following rule using parameters p and r: add each clause satisfied by both assignments with probability p (type A), and add each clause not satisfied by one of the assignments with probability r (type B). (Note that there is no clause that is satisfied by neither of a planted solution pair.) Note that there are n^2 type A clauses; hence, the number of clauses of type A added to the formula is *on average* pn^2. On the other hand, the formula has *on average* $rn(n+1)$ type B clauses. Clearly adding type B clauses is to make an obtained formula unsatisfiable, for making a formula nontrivial. (Recall that 2SAT problem is easy to solve.) It is easy to see that both members of the planted solution pair can satisfy all the type A clauses and the half of the type B clauses; that is, both fail to satisfy $rn(n+1)/2$ type B clauses on average.

This generating process can be expressed by the following polynomial-time computable generating function g_{sat}: given $\boldsymbol{a} \in \{0,1\}^n$, $\boldsymbol{r}_1 \in \{0,1\}^{n^2}$, and $\boldsymbol{r}_2 \in \{0,1\}^{n(n+1)}$, $g_{\text{sat}}(\boldsymbol{a}, \boldsymbol{r}_1, \boldsymbol{r}_2)$ is a formula obtained above where each bit of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is used to determine whether the corresponding clause is added to the formula. Thus, for the above distribution, we generate $\boldsymbol{r}_1 \in \{0,1\}^{n^2}$ following $\mathcal{B}(n^2, p)$

```
procedure algo_MAX-2SAT ($F$);
// An input $F = C_1 \wedge \cdots C_m$ is a 2CNF formula over variables $x_1, \ldots, x_n$.
// Let $S = S_+ \cup S_-$, where $S_+ = \{+1, \ldots, +n\}$ and $S_- = \{-n, \ldots, -1\}$.
begin
    construct $G = (V, E)$;    // See the text for the explanation.
    set $b(v_s)$ to 0 for all $s \in S$;
    $b(v_{+1}) \leftarrow +1$;  $b(v_{-1}) \leftarrow -1$;    // This is for the assumption that $x_1 = +1$.
    repeat MAXSTEP times do {
        for each $i \in \{2, \ldots, n\}$ in parallel do {
            $b(v_{+i}) \leftarrow \sum_{v_s \in N(v_{+i})} \min(0, b(v_s))$;   $b(v_{-i}) \leftarrow \sum_{v_s \in N(v_{-i})} \min(0, b(v_s))$;
             $b(v_{+i}) \leftarrow b(v_{+i}) - b(v_{-i})$;   $b(v_{-i}) \leftarrow -b(v_{+i})$;                 — (1)
        }
        if sg($b(v_i)$) is stabilized for all $i \in \{2, \ldots, n\}$ then break;
        $b(v_{+1}) \leftarrow 0$;  $b(v_{-1}) \leftarrow 0$;                                  — (2)
    }
    output( $+1, \mathrm{sg}(b(v_{+2})), \ldots, \mathrm{sg}(b(v_{+n}))$ );
end-procedure
```

Fig. 1. A message passing algorithm for the MAX-2SAT problem

and $\boldsymbol{r}_2 \in \{0,1\}^{n(n+1)}$ following $\mathcal{B}(n(n+1), r)$. Thus for the MLS problem, the probability that we need to minimize, for a given formula F, is

$$\Pr_{\boldsymbol{u}_1, \boldsymbol{u}_2}[\ F = g_{\mathrm{sat}}(\boldsymbol{a}, \boldsymbol{u}_1, \boldsymbol{u}_2)\ ] \ = \ p^{m_1} r^{m_2} (1-p)^{n^2 - m_1} (1-r)^{n(n+1) - m_2},$$

where m_1 and m_2 are the number of clauses of F of type A and type B w.r.t. a solution candidate $\boldsymbol{a}$, and the probability is on random variables $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ following distributions $\mathcal{B}(n^2, p)$ and $\mathcal{B}(n(n+1), r)$ respectively. Intuitively, we may consider that clauses are randomly generated from some reasoning $\boldsymbol{a}$; clauses of type A are constraints derived from $\boldsymbol{a}$ while clauses of type B are somewhat irregularly derived from $\boldsymbol{a}$. Then our goal is to find a most likely reasoning from observed constraints.

Since $m = m_1 + m_2$ is determined by F and we assume that $p > r$, this probability gets largest by an assignment $\boldsymbol{a}$ maximizing m_1. On the other hand, since any assignment satisfies all its type A clauses and it or its complement satisfies at least half of its type B clauses. Thus, if $\boldsymbol{a}$ is the most likely solution for F, either $\boldsymbol{a}$ or its complement $\boldsymbol{a}'$ satisfies at least $m_1 + m_2/2$ clauses. While this may not be the optimal from MAX-2SAT view point, we can easily prove that it indeed is the optimal with high probability if p and r are in an appropriate range.

Here again we expect that one of the planted solution pair is the optimal with high probability. In fact, by a relatively standard argument (though it is not so trivial as the first example), we can prove it and that no other assignment is as well as this optimal assignment with high probability, provided $p = \Omega(\ln n/n)$ and $p > 3r$ [WY06].

Now we consider a simple message passing algorithm stated in Figure 1. We first explain the outline of the algorithm. Below we use i and j to denote unsigned (i.e., positive) indices in $\{1, \ldots, n\}$, whereas s and t are used for signed indices in $S = \{-n, -(n-1), \ldots, -1, +1, \ldots, +(n-1), +n\}$. The algorithm is executed on a directed graph $G = (V, E)$ that is constructed from a given formula F as follows: A set V is a set of $2n$ vertices v_s, $s \in S$, each of which corresponds to literal x_s if s is positive and literal $\overline{x}_{|s|}$ if s is negative. A set E consists of two directed edges v corresponding to each clause $(\ell_i \vee \ell_j)$ of F, where $i < j$; e.g., an edge (v_{-i}, v_j) corresponds to clause $(x_i \vee x_j)$ $(= (\overline{x}_i \to x_j))$. On the other hand, only one edge, a self-loop, is added to E for each clause of type $(\overline{x}_i \vee x_i)$. Note that graph G has no multiple edge, while it may have some self-loops. Let $N(u)$ denote the set of vertices v having a directed edge to u.

The algorithm computes a "belief" $b(v_s)$ at each vertex v_s, an integral value indicating whether the Boolean variable $x_{|s|}$ should be assigned true (i.e., $+1$) or false (i.e., -1). More specifically, for an optimal assignment, the algorithm suggests, for each x_i, to assign $x_i = +1$ if the final value of $b(v_{+i})$ is positive and $x_i = -1$ if it is negative. Note that $b(v_{-i}) = -b(v_{+i})$; we may regard $b(v_{-i})$ as a belief for $\overline{x}_i$. These belief values are initially set to 0 except for one pair of vertices, e.g., v_{+1} and v_{-1} that are assigned $+1$ or -1 initially. In the algorithm of Figure 1, $b(v_{+1})$ (resp., $b(v_{-1})$) is set to $+1$ (resp., -1), which considers the case that x_1 is true in the optimal assignment. Clearly we need to consider the other case; that is, the algorithm is executed again with the initial assignment $b(v_{+1}) = -1$ and $b(v_{-1}) = +1$, and among two obtained assignments the one that satisfies more clauses is used as an answer. Now consider the execution of the algorithm. The algorithm updates beliefs based on messages from the other vertices. At each iteration, the belief of each vertex v_{+i} (resp., v_{-i}) is recomputed based on the last belief values of its neighbor vertices. More specifically, if there is an edge from v_{+i} to v_s, and $b(v_s)$ is negative, then this negative belief is sent to v_{+i} (from v_s) and used for computing the next belief of v_{+i}. The edge $v_{+1} \to v_s$ corresponds to a clause $(x_i \to \ell_{|s|})$ (where $\ell_{|s|}$ is the literal corresponding to v_s), and the condition that $b(v_s) < 0$ means that the literal $\ell_{|s|}$ is assigned false (under the current belief). Thus, in order to satisfy the clause $(x_i \to \ell_{|s|})$, we need to assign false to x_i. This is the reason for the message from v_s. Belief $b(v_{+i})$ at this iteration is defined as the sum of these messages. It should be remarked here that all belief values are updated *in parallel*; that is, updated beliefs are not used when updating the other beliefs in the same iteration, but those computed at the previous iteration are used. This update is repeated until no belief value has changed its sign after one updating iteration *or* the number of iterations reaches a bound specified as MAXSTEP. This is the outline of our algorithm. It is easy to see that each iteration can be executed in time $\mathcal{O}(n + m)$.

In [WY06], some theoretical justification to this algorithm is given. More precisely, a slightly simpler version modified as follows has been studied: (i) set MAXSTEP = 2; that is, beliefs are updated only twice, (ii) execute statement (1) only after the second iteration, and (iii) insert statement (2). For this algorithm, it is shown that the algorithm yields one of the planted solution pair with probability $1 - o(1)$ if $n = \Omega(\ln(n/\delta)/p^2)$ (or almost equivalently, $p = \Omega(n^{-1/2} \ln n)$)), where "order" is taken w.r.t. n.

4 Example 3: Graph Bisection Problem

For the last example, we consider a well-known graph partitioning problem, called *Graph Bisection problem*. The problem is to find an equal size partition of a given undirected graph with the smallest number of crossing edges. Again it is NP-hard in the worst case.

Here we consider undirected graphs with no loop nor multiple edges, and assume that the number of vertices is even. We use $2n$ and m to denote the number of vertices and edges respectively. For a graph $G = (V, E)$, an (*equal size*) *partition* is a pair of disjoint subsets V_+ and V_- of V such that $V = V_+ \cup V_-$ and $|V_+| = |V_-|$. For a given partition V_+ and V_- edges in $V_+ \times V_- \cap E$ are called crossing edges. The Graph Bisection problem is then defined as follows.

Graph Bisection problem	
Instance:	An undirected graph $G = (V, E)$ with $2n$ vertices.
Task:	Find an equal size partition of V with the smallest number of cut edges.

Consider our probability model. Here let us first define its instance generating function g_{part}. Let $V = \{v_1, ..., v_{2n}\}$ be a set of $2n$ vertices, and let $\boldsymbol{a} \in \{+1, -1\}^{2n}$ be an *assignment* denoting one partition V_+ and V_- of V; that is, v_i is in V_+ (resp., V_-) if $a_i = +1$ (resp., $a_i = -1$). A pair of vertices v_i and v_j of V is called a *like-class* pair (resp., a *diff-class* pair) if both are in either V_+ or V_- (resp., one is in V_+ and another is in V_-). Let m_1 and m_2 be the number of like-class and diff-class pairs of vertices in V respectively; let $\boldsymbol{r}_1 \in \{0, 1\}^{m_1}$ and $\boldsymbol{r}_2 \in \{0, 1\}^{m_2}$. For these $\boldsymbol{a}$, $\boldsymbol{r}_1$, and $\boldsymbol{r}_2$, we define $g_{\text{part}}(\boldsymbol{a}, \boldsymbol{r}_1, \boldsymbol{r}_2)$ to be a graph $G = (V, E)$, where E has an edge (v_i, v_j) if and only if the corresponding bit of $\boldsymbol{r}_1$ (resp., $\boldsymbol{r}_2$) is 1 for a pair (v_i, v_j) is like-class (resp., diff-class pair). Now with two probability parameters p and r ($p > r$) we assume that our problem instance is generated (from a given assignment $\boldsymbol{a}$) as $g_{\text{part}}(\boldsymbol{a}, \boldsymbol{r}_1, \boldsymbol{r}_2)$ for randomly generated $\boldsymbol{r}_1 : \mathcal{B}(m_1, p)$ and $\boldsymbol{r}_2 : \mathcal{B}(m_2, r)$. This is the distribution we consider for Graph Bisection problem. Note that this distribution or probability model is essentially the same as the planted solution model that has been used to discuss Graph Bisection problem [JS98]).

For this generating function, a MLS problem we can consider naturally is to find $\boldsymbol{a}$ minimizing the following probability for a given input graph G:

$$\begin{aligned}&\Pr_{\boldsymbol{u}_1, \boldsymbol{u}_2}[\ G = g_{\text{part}}(\boldsymbol{a}, \boldsymbol{u}_1, \boldsymbol{u}_2)\ ]\\ &\quad= \prod_{(v_i, v_j) \in E} p^{[a_i = a_j]} r^{[a_i \neq a_j]} \cdot \prod_{(v_i, v_j) \in \overline{E}} (1-p)^{[a_i = a_j]} (1-r)^{[a_i \neq a_j]},\end{aligned}$$

where random variables $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ follow $\mathcal{B}(m_1, p)$ and $\mathcal{B}(m_2, r)$ respectively. Note that we do not restrict $\boldsymbol{a}$ to those corresponding to some equal size partition, and m_1 and m_2 vary depending on $\boldsymbol{a}$, more precisely, the number of +1's in $\boldsymbol{a}$. As explained in the previous section, we can consider some scenario for this inference problem; see also, e.g., [DLP03].

```
procedure algo_BiSect ($G$, $p$, $r$);
begin
  set all $b_i$ to 0;
  repeat MAXSTEP times do {
    $b_1 \leftarrow +\infty$;
    for each $v_i \in V$ do in parallel {
      $b_i \leftarrow \sum_{v_j \in N_i} h_+ \cdot \mathrm{Th}_+(b_j)$
             $- \sum_{v_j \notin N_i} h_- \cdot \mathrm{Th}_-(b_j)$;
    }
    if all $b_i$'s get stabilized then break;
  }
  output $(+1, \mathrm{sg}(b_2), ..., \mathrm{sg}(b_{2n}))$;
end-procedure
```

parameters & functions

$$c_- = \frac{1-p}{1-r},\ c_+ = \frac{p}{r},$$

$$h_- = \left|\frac{c_- - 1}{c_- + 1}\right|,\ h_+ = \left|\frac{c_+ - 1}{c_+ + 1}\right|,$$

$$\mathrm{th}_- = \left|\frac{\ln c_-}{h_-}\right|,\ \mathrm{th}_+ = \left|\frac{\ln c_+}{h_+}\right|,$$

$$\mathrm{Th}_+(z) = \mathrm{sg}(z)\min(|z|, \mathrm{th}_+),$$

$$\mathrm{Th}_-(z) = \mathrm{sg}(z)\min(|z|, \mathrm{th}_-),$$

$\mathrm{sg}(z) =$ the sign of z, and

$N_i =$ the set of v_i's neighbors.

Fig. 2. Computation of pseudo beliefs for the MLP problem

Again the goal of this MLS problem is not always the same as that of Graph Bisection problem. Furthermore, for any choice of parameters p and r, we can find some example for which two problems ask for different answers. Nevertheless, we can show that the same solution is asked by these two problems with high probability for some range of parameters p and r. More specifically, we can show [Betal87] that if $p - r = \Omega(n^{-1/2})$, then with high probability, the planted solution is the unique optimal solution of Graph Bisection problem. On the other hand, the planted solution is also the unique optimal solution for the MLS problem with high probability if $p - r = \Omega(n^{-1/2})$ [Ons05]. Thus, for this range of p and r, with high probability, two problems share the same solution, which is indeed the planted solution.

Now we explain the algorithm algo_BiSect of Figure 2. This algorithm is derived from the standard belief propagation algorithm following the template given in [MMC98] applying two simplifications; see [OW05] for the derivation. It again updates beliefs for each vertex $v_i \in V_+$ at each round. An updated value of b_i is computed by summing up the beliefs of *all* vertices v_j, multiplied by either $h_+ > 0$ (if an edge (v_i, v_j) exists) and by $-h_- < 0$ (otherwise). This is intuitively reasonable because one can expect that two vertices v_i and v_j are in the same class (resp., in the different classes); if an edge exists (resp., does not exist) between them. The algorithm uses threshold functions $\mathrm{Th}_+(z)$ and $\mathrm{Th}_-(z)$ so that too large (or too small) beliefs are not sent to the other vertices. The algorithm terminates (before the time bound) if b_i gets stabilized for every i, i.e., either the change of b_i becomes small, or $|b_i|$ exceeds the threshold value $\max(\mathrm{Th}_+, \mathrm{Th}_-)$.

In [OW06] we give some theoretical analysis to this algorithm, again under the following further simplifications: (i) set MAXSTEP = 2; (ii) use some small $\theta < \min(\mathrm{th}_+, \mathrm{th}_-)$ for the initial value of b_1 (for avoiding the thresholding), and (iii) set $b_1 = 0$ before the second round (for ignoring the effect from v_1 in the second round).

For this version of `algo_BiSect`, we prove that it yields the planted solution with high probability if $p - r = \Omega(n^{-1/2} \log n)$. Note that this result is weaker than those for the other algorithms [Bop87, McS99, Coj06]; but we conjecture that the general version of our algorithm performs as well as the other ones.

References

[Bop87] Boppana, R.B.: Eigenvalues and graph bisection: an average-case analysis. In: Proc. Symposium on Foundations of Computer Science, pp. 280-285 (1987)

[Betal87] Bui, T., Chaudhuri, S., Leighton, F., Spiser, M.: Graph bisection algorithms with good average behavior. Combinatorica 7, 171–191 (1987)

[Coj06] Coja-Oghlan, A.: A spectral heuristic for bisecting random graphs. Random Struct. Algorithms 29(3), 351–398 (2006)

[DLP03] Dubhashi, D., Laura, L., Panconesi, A.: Analysis and experimental evaluation of a simple algorithm for collaborative filtering in planted partition models. In: Pandya, P.K., Radhakrishnan, J. (eds.) FST TCS 2003. LNCS, vol. 2914, pp. 168–182. Springer, Heidelberg (2003)

[Gal62] Gallager, R.G.: Low density parity check codes. IRE Trans. Inform. Theory IT-8(21), 21–28 (1962)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and Intractability, Bell Telephone Laboratories, Incorporated (1979)

[JS98] Jerrum, M., Sorkin, G.: The Metropolis algorithm for graph bisection. Discrete Appl. Math 82(1-3), 155–175 (1998)

[LMSS01] Luby, M., Mitzenmacher, M., Shokrollahi, M., Spielman, D.: Improved low-density parity-check codes using irregular graphs. IEEE Trans. on Information Theory 47(2), 585–598 (2001)

[Mac99] MacKay, D.: Good error-correcting codes based on very sparse matrices. IEEE Trans. Inform. Theory IT-45(2), 399–431 (1999)

[MMC98] McEliece, R., MacKay, D., Cheng, J.: Turbo decoding as an instance of Pearl's Belief Propagation algorithm In: EEE J. on Selected Areas in Comm. 16(2) (1998)

[McS99] McSherry, F.: Spectral partition of random graphs. In: Proc. 40th IEEE Sympos. on Foundations of Computer Science (FOCS'99), IEEE, NJ, New York (1999)

[Ons05] Onsjö, M.: Master Thesis (2005)

[OW05] Onsjö, M., Watanabe, O.: Simple algorithms for graph partition problems, Research Report C-212, Dept. of Math. and Comput. Sci. Tokyo Inst. of Tech (2005)

[OW06] Onsjö, M., Watanabe, O.: A simple message passing algorithm for graph partition problem. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 507–516. Springer, Heidelberg (2006)

[Pea88] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers Inc (1988)

[WST] Watanabe, O., Sawai, T., Takahashi, H.: Analysis of a randomized local search algorithm for LDPCC decoding problem. In: Albrecht, A.A., Steinhöfel, K. (eds.) SAGA 2003. LNCS, vol. 2827, pp. 50–60. Springer, Heidelberg (2003)

[WY06] Watanabe, O., Yamamoto, M.: Average-case analysis for the MAX-2SAT problem. In: Campilho, A., Kamel, M. (eds.) ICIAR 2006. LNCS, vol. 4142, pp. 277–282. Springer, Heidelberg (2006)

Turing Unbound: Transfinite Computation

Philip D. Welch

School of Mathematics, Unversity of Bristol, England
p.welch@bristol.ac.uk

Till thine Infinity shall be
A robe of envenomed agony;
And thine Omnipotence a crown of pain,
To cling like burning gold round thy dissolving brain.

Shelley, Act 1 *Prometheus Unbound*

1 Introduction

The intention of this talk is to look at various models of transfinite computation and give some calculations as to their comparative power. To clarify the kind of models that we are looking at, they will all be *discrete acting*: this will mean that they essentially perform simple discrete tasks in simple steps or stages. The reader should have in mind the paradigm of the *standard Turing Machine* which of course performs such simple actions as moving one cell left or right on the tape that it is reading, altering a symbol, changing a state *etc etc.* We are thus not considering any kind of machine or notional device that computes in an analogue fashion, nor any machine, such as neural network with nodes primed by infinitely precise real numbers, nor computations performed in chemistry beakers, across cell membranes, or in buckets of slime.

Our purpose here is purely *logico-mathematical*: to determine what these models can and can not do. Just as Turing established the range and capabilities of the Turing machine we wish to do likewise for the various models considered here. We are diagnostic of the formalisms proposed, but agnostic as to the desirability, notional feasability, and so forth, of them. The disclaimer is that mention of any product or products does not imply endorsement by the author.

Various machine proposals that fall under our rubric but which seek to compute functions beyond the standard recursive functions[1] are deliberately constructed to conform to some ambient physical theory or constraint (*cf.* [2], Davies). Apart from Sect.5, we shall not be entering into any discussion of *physical* viabilities, feasabilities and so forth.

In Section 2 we consider briefly some well-known facts about the standard model when the latter is allowed to run out to ω: *i.e.* infinitely many stages are computed and we then are allowed to inspect the tape.

Section 3 looks at a recent group of papers of Hogarth, [9], [10] and Etesi & Németi [2] which consider how to answer questions beyond the recursive using arrangements of Turing machine(s) in a class of general relativistic spacetimes known as *Malament-Hogarth* spacetimes. We delineate there the possible extent of computation in such universes.

