Paola Bonizzoni
Vasco Brattka
Benedikt Löwe (Eds.)

The Nature of Computation

Logic, Algorithms, Applications

LNCS 7921

9th Conference on Computability in Europe, CiE 2013
Milan, Italy, July 2013
Proceedings

Springer

Lecture Notes in Computer Science 7921

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Paola Bonizzoni Vasco Brattka
Benedikt Löwe (Eds.)

The Nature of Computation

Logic, Algorithms, Applications

9th Conference on Computability in Europe, CiE 2013
Milan, Italy, July 1-5, 2013
Proceedings

Springer

Volume Editors

Paola Bonizzoni
Università Degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milan, Italy
E-mail: bonizzoni@disco.unimib.it

Vasco Brattka
Universität der Bundeswehr
Institute for Theoretical Computer Science, Mathematics and Operations Research
Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany
and University of Cape Town, Department of Mathematics and Applied Mathematics
Private Bag X3, Rondebosch 7701, South Africa
E-mail: vasco.brattka@cca-net.de

Benedikt Löwe
Universiteit van Amsterdam, Institute for Logic, Language and Computation
Postbus 94242, 1090 GE Amsterdam, The Netherlands
and Universität Hamburg, Fachbereich Mathematik
Bundesstraße 55, 20146 Hamburg, Germany
E-mail: b.loewe@uva.nl

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-39052-4 e-ISBN 978-3-642-39053-1
DOI 10.1007/978-3-642-39053-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013940615

CR Subject Classification (1998): F.2, F.1, G.2, I.1

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

CiE 2013: The Nature of Computation

Computability in Europe 2013 (CiE 2013) followed the *Turing Centenary Conference* CiE 2012 in celebrating the enormous influence of Turing's work on the specific focus of the CiE conference series: the development of a multi-disciplinary and modern view of computation and computability. The interest for computation in nature (which was also the motivation for Turing's work on biological pattern formation) as reflected in the title of CiE 2013, *The Nature of Computation*, connects biology and computer science and has given rise to modern disciplines of research as well as new perspectives on computation.

In particular, CiE 2013 was focused on the unexpected changes that studies on nature have brought to several areas of mathematics, physics, and computer science. Two complementary research perspectives pervade the *Nature of Computation* theme. One is focused on the understanding of new computational paradigms, inspired by processes occurring in the biological world and resulting in a deeper and modern understanding of the theory of computation. The other perspective is on our understanding of how computations really occur in nature, on how we can interact with these computations, and their applications.

CiE 2013 was the ninth meeting in the conference series *Computability in Europe* organized by the Association CiE. The association promotes the development of computability-related science, ranging from mathematics, computer science and applications in various natural and engineering sciences, such as physics and biology, as well as the promotion of related fields, such as philosophy and history of computing. In particular, the conference series successfully brings together the mathematical, logical, and computer sciences communities that are interested in developing computability-related topics. This year this scope was strengthened by the co-location of CiE 2013 with UCNC 2013 (*Unconventional Computation and Natural Computation*), with Giancarlo Mauri as Chair of the Programme Committee.

The two conferences, CiE 2013 and UCNC 2013, were held at the University of Milano-Bicocca in Milan, Italy. They shared one plenary invited talk, given by Endre Szemerédi (Budapest and Piscataway NJ), winner of the Abel Prize in 2012, and two tutorials, one by Grzegorz Rozenberg (Leiden and Boulder CO) and one by Gilles Brassard (Montréal QC). Moreover, some satellite events were organized around the two conferences.

The eight previous CiE conferences were held in Amsterdam (The Netherlands) in 2005, Swansea (Wales) in 2006, Siena (Italy) in 2007, Athens (Greece) in 2008, Heidelberg (Germany) in 2009, Ponta Delgada (Portugal) in 2010, Sofia (Bulgaria) in 2011, and Cambridge (UK) in 2012. The proceedings of these meetings were all published in the Springer series *Lecture Notes in Computer Science*. The annual CiE conference has become a major event and is the largest international meeting focused on computability theoretic issues. The next meeting in 2014 will be held in Budapest (Hungary).

The series is coordinated by the CiE Conference Series Steering Committee consisting of Luís Antunes (Porto, Secretary), Arnold Beckmann (Swansea), Laurent Bienvenu (Paris), Natasha Jonoska (Tampa FL), Viv Kendon (Leeds), Benedikt Löwe (Amsterdam and Hamburg, Chair), Mariya Soskova (Sofia and Berkeley CA), and Peter van Emde Boas (Amsterdam).

The Programme Committee of CiE 2013 was responsible for the selection of the invited speakers, the special session organizers and for running the reviewing process of all submitted regular contributions.

The Programme Committee invited six speakers to give plenary lectures: Ulle Endriss (Amsterdam), Lance Fortnow (Atlanta GA), Anna Karlin (Seattle WA), Bernard Moret (Lausanne), Mariya Soskova (Sofia and Berkeley CA), and Endre Szemerédi (Budapest and Piscataway NJ; joint invitee of CiE 2013 and UCNC 2013).

These plenary speakers were invited to publish abstracts or papers in this volume. Karlin's lecture was the *2013 APAL Lecture* funded by Elsevier, Fortnow's lecture was the *2013 EACSL Lecture* funded by the *European Association for Computer Science Logic*, and Szemerédi's lecture was funded by the Department of Mathematics and its Applications of the University of Milano-Bicocca. In addition to the plenary lectures, the conference had two tutorials by Gilles Brassard (Montréal QC) and Grzegorz Rozenberg (Leiden and Boulder CO).

Springer-Verlag generously funded two awards that were given during the CiE 2013 conference. Nicolas de Rugy-Altherre was awarded the *Best Student Paper Award* for his paper "Determinant Versus Permanent: Salvation via Generalization?" Shankara Narayanan Krishna, Marian Gheorghe, and Ciprian Dragomir were awarded the *Best Paper on Natural Computing Award* for their paper "Some Classes of Generalised Communicating P Systems and Simple Kernel P Systems."

CiE 2013 had six special sessions: two sessions, *Computational Molecular Biology* and *Computation in Nature*, were devoted to the special focus of CiE 2013. In addition to this, new challenges arising in computations in the real world were faced in the session on *Data Streams and Compression*. The remaining

three sessions were on *Algorithmic Randomness*, *Computational Complexity in the Continuous World*, and *History of Computation*. Speakers in these special sessions were selected by the special session organizers and could contribute a paper to this volume.

We received 82 submissions which were reviewed by the Programme Committee and many expert referees. In the end, 31.7% of the submitted papers were accepted for publication in this volume. Without the help of our expert referees, the production of the volume would have been impossible. We would like to thank all the subreviewers for their excellent work; their names are listed in an appendix of this preface.

All authors who contributed to this conference were encouraged to submit significantly extended versions of their papers with unpublished research content to *Computability—The Journal of the Association CiE*.

The conference CiE 2013 was organized by Stefano Beretta (Milan), Paola Bonizzoni (Milan), Gianluca Della Vedova (Milan), Alberto Dennunzio (Milan), Riccardo Dondi (Bergamo), Giancarlo Mauri (Milan), Yuri Pirola (Milan), and Raffaella Rizzi (Milan).

The Steering Committee of the conference series CiE is concerned about the representation of female researchers in the field of computability. In order to increase female participation, the series started the *Women in Computability* (WiC) program in 2007, first funded by the *Elsevier Foundation*, then taken over by the publisher *Elsevier*. We were proud to continue this program with its successful annual WiC workshop and a grant program for junior female researchers in 2013.

The organizers of CiE 2013 would like to acknowledge and thank the following entities for their essential financial support (in alphabetic order): the *Association for Symbolic Logic* (ASL), the *Department of Mathematics and its Applications* and the *Department of Informatics, Systems and Communication*, both of the University of Milano-Bicocca, *Elsevier B.V.*, the *European Association for Computer Science Logic* (EACSL), the *European Association for Theoretical Computer Science* (EATCS), *IOS Press*, *Springer-Verlag* and the *University of Milano-Bicocca*. We would also like to acknowledge the support of our non-financial sponsors, the *Association Computability in Europe* (CiE).

May 2013

Paola Bonizzoni
Vasco Brattka
Benedikt Löwe

Organization

Programme Committee

Gerard Alberts	Amsterdam
Luís Antunes	Porto
Arnold Beckmann	Swansea
Laurent Bienvenu	Paris
Paola Bonizzoni	Milan, Co-chair
Vasco Brattka	Munich and Cape Town, Co-chair
Cameron Buckner	Houston TX
Bruno Codenotti	Pisa
Stephen Cook	Toronto ON
Barry Cooper	Leeds
Ann Copestake	Cambridge
Erzsébet Csuhaj-Varjú	Budapest
Anuj Dawar	Cambridge
Gianluca Della Vedova	Milan
Liesbeth De Mol	Ghent
Jérôme Durand-Lose	Orléans
Viv Kendon	Leeds
Bjørn Kjos-Hanssen	Honolulu HI
Antonina Kolokolova	St. John's NF
Benedikt Löwe	Amsterdam and Hamburg
Giancarlo Mauri	Milan
Rolf Niedermeier	Berlin
Geoffrey Pullum	Providence RI and Edinburgh
Nicole Schweikardt	Frankfurt
Sonja Smets	Amsterdam
Susan Stepney	York
S.P. Suresh	Chennai
Peter van Emde Boas	Amsterdam

Special Sessions

Algorithmic Randomness

Organizers. Mathieu Hoyrup (Nancy) and André Nies (Auckland).
Speakers. Johanna Franklin (Storrs CT), Noam Greenberg (Wellington), Joseph S. Miller (Madison WI), Nikolay Vereshchagin (Moscow).

Computational Complexity in the Continuous World

Organizers. Akitoshi Kawamura (Tokyo) and Robert Rettinger (Hagen).
Speakers. Mark Braverman (Princeton NJ), Daniel S. Graça (Faro), Joris van der Hoeven (Palaiseau), Chee K. Yap (New York NY).

Computational Molecular Biology
Organizers. Alessandra Carbone (Paris) and Jens Stoye (Bielefeld).
Speakers. Sebastian Böcker (Jena), Marília D.V. Braga (Duque de Caxias), Andrea Pagnani (Torino), Laxmi Parida (Yorktown Heights NY).

Computation in Nature
Organizers. Mark Delay (London ON) and Natasha Jonoska (Tampa FL).
Speakers. Jerome Durand-Lose (Orléans), Giuditta Franco (Verona), Lila Kari (London ON), Darko Stefanovic (Albuquerque NM).

Data Streams and Compression
Organizers. Paolo Ferragina (Pisa) and Andrew McGregor (Amherst MA).
Speakers. Graham Cormode (Florham Park NJ), Irene Finocchi (Rome), Andrew McGregor (Amherst MA), Marinella Sciortino (Palermo).

History of Computation
Organizers. Gerard Alberts (Amsterdam) and Liesbeth De Mol (Ghent).
Speakers. David Alan Grier (Washington DC), Thomas Haigh (Milwaukee WI), Ulf Hashagen (Munich), Matti Tedre (Stockholm).