Going beyond the ω'th stage requires either stacking up machines, or recycling the tape in some way or other. The Infinite Time Turing Machines (ITTM's) of Hamkins and Kidder [5] do precisely this. One might regard such a model as a laboratory to investigate what can be done transfinitely, just as the standard TM appears to be a suitable laboratory for finite discrete computation (strong forms of the Church-Turing thesis may claim that it *is* a suitable model for such computation). Section 4 considers these. Section 5 catalogues conceptual devices that extend this, or otherwise use "more sets" in their description. One may allow the tape to also be transfinite and one arrives at the ordinal tape machines of Koepke, Dawson, Siders; Koepke and Siders have considered *ordinal register machines*: these have finitely many registers, and a push-down stack, but allow ordinals (rather than integers) as objects to populate the machines. Naturally such devices will compute more relations than an ITTM will. As their authors have shown, one essentially can define Gödel's constructible hierarchy L from these. For completeness we mention here also finite state automata working on trees, or ordinals, (Rabin, Büchi, Neeman, Thomas *inter alia*) but their capabilities are well documented elsewhere, and so we do not pay them much attention in this article. Nor de we consider the extensive field of *computation on the reals*: we consider computation on infinite sequences, as ITTM's and others deal with (and so are capable of *higher type recursion*) but such models consider reals as elements of $\mathcal{P}(\mathbb{N})$ rather than elements of the real continuum $\mathbb{R}$. Older examples of such recursion are Kleene's recursion in $\mathbb{R}$. (See for example [11]).

The unifying thread is a simple statement: transfinite processes encroach on the infinitary world of subsytems of analysis, and of low level set theory, therefore analytical techniques, or those previously studied under the rubric of *generalised recursion theory* (such as α-recursion, higher type recursion) and *descriptive set theory* have a role to play. We want to urge that arguing at a higher conceptual level brings clarity and avoids repeating arguments from earlier eras or other fields.

[1] We shall reserve the term *recursive functions* for those standardly computed by TM's leaving "computable function" free for use for those functions computed by the model currently under consideration.

2 To the ω'th Station!

A Turing machine if left to its own devices can write a recursively enumerable, or Σ_1, set of integers to its output tape. We may thus consider as computing partial answers to whether $?n \in A?$ for any Σ_1 set A. If we suppose that there is an ω'th stage of time, then we may think of that as also having computed a *full* answer: if at time ω no affirmative answer to $?n \in A?$ has been received we may conclude that $n \notin A$. However we can do better than that. We suppose that our standard Turing machine's are writing an Accept/Reject 0/1 output to the first cell of their tape, without actually halting, but continue computing in case they wish to change their minds

Definition 1. *(Putnam [16])* $R \subset \mathbb{N}$ *is* a trial and error predicate *if there is a Turing machine* M_0 *so that*

$n \in R \Longleftrightarrow$ *the eventual value of* M_0*'s output tape on input* n *is 1*
$n \notin R \Longleftrightarrow$ *the eventual value of* M_0*'s output tape on input* n *is 0.*

The determining feature is that for such predicates the machine eventually settles down to a fixed value after some finite time. The standard interpretation is that for a trial and error predicate one can have a computational arrangement that provides the correct answer without one ever being in the position at a finite stage in time of knowing for sure that we have it. Nor can we have any recursive bound in terms of the input n on the number of time to expect a change of mind. Such predicates are known to be equivalent to Δ_2 predicates in the arithmetical hierarchy (*cf.* also Gold [3]).

One might be concerned that positing an ω'th stage in time might require a mechanism that writes to a particular cell on the tape infinitely often, when calculating such predicates, thereby raising the spectre of some Thomson Lamp like difficulties of what is written on the cell at time ω. However for Δ_2 predicates one can arrange matters so that this does not occur. (Of course this begs the question of where the read/write head will be at time ω!)

To decide Σ_2 predicates in ω steps, this will have to fail. We may *write* a Σ_2 predicate Q, or subset of $\mathbb{N}$, to a recursive slice of the tape, but if we are requiring that the first cell C_0 on the tape has the correct 0/1 answer after ω steps, it is easy to see we may have a positive answer (say given by 1) but for a negative answer in general there can be no finite stage at which "0" will be written to C_0 never to change later (else the predicate would be Δ_2).

So to *decide* Σ_2 questions requires a substantial modification, to which we shall turn after an interlude spent looking at computations in special spacetimes..

3 General Relativistic Models

As mentioned Hogarth, in [9], and [10], and Etesi & Németi [2] consider how to answer questions beyond the recursive using arrangements of Turing machine(s) in a class of general relativistic spacetimes known as *Malament-Hogarth* spacetimes. Both sets of authors query how far in the arithmetic hierarchy such computations can succeed. In [24] we consider the logico-mathematical limits of computation in their models.

Pitowsky [15] gives an account of an attempt to define spacetimes in which the effect of infinitely many tasks can be realised - essentially they allow the result of infinitely many computations by one observer O_r (he used the, as then unsolved, example of Fermat's Last Theorem) performed on their infinite (*i.e.* endless in proper time) world line γ_1, to check whether there exists a triple of integers $x^k + y^k = z^k$ for some $k > 2$ as a counterexample to the Theorem or not. If a counterexample was found a signal would be sent to another observer O_p travelling along a world line γ_2. The difference being that the proper time along γ_2 was finite, and thus O_p could know the truth or falsity of the Theorem in a (for them) finite time, depending on whether a signal was received or not. As Earman and Norton [1] mention, there are problems with this account not least that along γ_2 O_p must undergo unbounded acceleration.

Malament and Hogarth alighted upon a different spacetime example. The following definition comes from [1] (M is a pseudo-Riemannian manifold, g_{ab} a suitable metric):

Definition 2. $\mathcal{M}$=(M, g_{ab}) *is a* Malament-Hogarth (MH) spacetime *just in case there is a time-like half-curve* $\gamma_1 \subset M$ *and a point* $p \in M$ *such that* $\int_{\gamma_1} d\tau = \infty$ *and* $\gamma_1 \subset I^-(p)$.

(Here τ is proper time.[2]) This makes no reference to the word-line of an observer $\mathcal{O}_p$ travelling along their path γ_2, but point out that there will be in any case such a future-directed timelike curve γ_2 from a point $q \in I^-(p)$ to p such that $\int_{\gamma_2(q,p)} d\tau < \infty$, with q chosen to lie in the chronological future of the past endpoint of γ_2. (The important point is that the *whole* of γ_1 lies in the chronological past of $\mathcal{O}_p$. As Hogarth showed in [8] such spacetimes are not globally hyperbolic, thus ruling out many "standard" space-times (such as Minkowski space-time). Hogarth's diagram of a "toy MH space-time" is Figure 1 below. These are in general limited by assumptions they make concerning their physical set ups. Both sets of authors make

Assumption 1. *"no swamping":* no observer or part of the machinery of the system has to send or receive infinitely many signals.

Etesi & Németi then consider a particular spacetime (Kerr spacetime) and the case of a Turing machine sent along the world-line γ_1 searching for counterexamples to a Π_1 predicate (for example). A signal can be sent to $\mathcal{O}_p$ if one is found. Consequently they have a (real world (?)) procedure for deciding Π_1 queries. They observe that if the arrangment sends, for example, *two* signals to $\mathcal{O}_p$, then they can decide the difference of two Π_1 sets, and ask how far this can be taken. We show:

[2] We conform to the notation of Hawking & Ellis [7] and so $I^-(p)$ is the *chronological past* of p: the set of all points q from which a future-directed timelike curve meets p. The spacetimes, all derived from Malament and Hogarth's "toy spacetime", are differentible manifolds with a Lorentz metric g_{ab}, and are time-oriented.

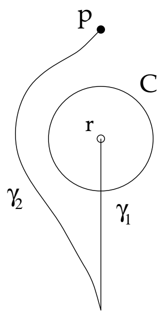


Fig. 1. A toy MH spacetime

Theorem 1. *[24] The relations $R \subseteq \mathbb{N}$ computable in the Etesi-Németi model form a subclass of the Δ_2 predicates of* $\mathbb{N}$*; this is a proper subclass if and only if there is a fixed finite bound on the number of signals sent to the observer $\mathcal{O}_p$.*

Clearly this is limited by Assumption 1: we know that there can be no recursive bound on the number of times a machine can change its mind (*i.e.* send a signal to $\mathcal{O}_p$) when computing a trial and error predicate, and so *a fortiori* no finite bound. If $\mathcal{O}_p$ is prepared to receive a *potentially* unbounded number of signals then the arrangement decides Δ_2. Note that $\mathcal{O}_p$ will not actually have to *receive* infinitely many signals, to decide a Δ_2 predicate, so Assumption 1 is not broken, but they had better be able in their physical arrangements not to have any boundedness restriction.

Hogarth is more ambitious, he considers preparing spacetime manifolds by arranging singularities as "MH-points" with observers or machines now $\mathcal{O}_{p_j}$, in open regions O_j of M, arranged with chains. By so doing he can decide any arithmetic predicate. He explicitly makes:

Assumption 2. *The regions O_j are all disjoint open regions of M.*

However with this formalism, the construction only scratches the surface of what is possible, one may show:

Theorem 2. *If H is any hyperarithmetic predicate on integers, then there is an MH spacetime in which any query $?n \in H?$ can be computed. Indeed there is a single spacetime which is* HYP*-Deciding, that can compute all such queries.*

However in one sense this is best possible. The last theorem is demonstrated by embedding, not just *chains* of MH spacetime components as in Fig. 1 in regions O_j, but embedding *recursive finite path trees*. Such can code the recursive *construction* of hyperarithemetic sets (a well known fact - see [17] 16.8 or [18])[3]

[3] Of course this is *not* to say that H is itself recursive, it is just that its *construction* has a recursive description. The set of codes of hyperarithmetic sets is not recursive, or r.e., or even arithmetic, it is complete Π^1_1 ; it thus requires a universal function quantification in analysis, or second order number theory (see for example, [17] Thm.*XX*.

Theorem 3. *Assuming the (modest and standard) requirement that space-time manifolds be paracompact and Hausdorff, for any MH spacetime $\mathcal{M}$ there will be a countable ordinal upper bound, $w(\mathcal{M})$, on the complexity of predicates in the Borel hierarchy resolvable in it.*

The reason is simple: just as in first year analysis, there can be at most countably many disjoint open intervals of $\mathbb{R}$, so there can be only countably many disjoint open regions $O_j \subseteq M$ (paracompactness and Hausdorff implies separable). So there cannot be for every countable ordinal α less than ω_1, regions of M in which there is finite path tree T_α of rank α, of singularities arranged *à la* Hogarth.

Of course you may prefer to believe that *our* spacetime $\mathcal{M}_{\mathrm{real}}$ has w number 0!

4 Beyond ω: Infinite Time Turing Machines (ITTM's)

Let us change the architecture. Suppose we allow cell values to change infinitely often. Let us arbitrarily declare that at stage ω the value of the cell C_i is its eventual value, if such exists, or otherwise, if the value has changed infinitely often let us fill the cell with B(lank) at time ω. Now our extended Turing machine (as we have gone beyond the standard Turing machine) will have a definite answer for us at time ω for answering $?n \in Q?$ for Σ_2 Q: the first cell will have a 1 for "yes" and a B for "no".[4]

Having gone this far we might as well go the whole hog and restart the machine using the current tape contents. We specify that the R/W head has magically reappeared on the first cell C_0. The cell values are as given in the last paragraph, and the machine has entered a special "Limit state" q_L. We thus enlarge the standard state set that the machine was equipped with, $q_1, \ldots, q_N$. Likewise the transition table, or program instruction list, or however we have set up our machines, will have instructions involving q_L which will tell the machine how to proceed to the $\omega+1$, or $\omega.2+1$ or any $\lambda+1$ stage (for any limit ordinal λ). Note that we still have *finite programs* and so can assume an enumeration $\langle P_e | e \in \mathbb{N}\rangle$ of all programs. At limit stages λ we regard the R/W head as reading C_0. Note also that we have essentially a machine acting on an alphabet of 3 letters $\{0, 1, B\}$. We may let this machine run indefinitely through any or all ordinal segments of time according to taste.

Noting that the machine may halt with the contents of its output tape essentially a member of $3^{\mathbb{N}}$ (identifying 3 with $\{0, 1, B\}$) or perhaps $2^{\mathbb{N}}$, we can think of the machine as outputting reals. Indeed we may think of the machine as having a sequence $y \in 3^{\mathbb{N}}$ on the tape to start with. So we could also think of the machines as performing some kind of higher type recursion computing functionals $F : 3^{\mathbb{N}} \longrightarrow 3^{\mathbb{N}}$, or we may require more conventional output and compute functions $F : 2^{\mathbb{N}} \longrightarrow 2^{\mathbb{N}}$. Suppose we denote by $P_e(n)$ the e'th computation

[4] One can show that actually one only needs the first cell alone to have the ability to change value infinitely often, the calculation can be done without more than a finite number of chages for the other cells.

on integer input of n, represented by an infinite string of n 1's followed by an infinite string of 0's. Several natural questions arise.

Q1: *What is* $\{e | P_e(0) \downarrow\}$*? (The halting problem on integers).*
Q2: *What is* $\{y \in 2^{\mathbb{N}} | \exists e \in \mathbb{N}\ P_e(0) \downarrow y\}$ *?*
Q3: *What are the halting times that arise? That is if* $P_e(0) \downarrow$*halts in* α *steps* $(P_e(0) \downarrow^{\alpha})$*how large is* α*? Is* α *a recursive ordinal?*
Q4: *What is the degree structure that arises, setting*

$$x \leq_\infty y \Longleftrightarrow \exists e \in \mathbb{N}\ P_e(y) \downarrow x?$$

The architecture we have here is essentially that of the *Infinite Time Turing Machines* of Hamkins and Kidder [5]. The difference between the presentation of a 1 tape machine here, and their 3 tape version is rather inessential: they have separate but parallel infinite tapes for input, scratch work, and output. A R/W head may scan the k'th cell from each of the tapes simultaneously, but it acts at limit stages in the same manner by returning to the leftmost triplet of cells. They work instead on an alphabet of $2 = \{0, 1\}$ and specify at limit times λ that $C_n(\lambda)$ has value the $limsup_{\alpha \to \lambda}\{C_n(\alpha)\}$. The answers to **Q1-Q4** above are unaffected by the choice between these two formalisms[5]. Likewise the class of functions $f : \mathbb{N} \longrightarrow \mathbb{N}$, or $F : 2^{\mathbb{N}} \longrightarrow 2^{\mathbb{N}}$ is the same for both types of machine. For most global questions it is immaterial which class of machines is employed.

In order to prove some kind of *Normal Form Theorem* (such as Kleene's T-predicate provides for standard Turing machines) one needs that the class of halting times of computations does not outstrip codes for ordinals that are potentially themselves the lengths of such halting computations. Fortunately this turns out to be the case [22] and we have

Theorem 4. *(Normal Form Theorem [23])* $\forall e \exists e' \forall x \in 2^{\mathbb{N}}$
$P_e(x) \downarrow \longrightarrow [P_{e'}(x) \downarrow y$ *where* $y \in 2^{\mathbb{N}}$ *codes a wellordered course-of-computation sequence for* $P_e(x) \downarrow]$.
Moreover the map $e \longrightarrow e'$ *is effective (in the usual Turing sense).*

Definition 3. *[5]*

$$x \subseteq \mathbb{N} \textit{ is (infinite time) semi-decidable iff } \exists e \forall n \in \omega [P_e(n) \downarrow 1 \leftrightarrow n \in x].$$

$$X \subseteq 2^{\mathbb{N}} \textit{is (infinite time) semi-decidable iff } \exists e \forall x \in 2^{\mathbb{N}} [P_e(x) \downarrow 1 \leftrightarrow x \in X].$$

[5] There are some tiny variations in some cases of answers to **Q3** on halting times, globally the supremum γ of halting times on integer inputs are the same, but to some halting times an addition of ω or so units of time is needed to get the sums to come out right. In fact the limit rule that [5] choose is quite robust: changing lim sup to liminf is (perhaps unsurprisingly) immaterial when considering the class of computable functions.

Definition 4

(i) $x \leq_\infty y \longleftrightarrow \exists e P_e(y) \downarrow x$; *we have a jump operation:*

$$x^\nabla =_{\mathrm{df}} \{e | P_e(x) \downarrow\}$$

(ii) $X {\leq_\infty} Y \longleftrightarrow \exists e \forall x [P_e^Y(x) \downarrow 1 \leftrightarrow x \in X \wedge P_e^Y(x) \downarrow 0 \leftrightarrow x \notin X]$; *we have again a jump operation:*

$$X^\blacktriangleright =_{\mathrm{df}} \{(e,x) | P_e(x) \downarrow\} \cup X.$$

In the second clause we have used the notation P_e^Y for the relativised oracle machine which answers queries $?y \in Y?$ for $y \in 2^{\mathbb{N}}$.

Theorem 5. *(Hamkins-Lewis) [5] Any arithmetical predicate is decidable by ITTM's in* $< \omega.\omega = \omega^2$ *steps.*

(ii) Any Π^1_1 *predicate on* $2^{\mathbb{N}}$ *is also ITTM decidable, in particular there is a machine* P_e *that decides whether an input* $y \in 2^{\mathbb{N}}$ *is in WO.*

(iii) The relations "$P_e(x) \downarrow y$*" and "*$P_e(x) \uparrow$*" are* Δ^1_2*. The ITTM semi-decidable predicates form a Spector class which is a proper subclass of the* Δ^1_2 *sets.*

Because of (ii) and (iii) above we see that the reducibility $\leq_\infty$ is, say on the sets of integers, intermediate between hyperarithmeticity and Δ^1_2. Indeed we have an ordinal assignment that satisfies a Spector Criterion:

Theorem 6. *Let* $\lambda^x =_{\mathrm{df}}$ *the least ordinal* λ *which has no code computable from* x*. Then:*

$$x \leq_\infty y \longrightarrow (x^\nabla \leq_\infty y \longleftrightarrow \lambda^x < \lambda^y).$$

4.1 Punch Tape Machines

Suppose you object to the notion of rewriting values to a cell on a tape infinitely often. The you might be happier with the concept of a *punch tape machine*. Such merely have a read/punch head now for writing to a blank cell by punching a hole. So, ignoring difficulties with "hanging chads", such cells are usable once only. Equipped with a standard-ish Turing program, one easily sees that in ω steps again Δ_2, or trial-and-error predicates are decidable. What if now we reset the head to the zero'th cell at time ω and let the machine continue? Could we calculate more? We have the following observation:

Proposition 1. *(S-D. Friedman-PDW) Precisely the arithmetical predicate are decidable by punch tape machines, and any such computation either halts by, or is in an infinite loop, by time* ω^2*.*

5 Larger Machines

One may consider ITTM's a starting point for wider classes of machines. Koepke and Dawson independently came up with the idea of allowing ordinal length tape on which $\{0, 1\}$ marks could be input or written. To make use of this tape we must not continually reset the head position to 0, to allow the head position at some limit ordinal time λ say, to be the *liminf* of its previous positions. They both allowed the machine at time λ to go into the state that was the again the *liminf* of the previous states. In programming terms this quite neatly puts the machine back at the head of the outermost loop it was cycling through before reaching time λ.

As sets can be coded by ordinals (assuming some form of Axiom of Choice) we have a means of working with sets. Conversely we can think of such machines as *producing* codes for sets by the marks they write. Dawson formulates an *Axiom of Computability* that asserts thet every set can appear coded on the output tape by some program. He then proves that the computable sets form a transitive class satisfying AC and moreover the Generalised Continuum Hypothesis. Clearly such machines have a very absolute nature, and so can be run inside the constructible hierarchy L. It then becomes clear that the class of computable sets in this sense, being a transitive ZF model containing all ordinals, can be none other than L itself which is the smallest such.

Koepke gave a detailed description [12], [13] of the organisation of such programs and a proof that a bounded truth function for L is ordinal computable by a halting program. Whereas Dawson was considering sets that appeared on a tape, Koepke considers *halting* computation from an input tape containing marks for finitely many ordinals.

Theorem 7. *(Koepke [12]) A set x of ordinals is ordinal computable from a finite set of of ordinal parameters if and only if it is an element of the constructible hierarchy L.*

We thus have another presentation of the constructible hierarchy to join those of Gödel, Jensen, and Silver.

5.1 Ordinal Register Machines (ORM)

It appears that the concept of a machine with finitely many registers, but with ordinals (rather than natural numbers) stored in them has arisen from time to time. Platek considered such when he formulated the axioms of KP, and the *metarecursion theory* which was an early example of generalising recursion theory from ω to ω_1^{ck} (see Part B of [18]). Siders considers [14] such a machine with a stack, and the authors show again that the recursive truth predicate for the constructible universe is decidable by such machines.

5.2 Higher Type- and α-Recursion

Kleene Recursion and ITTM semi-decidability. Kleene developed an equational calculus for developing the notion of *recursion on a higher type*:

$$x \in A \simeq \{e\}(x, y, B, {}^2E) \downarrow 1 \qquad A, B \subseteq \mathbb{R}\,(= 2^{\mathbb{N}})$$

(Here ${}^2E$ is the Type 2 functional evaluating equality between members of ${}^{\mathbb{N}}2$.) It has been characterised ([11]) as a model of computation in which a computational device had a

(i) *countably infinite memory,* and
(ii) *an ability to manipulate (search through, write to) that memory in finite time;* optionally
(iii) *an ability to quiz an oracle (for B) about its entire memory contents.*
We may think of this as a Turing machine with one (or more) infinite tapes on which reals (identified with infinite sequences of 0's,1's) are written and the ability to ask the oracle at any stage of the computation as to whether the current real under consideration is in some "oracle set" $B \subseteq \mathbb{R}$.

• This is not to be conceived as a computation that runs in transfinite segments of discrete time, but rather as one that makes calls for values from *subcomputations*; a computation thus has a *wellfounded finite path tree* structure.

• The course of computation may evolve its own tree structure as it progresses according to its instruction set; we may also view a "machine" as having a previously determined tree structure as part of its "instructions" or program. In short the machine may be viewed as determined by a (finite) program together with a (code, y, for) an infinite finite path-tree.