Additional Reviewers

Abu Zaid, Faried
Aizawa, Kenneth
Allo, Patrick
Almeida, Marco
Aman, Bogdan
Anderson, Matthew
Andrews, Paul
Arkhipov, Alex
Arrighi, Pablo
Badillo, Liliana
Barmpalias, George
Barr, Katie
Bauer, Andrej
Bauwens, Bruno
Bernardinello, Luca
Bès, Alexis
Besozzi, Daniela
Block, Alexander
Bonfante, Guillaume
Braverman, Mark
Bredereck, Robert
Brenguier, Romain
Bridges, Douglas
Brijder, Robert
Briseid, Eyvind
Bulteau, Laurent
Calvert, Wesley
Caravagna, Giulio
Carl, Merlin
Carlucci, Lorenzo
Carton, Olivier
Case, John
Chen, Jiehua
Chivers, Howard
Colombo, Riccardo
Cormode, Graham
Correia, Luis
Dahmani, François
De Paiva, Valeria
Delhommé, Christian
Dennunzio, Alberto
Diener, Hannes
Dondi, Riccardo
Dorais, Francois
Dowek, Gilles
Droop, Alastair
Dyckhoff, Roy
Eckert, Kai
Effros, Michelle
Eguchi, Naohi
Escardó, Martín
Ferretti, Claudio
Finkel, Alain
Fiorino, Guido
Fisseni, Bernhard
Fokina, Ekaterina
Franco, Giuditta
Franklin, Johanna N.Y.
Franks, Dan
Fraser, Robert
Freer, Cameron
Frehse, Goran
Friedman, Sy-David
Froese, Vincent
Gács, Péter
Gagie, Travis
Gärtner, Bernd
Gheorghe, Marian
Gherardi, Guido
Ghosh, Sujata
Gierasimczuk, Nina
Givors, Fabien
Goldberg, Paul
Gore, Rajeev
Górecki, Paweł

Goubault-Larrecq, Jean
Gutiérrez-Naranjo, Miguel A.
Haase, Christoph
Hansen, Jens Ulrik
Harizanov, Valentina
Hartung, Sepp
Hertling, Peter
Heunen, Chris
Hickinbotham, Simon
Hines, Peter
Hirschfeldt, Denis
Hölzl, Rupert
Hoyrup, Mathieu
Hüffner, Falk
Hutter, Marcus
Inamdar, Tanmay
Ivan, Szabolcs
Kabanets, Valentine
Kalimullin, Iskander
Kari, Jarkko
Kari, Lila
Kato, Yuki
Kawamura, Akitoshi
Kent, Tom
Kihara, Takayuki
Kishida, Kohei
Kleijn, Jetty
Klincewicz, Michal
Kolde, Raivo
Komusiewicz, Christian
Kulkarni, Manasi
Kullmann, Oliver
Lange, Karen
Lazic, Ranko
Le Gloannec, Bastien
Leal, Raul
Leporati, Alberto
Li, Zhenhao
Loeb, Iris
Lovett, Neil
Malcher, Andreas
Manea, Florin
Manzoni, Luca
Margenstern, Maurice
Marks, Andrew
Martiel, Simon
Martini, Simone
Maurino, Andrea
Michaelson, Greg
Miller, Russell
Montalban, Antonio
Moore, Cris
Morozov, Andrei
Mota, Francisco
Mummert, Carl
Nguyen, Nam
Nguyen, Paul K. L. V.
Nichterlein, André
Nies, André
Nobile, Marco
Noessner, Jan
Nordvall Forsberg, Fredrik
O'Keefe, Simon
Okhotin, Alexander
Panangaden, Prakash
Paun, Gheorghe
Pavesi, Giulio
Pescini, Dario
Pike, David
Pirola, Yuri
Porreca, Antonio
Poulding, Simon
Pouly, Amaury
Primiero, Giuseppe
Puzarenko, Vadim
Rizzi, Raffaella
Romashchenko, Andrei
Rothe, Jörg
Russell, Benjamin
Rute, Jason
Salomaa, Kai T.
Sampson, Adam
San Mauro, Luca
Sankur, Ocan
Sapir, Mark
Savani, Rahul
Sciortino, Marinella
Seisenberger, Monika
Seki, Shinnosuke
Sequoiah-Grayson, Sebastian
Sergioli, Giuseppe
Seyfferth, Benjamin
Shafer, Paul
Shao, Wen
Shen, Alexander
Shlapentokh, Alexandra
Simmons, Harold
Simon, Sunil Easaw
Simonnet, Pierre
Skordev, Dimiter
Solomon, Reed
Sorbi, Andrea
Sorge, Manuel
Soskova, Alexandra
Sourabh, Sumit
Souto, André
Sprevak, Mark
Stamatiou, Yannis
Stannett, Mike
Steffen, Bernhard
Stephan, Frank
Suchy, Ondra
Szymanik, Jakub
Thierauf, Thomas
Thomson, Jimmy
Tsigaridas, Elias
Uckelman, Joel
van Bevern, René
van den Berg, Benno
Vatev, Stefan
Verlan, Sergey
Vicary, Jamie
Wareham, Todd
Weihrauch, Klaus
Weller, Mathias
Yakoubsohn, Jean-Claude
Yokoyama, Keita
Zandron, Claudio
Ziegler, Martin
Zoppis, Italo

Ivan Nikolaev Soskov
(23 September 1954 – 5 May 2013)

During the preparations for CiE 2013, the shocking news reached the *Computability in Europe* community that our colleague Ivan Soskov had unexpectedly died. Ivan had been one of the important forces behind the CiE conference series, from its very beginnings as an informal European network, and his death will affect the conference series. We lost one of our staunch supporters, a very dear colleague and a good friend.

Ivan Soskov was born on 23 September 1954 in the southern Bulgarian city of Stara Zagora and went to the *National High School of Mathematics and Sciences* "Академик Любомир Чакалов", a school founded by the Faculty of Mathematics of Sofia University to offer an introduction to mathematics at the highest level to talented pupils. After his graduation, he studied at Sofia University where he graduated in 1979 with a degree in Mathematical Logic. He became a doctoral student of Dimiter Skordev and defended his PhD thesis entitled *Computability in Partial Algebraic Systems* in 1983.

Ivan spent his career at Sofia University, starting as a programmer in the *Computing Laboratory* of Sofia University, then becoming an Assistant Professor in the *Laboratory of Applied Logic*, later at the Faculty for Mathematics and Computer Science. In 1991, he was promoted to Associate Professor and spent two years at the University of California at Los Angeles (1991–1993). After he had obtained the higher doctorate (DSc) in 2001, he was promoted to Full Professor in 2005. He was head of the Department for Mathematical Logic and Applications from 2000 to 2007 and Dean of the Faculty of Mathematics and Computer Science from 2007 until his death. For many years, he has played a major role in the research administration of Sofia University, fulfilling many important tasks such as that of the chair of the Council of Deans of the University.