Kleene degrees: Let $A, B \subseteq \mathbb{R}$; we say that $A \leq_K B$ iff
there is some computational arrangement P such as above so that

$$\forall x \in \mathbb{R}(x \frac{\in}{\notin} \Longleftrightarrow P^B(x) \downarrow \frac{1}{0})$$

$$\begin{aligned}\text{iff} \quad & \text{there are } \Sigma_1\text{-formulae in } \mathcal{L}_{\in,\dot{X}}\ \varphi_1, \varphi_2, \text{ there is } y \in \mathbb{R} \text{ so that} \\ & \text{for any } x \in \mathbb{R}\,(x \in A \Longleftrightarrow L_{\omega_1^{B,y,x}}[B,y,x] \models \varphi_1[B,y,x] \\ & \qquad\qquad \Longleftrightarrow L_{\omega_1^{B,y,x}}[B,y,x] \models \neg\varphi_2[B,y,x]\ )\end{aligned}$$

(here $\omega_1^{B,y,x}$ is the least (B,y,x)-admissible ordinal).

0_K contains $\varnothing, \mathbb{R}$, and in fact consists of the *Borel sets*. $0'_K$ (the K-degree of a complete Kleene semi-recursive set of reals) contains WO the set of reals coding wellorders, and so a complete Π^1_1 set of reals. In fact it consists of the co-analytic, so $\boldsymbol{\Pi^1_1}$ sets.

• (Solovay) [20] AD (Axiom of Determinacy) *implies that the K-degrees are wellordered. Indeed a K-degree forms a boldface pointclass being closed under continuous preimages.*

•(Harrington-Steel) [21], [6] Determinacy(Bool(Π^1_1)) $\Longleftrightarrow \neg\exists A(0 <_K A <_K$ WO)

(Simpson) [11] $V = L \Longrightarrow$ there are many $<_K$-incomparable sets of reals below WO.

If one formulates the Definitions 3 and 4 of decidable and semi-decidable similarly with a real parameter y occurring just as for Kleene degrees, then one obtains similarly a notion of bold-face pointclass. The degrees so obtained are then closed unde continuous pre-images, in short they form a *Wadge degree* and their study is then amenable to methods of descriptive set theory.

Contrasting with the characterisation of Kleene's degrees above we have:

Theorem 8. *[23]* $A \leq_\infty B \Longleftrightarrow_{\mathrm{df}}$ *for some* $e \in \omega$*, some* $y \in \mathbb{R}$:

$$\forall x (x \overset{\in}{\underset{\notin}{}} A \longleftrightarrow P_e^{y,B}(x) \downarrow \tfrac{1}{0})$$

$\Longleftrightarrow$ *there are* Σ_1*-formulae in* $\mathcal{L}_{\in,\dot{X}}$ φ_1, φ_2*, and* $y \in \mathbb{R}$*, so that*

$$\forall x \in \mathbb{R}(x \in A \longleftrightarrow L_{\lambda^{B,y,x}}[B,y,x] \models \varphi_1[B,y,x]$$
$$\longleftrightarrow L_{\lambda^{B,y,x}}[B,y,x] \models \neg\varphi_2[B,y,x]).$$

Note the recurrence of the λ symbol: $\lambda^{B,y,x}$ is the supremum of all halting times of programs on input x with oracles for y, and B. The first equivalence here is The last equivalence of course has to be proven, but the reader can take it as a definition.

The $\leq_\infty$-degrees so formed are then boldface pointclasses within the Wadge hierarchy, with 0 and $0^{\blacktriangleleft}$ as the (degrees of the) ∞-recursive, and ∞-semirecursive sets of reals, respectively.

Definition 5. *Let* Γ_0 *be the class of* ∞*-semi-decidable sets of reals.*

Let F be the class of "quickly computable" reals: $x \in F \Longleftrightarrow x \in L_{\lambda^x}$ (Think of the "quickly constructible reals $Q = \{x | x \in L_{\omega_1^x}\}$.) The theorems of Simpson, and Harrington, Steel above become in this setting:

Theorem 9. *[23] (V=L)* $0 <_\infty F <_\infty 0^{\blacktriangleleft}$.

Theorem 10. *[23] (Det(Bool(*Γ_0*))) There is no* $A \subseteq \mathbb{R}$ *with* $0 <_\infty A <_\infty 0^{\blacktriangleleft}$.

ITTM's, ORM's and α-recursion. Clearly computing on ordinals invites comparison with the theory of α-recursion (see [18] Part C). One could consider computation on tapes of admissble ordinal length, or ORM's with entries restricted to ordinals below some admissible. Given the very absolute nature (in the sense of set theoretic absoluteness) of these machine models, and given the theorems just cited concerning the decidability of the bounded truth predicate for L one sees fairly immediately that proofs of theorems such as the Sacks-Simpson Theorem [19] on a solution to Post's problem, can be carried over, more or less by straight translation, to corresponding proofs concerning incomparable "semi-decidable" sets.

We may also compare the computations on ITTM's with say integer input and α-recursion theoretic results, with $\alpha = \lambda$. Here we are in the easiest case of Sacks-Simpson, as we find, like ω_1^{ck}, that λ is Σ_1-projectible to ω and $L_\lambda \models$"Everything is countable".

Bearing this in mind the following theorem of Hamkins and Lewis (which they proved by using Friedberg-Muchnik arguments) has a soft proof from α-recursion theory: one can simply replace the ordinals coded in the λ-r.e. incomparable sets A^*, B^* by their real codes and obtain ITTM semi-decidable (according to Def. 4) incomparable sets of reals A, B.

Theorem 11. *[4] (Hamkins-Lewis) There are ITTM semi-decidable sets* A, B *with* $0 <_\infty A, B <_\infty 0^{\blacktriangleright}$ *which are* $<_\infty$ *incomparable.*

(If the reader wonders why this appears to be a ZF theorem, which seems to conflict with 9 above, this is because the reducibility [4] uses is that of Def. 4, and the sets A, B are countable - in fact Σ_1-definable over L_λ - as one would expect from an α-recursion theoretic perspective. Hence if one uses the "boldface" reduction of 8 these sets simply disappear into a real parameter y.)

Similar results can be deduced from α-recursion theory for ordinal register or tape machines where the ordinals, or tape lengths come from other admissible α.

References

[1] Earman, J., Norton, J.D.: Forever is a day: Supertasks in Pitowsky and Malament-Hogarth spacetimes. Philosophy of Science 60, 22–42 (1993)

[2] Etesi, G., Németi, I.: Non-Turing computations via Malament-Hogarth space-times. International Journal of Theoretical Physics 41(2), 341–370 (2002)

[3] Gold, E.: Limiting recursion. Journal of Symbolic Logic 30(1), 28–48 (1965)

[4] Hamkins, J. D., Lewis, A.: Post's problem for supertasks has both positive and negative solutions. Archive for Mathematical Logic

[5] Hamkins, J.D., Lewis, A.: Infinite time Turing machines. Journal of Symbolic Logic 65(2), 567–604 (2000)

[6] Harrington, L.: Analytic determinacy and $0^{\#}$. JSL 43(4), 684–693 (1978)

[7] Hawking, S.W., Ellis, G.F.R.: The large scale structure of space-time. Cambridge University Press, Cambridge (1973)

[8] Hogarth, M.: Does general relativity allow an observer to view an eternity in a finite time? Foundations of Physics Letters 5(2), 173–181 (1992)

[9] Hogarth, M.: Non-Turing computers and non-Turing computability. PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association 1, 126–138 (1994)

[10] Hogarth, M.: Deciding arithmetic using SAD computers. British Journal for the Philosophy of Science 55, 681–691 (2004)

[11] Hrbacek, K., Simpson, S.: On Kleene degrees of analytic sets. In: Keisler, H.J., Barwise, J., Kunen, K. (eds.) Proceedings of the Kleene Symposium, Studies in Logic, pp. 347–352. North-Holland, Amsterdam (1980)

[12] Koepke, P.: Turing computation on ordinals. Bulletin of Symbolic Logic 11, 377–397 (2005)

[13] Koepke, P., Koerwien, M.: Ordinal computations. Mathematical Structures in Computer Science 16(5), 867–884 (2006)

[14] Koepke, P., Siders, R.: Computing the recursive truth predicate on ordinal register machines. In: Beckmann, A., et al. (ed.) Logical Approaches to Computational Barriers, Computer Science Report Series, p. 21. Swansea (2006)

[15] Pitowsky, I.: The physical Church-Turing thesis and physical computational complexity. Iyyun 39, 81–99 (1990)

[16] Putnam, H.: Trial and error predicates and the solution to a problem of Mostowski. Journal of Symbolic Logic 30, 49–57 (1965)

[17] Rogers, H.: Recursive Function Theory. Higher Mathematics. McGraw (1967)

[18] Sacks, G.E.: Higher Recursion Theory. In: Perspectives in Mathematical Logic, Springer, Heidelberg (1990)

[19] Sacks, G.E., Simpson, S.: The α-finite injury method. Annals of Mathematical Logic 4, 343–367 (1972)

[20] Solovay, R.M.: Determinacy and type 2 recursion. Journal of Symbolic Logic 36, 374 (1971)
[21] Steel, J.R.: Analytic sets and Borel isomorphisms. Fundamenta Mathematicae 108, 83–88 (1980)
[22] Welch, P.D.: The length of infinite time Turing machine computations. Bulletin of the London Mathematical Society 32, 129–136 (2000)
[23] Welch, P.D.: Post's and other problems in higher type supertasks. In: Löwe, B., Piwinger, B., Räsch, T. (eds.) Classical and New Paradigms of Computation and their Complexity hierarchies, Papers of the Conference Foundations of the Formal Sciences III, Trends in logic, October, vol. 23, pp. 223–237. Kluwer, Dordrecht (2004)
[24] Welch, P.D.: Turing Unbound: The extent of computations in Malament-Hogarth spacetimes. British Journal for Philosophy of Science (submitted)

Computability in Amorphous Structures*

Jiří Wiedermann[1] and Lukáš Petrů[2]

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

[2] Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague 1 Czech Republic
lukas.petru@st.cuni.cz

Abstract. Amorphous computing differs from the classical ideas about computations almost in every aspect. The architecture of amorphous computers is random, since they consist of a plethora of identical computational units spread randomly over a given area. Within a limited radius the units can communicate wirelessly with their neighbors via a single-channel radio. We consider a model whose assumptions on the underlying computing and communication abilities are among the weakest possible: all computational units are finite state probabilistic automata working asynchronously, there is no broadcasting collision detection mechanism and no network addresses. We show that under reasonable probabilistic assumptions non-uniform families of such amorphous computers can possess universal computing power with a high probability. To the best of our knowledge this is the first result showing the universality of such computing systems.

Keywords: computability, universality, amorphous computers, simulation, complexity.

1 Introduction

Classical models of universal computations, such as Turing machines, RAMs, etc., are rigorously defined mathematical structures in whose design there is no room for randomness. The situation is slightly different when the computing systems represented by networks of processors (such as the Internet, wireless networks, etc.) are considered: here, the network topology may result from a random process. In order to "compute" bold assumptions about such networks have been usually made: at least we require that all network nodes are connected by communication links, that prior to the start of computation each network node possesses a unique "network address", that there are communication primitives

* This research was carried out within the institutional research plan AV0Z10300504 and partially supported by the GA ČR grant No. 1ET100300517 and GD201/05/H014.

supporting message exchange and, last but not least, that each network node does posses a universal computing power. Such models have been the domain of the classical computational theory of the distributed system. However, recent developments in micro-electro-mechanical systems, wireless communications and digital electronics have brought yet a new challenge into the area of distributed computing systems. Their new instances integrate sensing, data processing and wireless communication capabilities. Typical representatives of such systems are sensor, mobile, or ad-hoc wireless networks (cf. [5]). At an extreme end, people consider exotic systems such as smart dust (cf. [8]) or amorphous computers (cf. [1], [2], [3]). In these systems the miniaturization is pushed to its limits resulting, presumably, into processors of almost molecular size with the respective communication and computing facilities adequately (and thus severely) restricted. These limitations call for the change of the basic computational and communication model of distributed computing systems which must subsequently be also reflected in the design of the corresponding algorithms.

It seems that so far the related research has mainly concentrated on the concrete hardware, software and algorithmic issues neglecting almost completely the computational and complexity aspects in that kind of computing (cf. [7]). Very often the designers of such algorithms have paid little attention to the underlying computational model and, e.g., they take for granted the universal computing power of all processors, synchronicity of time in all processors, the existence of unique node identifiers and those of communication primitives allowing efficient message delivery.

In our paper we focus on a computational model of a wireless communication network where such assumptions do not hold. This could be the case of e.g., the smart dust mentioned earlier. Our model, called *amorphous computer* works under very week assumptions: basically, it is a random graph which emerges by distributing the nodes randomly in the bounded planar area. The graph's nodes are processors represented by probabilistic finite state automata possessing no unique identifiers ("addresses"). The graph's edges exist only among the nodes within the bounded reach of each node's radio. Each node operates asynchronously, in either broadcasting or listening mode, hearing a message only if it is sent exactly by one of the node's neighbors. That is, there is no mechanism distinguishing the case of a multiple broadcast from the case of no broadcast. This model has been introduced recently by the authors in [6].

Due to its weak (and thus, general) underlying assumptions which correspond well to the case of amorphous computing as described in the literature, we believe that such a model presents a fundamental model of amorphous computing (cf. [1], [2], or an overview in [3]). Within the theory of computation a model of an amorphous computer, as given by our definition, represents an interesting object of study by itself since it contains elements of randomness built–in into both the computer's "set–up process" and its operations. The fundamental question is, of course, whether such a model does possess a universal computational power. The first steps towards this end have been taken in [6]. Here, under the above mentioned mild assumptions concerning the model of amorphous computing and under reasonable statistical assumptions on the underlying graph a scalable randomized auto-configuration protocol enabling message delivery from a source node to all other nodes has been designed. In this paper, a further modification of this protocol is used in designing an algorithm simulating a unit–cost RAM (for inputs of bounded size). This simulation shows that our model of amorphous computer does possess a universal computing power. The model represents an unusual instance of a non-uniform computational model since the choice of its parameters and the size of the network depends both on the space used by the standard computing (hence indirectly on the input size) but also on the computing time. To the best of our knowledge this is the first result of this kind.

A formal model of the amorphous computer is described in Section 2. In Section 3, for the sake of completeness, the asynchronous communication protocol from [6] is briefly presented followed by a new version of the broadcasting algorithm. The main result of the paper, i.e., an algorithm simulating a unit-cost RAM on our model of the amorphous computer, and its complexity analysis, is given in Section 4. Some useful properties of random graphs pertinent to our application are mentioned here as well. Section 5 is devoted to conclusions.

2 Model

In order to be able to prove universality of any model of computation we have to define this model quite rigorously: only then we can design plausible algorithms for it. To that end we give the definition of an amorphous computer as introduced in [6].

Definition 1. *An* **amorphous computer** *is a quintuple* $\mathcal{A} = (N, P, A, r, T)$ *where*

1. N *is the* number of processors *(also called* nodes*) in the underlying network. Each node is created by a RAM enhanced by a module for wireless sending and receiving. All nodes are identical, controlled by the same program, except of a single distinguished node called the* base station*. In addition to the standard node facilities (see below) this node is capable to send and receive data to/from a remote operator and is used to enter the data into the AC and to send the results of AC data processing to the operator.*
2. *Each RAM has a fixed number of registers holding numbers represented by* $O(\log N)$ *bits. Every RAM is equipped with a special read-only register called* rand, *a special read-only register* $\mathrm{r_{in}}$ *and a special write-only register* $\mathrm{r_{out}}$*. On each read, register* rand *delivers a new random number. The registers in all nodes are initialized by the same starting values.*
3. P *is a* random process *assigning to each node a position with continuous uniform distribution over a* planar area A*, independently for each node.*
4. r *gives the radius of a* communication neighborhood*. Any two nodes at distance at most* $r > 0$ *are called neighbors. All neighbors of a node form the* node's neighborhood.
5. $T > 0$ *is* transmission time *of a message within a neighborhood of any node.*

6. (Asynchronicity:*) In each RAM any instruction takes one unit of time. The actions (computations, communication) of all processors are not synchronized.*
7. *The nodes communicate according to the following rules:*
   - *all nodes broadcast on the same channel;*
   - *if a node writes a value representing a* message *to* $\mathrm{r}_{\mathrm{out}}$, *this message is broadcasted to its communication neighborhood;*
   - *if none of the given node's neighbors is broadcasting a message, then the given node register* r_{in} *contains an* empty message λ;
   - *if exactly one of a given node's neighbors is broadcasting a message* m, *then after time* T *register* r_{in} *in the given node contains* m;
   - *if two or more of the node's neighbors are broadcasting a message and the time intervals of broadcasting these message transfers overlap, then there is a so–called* collision *and the* r_{in} *register of the receiving node contains empty message* λ;
   - *the nodes have no means to detect a collision, i.e., to distinguish the case of no-broadcast from the case of a multiple broadcast.*

Note that since the register size of each RAM is bounded by $O(\log N)$ each RAM, inclusively its random number generator, can be seen as a probabilistic finite automaton of size polynomial in N (since each RAM has but a constant number of registers). That is, the size of automata grows with the size of the network. However, we have chosen to see each automaton as a "little RAM" since such a view will support the result we are after (i.e., a simulation of a unit-cost RAM) and corresponds more to practice.

An AC operates as follows. The input data enter the AC via its base station. From there, the data (which might also represent a program for the processors) spread to all nodes accessible via broadcasting. In a "real" AC additional data might also enter into individual processors via their sensors which, however, are not captured in our model since they do not influence the universality result. Then the data processing within processors and data exchange among processors begins. The results are delivered to the operator again via the base station.

3 Asynchronous Communication Protocol

In order to enable communication among all (or at least: a majority of) available processors the underlying communication graph of our AC must have certain desirable properties. The properties which are of importance in this case are: graph connectivity, graph diameter and the maximal degree of its nodes. Obviously, a good connectivity is a necessary condition in order to be able to harness a majority of all processors. Graph diameter bounds the length of the longest communication path. Finally, the node degree (i.e., the neighborhood size) determines the collision probability on the communication channel.

An instance of an amorphous computer $\mathcal{A}$ whose underlying computational graph has a maximal connected component of size N containing the base station is called a *well–formed instance* of $\mathcal{A}$ of size N.

Assuming that all nodes of an AC should participate in its computation there must exist a mechanism of node–to–node communication used by the nodes to coordinate their actions. Such a mechanism will consist of two levels. The lower level is given by a basic randomized broadcasting protocol enabling each node to broadcast a message to its neighborhood. Making use of this protocol we extend it, on the next level, to a broadcasting algorithm that can be used to broadcast a message from a given node to all other network nodes.

Protocol Send: A node is to send a message m with a given probability $\varepsilon > 0$ of failure. The protocol must work correctly under the assumption that all nodes are concurrently, asynchronously, in a non-coordinated way, using the same protocol, possibly interfering thus one with each other's broadcast.

The idea is for each node to broadcast sporadically, minimizing thus a communication collision probability in one node's neighborhood. This is realized as follows. Each node has a timer measuring *timeslots* (intervals) of length $2T$ (T is time to transfer a message between any two neighbors). During its own timeslot, each node is allowed either listen, or to send a message at the very beginning of its timeslot (and then listen till the end of this timeslot). Making use of its random number generator, a node keeps sending m at each start of the timeslot with probability p for k subsequent slots. The values of p and k are given in the proof of the following theorem. After performing the above algorithm each node waits for $2kT$ steps (so–called *safe delay*) before it can perform the next round of the protocol.

Theorem 1 (Sending a message). *Let $\mathcal{A}$ be a well–formed instance of an amorphous computer, let the underlying computational graph has the maximal neighborhood size bounded by Q. Let $1 > \varepsilon > 0$ be an a priori given allowable probability of failure. Assume that all nodes send their messages asynchronously according to the* Protocol Send. *Let X be a node sending message m and Y be any of X's neighbors. Then* Protocol Send *delivers m to Y in time $O(Q\log(1/\varepsilon))$ with probability at least $1 - \varepsilon$.*

Sketch of the proof: Thanks to our choice of the length of the timeslots, for each timeslot of a given node X there is exactly one corresponding timeslot of some other node Y such that if both nodes send asynchronously in their timeslots, only a single collision will occur. This is so because if X has started its sending at the beginning of its timeslot, X's and Y's sendings overlap if and only if Y had started a sending in a timeslot that was shifted w.r.t. the beginning of X's timeslot by less than T time units in either time direction. The timeslots of length shorter than $2T$ could cause more than a single broadcast collision between the arbitrary pairs of nodes, whereas longer timeslots would delay the communication.

We will treat message sendings as independent random events. Message m is correctly received by Y in one timeslot if X is transmitting m (the probability of such event is p) and none of Y's neighbors is transmitting (the corresponding probability is $(1-p)^Q$), giving the joint probability $p(1-p)^Q$. The value of $p(1-p)^Q$ is maximized for $p = 1/(Q+1)$. The probability of a failure after

k timeslots is $[1-p(1-p)^Q]^k = \varepsilon$. Hence, $k = \log \varepsilon / \log[1-p(1-p)^Q]$. The denominator in the latter expression equals $-\sum_{i=1}^{\infty}[p(1-p)^Q]^i/i \leq -p(1-p)^Q = -1/(Q+1)(1+1/Q)^{-Q} \leq -e^{-1}/(Q+1)$ leading to $k = O(Q\log(1/\varepsilon))$. □

In order to send a message to any node of an AC we use flooding, i.e, broadcasting the message to all nodes of the network.

Algorithm Broadcast. This algorithm is used to deliver a message m from a node X in the network to all remaining nodes which are in a *quiet state* w.r.t. some (previous) message $p \neq m$ (i.e., the nodes transmit any message except of p). To do so X sends m using *Protocol Send* with probability ε/N of failure and thereafter, it enters the quiet state w.r.t. m (by remembering m and rejecting its further transmission should it be received again). Upon receiving m, any other node which is not yet in the quiet state w.r.t. m also starts sending m using the previous procedure.