Ivan's research field was classical computability theory, in particular degree spectra and enumeration degrees. He has supervised 15 Master's students in subjects related to his research expertise and had three doctoral students, Stela Nikolova (1992), Vessela Baleva (2002), and Hristo Ganchev (2009); three

additional doctoral students were still working under Ivan's supervision at the time of his death. Ivan always encouraged his students to participate in CiE conferences, and in the lists of authors of papers accepted at our conferences, we repeatedly find the names of Ivan's students.

Ivan's involvement with CiE started with his role as the *coordinator* of the Bulgarian node in the informal CiE network that was formed in 2004 and developed into the conference series and association. He served as a member of the Programme Committee for the first five CiE conferences: Amsterdam in 2005, Swansea in 2006, Siena in 2007, Athens in 2008, and Heidelberg in 2009. Together with his colleagues in Sofia, he offered to host CiE 2011 in Sofia: Ivan served as one of the co-chairs of the Programme Committee for this seventh edition of CiE. His commitment to the conference series was not restricted to Programme Committees work; for this year's conference, CiE 2013, Ivan submitted a paper which is printed in this volume. Sadly, we will not be able to hear him present it in Milan. We shall miss him.

S. Barry Cooper (*President Association CiE*)
Benedikt Löwe (*Chairman Steering Committee CiE-CS*)
Stela Nikolova (*Sofia University*)

Table of Contents

Real Benefit of Promises and Advice 1
Klaus Ambos-Spies, Ulrike Brandt, and Martin Ziegler

Computability and Computational Complexity of the Evolution of Nonlinear Dynamical Systems 12
Olivier Bournez, Daniel S. Graça, Amaury Pouly, and Ning Zhong

An Overview of Genomic Distances Modeled with Indels 22
Marília D.V. Braga

Noise versus Computational Intractability in Dynamics 32
Mark Braverman

Cluster Editing 33
Sebastian Böcker and Jan Baumbach

Beyond Rogers' Non-constructively Computable Function 45
John Case and Michael Ralston

Constructing Continuous Systems from Discrete Cellular Automata 55
Julien Cervelle

Latency-Bounded Target Set Selection in Social Networks 65
Ferdinando Cicalese, Gennaro Cordasco, Luisa Gargano, Martin Milanič, and Ugo Vaccaro

Summary Data Structures for Massive Data 78
Graham Cormode

Determinant versus Permanent: Salvation via Generalization? 87
Nicolas de Rugy-Altherre

Aligning and Labeling Genomes under the Duplication-Loss Model 97
Riccardo Dondi and Nadia El-Mabrouk

Irrationality Is Needed to Compute with Signal Machines with Only Three Speeds 108
Jérôme Durand-Lose

Processes Inspired by the Functioning of Living Cells: Natural Computing Approach 120
Andrzej Ehrenfeucht and Grzegorz Rozenberg

Recent Developments in Collective Decision Making in Combinatorial Domains 123
Ulle Endriss

Software Streams: Big Data Challenges in Dynamic Program Analysis 124
Irene Finocchi

On λ-Definable Functions on Ordinals 135
Tim Fischbach and Benjamin Seyfferth

A Personal View of the P versus NP Problem 147
Lance Fortnow

An Investigation on Genomic Repeats 149
Giuditta Franco and Alessio Milanese

Local Computability for Ordinals 161
Johanna N.Y. Franklin, Asher M. Kach, Russell Miller, and Reed Solomon

A Note on the Sequential Version of Π^1_2 Statements 171
Makoto Fujiwara and Keita Yokoyama

On Conservative Learning of Recursively Enumerable Languages 181
Ziyuan Gao, Sanjay Jain, and Frank Stephan

Topology of Asymptotic Cones and Non-deterministic Polynomial Time Computations 191
Anthony Gasperin

On Decidable and Computable Models of Theories 200
Alexander Gavruskin and Bakhadyr Khoussainov

Discovering Hidden Repetitions in Words 210
Paweł Gawrychowski, Florin Manea, and Dirk Nowotka

Language Forbidding-Enforcing Systems Defining DNA Codewords 220
Daniela Genova

Computing K-Trivial Sets by Incomplete Random Sets 230
Noam Greenberg

Cardinal-Recognizing Infinite Time Turing Machines 231
Miha E. Habič

'Stored Program Concept' Considered Harmful: History and Historiography 241
Thomas Haigh

The Complexity of Interior Point Methods for Solving Discounted Turn-Based Stochastic Games 252
Thomas Dueholm Hansen and Rasmus Ibsen-Jensen

The Computation of Nature, Or: Does the Computer Drive Science and Technology? 263
Ulf Hashagen

Negative Glues and Non-determinism in Nanocomputations by Self-assembly 271
Lila Kari

Structures without Scattered-Automatic Presentation 273
Alexander Kartzow and Philipp Schlicht

Some Classes of Generalised Communicating P Systems and Simple Kernel P Systems 284
Shankara Narayanan Krishna, Marian Gheorghe, and Ciprian Dragomir

Closed Choice for Finite and for Convex Sets 294
Stéphane Le Roux and Arno Pauly

Realizability Models Separating Various Fan Theorems 306
Robert S. Lubarsky and Michael Rathjen

Towards a Theory of Homomorphic Compression 316
Andrew McGregor

The Classification Problem for Compact Computable Metric Spaces 320
Alexander G. Melnikov and André Nies

Exploiting Co-evolution across Protein Families for Predicting Native Contacts and Protein-Protein Interaction Surfaces 329
Andrea Pagnani

A Compositional Semantics of Reaction Systems with Restriction 330
Giovanni Pardini, Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Simone Tini

Using Random Graphs in Population Genomics 340
Laxmi Parida

The Tarski-Lindenbaum Algebra of the Class of All Strongly Constructivizable Countable Saturated Models 342
Mikhail G. Peretyat'kin

The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words 353
Giovanna Rosone and Marinella Sciortino

A Note on ω-Jump Inversion of Degree Spectra of Structures 365
Ivan N. Soskov

The Turing Universe in the Context of Enumeration Reducibility 371
Mariya I. Soskova

Computing Game Strategies 383
Darko Stefanovic and Milan N. Stojanovic

On Processes and Structures 393
Alexey Stukachev

Various Regularity Lemmas in Graphs and Hypergraphs 403
Endre Szemerédi

Three Debates about Computing 404
Matti Tedre

Another Jump Inversion Theorem for Structures 414
Stefan Vatev

On Algorithmic Strong Sufficient Statistics 424
Nikolay Vereshchagin

Analytic Root Clustering: A Complete Algorithm Using Soft Zero Tests 434
Chee Yap, Michael Sagraloff, and Vikram Sharma

Author Index 445

Real Benefit of Promises and Advice*

Klaus Ambos-Spies[1], Ulrike Brandt[2], and Martin Ziegler[3]

[1] Department of Mathematics and Computer Science, Ruprecht-Karls-Universität Heidelberg, Heidelberg, Germany
[2] Department of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany
[3] Department of Mathematics, Technische Universität Darmstadt, Darmstadt, Germany

Abstract. Promises are a standard way to formalize partial algorithms; and advice quantifies nonuniformity. For decision problems, the latter is captured in common complexity classes such as $\mathcal{P}/\mathrm{poly}$, that is, with advice growing in size with that of the input. We advertise constant-size advice and explore its theoretical impact on the complexity of classification problems – a natural generalization of promise problems – and on real functions and operators. Specifically we exhibit problems that, without any advice, are decidable/computable but of high complexity while, with (each increase in the permitted size of) advice, (gradually) drop down to polynomial-time.

1 Motivation

The Boolean satisfiability problem SAT is $\mathcal{NP}$–complete; but drops down to $\mathcal{P}$ when restricting to terms in 2-conjunctive normal form (2SAT). More strikingly, restricted to each 'slice' $\{0,1\}^n$ the $\mathcal{NP}$–complete Knapsack problem can be decided by an algorithm $\mathcal{A}_n$ of running time polynomial in n; cf. [2, Corollary 3.27]. In both above examples, a computable problem becomes easier when promising inputs to belong to, and thus permitting algorithms tailored for, a certain subset. Such promises occur frequently in the Theory of Computing [10, 12] formalized as Items a) & f) of

Definition 1. *Fix a subset X of $\{0,1\}^*$ or of $\mathbb{N}$.*

a) A promise problem *over X is a pair (A,B) of disjoint subsets of X. An algorithm* solves *(A,B) within time $t(n)$ if on inputs $\vec{x} \in A$ it reports 0 and 1 on inputs $\vec{x} \in B$, both after at most $\mathcal{O}\big(t(|\vec{x}|)\big)$ steps, where $|\vec{x}|$ denotes the (binary) length of $\vec{x}$. Note that the algorithm may behave arbitrarily, and even diverge, on inputs $\vec{x} \notin A \cup B$.*

* Supported in part by the *Marie Curie International Research Staff Exchange Scheme Fellowship* 294962 within the 7th European Community Framework Programme and by the *German Research Foundation* (DFG) with project Zi 1009/4-1. We acknowledge seminal discussions with Vassilis Gregoriades, Thorsten Kräling, and Hermann K.-G. Walter.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 1–11, 2013.

b) A classification problem *over X is a (finite or countable) family $\mathcal{C} = (C_j)_{j\in J}$ of subsets of X. $\mathcal{C}$ is* straight *if the C_j are pairwise disjoint. $\mathcal{C}$ is* total *if X coincides with $\bigcup\mathcal{C} := \bigcup_{j\in J} C_j$.*

c) An algorithm solves *$\mathcal{C}$ if, upon input of $\vec{x} \in \bigcup\mathcal{C}$, it produces some $j \in J$ with $\vec{x} \in C_j$, that is, if it computes (some selection of) the (multivalued, unless $\mathcal{C}$ is straight)* generalized characteristic function $\mathbf{1}_{\mathcal{C}} : \bigcup\mathcal{C} \rightrightarrows J$. *Again, the behaviour on inputs $\vec{x} \notin \bigcup\mathcal{C}$ is unspecified and in particular not necessarily divergent.*

d) A multivalued *mapping $f : X \rightrightarrows Y$ is a relation $f \subseteq X \times Y$ satisfying: $\forall \vec{x} \in X\ \exists \vec{y} \in Y :\ (\vec{x}, \vec{y}) \in f$. We identify such f with the set-valued function $f : X \to 2^Y \setminus \{\emptyset\}$.*

e) Let $\mathcal{C} = (C_j)_{j\in J}$ and $\mathcal{B} = (B_j)_{j\in J}$ denote classification problems over X and fix $Y \subseteq X$. Abbreviate $\mathcal{C} \cap Y := (C_j \cap Y)_{j\in J}$ and $\mathcal{C} \oplus \mathcal{B} := \big((\mathtt{0} \circ C_j) \cup (\mathtt{1} \circ B_j)\big)_{j\in J}$. We write "$\mathcal{B} \subseteq \mathcal{C}$" if it holds $B_j \subseteq C_j$ for every $j \in J$.

f) Let $\bar{X} = (X_k)_{k\in K}$ be a (finite or countable) partition of X. An algorithm $\mathcal{A}$ computes a (possibly multivalued) mapping $f : X \rightrightarrows Y$ with advice $\bar{X}$ *if, upon input of $(\vec{x}, k)$ with $\vec{x} \in X_k$, it produces some $\vec{y} \in f(\vec{x})$. In this case we say that $\mathcal{A}$ computes f* with $|K|$–fold advice.

g) Fix a family $\mathcal{L}$ of subsets of X (such as $\mathcal{P}$ or $\mathcal{RE}$, the recursively enumerable languages) and call a classification problem $\mathcal{C} = (C_1, \ldots, C_J)$ $\mathcal{L}$–separable *if there exist pairwise disjoint $C'_1, \ldots, C'_J \in \mathcal{L}$ such that $C_j \subseteq C'_j$.*

So promise problems are precisely the straight classification problems of cardinality two [9]; and c) describes the computational problem of *providing* the kind of advice employed in f). See also Manifesto 3 below...