Theorem 2 (Broadcasting). *Let D denote the diameter of the communication graph. Then, for any $\varepsilon : N/2^{N/Q} \leq \varepsilon \leq 1$,* Algorithm Broadcast *delivers m to each node in time $O(DQ\log(N/\varepsilon))$ and with probability $1-\varepsilon$. Afterwards, all nodes will be in a quiet state w.r.t. m.*

Sketch of the proof: Starting in X, m spreads through the network as a random breadth-first search algorithm of the communication graph starting in X would do. This is because each node, after receiving and re-sending m via *Protocol Send* with probability ε/N of failure, stops re–sending of m. Obviously, after repeating this process at most D times, m will reach all nodes and all nodes will be in a quiet state w.r.t. m. The algorithm thus takes time $O(DQ\log(N/\varepsilon))$. For one node the failure probability is ε/N and for the whole network this probability will rise to ε. The boundary on ε follows from the bounded memory size of each processor. □

Note that the previous algorithm cannot process two identical messages sent one after the other.

4 The Universal Computing Power of an AC

We show the universal computing power of on AC by letting it simulate a unit–cost RAM. We will assume that the entire RAM program as well as two RAM accumulators are stored in the base station. The input data are provided by the operator upon request. The contents of the RAM registers will be held in the individual AC processors (assuming that the contents of any RAM register will fit any AC processor). Obviously, in order to be able to address the registers we must allocate an address to each of them.

The RAM program will consist of the usual kind of instructions. For simplicity we will assume that all instructions requiring two operands (indirect addressing, arithmetical operations) are realized in the following way. The first operand is assumed to be in the first accumulator. The second operand (if any) is to be delivered into the second accumulator. An instruction moving the register contents between a register and the accumulator is realized as follows. The base station broadcasts the current instruction holding the address of the register to which the instruction is pertinent to all nodes of the AC. The instruction is realized in the requested register (processor) which then sends back the confirmation along with the current contents of that register.

Thus, in order to the previous ideas to work we must ensure that:

- enough registers with the different addresses get allocated in the AC;
- at the same time, no two different messages occur in the net (since *Algorithm Broadcast* can handle only one message at a time).

Addresses Allocation: Assume that we need an address space of size $M > 0$. The following randomized algorithm makes use of $N = 2M$ processors in order to generate the addresses for at least M different registers with a high probability (the occurrence of registers with the same address does not harm).

Algorithm Generate_Addresses

```
1. All processors randomly generate and store a binary string of length $\lceil \log(2M+1) \rceil$ in a variable called $address$;
2. The base station initializes two variables, $round := 1$ and $max := 1$;
3. Using Algorithm Broadcast, the base broadcasts pair $(round, max)$ to all processors;
4. Upon receiving this message, each processor whose $address = round$ sends a confirmation — message "0" — back to the base and resets its $address$ to $max$;
5. If within the waiting period the base receives at least one confirmation, $max$ is increased by 1;
6. After the waiting time has elapsed, $round$ is increased by 1.
7. If $round < 2M$ then go to step 3;
8. If $max \geq M$ then HALT else go to step 1.
```

Clearly, variable *max* counts the number of processors with different addresses. Algorithm *Generate_Addresses* is repeated until at least M different addresses are generated. The probability that this happens already after the first trial is high and tends to 1 with the increasing M:

Lemma 1. *Choosing* $2M$ *random numbers uniformly distributed in interval* 1 *to* $2M$, *the probability that only* M *or less different numbers were chosen is less than* $1/\sqrt{M+1}$.

Proof: There are $(2M)^{2M}$ sequences of length $2M$ over $\{1, 2, \ldots, 2M\}$. Among them, there are M^{2M} sequences "made of" at most M different numbers which can be selected in $\binom{2M}{M}$ different ways. Hence the probability that a sequence contains M or less different numbers is $\binom{2M}{M} M^{2M}/(2M)^{2M}$. By induction, one can prove that $\binom{2M}{M} < 2^{2M}/\sqrt{M+1}$. The claim of the theorem follows. □

Simulation: Prior to the start of the simulation, using *Algorithm Broadcast* the base station sends the values of parameters k, Q, D, N and ε to all nodes in the network. These parameters are used by the nodes in order to fix the duration of the safe delay and that of the break (in *Protocol Send*). Then the simulation proceeds in rounds. In general, each round is performed with a different failure probability ε_t. In each round, the base station issues the instruction to be realized. The network is "flooded" by this instruction using *Algorithm Broadcast.* Upon arriving into any processor holding the respective register the instruction is realized. Subsequently, after making a *big break* (of duration $O(DQ\log(N/\varepsilon_t))$, a confirmation is broadcasted back to the base station, again by using the broadcast algorithm. The purpose of the big break is to allow all nodes to enter the quiet state.

In more detail, the simulation of the t-the instruction in the t-th round proceeds as follows.

Algorithm Simulate: At the beginning of the t–th round, we assume that the following invariant holds: with probability $1 - \sum_{i=1}^{t-1} \varepsilon_i$, each register represented in the processors of the AC holds the correct value (corresponding to the values of the RAM registers after realizing the first $t-1$ instructions) and that all processors are in a quiet state w.r.t. the lastly sent message.

In order to realize an instruction requiring a load from register i, or a store of value *contents* into register i, the base station broadcasts a quadruple of form $(t\ mod\ N, i, instr, contents)$, where t denotes the order of the instruction in the instruction sequence to be performed, *instr* is either *LOAD* or *STORE*. In the former case, *contents* is empty, whereas in the latter case *contents* holds the value to be stored in the i-th register. Upon arriving at any processor which is in a quiet state, the processor starts the re-sending of the quadruple using *Protocol Send.* Moreover, upon arriving at a register whose $address = i$, after the big break a load instruction makes the processor to broadcast a confirmation triple of a form $(t\ mod\ N, LOAD, reg[i])$ where $reg[i]$ is the contents of the i-th register. A store instruction is realized by performing the assignment $reg[i] := contents$ within the processor and again, after the big break, by broadcasting a confirmation triple of a form $(t\ mod\ N, STORE, empty)$. The base station also allows to elapse the big break within which it should receive a confirmation of the t-th instruction realization. If during the big break no confirmation is obtained, the base station allows for another big break. Of course, with a small probability it may happen that the base station, expecting a confirmation at time t, receives a confirmation issued at some earlier time $t_1 < t$. Such a confirmation is neglected by the base station. This, of course, does not mean that the time stamps in the confirmations are superfluous: the base station has always wait until the confirmation of the current instruction reaching its goal is obtained. Without the time stamps, a confirmation from an earlier instruction could cause the base station to issue the next instruction what could lead to an incorrect simulation. Note that due to the counting modulo N some care is necessary when comparing the values of time stamps.

By that time, the t-th instruction has been carried out with probability $1 - \varepsilon_t$, where ε_t is the failure probability of protocol *Broadcast.* With the same probability, the base station has obtained the respective confirmation. Moreover, again with probability $1 - \varepsilon_t$, the network nodes are in a quiet state w.r.t. the lastly sent message. This means that the invariant has been re-established and the simulation of the next, $t + 1$-st instruction can start. □

The previous algorithm consists of a series of $T(n)$ rounds, where $T(n)$ denotes the time complexity of the original RAM algorithm. If each round will fail with the same probability $O(\varepsilon_1)$, the entire simulation will fail with probability $O(\varepsilon_1 T(n))$. Thus, choosing $\varepsilon_1 = O(\varepsilon/T(n))$ leads to a simulation algorithm with the probability of failure at most ε.

If $T(n)$ is not known in advance we perform each next simulation round with a decreased probability of failure. If the t-th round is realized with the probability of error ε/t^2, the probability of an error after performing $T(n)$ rounds is bounded by $\min\{1, \sum_{t=1}^{T(n)} \varepsilon/t^2\} < \min\{1, \sum_{t=1}^{\infty} \varepsilon/t^2\} = \min\{1, \varepsilon\pi^2/6\}$.

Assume now that we want to simulate with failure probability $\varepsilon > 0$ a RAM $\mathcal{R}$ of time complexity $T(n)$, of space complexity $S(n)$, with register size $c \log T(n)$ for some $c > 0$. Suppose that our simulating AC $\mathcal{A}$ has $s \geq 2$ registers of size $O(\log N)$ available in each processor for computing and storing the value $DQ \log(Nt^2/\varepsilon)$ of the big break, for $1 \leq t \leq T(n)$, and for storing each RAM register contents. For this to happen it is enough that $NQ \log(NT^2(n)/\varepsilon) + c \log T(n) \leq N^s$ leading to $\varepsilon \geq NT^2(n)/2^{2^{N^{s-1}/Q}}$. Since $s \geq 2$ and $c \log T(n) = O(\log N)$, for sufficiently large n's there is enough room in AC processors to compute with any given $\varepsilon > 0$.

Putting all the results together we see that in order to simulate $\mathcal{R}$ for an input of size n, we must first "set up" a well-formed instance of an amorphous computer with at least $max\{2S(n), N^{O(1)}\}$ processors. Since the time complexity of *Broadcast* in the t–th round is $O(DQ \log(Nt^2/\varepsilon))$ and $\sum_{t=1}^{T(n)} \log(Nt^2/\varepsilon))) \leq T(n) \log(NT^2(n)/\varepsilon)$, for the complexity of our simulation algorithm we get the following result:

Theorem 3 (Simulation). *Under the previous notation and assumptions, for any $\varepsilon > 0$ and sufficiently large inputs of size n there exists an AC $\mathcal{A}$ simulating $\mathcal{R}$ in time $O(T(n)DQ \log(NT^2(n)/\varepsilon))$, with probability of failure at most $\min\{\varepsilon\pi^2/6, 1\}$.*

In [6], the maximal connected component size, the diameter, and the maximal neighborhood size of a communication graph of an amorphous computer have been studied using both experiments and combinatorial tools. The respective results show that under reasonable statistical assumptions chances are high that an amorphous computer with good expected properties will emerge. For instance, it has been experimentally verified that for an AC spread over a square A with density $d = 6$, where the density is defined as $d = N\pi r^2/a$ (a denotes the size of area A), one can expect that with a high probability, a connected component containing $0.48N$ processors will arise. Furthermore, for such areas it has been experimentally verified that the diameter of the resulting graph is bounded by $O(\sqrt{N})$. Finally, it has been analytically shown that the maximal neighborhood size in such graphs is $O(\log N/ \log\log N)$. Interestingly, it is not known whether the first two results can also be derived analytically (for references, cf. [6]).

Using these results one can derive a corollary of the previous theorem stating the efficiency of our RAM–simulation for the case when the previous estimates of random graph parameters hold as $O(T(n)\log(NT^2(n)/\varepsilon)\sqrt{N}\log N/\log\log N))$. More research is needed in order to better understand important properties of random graph underlying amorphous computers.

5 Conclusion

The main departure point of the amorphous computing structures from other models of wireless networks or distributed computing is the randomness of the underlying network topology, anonymity of processors not possessing universal computing power and a principal lack of synchronicity combined with the impossibility to detect broadcasting collisions. Unlike the majority of the known models which work whenever appropriately programmed, this need not be the case with an amorphous computer since its nodes can be dispersed in an unlucky manner that does not support the computer's functionality. For our model we have designed and analyzed a probabilistic algorithm simulating a unit-cost RAM. To the best of our knowledge, our simulation algorithm seems to be the first result showing the universal computing power of a non–uniform family of amorphous computers of the type we have considered.

References

1. Abelson, H., Allen, D., Coore, Ch., Hanson, G., Homsy, T.F., Knight Jr, R., Nagpal, E., Rauch, G.J., Sussman, R.: Weiss.: Amorphous Computing. Communications of the ACM 43(5), 74–82 (2000)
2. Abelson, H., et al.: Amorphous Computing. MIT Artificial Intelligence Laboratory Memo No. 1665, August (1999)
3. Coore, D.: Introduction to Amorphous Computing. In: Banâtre, J.-P., Fradet, P., Giavitto, J.-L., Michel, O. (eds.) UPP 2004. LNCS, vol. 3566, pp. 99–109. Springer, Heidelberg (2005)
4. D'Hondt, E.: Exploring the Amorphous Computing Paradigm. Master's Thesis, Vrije University (2000)
5. Nikoletseas, S.: Models and Algortihms for Wireless Sensor Networks (Smart Dust). In: Wiedermann, J. (ed.) SOFSEM 2006. LNCS, vol. 3831, pp. 65–83. Springer, Heidelberg (2007)
6. Petru, L., Wiedermann, J.: A Model of an Amorphous Computer and its Communication Protocol. In: Proc. SOFSEM'07. LNCS, vol. 4362, Springer, Berlin (2007)
7. Spirakis, P.G.: Algorithmic and Foundational Aspects of Sensor Systems (Invited Talk). In: Nikoletseas, S.E., Rolim, J.D.P. (eds.) ALGOSENSORS 2004. LNCS, vol. 3121, pp. 3–8. Springer, Heidelberg (2004)
8. Warneke, B., et al.: Smart dust: communicating with a cubic-millimeter computer. Computer 34(1), 44–51 (2001)

The Complexity of Small Universal Turing Machines

Damien Woods[1] and Turlough Neary[2]

[1] Department of Computer Science,
University College Cork, Ireland
d.woods@cs.ucc.ie
[2] TASS, Department of Computer Science,
National University of Ireland Maynooth, Ireland
tneary@cs.may.ie

Abstract. We survey some work concerned with small universal Turing machines, cellular automata, and other simple models of computation. For example it has been an open question for some time as to whether the smallest known universal Turing machines of Minsky, Rogozhin, Baiocchi and Kudlek are efficient (polynomial time) simulators of Turing machines. These are some of the most intuitively simple computational devices and previously the best known simulations were exponentially slow. We discuss recent work that shows that these machines are indeed efficient simulators. As a related result we also find that Rule 110, a well-known elementary cellular automaton, is also efficiently universal. We also mention some new universal program-size results, including new small universal Turing machines and new semi-weakly universal Turing machines. We then discuss some ideas for future work arising out of these, and other, results.

1 Introduction

Shannon [38] was the first to consider the question of finding the smallest possible universal Turing machine, where size is the number of states and symbols. In the early sixties Minsky and Watanabe had a running competition to see who could come up with the smallest universal Turing machine [20,21,40,41]. Early attempts [9,41] gave small universal Turing machines that efficiently (in polynomial time) simulated Turing machines. In 1962, Minsky [21] found a small 7-state, 4-symbol universal Turing machine. Minsky's machine worked by simulating 2-tag systems, which where shown to be universal by Cocke and Minsky [3]. Rogozhin [35] extended Minsky's technique of 2-tag simulation and found small machines with a number of state-symbol pairs. Subsequently, some of Rogozhin's machines were reduced in size or improved by Robinson [34], Rogozhin [36], Kudlek and Rogozhin [11], and Baiocchi [2]. All of the smallest known 2-tag simulators are plotted as circles in Figure 1.

Unfortunately, Cocke and Minsky's 2-tag simulation of Turing machines was exponentially slow. The exponential slowdown was essentially caused by the use

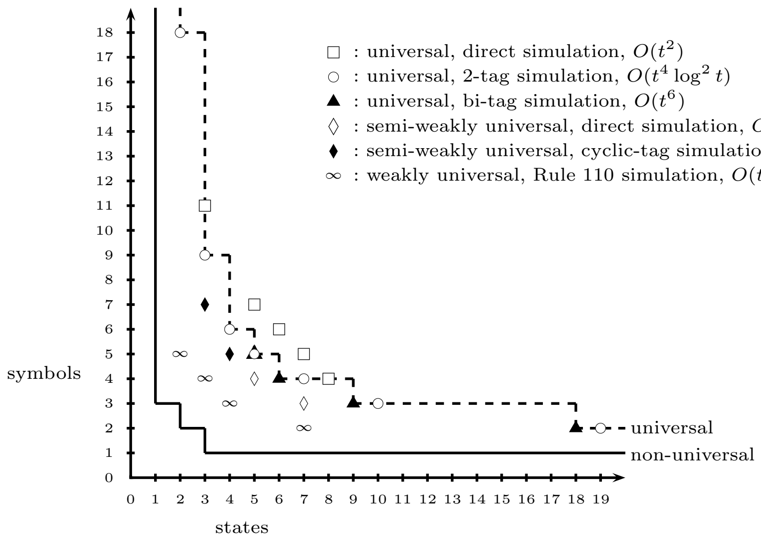


Fig. 1. State-symbol plot of small universal Turing machines. The type of simulation is given for each group of machines. Also we give the simulation overheads in terms of simulating a single tape, deterministic Turing machine that runs in time t.

of a unary encoding of Turing machine tape contents. Therefore, for many years it was entirely plausible that there was an exponential trade-off between program size complexity on the one hand, and time/space complexity on the other; the smallest universal Turing machines seemed to be exponentially slow.

Figure 1 shows a non-universal curve. This curve is a lower bound that gives the state-symbol pairs for which it is known that we cannot find a universal machine. The 1-symbol case is trivial, and the 1-state case was shown by Shannon [38] and, by using another method, Hermann [8]. Pavlotskaya [30] and, via another method, Kudlek [10] have shown that there are no universal 2-state, 2-symbol machines, where one transition rule is reserved for halting. Pavlotskaya [31] has also shown that there are no universal 2-state, 3-symbol machines, and also claimed [30], without proof, there are no universal machines for the 3-state, 2-symbol case. Again, both of these cases assume that a transition rule is reserved for halting.

2 Time Efficiency of Small Machines

As mentioned above, some of the very earliest small Turing machines were polynomial time simulators. Subsequently attention turned to the smaller, but exponentially slower, 2-tag simulators given by Minsky, Rogozhin and others. Recently [28] we have given small machines that are efficient polynomial time simulators. More precisely, if M is a deterministic single-tape Turing machine that runs in time t, then there are machines, with state-symbol pairs given by the squares in Figure 1, that directly simulate M in polynomial time $O(t^2)$. These machines define a $O(t^2)$ curve. They are currently the smallest known universal Turing machines that (a) simulate Turing machines directly and (b) simulate in $O(t^2)$ time.

Given these efficient $O(t^2)$ simulators it still remained the case that the smallest machines were exponentially slow. However we have recently shown [45] that 2-tag systems are in fact efficient simulators of Turing machines. More precisely, if M is a deterministic single-tape Turing machine that runs in time t then there is a 2-tag system that simulates M and runs in polynomial time $O(t^4 \log^2 t)$. The small machines of Minsky, Rogozhin, and others have a quadratic time overhead, when simulating 2-tag systems, hence by the result in [45] they simulate Turing machines in time $O(t^8 \log^4 t)$. It turns out that the time overhead can be improved to $O(t^4 \log^2 t)$, giving the $O(t^4 \log^2 t)$ machines in Figure 1 (this improvement is as yet unpublished). Thus, there is currently little evidence for the claim of an exponential trade-off between program size complexity, and time/space complexity.

In a further improvement Neary and Woods [24,25] have given four Turing machines that are presently the smallest known machines with 2, 3, 4 and 5 symbols. The 5-symbol machine improves on the 5-symbol machine of Rogozhin [36] by one transition rule. The remainder of these machines improve on the 2- and 4-symbol machines of Baiocchi [2], and the 3-symbol machine of Rogozhin [36], by one state each. They simulate our universal variant of tag systems called a *bi-tag system* [26]. These small machines simulate Turing machines in polynomial time $O(t^6)$ and are illustrated as triangles in Figure 1. Bi-tag systems are essentially 1-tag systems (and so they read and delete one symbol per timestep) augmented with additional context sensitive rules that read, and delete, two symbols per timestep. On the one hand bi-tag systems are universal, while on the other hand they are sufficiently 'simple' to be simulated by such small machines.

Improving the time efficiency of 2-tag systems has implications for a number of models of computation, besides small universal Turing machines. Following our result, the simulation efficiency of many biologically inspired models of computation, including neural networks, H systems and P systems, has been improved from exponential to polynomial. For example, Siegelmann and Margenstern [39] give a neural network that uses only nine high-order neurons to simulate 2-tag systems. Taking each synchronous update of the nine neurons as a single parallel timestep, their neural network simulates 2-tag systems in linear time. They note that "tag systems suffer a significant slow-down ... and thus our result proves only Turing universality and should not be interpreted complexity-wise as a Turing equivalent." Our work shows that their neural network is in fact efficiently universal. Rogozhin and Verlan [37] give a tissue P system with eight rules that simulates 2-tag systems in linear time, and thus we have improved its simulation time overhead from exponential to polynomial. This system uses splicing rules (from H systems) with membranes (from P systems) and is non-deterministic. Harju and Margenstern [7] gave an extended H-system with 280 rules that generates recursively enumerable sets using Rogozhin's 7-state, 4-symbol universal Turing machine. Using our result from 2-tag systems, the time efficiency of their construction is improved from exponential to polynomial, with a possible small constant increase in the number of rules. The technique of simulation via 2-tag systems is at the core of many of the universality proofs in Margenstern's survey [16]. Our work exponentially improves the time overheads in these simulations, such as Lindgren and Nordahl's cellular automata [12], Margenstern's non-erasing Turing machines [13,14], and Robinson's tiling [33].

3 Weak Universality and Rule 110

So far we have been discussing results for universal Turing machines that have one tape, one tape head, and are deterministic. Of course one can consider results for other variants of the model [32]. An interesting case is when we stick to the above conventions, but we allow the blank portion of the tape to contain a word, that is constant (independent of the input), and is repeated infinitely often in one direction, say to the left of the input. We call such universal Turing machines *semi-weakly universal*. Some of the earliest small universal Turing machines were semi-weak [41,42]. Sometimes another word is also repeated infinitely often to the right. Universal machines that use this kind of encoding are called *weakly universal* [17].

It is not difficult to see how this generalisation can help to reduce program size. For example, it is typical of small universal Turing machine computations that the program being simulated is stored on the tape. When reading an instruction we often mark certain symbols. At a later time we then restore marked symbols to their original values. If the simulated program is repeated infinitely often, say to the left of the input, things may be much easier as we can simply skip the 'restore' phase of our algorithm and access a new copy of the program when simulating the next instruction, thus reducing the universal program's size.