1.1 Real Computation and Complexity

Computable Analysis is the theory of real computation by approximation up to guaranteed prescribable absolute error. Initiated by Turing in the very same publication that introduced 'his' machine, it formalizes validated numerics in unbounded precision and more precisely interval computations of arbitrarily prescribable absolute output error; cf., e.g., [7].

Formally, a real number x is considered *computable* if any (equivalently: all) of the following conditions hold [38, §4.1 & Lemma 4.2.1]:

(a1) The (or any) binary expansion $b_n \in \{0, 1\}$ of $x = \sum_{n\geq -N} b_n 2^{-n}$ is recursive.

(a2) A Turing machine can produce dyadic approximations to x up to prescribable error 2^{-n}, that is, compute an integer sequence $a : \mathbb{N} \ni n \mapsto a_n \in \mathbb{Z}$ (output encoded in binary) with $|x - a_n/2^{n+1}| \leq 2^{-n}$.

(a3) A Turing machine can output rational sequences (c_n) and (ϵ_n) with $|x - c_n| \leq \epsilon_n \to 0$.

(a4) x admits a recursive *signed digit expansion*, that is a sequence $s_n \in \{0, -1, +1\}$ with $x = \sum_{n\geq -N} s_n 2^{-n}$.

Moreover the above conditions are, similarly to Shoenfield's Limit Lemma, one Turing jump stronger than the following

(a5) A Turing machine can output rational sequences (c'_n) with $c'_n \to x$.

Under the refined view of complexity, (a2) and (a4) remain uniformly polynomial-time equivalent but not (a1) nor (a3). Let $\mathbb{D}_n := \{a/2^{-n} : a \in \mathbb{Z}\}$ and $\mathbb{D} := \bigcup_n \mathbb{D}_n$ denote the set of dyadic rationals (of precision n). Proceeding from single reals to (possibly partial) functions $f :\subseteq [0;1] \to \mathbb{R}$, the following conditions are known equivalent and thus a reasonable notion of computability [14], [35, §0.7], [38, §6.1], [21, §2.3]:

(b1) A Turing machine can, upon input of every sequence $a_n \in \mathbb{Z}$ with $|x - a_n/2^{n+1}| \leq 2^{-n}$ for $x \in \mathrm{dom}(f)$, output a sequence $b_m \in \mathbb{Z}$ with $|f(x) - b_m/2^{m+1}| \leq 2^{-m}$.

(b2) There exists an oracle Turing machine $\mathcal{M}^?$ which, upon input of each $n \in \mathbb{N}$ and for every (discrete function) oracle $\mathcal{O} = \mathcal{O}(x)$, $x \in \mathrm{dom}(f)$, answering queries "$m \in \mathbb{N}$" with some $a \in \mathbb{D}_{m+1}$ such that $|x - a| \leq 2^{-m}$, prints some $b \in \mathbb{D}_{n+1}$ with $|f(x) - b| \leq 2^{-n}$.

It follows that every (even relatively, i.e. oracle) computable $f :\subseteq [0;1] \to \mathbb{R}$ is necessarily continuous.

Concerning complexity, (b1) and (b2) have turned out as polynomial-time equivalent; cf. [21, §8.1] and [38, Theorem 9.4.3]. Here the running time is measured in terms of the output precision parameter n in (b2); and for (b1) in terms of the time until the n-th digit of the infinite binary output string appears: in both cases uniformly in (i.e. w.r.t. the worst-case over all) $x \in \mathrm{dom}(f)$. (Efficient) computability is tied to (quantitative) topological properties; specifically we record from [21, Theorem 2.19]:

Fact 2. *If $f : [0;1] \to \mathbb{R}$ is computable in time $t(n)$, then $\mu(n) := t(n+2)$ constitutes a modulus of continuity for f in the sense that it holds*

$$\forall x, y \in [0;1] : \quad |x - y| \leq 2^{-\mu(n)} \;\Rightarrow\; |f(x) - f(y)| \leq 2^{-n} \quad .$$

For further details and prototype results see, e.g., [20, 29, 30, 33, 21, 31, 22, 6, 34, 17, 23], and [38, §7]. We also emphasize the close relation of the above notions to actual implementations of exact real arithmetic [32].

Multivalued mappings (Definition 1d) arise in Computable Analysis both for structural reasons [4, §1.4] and from practical applications. For instance the Fundamental Theorem of Algebra yields to every monic degree-d polynomial p a d-tuple of complex roots including multiplicities — up to permutation [36], that is, a *multivalued* mapping

$$\mathbb{C}^d \ni (a_0, \ldots a_{d-1}) \Rightarrow (\lambda_1, \ldots \lambda_d) \text{ s.t. } a_0 + a_1 z + \cdots a_{d-1} z^{d-1} + z^d = \prod_{j=1}^{d} (z - \lambda_j).$$

Or when diagonalizing a given real symmetric matrix, one is interested in *some* basis of eigenvectors, not a specific one. It is thus natural to consider computations which, given (some encoding of) x, intensionally choose and output *some*

value $y \in f(x)$. Indeed, a multifunction may well be computable yet admit no computable single-valued *selection*; cf. e.g. [38, Exercise 5.1.13] or [26].

1.2 Advice in Computer Science, Analysis, and Logic

Theoretical computer science regularly employs advice to formalize nonuniformity. Here an algorithm is presented not only with the input instance $\vec{x}$ but also with some additional integer $k = k(\vec{x})$. $\mathcal{P}/\mathrm{poly}$ for example consists precisely of those languages $L \subseteq \{0,1\}^*$ decidable in polynomial-time with advice k

- depending only on the binary length $|\vec{x}|$ of $\vec{x}$
- and of 'size' $\log k$ polynomially bounded in $|\vec{x}|$;

equivalently: languages L decidable by polynomial-size circuit families, i.e., one separate circuit for each input length n [13, §3.1]. These classes have received recent attention for instance in connection with the *Isomorphism Conjecture* [1, §2.7].

Manifesto 3. *We are interested in the benefit of constant-size advice (that is, with $k \in K$ for a fixed finite K) but permit full dependence on $\vec{x}$ (rather than on $|\vec{x}|$ only—which would not make sense for $\vec{x}$ being real numbers, anyway), recall Definition 1f) and see Section 2.2 below. In practice this means that the input $\vec{x}$ is accompanied with some integer $k \in K$ known from the application that generated $\vec{x}$ but not necessarily computable from said $\vec{x}$ alone. Put differently, $|K|$ separate algorithms $(\mathcal{A}_1, \ldots, \mathcal{A}_{|K|})$ may 'jointly' solve a problem in the sense that, on each input $\vec{x}$, at least one of them (namely $\mathcal{A}_{k(\vec{x})}$) terminates and produces the correct output. This is motivated by many problems, mostly over real numbers where finite advice makes the difference between computability and incomputability; see for instance Example 5 below.*

Indeed, already in the discrete realm $\mathcal{P}/\mathrm{const}$ contains undecidable problems. The class $\mathcal{P}_{\mathbb{R}}/\mathrm{const}$ in algebraic complexity theory [2, 28] means algorithms having stored (a constant number of) real constants and is unrelated to advice in our sense.

Remark 4. *Another superficially similar, but logically independent, relaxation of classical decision problems is* semi-membership*: Here an algorithm receives as input two arguments $(\vec{x}, \vec{y})$, with the promise that at least one of them belongs to L, and has to output $\vec{z} \in \{\vec{x}, \vec{y}\}$ with $\vec{z} \in L$ [15]. In the setting of Definition 1, this corresponds to the classification problem $(L \times X, X \times L)$.*

Note that Definition 1f) applies also to the realm $X = \mathbb{R} = Y$ of Computable Analysis — where, indeed, discrete advice occurs as a natural means against weak ineffectivity: Many practical problems over real numbers are trivially incomputable for continuity reasons in violation of the sometimes so-called *Main Theorem of Computable Analysis* [38, Theorem 4.3.1]. (Seemingly few) others are ineffective in the stronger sense of mapping computable arguments to incomputable ones – such as the differentiation operator [38, Example 6.4.9]. Ineffectivity of the former kind can easily be mended by providing, in addition

to the 'continuous' argument $x \in X$ from the connected space X, some discrete information k from a countable universe, say, $\mathbb{N}$. Such additional data is known in logic as *enrichment* [24, p.238/239]; cf. also [3]. Observe that, indeed, non-constant functions $k : X \to \mathbb{N}$ constitute 'prototypes' of nonuniformly computable but uniformly incomputable mappings; cf. [5]. Now previous work [40] has determined the precise amount of advice (i.e. the size of K) sufficient and necessary in order to render classical tasks in real linear algebra (continuous and) computable:

Example 5. *a) In order to compute, given a singular $d \times d$ matrix A, some non-zero solution $\vec{x}$ to the homogeneous system $A \cdot \vec{x} = 0$ of linear equations, knowing $k := \mathrm{rank}(A) \in \{0, 1, \ldots, d-1\} =: K$ is sufficient* [39, Theorem 11]*; and $|K| \geq d$ also necessary* [40, Theorem 46].

b) In order to compute, given a symmetric real $d \times d$ matrix A, some basis of eigenvectors, knowing $k := \mathrm{Card}\,\sigma(A) \in \{1, \ldots, d\} =: K$ is sufficient [39, Theorem 19]*; and $|K| \geq d$ also necessary* [40, Theorem 47].

c) In order to compute, given a symmetric real $d \times d$ matrix A, some single eigenvector, $\lfloor 1 + \log_2 d \rfloor$-fold advice is sufficient and necessary [40, Theorem 49].

Note that these investigations on quantitative nonuniformity exhibit examples which, with insufficient advice, are not even computable; see also [8]. [40, Example 7] on the other hand has constructed a computable smooth $h_2 : [0; 1] \to [0; 1]$ which, with two-fold advice, can be evaluated within polynomial, but without requires exponential, time in the sense of Section 1.1.

The present work extends this trade off between computational complexity and (the amount of) additional discrete information. More precisely we construct (Theorem 11) for each $d \in \mathbb{N}$ a smooth function $h_d : [0; 1] \to [0; 1]$ computable but with time complexity an exponential tower of height d; which, when providing k–fold advice, drops to height $d - k$ for each $k = 1, 2, \ldots, d$.

2 Discrete Classification Problems and Advice

Concerning trade-offs between advice and computational complexity, we have

Theorem 6. *Let* $\exp_0(n) := n$, $\exp_1(n) := 2^n$, $\exp_2(n) := 2^{2^n}$, *and* $\exp_{j+1}(n) := \exp_j(2^n) = 2^{\exp_j(n)}$ *denote the tower of iterated exponention also known as* tetration. *For each $J \geq 2$ there is a total straight classification problem of size J*

- *solvable in time* $\exp_{J-1}(\mathrm{poly}\, n)$ *but not in time* $\exp_{J-2}(\mathrm{poly}\, n)$
- *with 2–fold advice solvable in time* $\exp_{J-2}(\mathrm{poly}\, n)$ *but not in time* $\exp_{J-3}(\mathrm{poly}\, n)$
- *and more generally with j–fold advice solvable in time* $\exp_{J-j}(\mathrm{poly}\, n)$ *but not in time* $\exp_{J-j-1}(\mathrm{poly}\, n)$, $1 \leq j \leq J$.

2.1 On Hard Cores of Promise and Classification Problems

Nancy Lynch observed that any recursive decision problem L that cannot be solved in polynomial time contains an infinite recursive subset L' such that any algorithm semi-deciding L makes superpolynomially many steps on L' [27, Lemma 1]; cf. [11].