This was the strategy used by Watanabe [41,42] to find the semi-weak, direct Turing machine simulators shown in Figure 1 as hollow diamonds. Recently [44] we have given two new semi-weakly universal machines and these are shown as solid diamonds in Figure 1. These machines simulate cyclic tag systems, which were first used by Cook [5] to show that Rule 110 is universal. It is interesting to note that our machines are symmetric with those of Watanabe, despite the fact that we use a different simulation technique. Our 4-state, 5-symbol machine has only 17 transition rules, making it the smallest known semi-weakly universal machine (Watanabe's 5-state, 4-symbol machine has 18 transition rules). The time overhead for our machines is polynomial. More precisely, if M is a single-tape deterministic Turing machine that runs in time t, then M is simulated by either of our semi-weak machines in time $O(t^4 \log^2 t)$.

Matthew Cook [5] dramatically changed the situation in Figure 1 by finding weakly universal Turing machines that are substantially smaller than those found to date. Cook's machines work by simulating Rule 110, a very simple kind of cellular automaton. Rule 110 is an elementary cellular automaton, which means that it is a one-dimensional, nearest neighbour, binary cellular automaton [43]. More precisely, it is composed of a sequence of cells $\ldots p_{-1}p_0p_1\ldots$ where each cell has a binary state $p_i \in \{0,1\}$. At timestep $t+1$ the value of cell $p_{i,t+1} = F(p_{i-1,t}, p_{i,t}, p_{i+1,t})$ is given by the synchronous local update function F

$$\begin{aligned} F(0,0,0) &= 0 & \qquad F(1,0,0) &= 0 \\ F(0,0,1) &= 1 & \qquad F(1,0,1) &= 1 \\ F(0,1,0) &= 1 & \qquad F(1,1,0) &= 1 \\ F(0,1,1) &= 1 & \qquad F(1,1,1) &= 0 \end{aligned}$$

Cook showed that Rule 110 is universal via an impressive simulation. Since Rule 110 is so ‘simple’ Cook was able to write very small, weakly universal machines that simulate it, these are illustrated as ∞ symbols in Figure 1. The use of weak universality here is very important, since (a) Cook encodes a (possibly universal) program in one of these repeated words and (b) it is known from a result of Codd [4] that one-dimensional elementary cellular automata (e.g. Rule 110) on finite initial configurations are not universal.

Rule 110 was shown to be universal by simulating Cook's cyclic tag systems, which in turn simulate 2-tag systems. The chain of simulations included the exponentially slow 2-tag algorithm of Cocke and Minsky, thus Rule 110, and Cook's universal machines, were exponentially slow. In a recent paper [27] we have improved their simulation time overhead to polynomial by showing that cyclic tag systems are efficient simulators of Turing machines. This result has interesting implications for Rule 110. For example, given an initial configuration of Rule 110, and a value t given in unary, predicting t timesteps of a Rule 110 computation is P-complete. Therefore, unless P = NC, which is widely believed to be false, we cannot hope to quickly (in polylogarithmic time) predict the evolution of this simple cellular automaton even if we have a polynomial amount of parallel hardware. The question of whether Rule 110 prediction is P-complete has been asked, either directly or indirectly, in a number of previous works (for example [1,22,23]). Rule 110 is the simplest (one-dimensional, nearest neighbour) cellular automaton that has been shown to have a P-complete prediction problem. In particular Ollinger's [29] intrinsic universality result shows that prediction for one dimensional nearest neighbour cellular automata is P-complete for six states, and our result improves this to two states.

It is currently not known if all of the lower bounds in Figure 1 hold for machines that are of the style of Cook's. For example, the non-universality results of Pavlotskaya were proven for the case where one transition rule is reserved for halting, whereas Cook's machines do not halt.

4 Further Work

There are many avenues for further work in this area, here we highlight a few examples.

Applying computational complexity theory to the area of small universal Turing machines allows us to ask a number of questions that are more subtle than the usual questions about program size. As we move towards the origin, the universal machines have larger (but polynomial) time overheads. Can the time overheads in Figure 1 be further improved (lowered)? Can we prove lower bounds on the simulation time of machines with a given state-symbol pair? Proving non-trivial simulation time lower bounds seems like a difficult problem. Such results could be used to prove that there is a polynomial trade-off between simulation time and universal program size.

As we move away from the origin, the non-universal machines seem to have more power. For example Kudlek's classification of 2-state, 2-symbol machines shows that the sets accepted by these machines are regular, with the exception of one context free language ($a^n b^n$). Can we hope to fully characterise the sets accepted by non-universal machines (e.g. in terms of complexity or automata theoretic classes) with given state-symbol pairs or other program restrictions?

When discussing the complexity of small machines the issue of encodings becomes very important. For example, when proving that the prediction problem for a small machine is P-complete [6] then the relevant encodings should be in logspace, and this is the case for all of the polynomial time machines in Figure 1.

For results on more general definitions of small universal Turing machines (higher dimensional tapes, more tape heads, etc.), see for example Priese [32] and the references therein. Margenstern and Pavlotskaya show tight lower and upper bounds for universality on other variants on the Turing machine model including non-erasing Turing machines [13,14,15] and Turing machines coupled with a finite automaton [18].

The space between the non-universal curve and the smallest non-weakly universal machines in Figure 1 contains some complicated beasts. These lend weight to the feeling that finding new lower bounds on universal program size is tricky. Most noteworthy are the weakly and semi-weakly universal machines discussed in the previous section. Also of importance are the small machines of Margenstern [16] and Michel [19] that live in this region and simulate iterations of the $3x + 1$ problem. So it seems that there are plenty of animals yet to be tamed.

Acknowledgements

We thank Olivier Bournez and Paola Bonizzoni for the invitation to speak at CiE 2007. During this work the authors were partially funded by the Irish Research Council for Science, Engineering and Technology.

References

1. Aaronson, S.: Book review: A new kind of science. Quantum Information and Computation 2(5), 410–423 (2002)
2. Baiocchi, C.: Three small universal Turing machines. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 1–10. Springer, Heidelberg (2001)
3. Cocke, J., Minsky, M.: Universality of tag systems with $P = 2$. Journal of the Association for Computing Machinery 11(1), 15–20 (1964)
4. Codd, E.: Cellular Automata. Academic Press, New York (1968)

5. Cook, M.: Universality in elementary cellular automata. Complex Systems 15(1), 1–40 (2004)
6. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to parallel computation: P-completeness theory. Oxford university Press, Oxford (1995)
7. Harju, T., Margenstern, M.: Splicing systems for universal Turing machines. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA Computing. LNCS, vol. 3384, pp. 149–158. Springer, Heidelberg (2005)
8. Hermann, G.T.: The uniform halting problem for generalized one state Turing machines. In: Proceedings of the ninth annual Symposium on Switching and Automata Theory (FOCS), Oct. 1968, pp. 368–372. IEEE Computer Society Press, Schenectady, New York (1968)
9. Ikeno, N.: A 6-symbol 10-state universal Turing machine. In: Proceedings, Institute of Electrical Communications, Tokyo (1958)
10. Kudlek, M.: Small deterministic Turing machines. Theoretical Computer Science 168(2), 241–255 (1996)
11. Kudlek, M., Rogozhin, Y.: A universal Turing machine with 3 states and 9 symbols. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 311–318. Springer, Heidelberg (2002)
12. Lindgren, K., Nordahl, M.G.: Universal computation in simple one-dimensional cellular automata. Complex Systems 4(3), 299–318 (1990)
13. Margenstern, M.: Non-erasing Turing machines: A frontier between a decidable halting problem and universality. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 375–385. Springer, Heidelberg (1993)
14. Margenstern, M.: Non-erasing Turing machines: a new frontier between a decidable halting problem and universality. In: Baeza-Yates, R.A., Poblete, P.V., Goles, E. (eds.) LATIN 1995. LNCS, vol. 911, pp. 386–397. Springer, Heidelberg (1995)
15. Margenstern, M.: The laterality problem for non-erasing Turing machines on $\{0, 1\}$ is completely solved. Theoretical Informatics and Applications 31(2), 159–204 (1997)
16. Margenstern, M.: Frontier between decidability and undecidability: a survey. Theoretical Computer Science 231(2), 217–251 (2000)
17. Margenstern, M.: An algorithm for building intrinsically universal cellular automata in hyperbolic spaces. In: Proceedings of the 2006 International Conference on Foundations of Computer Science (FCS), pp. 3–9. CSREA Press, Las Vegas, NV (2006)
18. Margenstern, M., Pavlotskaya, L.: On the optimal number of instructions for universality of Turing machines connected with a finite automaton. International Journal of Algebra and Computation 13(2), 133–202 (2003)
19. Michel, P.: Small Turing machines and generalized busy beaver competition. Theoretical Computer Science 326, 45–56 (2004)
20. Minsky, M.: A 6-symbol 7-state universal Turing machines. Technical Report 54-G-027, MIT, Aug (1960)
21. Minsky, M.: Size and structure of universal Turing machines using tag systems. In: Recursive Function Theory: Proceedings, Symposium in Pure. Mathematics, Provelence, AMS vol. 5, pp. 229–238 (1962)
22. Moore, C.: Quasi-linear cellular automata. Physica D 103, 100–132 (1997)
23. Moore, C.: Predicting non-linear cellular automata quickly by decomposing them into linear ones. Physica D 111, 27–41 (1998)
24. Neary, T.: Small polynomial time universal Turing machines. In: Fourth Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT'06), pp. 325–329. University College Cork, Ireland (2006)

25. Neary, T., Woods, D.: Four small universal Turing machines. In: Machines, computations and universality(MCU) 2007, Sept. Springer LNCS. (Accepted)
26. Neary, T., Woods, D.: A small fast universal Turing machine. Technical Report NUIM-CS-TR-2005-12, Department of Computer Science, NUI Maynooth (2005)
27. Neary, T., Woods, D.: P-completeness of cellular automaton Rule 110. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 132–143. Springer, Heidelberg (2006)
28. Neary, T., Woods, D.: Small fast universal Turing machines. Theoretical Computer Science 362(1–3), 171–195 (2006)
29. Ollinger, N.: The quest for small universal cellular automata. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 318–329. Springer, Heidelberg (2002)
30. Pavlotskaya, L.: Solvability of the halting problem for certain classes of Turing machines. Mathematical Notes (Springer), vol. 13(6) pp. 537–541, June 1973 (Translated from Matematicheskie Zametki, Vol. 13, No. 6, pp. 899–909, June 1973)
31. Pavlotskaya, L.: Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. Avtomaty i Mashiny, pp. 91–118 (Sufficient conditions for the halting problem decidability of Turing machines) (in Russian) (1978)
32. Priese, L.: Towards a precise characterization of the complexity of universal and nonuniversal Turing machines. SIAM J. Comput. 8(4), 508–523 (1979)
33. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. Inventiones Mathematicae 12(3), 177–209 (1971)
34. Robinson, R.M.: Minsky's small universal Turing machine. International Journal of Mathematics 2(5), 551–562 (1991)
35. Rogozhin, Y.: Sem' universal'nykh mashin T'juringa. Systems and theoretical programming, Mat. Issled (Seven universal Turing machines. In Russian) 69, 76–90 (1982)
36. Rogozhin, Y.: Small universal Turing machines. Theoretical Computer Science 168(2), 215–240 (1996)
37. Rogozhin, Y., Verlan, S.: On the rule complexity of universal tissue P systems. In: Freund, R. (ed.) WMC 2005. LNCS, vol. 3850, pp. 356–362. Springer, Heidelberg (2005)
38. Shannon, C.E.: A universal Turing machine with two internal states. Automata Studies, Annals of Mathematics Studies 34, 157–165 (1956)
39. Siegelmann, H.T., Margenstern, M.: Nine switch-affine neurons suffice for Turing universality. Neural Networks 12(4–5), 593–600 (1999)
40. Watanabe, S.: On a minimal universal Turing machines. Technical report, MCB Report, Tokyo, Aug (1960)
41. Watanabe, S.: 5-symbol 8-state and 5-symbol 6-state universal Turing machines. Journal of the ACM 8(4), 476–483 (1961)
42. Watanabe, S.: 4-symbol 5-state universal Turing machines. Information Processing Society of Japan Magazine 13(9), 588–592 (1972)
43. Wolfram, S.: Statistical mechanics of cellular automata. Reviews of Modern Physics 55(3), 601–644 (1983)
44. Woods, D., Neary, T.: Small semi-weakly universal Turing machines. In: Machines, computations and universality(MCU) 2007, Sept. Springer LNCS. (Accepted)
45. Woods, D., Neary, T.: On the time complexity of 2-tag systems and small universal Turing machines. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Oct, pp. 439–446. IEEE, Berkeley, California (2006)

Approximating Generalized Multicut on Trees

Peng Zhang[⋆]

State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, P.O.Box 8718, Beijing, 100080, China
Graduate University of Chinese Academy of Sciences, Beijing, 100049, China
zhangpeng04@iscas.cn

Abstract. Given a tree T with costs on edges and a collection of terminal sets $X = \{S_1, S_2, \ldots, S_l\}$, the generalized Multicut problem asks to find a set of edges on T whose removal cuts every terminal set in X, such that the total cost of the edges is minimized. The standard version of the problem can be approximately solved by reducing it to the classical Multicut on trees problem. For the prize-collecting version of the problem, we give a primal-dual 3-approximation algorithm and a randomized 2.55-approximation algorithm (the latter can be derandomized). For the k-version of the problem, we show an interesting relation between the problem and the Densest k-Subgraph problem, implying that approximating the k-version of the problem within $O(n^{1/6-\epsilon})$ for some small $\epsilon > 0$ is hard. We also give a $\min\{2(l-k+1), k\}$-approximation algorithm for the k-version of the problem via a nonuniform approach.

Keywords: Generalized Multicut, Tree, Approximation Algorithm, Combinatorial Optimization.

1 Introduction

The cut problem is a classical and active topic in combinatorial optimization. We propose and study the problem *generalized Multicut on trees*. Given an undirected graph $G = (V, E)$ with costs on edges and a collection of terminal sets $X = \{S_1, S_2, \ldots, S_l\}$, the generalized Multicut problem (GMC for short) asks to find a set of edges in E whose removal disconnects every terminal set in X, such that the total cost of the picked edges is minimized. Denote by GMC(G) the problem on general (undirected) graphs, and by GMC(T) the problem on trees. The generalized Multicut problem is of its own interest since the problem is a natural generalization of the classical Multicut problem, and also is the dual of the generalized Steiner Forest problem.

The GMC problem is a natural generalization of the classical Multicut problem. If in the instance of the GMC problem every terminal sets is restricted to be of cardinality 2, then the problem is just the classical Multicut problem.

[⋆] Supported by NSFC grants No. 60325206 and No. 60310213. This work is part of the author's Ph.D. thesis prepared at Institute of Software, Chinese Academy of Sciences under the supervision of Prof. Angsheng Li.

The GMC problem generalizes the Multiway Cut problem too, since if l is restricted to be 1 then the problem is reduced to the Multiway Cut problem. On the other hand, the GMC problem is the dual of the generalized Steiner Forest problem (GSF). The instance of the GSF problem is the same as that of the GMC problem, but the goal is to connect every terminal set in X by edges with minimized total cost. Obviously, the optimal solution to GSF is a forest, while the optimal solution to GMC is just a set of edges, which destroys any subgraph interconnecting any terminal sets.

1.1 Related Work

For the problem Multicut on trees (MC(T)), Garg, Vazirani and Yannakakis [4] gave a primal-dual approximation algorithm (which is referred to as the GVY algorithm in this paper) with factor 2. This is also the best known performance ratio for MC(T). Recently, Levin and Segev [8] considered the problem prize-collecting Multicut on trees (pc-MC(T)) and the problem k-Multicut on trees (k-MC(T)). They reduced pc-MC(T) to MC(T) and proved that the GVY algorithm for pc-MC(T) possesses the so-called *Lagrangian Multiplier Preserving* [7] property, and thus gave a $\frac{8}{3} + \epsilon$-approximation for k-MC(T) for arbitrarily small $\epsilon > 0$ via Lagrangian relaxation. The same result for k-MC(T) was also obtained independently by Golovin, Nagarajan and Singh [5].

For the problem Multicut on general undirected graphs (MC(G)), Garg, Vazirani and Yannakakis [3] gave an $O(\log l)$-approximation algorithm through the region growth based LP-rounding technique. Later, Avidor and Langberg [1] extended the region growth technique to deal with the GMC(G) problem, and obtained the same performance ratio $O(\log l)$ for GMC(G).

1.2 Our Results

We consider the problem generalized Multicut on trees and its prize-collecting variant (pc-GMC(T)) and k-version variant (k-GMC(T)). Recall that by the results of Garg, Vazirani and Yannakakis [4] the problem Multicut on trees can be approximated within factor 2. Garg et al. [4] proved that MC(T) is **NP**-hard and **MAX SNP**-hard even for trees of height 1 and unit costs on edges. This implies the same hardness results for GMC(T). The GMC(T) problem can be easily reduced to MC(T) by the following way: Given the instance of GMC(T), for each terminal set $S_i \in X$, just $\binom{|S_i|}{2}$ possible terminal pairs are added to the reduced instance. Obviously this reduction can be finished in polynomial time. Then by the results of [4], GMC(T) admits 2-approximation.

Proposition 1. *The problem generalized Multicut on trees is both* **NP***-hard and* **MAX SNP***-hard.* □

Proposition 2. *There is a polynomial time 2-approximation algorithm for the problem generalized Multicut on trees.* □

In pc-GMC(T), every terminal set S_i is either cut by the picked edges or not cut but has to pay for the penalty π_i. Note that only if all terminals in S_i are separated from each other, we say that the terminal set S_i is cut. The goal of pc-GMC(T) is to minimize the sum of the total cost of picked edges and the penalties of uncut terminal sets. We extend the GVY algorithm to deal with pc-GMC(T), resulting in a primal-dual 3-approximation algorithm for pc-GMC(T). The main difference of our algorithm from the GVY algorithm is that our algorithm may raise many dual variables simultaneously, whereas the GVY algorithm raises each dual variable individually. We also give a randomized 2.55-approximation algorithm for pc-GMC(T), which can be derandomized and thus the final algorithm is deterministic. The randomized algorithm needs to solve linear program, whereas the primal-dual algorithm does not and is purely combinatorial.

The k-GMC(T) problem asks to find a solution with minimized cost to cut at least k terminal sets in X. Könemann, Parekh and Segev [10] gave a unified framework based on Lagrangian relaxation to approximate a class of Set Cover-like k-version optimization problems. Although the k-MC(T) problem can be viewed as a Set Cover-like problem and thus can be casted in the framework of [10], we find that the k-GMC(T) problem can not be solved by their framework since if one converts k-GMC(T) to Set Cover then the set family is of exponential size in l. We give a polynomial time $\min\{2(l-k+1), k\}$-approximation algorithm for k-GMC(T) via a nonuniform approach. The performance ratio $\min\{2(l-k+1), k\}$ behaves well when k is near to 1 or l. we also find an interesting relation between k-GMC(T) and the Densest k-Subgraph problem, implying that approximating k-GMC(T) within factor $O(n^{1/6-\epsilon})$ for some small $\epsilon > 0$ is hard.

2 Prize-Collecting Generalized Multicut on Trees

2.1 The Linear Program Formulations

First we give the fractional linear program formulation for the GMC(T) problem, as shown in (LP_s) (where the index s means the standard version of the problem).

$$(LP_s) \qquad \min \sum_{e\in E} c_e x_e \tag{1}$$

subject to

$$\sum_{e\in p} x_e \geq 1, \qquad \forall i \in [l], \forall p \in P_i \tag{2}$$

$$x_e \geq 0, \qquad \forall e \in E$$

To better understand (LP_s), consider its integral version. The variable x_e is defined for every edge $e \in E$ to indicate whether edge e is picked in the solution. For simplicity, we use the notation $[l]$ to denote the set $\{1, 2, \ldots, l\}$. For every terminal set S_i, there are $\binom{|S_i|}{2}$ possible terminal pairs in total. Since GMC(T) is defined on tree, for each terminal pair there is a unique path between them. For a terminal pair (t, t') in S_i, denote by $[t, t']$ the unique path between t and t'. The notation P_i in constraint (2) denotes the set of unique paths for every possible terminal pairs in S_i. Then constraint (2) states that for every terminal set S_i and for every path in P_i there is at least one edge e with $x_e = 1$ (and hence S_i is cut by edges $\{e \in E \colon x_e = 1\}$). Since the objective function (1) is to minimize the total cost of picked edges, an integral optimal solution to (LP_s) is just an optimal solution to GMC(T), and vice versa.

The fractional linear program for pc-GMC(T) can be written as (LP_p) (where the index p means the prize-collecting version of GMC(T)).

$$(LP_p) \quad \min \sum_{e \in E} c_e x_e + \sum_{i \in [l]} \pi_i z_i$$

subject to

$$\sum_{e \in p} x_e + z_i \geq 1, \qquad \forall i \in [l], \forall p \in P_i \tag{3}$$

$$x_e \geq 0, \qquad \forall e \in E$$

$$z_i \geq 0, \qquad \forall i \in [l]$$

In the linear program (LP_p), the variable z_i is defined for every terminal set to indicate whether the terminal set S_i is cut by edges $\{e \in E \colon x_e = 1\}$. If the terminal set S_i is uncut, then $z_i = 1$ and S_i has to pay for the penalty π_i. The constraint (3) states that for every terminal set S_i, either for every path in P_i there is at least one edge e with $x_e = 1$, or $z_i = 1$ (and thus S_i has to pay for the penalty π_i).

The dual program of (LP_p) is shown as the following linear program (DP_p).

$$(DP_p) \qquad \max \sum_{i \in [l]} \sum_{p \in P_i} f_p$$

subject to

$$\sum_{i \in [l]} \sum_{\substack{p \colon e \in p \\ p \in P_i}} f_p \leq c_e, \qquad \forall e \in E \tag{4}$$

$$\sum_{p \in P_i} f_p \leq \pi_i, \qquad \forall i \in [l] \tag{5}$$

$$f_p \geq 0, \qquad \forall i \in [l], \forall p \in P_i$$

The variable f_p in (DP_p), which is defined for every path in every terminal set, can be interpreted as the flow value on the path. In fact linear program (DP_p) describes the flow problem which is the dual to pc-GMC(T). Last but not least, note that a terminal pair, say (t, t'), may appear in more than one terminal sets. So, in dual program (DP_p) the path $[t, t']$ may have more than one delegate paths, with each delegate path p corresponding to a terminal set in which $[t, t']$ appears. The variable f_p is defined actually for every delegate path.

2.2 The Primal-Dual Approximation Algorithm

Given the primal program and the dual program for pc-GMC(T), we design an approximation algorithm for pc-GMC(T) based on the primal-dual scheme, as shown in algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ is extended from the GVY algorithm by adding the process for penalties in each iteration. A distinct difference of algorithm $\mathcal{A}$ from the GVY algorithm is that algorithm $\mathcal{A}$ may raise many dual variable f_p simultaneously, whereas the GVY algorithm raises each dual variable individually.

Algorithm $\mathcal{A}$

```
    *input:* Tree $T$, terminal sets $\{S_i\}$ and their penalties $\{\pi_i\}$.
    *output:* Edge set $D$ and index set $N$, such that removing $D$ from $T$ cuts
    every terminal set in $\{S_i \colon i \notin N\}$.

 1. **let** $D \leftarrow \emptyset$, $f_p \leftarrow 0$, $N \leftarrow \emptyset$.
 2. Root the tree at any vertex.
 3. **while** there exists non-processed vertices **do**
 4.     Let $v$ be the deepest non-processed vertex, breaking ties arbitrarily.
 5.     **for each** $(t,t')$ in some $S_i$ such that $i \notin N$ and $lca(t,t') = v$ **do**
 6.         Suppose that the terminal sets which contain $(t,t')$ and are not fully
            paid for are $S_{i_1}, S_{i_2}, \ldots, S_{i_r}$ (i.e. for every $j \in [r]$, $\{t,t'\} \subseteq S_{i_j}$ and
            $i_j \notin N$). For each $j \in [r]$, let $p_j$ be the delegate of path $[t,t']$ in $P_{i_j}$.
 7.         Simultaneously increase $f_{p_j}$ for all $1 \le j \le r$. And in this process,
 8.         **if** for some $j \in [r]$ the penalty $\pi_{i_j}$ is fully paid for, **then** stop
            increasing $f_{p_j}$ and **let** $N \leftarrow N \cup \{i_j\}$.
 9.         **if** some edge $e \in [t,t']$ is saturated **then** stop increasing all $f_{p_j}$
            and **let** $D \leftarrow D \cup \{e\}$.
10.     **end**
11. **end**
12. Let $e_1, e_2, \ldots, e_t$ be the ordered list of edges in $D$, and $A$ be the index set of
    terminal sets in $X$ which are cut by $D$.
13. **for** $j = t$ **downto** $1$ **do**
14.     **if** $D - \{e_j\}$ still cut every terminal set with index in $A$ **then** remove $e_j$
        from $D$.
15. **end**
16. **output** $D$ and $N$.
```

In algorithm $\mathcal{A}$, D denotes the edges picked by the algorithm, and N denotes the set of indices of terminal sets that have to pay for their penalties. The algorithm starts with an infeasible primal solution ($\mathbf{x} = \mathbf{0}, \mathbf{z} = \mathbf{0}$) and a trivial feasible dual solution $\mathbf{f} = \mathbf{0}$. In each iteration, algorithm $\mathcal{A}$ continues to improve the feasibility of the primal solution $(\mathbf{x}, \mathbf{z})$ and the optimality of the dual solution $\mathbf{f}$. Eventually, algorithm $\mathcal{A}$ finds an integral feasible solution to (LP_p) and an as optimal as possible solution to (DP_p), meanwhile the final primal solution and dual solution satisfy the complementary slackness conditions.

Garg et al. in [4] proved a structural property about the solution found by the GVY algorithm in the setting of MC(T). For each disconnected pair (s_i, t_i), if the flow on the path $[s_i, t_i]$ is nonzero, then at most one edge is picked by the GVY algorithm in each of the two paths $[s_i, v]$ and $[v, t_i]$, where v is the least common ancestor of s_i and t_i. For algorithm $\mathcal{A}$ although there may be many paths with non-zero flows and the algorithm does not pick any edge on these paths (these paths are not cut and the flows on these paths pay for penalties of uncut terminal sets), we can show that for every disconnected terminal pair the structural property still holds and thus introduce the structural property into the setting of pc-GMC(T). The proof of Lemma 1 is essentially the same as that of Lemma 18.5 in [11](page 149), and is omitted here.

Lemma 1 ([4,11]). *For every disconnected terminal pair (t, t') in some terminal set, if the flow on the path $[t, t']$ is nonzero, then at most one edge is picked by algorithm $\mathcal{A}$ in each of the two paths $[t, v]$ and $[v, t']$, where $v = lca(t, t')$ is the least common ancestor of t and t'.* □

Now, we are ready to prove the main theorem for algorithm $\mathcal{A}$.

Theorem 1. *Algorithm $\mathcal{A}$ is a polynomial time 3-approximation algorithm for the problem prize-collecting generalized Multicut on trees.*

Proof. It is obvious that the output of algorithm $\mathcal{A}$ is a feasible solution to pc-GMC(T). By the primal-dual scheme of algorithm $\mathcal{A}$, the primal solution $(\mathbf{x}, \mathbf{z})$ and the dual solution $\mathbf{f}$ found by $\mathcal{A}$ satisfy that $x_e = 1$ iff $\sum_{i\in[l]} \sum_{p:\, e\in p, p\in P_i} f_p = c_e$ and $z_i = 1$ iff $\sum_{p:\, p\in P_i} f_p = \pi_i$. So, for the set D of picked edges, we have

$$\sum_{e\in D} c_e = \sum_{e\in D} \sum_{i\in[l]} \sum_{\substack{p:\, e\in p \\ p\in P_i}} f_p = \sum_{i\in[l]} \sum_{p\in P_i} f_p \cdot |D \cap p| \le 2 \sum_{i\in[l]} \sum_{p\in P_i} f_p. \tag{6}$$

In inequality (6), the second equality holds since the summation order is changed. The last inequality is relaxed from the following two aspects: First, by Lemma 1 we know that for every cut path p, $f_p > 0$ implies that $|D \cap p| \le 2$. Second, there are still some paths p satisfying that $f_p > 0$ and $|D \cap p| = 0$.

Then, for the index set N of uncut terminal sets, we have

$$\sum_{i\in N} \pi_i = \sum_{i\in N} \sum_{p\in P_i} \le \sum_{i\in[l]} \sum_{p\in P_i} f_p. \tag{7}$$

By inequalities (6) and (7), we know that $A(I)$, the value of the output of algorithm $\mathcal{A}$, satisfying that

$$A(I) = \sum_{e\in D} c_e + \sum_{i\in N} \pi_i \le 3 \sum_{i\in[l]} \sum_{p\in P_i} f_p \le 3 \cdot OPT_f(LP_p) \le 3 \cdot OPT, \tag{8}$$

where $OPT_f(LP_p)$ denotes the optimum of factional linear program (LP_p), and OPT denotes the optimum of the instance of pc-GMC(T). The second inequality in (8) is due to the primal-dual theorem. This completes the proof of the theorem. □

2.3 The Randomized Rounding Approximation Algorithm

Using the randomized rounding with scaling technique, we design a 2.55-approximation algorithm for pc-GMC(T). Similar technique also appears in [6] to deal with the problem prize-collecting Steiner Forest on graphs. The proof of Theorem 2 is omitted in this extended abstract.

Theorem 2. *There is a polynomial time 2.55-approximation algorithm for the problem prize-collecting Multicut on trees.* □

3 *k*-Generalized Multicut on Trees

The k-GMC(T) problem is the natural k-version of the GMC(T) problem. We find an interesting relation between the k-GMC(T) problem and the Densest k-Subgraph problem, implying that simply adding the parameter k significantly increase the hardness of the problem generalized Multicut on trees. Then we give a polynomial time approximation algorithm to k-GMC(T) through a nonuniform approach, with performance ratio $\min\{2(l-k+1), k\}$.

3.1 Hardness of *k*-GMC(T)

We show that the hardness of k-GMC(T) can be reduced to that of the Densest k-Subgraph problem. First we show that the Minimum k-Edge Coverage problem can be reduced to k-GMC(T) by approximation factor preserving reduction. The instance of the Minimum k-Edge Coverage problem (k-MEC) consists of a graph G and an integer $k > 0$, and the goal of the problem is to find the minimum number of vertices in graph G whose induced subgraph has at least k edges.

Lemma 2. *There is an approximation factor preserving reduction from the k-MEC problem to the k-GMC(T) problem.*

Proof. The instance of k-GMC(T) is constructed as follows. The root of the reduced instance is r. For every vertex v_i in the instance of k-MEC, there is a vertex v_i in the reduced instance whose parent is r. For every edge $e = (v_i, v_j)$ in the instance of k-MEC, there is a terminal set $S = \{r, v_i, v_j\}$ in the instance of k-GMC(T). Then one can see that if in the instance of k-MEC there is a vertex set V' which strictly covers at least k edges, then in the instance of k-GMC(T) there is an edge set (in which each edge corresponds to a vertex in V') whose removal cuts at least k terminal sets (each cut terminal set corresponds to an edge that is strictly covered in k-MEC), and vice versa. The lemma follows. □

Given a graph G and an integer $k > 0$, the Densest k-Subgraph problem asks to find a set of k vertices with maximum number of induced edges. Hajiaghayi and Jain [6] proved that if the k-MEC problem can be approximated within α, then the Densest k-Subgraph problem can be approximated within $2\alpha^2$. The currently best known performance ratio for the Densest k-Subgraph problem is $O(n^{1/3-\epsilon})$ for some small $\epsilon > 0$ and the improvement is believed to be hard [2,9]. By these facts and Lemma 2 we have Theorem 3.

Theorem 3. *If the k-GMC(T) problem can be approximated within factor $O(n^{1/6-\epsilon})$ in polynomial time for some small $\epsilon > 0$, then there is an $O(n^{1/3-2\epsilon})$-approximation algorithm for the Densest k-Subgraph problem.* □

3.2 Approximating *k*-GMC(T)

In this subsection we design a $\min\{2(l-k+1), k\}$-approximation algorithm for k-GMC(T) through a nonuniform approach. In fact we design two approximation algorithms for k-GMC(T). One is based on greedy approach which deals with the case that k is small, and the other is based on LP-rounding technique which deals with the case that $l-k$ is small.

If the parameter l is restricted to be 1 in k-GMC(T), then the problem is reduced to the Multiway Cut problem on trees. We show that the problem Multiway Cut on trees can be optimally solved in polynomial time. The proof of Lemma 3 is omitted in this extended abstract.

Lemma 3. *The problem Multiway Cut on trees is polynomial time solvable.* □

The polynomial time solvability of the problem Multiway Cut on trees suggests a simple greedy approximation to the k-GMC(T) problem.

Lemma 4. *The k-GMC(T) problem can be approximated within factor k in polynomial time.*

Proof. Solve the Multiway Cut instance for every terminal set S_i in X. Suppose that the solution to the instance corresponding to S_i is edge set F_i. Then the union of the lightest k sets F_i is a k-approximation to k-GMC(T), since the optimal solution to k-GMC(T) must cut at least k terminal sets, and hence for every $1 \le i \le k$ the cost of F_i is no more than the optimum of k-GMC(T). □

Then we give the fractional linear program formulation for k-GMC(T), as shown in (LP_k) (where the index k means the k-version of GMC(T)).

$$(LP_k) \qquad \min \sum_{e \in E} c_e x_e$$

subject to

$$\sum_{e \in p} x_e + z_i \ge 1, \qquad \forall i \in [l], \forall p \in P_i \tag{9}$$

$$\sum_{i \in [l]} z_i \le l - k, \tag{10}$$

$$x_e \ge 0, \qquad \forall e \in E$$

$$z_i \ge 0, \qquad \forall i \in [l]$$

Consider the integral version of linear program (LP_k). The variable z_i indicates whether the terminal set S_i is cut by the edges $\{e \in E \colon x_e = 1\}$. If S_i is not cut, then $z_i = 1$. The constraint (10) states that the number of terminal sets that is cut must be at least k. The objective function of (LP_k) is to minimize the total cost of edges that cut at least k terminal sets. A straightforward observation of (LP_k) is the following Lemma 5.

Lemma 5. *For the optimal solution* $(\mathbf{x}^*, \mathbf{z}^*)$ *to* (LP_k)*, the constraint* (10) *holds with equality.* □

Then we give an approximation guarantee with respect to the parameter l and k for the k-GMC(T) problem. The technique is LP-rounding with scaling.

Lemma 6. *The k-GMC(T) problem can be approximated within factor* $2(l-k+1)$ *in polynomial time.*

Proof. Without loss of generality, by reordering the indices, suppose that for the optimal solution $(\mathbf{x}^*, \mathbf{z}^*)$ the variables z_i^*'s are in the order $z_1^* \geq z_2^* \geq \cdots \geq z_l^*$. Then we round the variables $z_1^*, \ldots, z_{l-k}^*$ to 1 and the variables $z_{l-k+1}^*, \ldots, z_l^*$ to 0. That is, we leave every terminal set $S \in \{S_1, \ldots, S_{l-k}\}$ not cut. Denote by $\hat{\mathbf{z}}$ the rounded $\mathbf{z}^*$, and by Q the index set $\{i \colon \hat{z}_i = 0\}$. By Lemma 5, $\sum_{i\in[l]} z_i^* = l-k$. We argue that $z_i^* \leq (l-k)/(l-k+1)$ holds for every $i \in Q$, since otherwise $z_1^* + \cdots + z_{l-k+1}^* > l-k$, which contradicts constraint (10). So, by constraint (9), we know that $\forall i \in Q, \forall p \in P_i$,

$$\sum_{e\in p} x_e \geq 1 - z_i \geq 1 - \frac{l-k}{l-k+1}.$$

Let $a = (l-k)/(l-k+1)$. Now, define $x'_e = \min\{\frac{1}{1-a}x_e^*, 1\}$ for every e. Then, $\mathbf{x}'$ is a feasible solution to the linear program (LP_s) on Q. Notice that (LP_s) on Q is just the linear program for the problem Multicut on trees with terminal sets $X = \{S_i \colon i \in Q\}$. By the GVY algorithm, an integral 2-approximate solution $\hat{\mathbf{x}}$ to (LP_s) can be found in polynomial time. Now, $(\hat{\mathbf{x}}, \hat{\mathbf{z}})$ is a feasible solution to (LP_k), with solution value

$$\sum_{e\in E} c_e \hat{x}_e \leq 2 \cdot OPT_f(LP_s) \leq 2\sum_{e\in E} x'_e \leq \frac{2}{1-a}\sum_{e\in E} x_e^* \leq \frac{2}{1-a} OPT, \qquad (11)$$

where $OPT_f(LP_s)$ denotes the value of an optimal fractional solution to (LP_s), and OPT denotes the optimum to the instance of k-GMC(T). In inequality (11), the second inequality holds since $\mathbf{x}'$ is a feasible solution to (LP_s), the third inequality holds since by definition $x'_e \leq \frac{1}{1-a}x_e^*$ for every edge e. Finally, we know that $2/(1-a) = 2(l-k+1)$. The lemma follows. □

Lemma 4 and Lemma 6 establish the main theorem of this subsection.

Theorem 4. *There is a polynomial time* $\min\{2(l-k+1), k\}$*-approximation algorithm for the k-GMC(T) problem.* □

Note that the performance ratio $\min\{2(l-k+1), k\}$ in Theorem 4 behaves well when k is near to 1 or l. Since k ranges over $\{1, \ldots, l\}$, we always have that $\min\{2(l-k+1), k\} \leq (2/3)l + 2/3$.

4 Discussion

We obtain an instance, which takes advantage of the penalties, showing that the integrality gap of (LP_p) for pc-GMC(T) is at least 2. Then by the results in section 2 we know that the integrality gap for pc-GMC(T) is between 2 and 2.55. Thus getting an instance to show that the integrality gap for (LP_p) is strictly greater than 2 or reducing the performance ratio for pc-GMC(T) is an interesting problem. Moreover, although the performance ratio $\min\{2(l-k+1), k\}$ for k-GMC(T) is non-trivial, it seems that there is still a large space to improve the ratio.

Acknowledgement. We would like to thank the anonymous referees for their helpful comments, which help us significantly improve the presentation of the paper.

References

1. Avidor, A., Langberg, M.: The multi-multiway cut problem. In: Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04), pp. 273–284 (2004)
2. Feige, U.: Relations between average case complexity and approximation complexity. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02), pp. 534–543 (2002)
3. Garg, N., Vazirani, V., Yannakakis, M.: Approximate max-flow min-(multi)cut theorems and their applications. SIAM Journal on Computing 25, 235–251 (1996)
4. Garg, N., Vazirani, V., Yannakakis, M.: Primal-dual approximation algorithm for integral flow and multicut in trees. Algorithmica 18, 3–20 (1997)
5. Golovin, D., Nagarajan, V., Singh, M.: Approximating the k-multicut problem. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA'06), pp. 621–630 (2006)
6. Hajiaghayi, M., Jain, K.: The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06), pp. 631–640 (2006)
7. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.: Greedy facility location algorithms analyzied using dual fitting with factor-revealing LP. Journal of the ACM 50(6), 795–824 (2003)
8. Levin, A., Segev, D.: Partial multicuts in trees. In: Erlebach, T., Persinao, G. (eds.) Approximation and Online Algorithms. LNCS, vol. 3879, pp. 320–333. Springer, Heidelberg (2006)
9. Khot, S.: Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In: Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'04), pp. 136–145 (2004)
10. Könemann, J., Parekh, O., Segev, D.: A unified approach to approximating partial covering problems. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 468–479. Springer, Heidelberg (2006)
11. Vazirani, V.: Approximation Algorithms, 2nd edn. Springer, Berlin (2003)

(Short) Survey of Real Hypercomputation

Martin Ziegler*

University of Paderborn, Germany
ziegler@upb.de

Abstract. We survey and compare models of computation on real numbers exceeding the Church–Turing Hypothesis.

1 Computability

Any theory of computability must rely on some model of computation, that is a mathematical abstraction and idealization of the kind of actual devices whose fundamental power the theory is to describe. On the discrete realm several major such notions of effectivity (Turing machines, `WHILE`–programs, λ–calculus, μ–recursivity, ...) have turned out as equivalent and are now widely agreed upon as appropriately capturing a, say, PC's capabilities operating on bits or integers. This model has been extended in two fundamental directions whose fusion the present text surveys on: (discrete) hypercomputation and real computability theory.

1.1 Real Computability

deals with questions of computability for problems involving real numbers. This departure from the discrete realm has borne three major notions, each one reflecting some while (as necessary for any idealization) neglecting other aspects of actual real computing devices:

Recursive Analysis as initiated by Alan M. Turing himself [Turi36] considers effective approximability of real numbers and function values with fractions of (discrete) integers by a Turing machine (Section 2.2).

Algebraic/symbolic computation extends Turing machines to store in each cell and operate in each step arithmetically on a real number (Section 2.1)

Analog computation, including real recursion theory, is however beyond our scope; instead see e.g. [Moor96, Orpo97, CMC02, GrCo03, Kawa05, BoCa07].

They are largely *in*equivalent to one another. For example the exponential function is computable in Recursive Analysis but not in the algebraic model [Brat00], whereas the situation for sign function is vice versa. On the other hand certain natural extensions of these models do admit instructive ways of comparing their power [BoVi99]. Such extensions bring us to the topic of

* Supported by the German Research Foundation (DFG) project `Zi 1009/1-1`.

1.2 Hypercomputation

In classical computability over bits and integers,

a) the aforementioned proven equivalence of several natural notions,
b) the continuous failure to come up with a device capable of solving the Turing–undecidable Halting problem H,
c) and the success of having a vast variety of physical systems simulated on a Turing machine

all support the conclusion that this is not just a rather appropriate but actually the universal model of practical computation:

> "*every function which would naturally be regarded as computable can be computed by a Turing Machine.*"

This has become known as "the Church–Turing Thesis" (CTH). However every single word of this name is misleading:

- More a hypothesis than a 'thesis', it has *not* been and *cannot* be proven — simply because it is far too informal with ambiguous notions like 'naturally'.
- Neither Alonzo Church nor Alan Turing have actually put forth a claim as bold as the one above,
- and, as opposed to 'the', current science more carefully distinguishes *various* variants of this hypothesis; see e.g. [Ord02, Section 2.2] or [Cope02].

As a matter of fact the above supports of CTH are nowadays known to fail at least in principle: c) in Quantum Gravitation [Gero86, p.547]; b) in Relativity Theory [Hoga92, EtNe02, WiLe02], Quantum Mechanics [ACP04, Kie04b, Zie05b] or even within Classical Physics [Smit06, BeTu06]; and a) when turning to real numbers.

This suggests to study models of computation exceeding the Turing machine, so-called hypercomputers.

1.3 Real Hypercomputation

Real Hypercomputation applies, investigates, and compares models of real number computability beyond the Church–Turing Hypothesis.

2 Models of Real Computation

The classical Turing machine can store and operate on (e.g. add, subtract, multiply, divide, and compare) rational numbers within finite time; it is equivalent to the RAM model [Scho79]. This observation suggests two different directions of generalization to the real numbers which nicely complement each other.

2.1 Algebraic Computation

Considering $\mathbb{Q}$ and $\mathbb{R}$ as algebraic structures (ordered fields), a machine which can store reals and perform on them a finite number of arithmetic operations and comparisons is called an $\mathbb{R}$–machine or BCSS–machine [Tuck80], [BSS89], [BCSS98], and [TuZu00]. This is the standard model in Computational Geometry [BKOS00] (there named "real-RAM") and ubiquitous in Computer Algebra [BCS97], [GaGe03] as well as Algebraic Geometry [BPR03]. Gaussian Elimination and Simplex Algorithm typically pertain to this kind of machine. Regarding (un-)computability, we mention

Example 1.

a) The set $\mathbb{N}$ of integers is decidable[1]

b) Every discrete problem $P \subseteq \mathbb{N}$ is decidable to a BCSS–machine by storing $c := \sum_{n \in P} 2^{-n}$ as a real constant [BSS89, Example 6].

c) The sets $\mathbb{Q}$ of rational and $\mathbb{A}$ of algebraic real numbers are semi-decidable yet undecidable.

d) Also the Mandelbrot Set has been proven undecidable as well as the set of starting points for convergence of the Newton iteration [BCSS98, Section 2.4].

e) Both exponential $\mathbb{R} \ni x \mapsto \exp(x)$ and square root function $[0, \infty) \mapsto \sqrt{x}$ are uncomputable.

f) Feasibility of a given system of real polynomial in-/equalities

$$\left\{ \vec{x} \in \mathbb{R}^m \ : \ p_1(\vec{x}) = \ldots = p_k(\vec{x}) = 0 \ \wedge \ q_1(\vec{x}) > 0 \wedge \ldots \wedge q_\ell(\vec{x}) > 0 \right\} \tag{1}$$

is decidable by means of Tarski's Quantifier Elimination [BPR03, §2.5.1].

Claim f) asserts that $\mathsf{NP}_\mathbb{R} \subseteq \mathsf{EXP}_\mathbb{R} \subseteq \mathsf{REC}_\mathbb{R}$: real non-determinism can be simulated deterministically in exponential time. The problem in f) can also be regarded as a real counterpart to the feasibility of a system of polynomials over integers; the latter is *un*decidable to a Turing machine by the famous result [Mati70]. Also, there exists no computable real pairing function $\langle \cdot, \cdot \rangle : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. On the other hand many properties from discrete computability do carry over to $\mathbb{R}$:

Fact 2

a) A language $\mathbb{L} \subseteq \mathbb{R}^*$ is decidable iff both $\mathbb{L}$ and its complement are semi-decidable.

b) Non-empty $\mathbb{L}$ is semi–decidable iff it is enumerable *in the sense of coinciding with the range of a computable total function $f : \mathbb{R}^* \to \mathbb{R}^*$.*

c) One can naturally encode a BCSS–machine $\mathbb{M}$ as $\langle \mathbb{M} \rangle \in \mathbb{R}^*$ and thus obtain a real Gödelization together with a universal *BCSS–machine, and obtain SMN and UTM properties.*

[1] However the number of steps performed will depend on the *value* of the input. Such behavior disappears in computations over $\mathbb{C}$ where, similar to Turing machines over bit strings $\{0,1\}^n$, any algorithm terminating for all inputs from $\mathbb{C}^n$ does so in time bounded by some function of the *size* n of the input only [Ross93]. In particular, $\mathbb{N}$ is *un*decidable over $(\mathbb{C}, +, -, \times, \div, =)$.

d) The real Halting problem $\mathbb{H}$ *is undecidable, where*

$$\mathbb{H} \quad = \quad \{\langle\mathbb{M}\rangle : \mathbb{M} \text{ terminates on empty input}\} \quad \subseteq \quad \mathbb{R}^* \ .$$

A source of criticism against this model arises from its ability to compare arbitrary reals exactly [BoVi99] which leads to strange functions being computable [Weih00, Example 9.7.2]. As a remedy, [BrHe98] proposes a modified semantics for comparisons. Example 1b) can be avoided by restricting the use of real machine constants, cf. Section 4.3.

2.2 Recursive Analysis

Every real is the limit of a rational Cauchy sequence; so consider real computation in terms of classical Turing computations on Cauchy sequences of rationals.

Definition 3

a) Call a number $x \in \mathbb{R}$ naively computable *if there exists a Turing–computable sequence* $(q_n) \subseteq \mathbb{Q}$ *with* $x = \lim_n q_n$.
b) The number x *is* computable *if there exists a Turing–computable sequence* $(q_n) \subseteq \mathbb{Q}$ *with* $|x - q_n| \leq 2^{-n}$.
c) Lower computability *of* x *means existence of a Turing–computable sequence* $(q_n) \subseteq \mathbb{Q}$ *such that* $x = \sup_n q_n$.
d) Function $f : \mathbb{R} \to \mathbb{R}$ *is* computable *if, upon input of* $(q_n) \subseteq \mathbb{Q}$ *with* $|x - q_n| \leq 2^{-n}$, *a Turing machine can output* $(p_m) \subseteq \mathbb{Q}$ *such that* $|f(x) - p_m| \leq 2^{-m}$.
e) A set $U \subseteq \mathbb{R}^k$ *is* semi-decidable *if there exists a Turing machine which, on input of* $(q_n) \subseteq \mathbb{Q}^k$ *with* $|x - q_n| \leq 2^{-n}$, *terminates in case* $x \in U$ *and diverges in case* $x \notin U$.

Notion b) is equivalent to the one considered already by Turing [Turi36, Turi37]. Notice that, although an admissible computation takes infinitely long, one may abort the execution after attaining the desired absolute precision. In a) however, rational approximations need not satisfy computable error bounds; hence a finite part of the output does not permit any conclusion about the final result. For the Halting Problem $H \subseteq \mathbb{N}$, the real number $\sum_{n \in H} 2^{-n}$ is naively computable and lower computable yet uncomputable [Spec49]; see also Section 3.2.

Notion d) dates back to [Grze57] and is equivalent to many other reasonable notions [Laco57, PERi89, Ko91, Brat96], cf. [Weih95, Section 10] or [Weih00, Section 9]. For instance we mention [PERi89, Section 0.7]

Proposition 4 (Effective Weierstraß Theorem). *A function* $f : [0,1] \to \mathbb{R}$ *is computable iff there exists a Turing–computable sequence of (degrees and coefficients of) rational polynomials* $(P_m) \subseteq \mathbb{Q}[X]$ *such that* $\|f - P_m\| := \sup_{0 \leq x \leq 1} |f(x) - P_m(x)| \leq 2^{-m}$.

Most analytic functions from practice are computable: addition, multiplication, division, exponentiation, logarithms [Weih00, Section 4.3]. Uncomputable is the constant function $f(x) \equiv \sum_{n \in H} 2^{-n}$; also notice [Weih00, Theorem 4.3.1]:

Fact 5 ('Main Theorem'). *Every discontinuous function is uncomputable.*

Decidability thus fails for equality "=" and every non-empty strict subset of $\mathbb{R}$:

Lemma 6. *Any semi-decidable set $U \subseteq \mathbb{R}^k$ is necessarily open: U is semi-decidable iff* recursively enumerable *(r.e. open) in the following sense: a Turing machine can output rational centers (q_n) and radii (r_n) of open rational 'balls' $B(q,r) = \{x \in \mathbb{R}^k : |x-q| < r\}$ whose union $\bigcup_n B(q_n, r_n)$ coincides with U.*

Fact 5 raises criticism from supporters of the algebraic model [Koep01] as it prevents functions as simple as the sign from being computable. Regarding the topic of this survey, an interesting question asks whether hypercomputation may lift the continuity requirement (Section 3.3). Another characterization of computable (and thus continuous) functions proceeds in terms of *effective* continuity:

Proposition 7 ([Laco57]). *A function $f : \mathbb{R} \to \mathbb{R}$ is computable iff there exists a Turing machine which, upon input of (centers and radii of) open rational balls $B(q_n, r_n)$ exhausting $U \subseteq \mathbb{R}$, outputs corresponding balls $B(p_m, s_m)$ exhausting $f^{-1}[U] \subseteq \mathbb{R}$.*

The present text focuses on computability on real numbers, (continuous) real functions, and (open) subsets of reals. However the concept underlying Definition 3 extends to arbitrary spaces X of continuum cardinality: Fix an encoding of each element $x \in X$ as an infinite string of bits or integers (Cantor or Baire space) and define computability on X in terms of these 'names'. The study and comparison of these encodings, so-called *representations*, leads to WEIHRAUCH's (meta-) theory of computability called TTE [Weih00].

2.3 Further Models: Domain Theory and Analytic Machines

In view of Definition 3 and Proposition 4, Recursive Analysis can be considered as a foundation for interval computation; conversely it has indeed been implemented in practice [Muel01, Lamb05]. Another formalization, Domain Theory [Scot70] is often equivalent but in detail subtly different by requiring (in contrast to Definition 3d) each single input interval to yield a corresponding output interval; cf. [Weih00, SECTION 9.5] for a comparison.

A proof establishing computability of some problem strictly speaking only asserts the *existence* of an algorithm. Practical implementations like the above raise the issue of actually devising one: and of doing so in an elegant and intuitive language [BrHe98]. In discrete computability theory, λ–calculus had led to the concept of functional programming (`LISP`, `Haskell` etc); [Esca96] has devised a similar language including data types over reals.

Analytic Machines form an entire family of models combining various aspects of BCSS–machines with Recursive Analysis [HVS95, ChHo99]. Classical Turing–computability amounts to $\mathbb{Q}$–machines, $\mathbb{R}$–machines are BCSS–machines, and Definition 3d) is realized by so-called *robust strongly δ–$\mathbb{Q}$–analytic* machines. The powerful *(strongly)* $\mathbb{R}$*–analytic* machine finally may use exact real arithmetic on $x \in \mathbb{R}$ in order to approximate the result $f(x)$ in the limit (with error bounds 2^{-n}). They however lack closure under composition.

An interesting in-between constitutes the class of *robust quasi-strongly* δ–$\mathbb{Q}$–*analytic* machines. Their output approximations $q_n \in \mathbb{Q}$ may violate error bound 2^{-n} for an arbitrary but finite number of times. This notion does ensure closure under composition while adding to all functions satisfying Definition 3d) the capability to compute also discontinuous ones. More precisely using so-called *conservative branching*, any function realized by a BCSS–machine (using only computable constants) can be shown robust quasi-strongly δ–$\mathbb{Q}$–analytic [ChHo99, THEOREM 3].

3 Arithmetical Hierarchies of Hypercomputers

Even the study of (discrete) hypercomputers dates back to Turing with his dissertation [Turi39] about oracle machines. For example, the power of oracle access to the Halting problem is characterized in

Fact 8 (Shoenfield's Limit Lemma). *For an arbitrary oracle $\mathcal{O}$, a (discrete) function $f : \mathbb{N} \to \mathbb{N}$ is computable relative to $\mathcal{O}'$ iff $f(n) = \lim_m g(n, m)$ is the pointwise limit of a function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ computable relative to $\mathcal{O}$.*

$\mathcal{O}'$, called the *jump* of $\mathcal{O}$, denotes the Halting problem for machines equipped with oracle $\mathcal{O}$. By iteration one obtains the Arithmetical Hierarchy [Soar87]

$$\emptyset \qquad \emptyset' \equiv H \qquad \emptyset'' =: \emptyset^{(2)} \qquad \ldots \qquad \emptyset^{(d)} \qquad \ldots \tag{2}$$

"$X \quad Y$" here means many-one reducibility from X to Y but lack of Turing-reducibility from Y to X. A *syntactical* formulation proceeds in terms of iterated alternating quantifiers: (light-face) Σ_d is the class of languages of the form

$$L \;=\; \big\{x \in \mathbb{N} \;\big|\; \exists y_1 \in \mathbb{N}\, \forall y_2 \in \mathbb{N}\, \exists y_3 \in \mathbb{N} \ldots \theta_d y_d \in \mathbb{N} : \langle x; y_1, \ldots, y_d\rangle \in R\big\} \tag{3}$$

for a decidable problem $R \subseteq \mathbb{N}$. Σ_1 thus coincides with the class of semi-decidable languages, that is, (the empty set and) images of computable total functions. By Post's Theorem, a problem belongs to Σ_{d+1} iff it is semi-decidable relative to $\emptyset^{(d)}$. The following question FIN is Σ_2–complete under many-one reduction [Soar87, THEOREM §IV3.2]:

"*Given an (encoding of a) Turing machine, is the set it accepts finite?*"

A set is defined to belongs to Δ_d if it, as well as its complement, lie in Σ_d. The Arithmetical Hierarchy

$$\Delta_1 \subsetneq \Sigma_1 \subsetneq \Delta_2 \subsetneq \Sigma_2 \subsetneq \ldots \subsetneq \Delta_d \subsetneq \Sigma_d \subsetneq \ldots \tag{4}$$

has been carried from the discrete setting over to both the BCSS model of real number computation (Section 3.5) and to Recursive Analysis (Sections 3.1–3.4):

3.1 Effective Borel Hierarchy

The Borel Hierarchy of a topological space X starts with the (bold-face) class $\mathbf{\Sigma}_1(X)$ of open subsets; next comes the class $\mathbf{\Sigma}_2(X)$ of F_σ–subsets, that is, of countable unions over closed sets; and, inductively, $\mathbf{\Sigma}_{d+1}(X)$ consists of all sets $\bigcup_{n\in\mathbb{N}}(X \setminus S_n)$ where $S_n \in \mathbf{\Sigma}_d(X)$; cf. e.g. [Kech95]. Finally $\mathbf{\Delta}_d(X) := \{S \in \mathbf{\Sigma}_d(X) : X \setminus S \in \mathbf{\Sigma}_d(X)\}$ constitute the *ambiguous classes*. Then it holds for $X = \mathbb{N}$, $X = \mathbb{R}$, $X = \{0,1\}^{\mathbb{N}}$, and $X = \mathbb{N}^{\mathbb{N}}$:

$$\mathbf{\Delta}_1(X) \subsetneq \mathbf{\Sigma}_1(X) \subsetneq \mathbf{\Delta}_2(X) \subsetneq \mathbf{\Sigma}_2(X) \subsetneq \ldots \mathbf{\Delta}_d(X) \subsetneq \mathbf{\Sigma}_d(X) \subsetneq \ldots$$

Many similarities between the Borel and the Arithmetical hierarchy have been observed long ago [Hinm78] and led to study of *effective* descriptive set theory [Mosc80]. For instance, Lemma 6 and Definition 3e) can be regarded as equipping $\mathbf{\Sigma}_1(\mathbb{R}^k)$ with a notion of effectivity. This has been generalized [Weih00, EXERCISES 4.3.17+18] to (complements of) $\mathbf{\Sigma}_2(\mathbb{R}^k)$ and [Brat05] to arbitrary (finite) levels $\mathbf{\Sigma}_d(\mathbb{R}^k)$, see also [Mosc80, CHAPTER 3]:

Definition 9. *A set $S \subseteq \mathbb{R}^k$ is* $\mathbf{\Sigma}_d$–computable *if it coincides with*

$$\bigcup_{m_1}\bigcap_{m_2}\bigcup_{m_3}\cdots\bigodot_{m_d} B(p_{(m_1,\ldots,m_d)}, s_{(m_1,\ldots,m_d)}) \tag{5}$$

for Turing–computable rational (multi-)sequences $p = (p_{m_1,\ldots,m_d})$ and $s = (s_{m_1,\ldots,m_d})$. "$\bigodot_m S_m$" means "$\bigcup_m S_m$" for odd d and "$\bigcap_m(\mathbb{R}^k \setminus S_m)$" for d even.

The class of $\mathbf{\Sigma}_d$–computable sets is a (countable hence strict) subclass of $\mathbf{\Sigma}_d$. On the other hand, $\mathbf{\Sigma}_d$–computable sets exceed $\mathbf{\Delta}_d$ [Hinm78, EXERCISE III.1.28]. In fact a set belongs to $\mathbf{\Sigma}_d$ iff it is $\mathbf{\Sigma}_d$–computable *relative* to some oracle $\mathcal{O}$. Replacing in Definition 9 the space $\mathbb{R}^k$ by the natural numbers $\mathbb{N}$, one recovers the discrete Hierarchy (3). This has led to the application of the light-face notion "Σ_d" to refer more generally to perfect Polish spaces including $\mathbb{N}$ and $\mathbb{R}^k$ but also Cantor $\{0,1\}^{\mathbb{N}}$ and Baire space $\mathbb{N}^{\mathbb{N}}$ [Mosc80].

3.2 Hypercomputable Real Numbers

Recall that, as opposed to the BCSS model, computability of single real numbers is a non-trivial topic and in fact origin of Recursive Analysis [Turi36]. We have mentioned for instance (§2.2) that $\sum_{n\in H} 2^{-n}$ lacks computability but is naively computable and does become computable by means of oracle-access to H.

Proposition 10 ([Ho99, Theorem 9]). *For an arbitrary oracle $\mathcal{O}$, a real number x is naively computable relative to $\mathcal{O}$ iff it x computable relative to $\mathcal{O}'$.*

That is, a jump makes the difference between approximations *with* or with*out* effective error bounds: fast 2^{-n} convergence versus ultimate convergence of a Turing-computable function $f : \mathbb{N} \to \mathbb{Q}$. This can also be regarded as a continuous variant of Shoenfield's Limit Lemma (Fact 8)! [ZhWe01] has extended HO's above result from a single jump to arbitrary levels of the Arithmetical Hierarchy:

Theorem 11. *For $x \in \mathbb{R}$, oracle $\mathcal{O}$, and $d \in \mathbb{N}$, the following are equivalent:*

- *x is computable relative to $\mathcal{O}^{(d)}$*
- *$x = \lim\limits_n q_n$ for a rational sequence q_n computable relative to $\mathcal{O}^{(d-1)}$*
- *$x = \lim\limits_n \lim\limits_m q_{\langle n,m\rangle}$ for a sequence $q_{\langle n,m\rangle} \subseteq \mathbb{Q}$ computable relative to $\mathcal{O}^{(d-2)}$*
- *...*
- *$x = \lim\limits_{n_1} \ldots \lim\limits_{n_d} q_{\langle n_1,\ldots,n_d\rangle}$ for a rational sequence computable relative to $\mathcal{O}$.*

Let the class $\Delta_d(\mathbb{R})$ consist of all $x \in \mathbb{R}$ satisfying one (and thus all) conditions from Theorem 11 for $\mathcal{O} = \emptyset$. Similar characterizations hold for lower semi-computable real numbers, that is, suprema (rather than arbitrary limits) of rational sequences. These lead to a hierarchy of real numbers similar to (4):

$$\Delta_1(\mathbb{R}) \subsetneqq \Sigma_1(\mathbb{R}) \subsetneqq \Delta_2(\mathbb{R}) \subsetneqq \Sigma_2(\mathbb{R}) \subsetneqq \ldots \Delta_d(\mathbb{R}) \subsetneqq \Sigma_d(\mathbb{R}) \subsetneqq \ldots$$

Its relation to the discrete counterpart is even closer for Δ–classes: For $S \subseteq \mathbb{N}$, $\sum_{n\in S} 2^{-n}$ belongs to $\Delta_d(\mathbb{R})$ iff $S \in \Delta_d$ [ZhWe01, Theorem 7.8]. The corresponding claim for the Σ–classes however fails already for $d = 1$: $\Sigma_d(\mathbb{R})$ satisfies closure under e.g. addition whereas the sum of two reals with semi-decidable binary expansion in general does not have a semi-decidable expansion.

3.3 Hypercomputable Real Functions

[Ho99] has studied in addition to real numbers also real function computability (Definition 3d) relative to $\emptyset'$ and obtained for instance the following analogue of Proposition 4 with *non*-effective uniform convergence:

Proposition 12 ([Ho99, Corollary 17]). *A function $f : [0,1] \to \mathbb{R}$ is computable relative to $\emptyset'$ iff there exists a Turing–computable sequence of (degrees and coefficients of) rational polynomials $(P_m) \subseteq \mathbb{Q}[X]$ such that $\|f - P_m\| \to 0$.*

In particular the Halting oracle, employed in this way, does not lift the continuity condition Fact 5. More generally it immediately follows from the Kreitz-Weihrauch Representation Theorem [KrWe85]

Corollary 13. *A function $f : \mathbb{R} \to \mathbb{R}$ is continuous iff there exists an oracle $\mathcal{O}$ such that f is computable relative to $\mathcal{O}$.*

A different approach to real function computability on the Arithmetical Hierarchy however does include discontinuous examples: Based on Theorem 11, relax Definition 3d) as follows:

Definition 14. *Function $f : \mathbb{R} \to \mathbb{R}$ is (k,ℓ)–*hypercomputable *if, upon input of $(q_n) \subseteq \mathbb{Q}$ with $x = \lim_{n_1} \ldots \lim_{n_k} q_{\langle n_1,\ldots,n_k\rangle}$ ($|x - q_n| \leq 2^{-n}$ in case $k = 0$) a Turing machine can output $(p_m) \subseteq \mathbb{Q}$ such that $f(x) = \lim_{m_1} \ldots \lim_{m_\ell} p_{\langle m_1,\ldots,m_\ell\rangle}$ ($|f(x) - p_m| \leq 2^{-m}$ in case $\ell = 0$).*

The example of the identity function reveals that only the case $\ell \geq k$ makes sense. By [Zie05a, Section 3.2], the discontinuous Heaviside Function is (k,ℓ)–hypercomputable whenever $\ell > k$. On the other hand in case $\ell = k = 0$, Fact 5 requires continuity. This has been extended to $\ell = k = 1$ in [BrHe02, Section 6], then to $\ell = k = 2$ in [Zie05a, Theorem 10], and finally to arbitrary $\ell = k$ [Zieg06, Corollary 2.11].

3.4 Effective Borel Measurability

A function $f : X \to Y$ is continuous iff $f^{-1}[V] \in \mathbf{\Sigma}_1(X)$ for every open $V \subseteq Y$; it is called $\mathbf{\Sigma}_d$*–measurable* iff $f^{-1}[V] \in \mathbf{\Sigma}_d(X)$ for every open $V \subseteq Y$. This has led [Brat05] to the following

Definition 15. *A function $f : \mathbb{R}^k \to \mathbb{R}^\ell$ is* effectively $\mathbf{\Sigma}_d$–measurable *if there exists a Turing machine which, upon input of (centers and radii of) open rational balls $B(q_n, r_n)$ exhausting $U \subseteq \mathbb{R}^\ell$, output centers p and radii s such that $f^{-1}[U]$ coincides with (5); compare* [Mosc80, Chapter 3D].

Effective continuity (Proposition 7) thus amounts to effective $\mathbf{\Sigma}_1$–measurability. Its generalization to $d \geq 2$ constitutes a notion of real hypercomputation which does include discontinuous functions. In fact we could establish [Zieg06] the following characterization relating Definitions 14 and 15:

Theorem 16. *A function $f : \mathbb{R} \to \mathbb{R}$ is $(0,d)$–hypercomputable iff it is effectively $\mathbf{\Sigma}_{d+1}$–measurable.*

3.5 Arithmetical Hierarchy of BCSS Machines

The diagonalization argument underlying the undecidability of the real Halting Problem $\mathbb{H}$ (Fact 2d) relativizes and yields by iteration a strict hierarchy of problems BCSS–$\emptyset^{(d)}$ similar to Equation (2).

However regarding a syntactical definition, any problem L of the form (3) is BCSS–decidable by Example 1b). And replacing $\mathbb{N}$ with $\mathbb{R}$ does not help either because d–fold application of Example 1f) eliminates all quantifiers. Instead, [Cuck92] considers some arbitrary finite field extension $F = \mathbb{Q}(c_1, \ldots, c_k)$ and a double sequence $A_{n,m} \subseteq \mathbb{R}^m$ of real subsets algebraic <u>over F</u>, i.e., each $A_{n,m}$ is the set of solutions $\vec{x} \in \mathbb{R}^m$ of some system $\varphi_{n,m}(\vec{x})$ of in-/equalities (1) of polynomials in m variables <u>with coefficients in F</u>. Now define the class BCSS–Σ_k to consist of all sets of the form

$$\left\{\vec{x} \in \mathbb{R}^m \;\middle|\; m \in \mathbb{N},\ \exists n_1 \in \mathbb{N} \forall n_2 \in \mathbb{N}\, \exists n_3 \in \mathbb{N} \ldots \theta_d n_d \in \mathbb{N} : \mathbb{R} \models \varphi_{\langle n_1,\ldots,n_d\rangle, m}(\vec{x})\right\}$$

where $(\varphi_{n,m})$ ranges over all double sequences of systems of in-/equalities of polynomials over (the same but also arbitrarily varying) finite real number field extension.

Proposition 17 ([Cuck92, Theorem 2.11]). *A set $\mathbb{L} \subseteq \mathbb{R}^*$ belongs to BCSS-Σ_{d+1} iff it is BCSS–semidecidable relative to BCSS–$\emptyset^{(d)}$.*

Interestingly this result, although a counterpart to Post's Theorem, is proven very differently from the latter. [Cuck92, THEOREM 2.15] also establishes a real analogue of FIN to be Σ_2–complete in the BCSS setting.

Every open subset of $\mathbb{R}^m$ is BCSS semi-decidable. Conversely every semi-decidable subset of $\mathbb{R}^m$ is a countable union of closed sets and hence in $\mathbf{\Sigma}_2(\mathbb{R}^k)$, from which [Cuck92, THEOREM 4.3] concludes that the BCSS Arithmetical Hierarchy coincides with the Borel Hierarchy $\bigcup_{d\in\mathbb{N}} \mathbf{\Sigma}_d$.

3.6 Transfinite Levels

The finite Borel classes $\mathbf{\Sigma}_d$, $d \in \mathbb{N}$, can and must be extended to *trans*finite countable ordinals d in order to obtain closure under (both complement *and*) countable unions and to yield the σ–algebra of Borel sets [Kech95, SECTION 22]. Strictly exceeding it, the class $\mathbf{\Sigma}^1_1$ of so-called analytic[2] sets consists of all projections of closed subsets of $\mathbb{R}^k \times (\mathbb{N}^\mathbb{N})$ [Kech95, SECTION 14.A]. However by SOUSLIN'S THEOREM, a set is Borel iff it as well as its complement is analytic [Kech95, SECTION 14.C]. This is the starting point of the Projective Hierarchy $\mathbf{\Delta}^1_1 \subsetneq \mathbf{\Sigma}^1_1 \subsetneq \mathbf{\Delta}^1_2 \subsetneq \mathbf{\Sigma}^1_2 \subsetneq \ldots$ It is complemented on the effective (i.e. light-face) side by the Analytical Hierarchy $\Delta^1_1 \subsetneq \Sigma^1_1 \subsetneq \Delta^1_2 \subsetneq \Sigma^1_2 \subsetneq \ldots$ As with the bold-face variant, the class Δ^1_1 of so-called hyperarithmetical sets *strictly* contains the finite arithmetical hierarchy $\bigcup_{d\in\mathbb{N}} \Sigma_d$ and coincides with the case of d running through all *recursive* ordinals [Roge67, §16.4].

[Barm03] has extended ZHENG and WEIHRAUCH's arithmetical hierarchy of real numbers (recall Section 3.2) to transfinite levels. *Non*-deterministic real computation in the sense of IMMERMAN–SZELEPSCÉNYI [Zie05a, SECTION 5] has been revealed to have power coinciding with Δ^1_1 [Zieg06].

Nondeterministic BCSS computation can, by means of real quantifier elimination, be simulated deterministically. Tarski however does not apply to *oracle* computation. As a matter of fact, nondeterministic BCSS computation relative to $\mathbb{H}$ already has the capability to decide every Borel set, i.e. whole $\mathbf{\Delta}^1_1$ [Cuck92, COROLLARY 4.5].

4 Real Hypercomputation Below the Halting Problem

Section 3 has focused on hypercomputers at least as powerful the Halting problem. An opposite direction of research investigates the recursive fine structure below H, initiated by

4.1 Post's Problem

Undecidability proofs usually proceed by reduction from the Halting problem; compare e.g. the famous results regarding Hilbert's Tenth [Mati70] or the Word Problem for Groups [Novi59, Boon58]. Already in 1944, EMIL L. POST asked whether a set $L \subseteq \mathbb{N}$ can be undecidable (and semi-decidable) yet lack reducibility

[2] Not to be confused with sets in the analytic*al* hierarchy.

from H. It was answered to the positive in 1956/57 independently by MUCHNIK and FRIEDBERG [Frie57]. Their proofs are based on a new method of diagonalization, cf. e.g. [Soar87, CHAPTERS V to VII] or [ScPr98, THEOREM 1.1].

A far more explicit solution is feasible for BCSS machines: one can show that the subset $\mathbb{Q}$ of rational reals is undecidable and semi-decidable but *not* reducible from the BCSS Halting problem $\mathbb{H}$ [MeZi05]. Similar results have been obtained in [MeZi06, SECTION 2] and [Gass06] for a *linear* variant of BCSS machines lacking multiplication and division. This brings us to the topic of

4.2 BCSS Machines with Alternative Sets of Operations

A 'classical' BCSS machine can perform addition, subtraction, multiplication, division, and (exact) comparisons on real numbers. Other sets of operations and their computational power have been studied as well. We have already mentioned *linear* BCSS Machines which, by omitting the capability of obtaining exponentially large numbers via repeated multiplication, are more closely related to the bit-oriented Turing machines [Koir94]. Also in order to make the model more realistic, [BrHe98] has restricted the semantics of real comparisons.

A simple but very interesting result with respect to real hypercomputation, the sine-function as a primitive enables a BCSS machine to decide the (otherwise undecidable, recall Section 4.1) set $\mathbb{Q}$ of rational numbers [Meer93, THEOREM 1.1]. Closely related, one may include the exponential function as an operation (e.g. in order to obviate criticism of the model like [Brat00]). Then, however, quantifier elimination provably fails [Drie84]. The question "$\mathsf{NP}_{\mathbb{R}}^{\exp} \subseteq \mathsf{REC}_{\mathbb{R}}^{\exp}$?" whether *non*-deterministic BCSS computation *with* exponentiation can still be simulated deterministically—e.g. by means of Schanuel's conjectured generalization of the Lindemann–Weierstraß Theorem [Maci91]—remains open [MaWi96].

4.3 BCSS Machines with Restricted Constants

Because of Example 1b), the BCSS machine is sometimes already counted as a hypercomputer. On the other hand, this example does not apply to the *real* Halting problem. So, even equipped with arbitrary non-recursive constants, the computational power remains below $\mathbb{H}$. On the other hand each additional permitted such constant provably does increase the capabilities of a BCSS machine [MeZi06, THEOREMS 5 and 11]. Recall from Section 2.1 that the lack of a real pairing function prevents us 'combining' several reals into a single one.

4.4 Below Naively Recursive Reals

Concerning Recursive Analysis, we have seen that oracle access to H corresponds to proceeding from computable to naively computable real numbers; recall Proposition 10. Hence a class of reals located *strictly* between $\Delta_1(\mathbb{R})$ and $\Delta_2(\mathbb{R})$ constitutes a notion hypercomputation below the Halting problem. The vast variety of results in that respect, mostly obtained by X. ZHENG et al., is the topic of an entire survey of its own [Zhen07].

5 Conclusion

Real Hypercomputation is a very interesting area of research combining real computability theory with hypercomputation. Different from the discrete realm, there are (at least) *two* major models of real computation to start with. A standard way of enhancing such a machine beyond the Church-Turing Hypothesis proceeds by equipping it with access to some undecidable oracle, leading to the notion of *degree*s. However, due to the nature of real number computation, other natural extensions make sense as well: recall Sections 3.4, 4.2, and 4.3. A systematic investigation of their relation to degrees has only started [Meer93, Section 2], [Mark96], [BoVi99], [MeZi06, Section 3].

Reflecting the real number background, proofs are generally based on logical as well as on topological and algebraic arguments.

5.1 Omissions

For reasons of conciseness the present survey has, deliberately as well as inadvertendly, neglected many contributions related to real hypercomputation; e.g. infinite time machines [HaLe00] or higher recursion theory [Sack90].

References

[ACP04] Adamyan, V.A., Calude, C.S., Pavlov, B.S.: Transcending the limits of Turing computability. In: Hida, T., Saito, K., Si, S. (eds.) Proc. Meijo Winter School, pp. 119–137. World Scientific, Singapore (2003)

[Barm03] Barmpalias, G.: A Transfinite Hierarchy of Reals. Mathematical Logic Quarterly 49(2), 163–172 (2003)

[BCS97] Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. Springer, Heidelberg (1997)

[BCSS98] Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, Heidelberg (1998)

[BeTu06] Beggs, E.J., Tucker, J.V.: Can Newtonian systems, bounded in space, time, mass and energy compute all functions? Theoretical Computer Science (to appear 2007)

[BKOS00] de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Heidelberg (2000)

[BoCa07] Bournez, O., Campagnolo, M.: A Survey On Continuous Time Computations. In: submitted as a chapter of the book New Computational Paradigms, Springer, Heidelberg (2007)

[Boon58] Boone, W.W.: The word problem. Proc. Nat. Acad. Sci. U.S.A 44, 265–269 (1958)

[BoVi99] Boldi, P., Vigna, S.: Equality is a Jump. in Theoretical Computer Science 219, 49–64 (1999)

[BPR03] Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer, Heidelberg (2003)

[Brat96] Brattka, V.: Recursive Characterization of Computable Real-Valued Functions and Relations. in Theoretical Computer Science 162, 45–77 (1996)

[Brat00] Brattka, V.: The Emperor's New Recursiveness: the Epigraph of the Exponential Function in Two Models of Computability. In: Ito, M., Imaoka, T. (eds.) Words, Languages & Combinatorics, vol, vol. III, pp. 63–72. World Scientific Publishing, Singapore (2000)

[Brat05] Brattka, V.: Effective Borel measurability and reducibility of functions. in Mathematical Logic Quarterly 51, 19–44 (2005)

[BrHe98] Brattka, V., Hertling, P.: Feasible real random access machines. Journal of Complexity 14(4), 490–526 (1998)

[BrHe02] Brattka, V., Hertling, P.: Topological Properties of Real Number Representations. Theoretical Computer Science 284, 241–257 (2002)

[BSS89] Blum, L., Shub, M., Smale, S.: On a Theory of Computation and Complexity over the Real Numbers: $\mathcal{NP}$-Completeness, Recursive Functions, and Universal Machines. in Bulletin of the American Mathematical Society (AMS Bulletin) 21, 1–46 (1989)

[ChHo99] Chadzelek, T., Hotz, G.: Analytic Machines. Theoretical Computer Science 219, 151–165 Elsevier, Amsterdam (1999)

[CMC02] Campagnolo, M.L., Moore, C., Costa, J.F.: An analog characterization of the Grzegorczyk hierarchy. in Journal of Complexity 18, 977–1000 (2002)

[Cope02] Copeland, J.: Hypercomputation. In: Minds and Machines, vol. 12, pp. 461–502. Kluwer, Dordrecht (2002)

[Cuck92] Cucker, F.: The Arithmetical Hierarchy over the Reals. Journal of Logic and Computation 2(3), 375–395 (1992)

[Drie84] van den Dries, L.: Remarks on Tarski's problem concerning $(\mathbb{R}, +, \times, \exp)$. In: Longi, G., Longo, G., Marcja, A. (eds.) Logic Colloquium '82, North-Holland, Amsterdam (1984)

[Esca96] Escardó, M.H.: PCF extended with real numbers. Theoretical Computer Science 162, 79–115 (1996)

[EtNe02] Etesi, G., Németi, I.: Non-Turing Computations Via Malament-Hogarth Space-Times. in International Journal of Theoretical Physics 41(2), 341–370 (2002)

[Frie57] Friedberg, R.M.: Two recursively enumerable sets of incomparable degrees of unsolvability. Proc. Natl. Acad. Sci. 43 43, 236–238 (1957)

[GaGe03] Gathen, J. v. z., Gerhard, J.: Modern Computer Algebra 2nd Edition, Cambridge (2003)

[Gass06] Gassner, C.: The Addititve Halting Problem is Not Decidable by means of the Rationals as an Oracle, pre-print

[Gero86] Geroch, R., Hartle, J.B.: Computability and Physical Theories. Foundations of Physics 16(6), 533–550 (1986)

[GrCo03] Graça, D.S., Costa, J.F.: Analog computers and recursive functions over the reals. Journal of Complexity 19, 644–664 (2003)

[Grze57] Grzegorczyk, A.: On the Definitions of Computable Real Continuous Functions. Fundamenta Mathematicae 44, 61–77 (1957)

[HaLe00] Hamkins, J.D., Lewis, A.: Infinite Time Turing machines. Journal of Symbolic Logic 65(2), 567–604 (2000)

[Hinm78] Hinman, P.G.: Recursion-Theoretic Hierarchies. In: Perspectives in Mathematical Logic, Springer, Heidelberg (1978)

[Ho99] Ho, C.-K.: Relatively recursive reals and real functions. Theoretical Computer Science 210, 99–120 (1999)

[Hoga92] Hogarth, M.L.: Does General Relativity Allow an Observer to View an Eternity in a Finite Time? Foundations of Physics Letters 5(2), 173–181 (1992)

[HVS95] Hotz, G., Vierke, G., Schieffer, B.: Analytic Machines, Electronic Colloquium on Computational Complexity vol. 025 (1995)

[Kawa05] Kawamura, A.: Type-2 Computability and Moore's Recursive Functions. Electronic Notes in Theoretical Computer Science 120, 83–95 (2005)

[Kech95] Kechris, A.S.: Classical Descriptive Set Theory. In: Graduate Texts in Mathematics, Springer, Heidelberg (1995)

[Kie04b] Kieu, T.D.: A reformulation of Hilbert's tenth problem through quantum mechanics. Proc. Royal Soc. A 460, 1535–1545 (2004)

[Koir94] Koiran, P.: Computing over the Reals with Addition and Order. Theoretical Computer Science 133, 35–48 (1994)

[Ko91] Ko, K.: Complexity Theory of Real Functions, Birkhäuser (1991)

[Koep01] Koepf, W.: Besprechungen zu Büchern der Computeralgebra: Klaus Weihrauch Computable Analysis. Computeralgebra Rundbrief 29, 29 (2001), http://fachgruppe-computeralgebra.de/CAR/CAR29/node19.html

[KrWe85] Kreitz, C., Weihrauch, K.: Theory of representations. Theoretical Computer Science 38, 35–53 (1985)

[Laco57] Lacombe, D.: Les ensembles récursivement ouverts ou fermés, et leurs applications à l'analyse récursive, pp. 1040–1043 in Compt. Rend. Acad. des Sci. Paris vol. 245 (1957); sequel pp. 28–31 in Compt. Rend. Acad. des Sci. Paris, vol. 246 (1958)

[Lamb05] Lambov, B.: RealLib: an Efficient Implementation of Exact Real Arithmetic, to appear in Mathematical Structures in Computer Science

[Maci91] Macintyre, A.: Schanuel's conjecture and free exponential rings. Ann. Pure Appl. Logic 51, 241–246 (1991)

[Mark96] Marker, D.: Model Theory and Exponentiation, Notices of the AMS, 753–759 (1996)

[MaWi96] Macintyre, A., Wilkie, A.J.: On the decidability of the real exponential field, in Kreiseliana. About and Around Georg Kreisel, A.K. Peters pp. 441–467 (1996)

[Mati70] Matiyasevich, Y.V.: Enumerable Sets are Diophantine. Soviet Math.Dokl 11, 354–357 (1970)

[Meer93] Meer, K.: Real Number Models under Various Sets of Operations. Journal of Complexity 9, 366–372 (1993)

[MeZi05] Meer, K., Ziegler, M.: An Explicit Solution to Post's Problem over the Reals. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 456–467. Springer, Heidelberg (2005)

[MeZi06] Meer, K., Ziegler, M.: Uncomputability below the Real Halting Problem. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 368–377. Springer, Heidelberg (2006)

[Moor96] Moore, C.: Recursion theory on the reals and continuous-time computation. Theoretical Computer Science 162, 23–44 (1996)

[Mosc80] Moschovakis, Y.N.: Descriptive Set Theory. In: Studies in Logic, North-Holland, Amsterdam (1980)

[Muel01] Müller, N.: The iRRAM: Exact Arithmetic in C++. In: Blank, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 222–252. Springer, Heidelberg (2001)

[Novi59] Novikov, P.S.: On the algorithmic unsolvability of the word problem in group theory. Trudy Mat. Inst. Steklov 44, 1–143 (1959)

[Ord02] Ord, T.: Hypercomputation: computing more than the Turing machine, Honour's Thesis, University of Melbourne (2002)

[Orpo97] Orponen, P.: A survey of continuous-time computation theory. In: Du, D.-Z., Ko, K.-I. (eds.) Advances in Algorithms, Languages, and Complexity, pp. 209–224. Kluwer, Dordrecht (1997)

[PERi89] Pour-El, M.B., Richards, J.I.: Computability in Analysis and Physics. Springer, Heidelberg (1989)

[Roge67] Rogers, J.H.: Theory of Recursive Functions and Effective Computability. Series in Higher Mathematics. McGraw-Hill, New York (1967)

[Ross93] Cucker, F., Rosselló, F.: Recursiveness over the Complex Numbers is Time-Bounded. In: Shyamasundar, R.K. (ed.) Foundations of Software Technology and Theoretical Computer Science, vol. 761, pp. 260–267. Springer, Heidelberg (1993)

[Sack90] Sacks, G.: Higher Recursion Theory. Springer, Heidelberg (1990)

[Scho79] Schönhage, A.: On the Power of Random Access Machines. In: Maurer, H.A. (ed.) Automata, Languages, and Programming. LNCS, vol. 71, pp. 520–529. Springer, Heidelberg (1979)

[Scot70] Scott, D.S.: Outline of a Mathematical Theory of Computation. In: Technical Monograph PRG-2, Oxford University, Oxford (1970)

[ScPr98] Schöning, U., Pruim, R.: Gems of Theoretical Computer Science. Springer, Heidelberg (1998)

[Smit06] Smith, W.D.: Church's Thesis meets the N-body Problem. J. Applied Mathematics and Computation 178, 154–183 (2006)

[Soar87] Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, Heidelberg (1987)

[Spec49] Specker, E.: Nicht konstruktiv beweisbare Sätze der Analysis. Journal of Symbolic Logic 14(3), 145–158 (1949)

[Tuck80] Tucker, J.V.: Computability and the algebra of fields. J. Symbolic Logic 45, 103–120 (1980)

[Turi36] Turing, A.M.: On Computable Numbers, with an Application to the Entscheidungsproblem. Proc. London Math. Soc 42(2), 230–265 (1936)

[Turi37] Turing, A.M.: On Computable Numbers, with an Application to the Entscheidungsproblem. A correction. Proc. London Math. Soc 43(2), 544–546 (1937)

[Turi39] Turing, A.M.: Systems of Logic Based on Ordinals. Proc. London Math. Soc 45, 161–228 (1939)

[TuZu00] Tucker, J.V., Zucker, J.I.: Computable functions and semicomputable sets on many-sorted algebras. In: Abramsky, S., Gabbay, D.M., Maybaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. 5, pp. 317–523. Oxford Science Publications, Oxford (2000)

[Weih95] Weihrauch, K.: A Simple Introduction to Computable Analysis, Monographs of the Electronic Colloquium on Computational Complexity (1995)

[Weih00] Weihrauch, K.: Computable Analysis. Springer, Heidelberg (2000)

[WiLe02] Wiedermann, J., van Leeuwen, J.: Relativistic Computers and Non-uniform Complexity Theory. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.) UMC 2002. LNCS, vol. 2509, pp. 287–299. Springer, Heidelberg (2002)

[Zhen07] Zheng, X.: On the hierarchy of Δ_2 real numbers. Theoretical Informatics and Application (to appear)

[ZhWe01] Zheng, X., Weihrauch, K.: The Arithmetical Hierarchy of Real Numbers. Mathematical Logic Quarterly 47, 51–65 (2001)

[Zie05a] Ziegler, M.: Computability and Continuity on the Real Arithmetic Hierarchy. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 562–571. Springer, Heidelberg (2005)

[Zie05b] Ziegler, M.: Computational Power of Infinite Quantum Parallelism. International Journal of Theoretical Physics 44, 2059–2071 (2005)

[Zieg06] Ziegler, M.: Revising Type-2 Computation and Degrees of Discontinuity. In: Proc. 3rd International Conference on Computability and Complexity in Analysis pp. 347–366

Author Index