Definition 7. *Let $\mathcal{C} = (C_1, \ldots, C_J)$ denote a classification problem over X.*

a) $\mathcal{C}$ is hard *if none of the promise problems (C_i, C_j), $i < j$, can be solved in polynomial time.*
b) $\mathcal{C}$ is a hard core *if all C_j are infinite and if, for every choice of infinite $B_j \subseteq C_j$, $\mathcal{B} := (B_1, \ldots, B_J)$ is hard.*
c) $\mathcal{C}$ contains *a hard core if there exists a $\mathcal{B} \subseteq \mathcal{C}$ that is a hard core.*

[9, Theorem 3.4] establishes that each promise (i.e. 2-fold classification) problem unsolvable in polynomial time contains a superpolynomial-time hard core. We demonstrate that this fails for 3-fold classification problems:

Example 8. *Let $A, B, C \subseteq \{0,1,2\}^*$ denote classical decision problems – i.e. total promise problems $(A, \bar{A})$, $(B, \bar{B})$, and $(C, \bar{C})$, where $\bar{A} := \{0,1,2\}^* \setminus A$ – decidable in exponential but not in polynomial time. Now consider the classification problem*

$$\Big((2 \circ \bar{C}) \cup (0 \circ A),\ (0 \circ \bar{A}) \cup (1 \circ B),\ (1 \circ \bar{B}) \cup (2 \circ C) \Big) \quad :$$

It clearly is hard but does not contain a hard core. □

2.2 Advice versus Circuit Families: Incompressibility

This section clarifies the relation between advice in the sense of Definition 1f) and the classical model of nonuniform computation. To this end recall that $\mathcal{FP}/\mathrm{const}$ consists precisely of functions $f : \{0,1\}^* \to \{0,1\}^*$ such that

- there exists a polynomial-time Turing machine $\mathcal{M}$
- and a sequence $\vec{a}_n$ of strings from a fixed finite domain
- such that, for every $\vec{x} \in \{0,1\}^n$, $\mathcal{M}(\vec{x}, \vec{a}_n)$ outputs $f(\vec{x})$.

Similarly for other classes such as $\mathcal{REC}/\mathrm{poly}$: all languages decidable with the help of strings $\vec{a}_n$ of polynomial length... Here all inputs $\vec{x}$ of the same length n share the same advice – a real restriction compared to Definition 1f) as we now demonstrate using the Incompressibility Method [25]:

Proposition 9. *Let $\bar{\sigma} \in \{0,1\}^\omega$ denote an infinite random sequence in the sense of Martin-Löf and consider $L := \left\{ \vec{x} : \sigma_{\mathrm{bin}(1\vec{x})} = 1 \right\} \subseteq \{0,1\}^*$, its encoding into a binary language. Then L is trivially decidable (even in constant time) with 2-fold advice; but does not belong to $\mathcal{REC}/\mathrm{poly}$.*

The integer Kolmogorov complexity function $C : \mathbb{N} \to \mathbb{N}$ is well-known uncomputable [25, Theorem 2.3.2] – yet its values can be encoded as suitable, though unbounded, advice. In fact its tally version $C' : \{0,1\}^* \ni \vec{x} \mapsto C(|\vec{x}|)$, belongs to $\mathcal{FP}/\log \subseteq \mathcal{FP}/\operatorname{poly}$ via $\operatorname{bin}(\vec{a}_{|\vec{x}|}) := C'(\vec{x}) \leq |\vec{x}|$; whereas we have

Theorem 10. *Neither C nor C' is computable with finite advice.*

Together with Proposition 9 this demonstrates that (e.g. polynomial-time) computation with finite advice lies skewly to the standard nonuniform circuit complexity classes. We wonder whether $C \in \mathcal{FP}/\operatorname{poly}$ holds.

3 Real Function Complexity with Advice

Example 5 has demonstrated how (and how much) finite advice can make a difference between real computability and incomputability. Our main result provides analogous (although admittedly less natural) examples for the refined view of complexity by connecting discrete classification problems to real functions:

Theorem 11. *a) Fix a finite total straight classification problem $\mathcal{C} = (C_1, \ldots c_J)$ over $\mathbb{N}$. There exists a smooth (i.e. C^∞: infinitely often differentiable) function $h : [0;2] \to [0;J]$ such that the following holds for each $K \in \mathbb{N}$ and superlinear nondecreasing $t : \mathbb{N} \to \mathbb{N}$: $\mathcal{C}$ is solvable with K–fold advice within time $\mathcal{O}\big(t(\operatorname{poly} n)\big)$ iff h is computable with K–fold advice within time $\mathcal{O}\big(t(\operatorname{poly} n)\big)$.*

b) For each $J \in \mathbb{N}$ there is a smooth $f : [0;1] \to [0;1]$ computable with j–fold advice in time $\exp_{J-j}(\operatorname{poly} n)$ but not in time $\exp_{J-j-1}(\operatorname{poly} n)$, $1 \leq j \leq J$.

This (corrects a mistake in, and) generalizes [40, Example 7].

3.1 Benefit of 2-Fold Advice to Function Maximization

The computable functions constructed in Theorem 11b) exhibit the positive effects (in the sense of complexity-reducing) of discrete advice to real computation – but may arguably be considered artificial. For a more natural example, we now consider the generic problem of continuous optimization: computing the maximum of a given (say, non-expansive) $f : [0;1] \to [0;1]$. This constitutes a functional $\operatorname{Max} : \operatorname{Lip}_1 \to [0;1]$, $f \mapsto \max_{0 \leq x \leq 1} f(x)$ on $\operatorname{Lip}_1 := \big\{ f : [0;1] \to [0;1], |f(x) - f(y)| \leq |x - y| \big\}$ computable with respect to any reasonable encoding of Lip_1 into infinite binary strings [38, Corollary 6.2.5 & Lemma 6.1.7]. For instance $f \in \operatorname{Lip}_1$ is uniquely determined via the real sequence $\big(f(d_m)\big)_m$ where

$$(d_m)_m \quad := \quad \big(0, 1, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{3}{4}, \tfrac{1}{8}, \tfrac{3}{8}, \tfrac{5}{8}, \tfrac{7}{8}, \tfrac{1}{16}, \tfrac{3}{16}, \ldots\big) \quad \subseteq \quad \mathbb{D}$$

is dense in $\operatorname{dom}(f)$. And, extending (b1) from Section 1.1, such a real sequence $(y_m)_m$ can be 'represented' (in a sense formalized in the *Type-2 Theory of Effectivity*, TTE) as an integer (double) sequence $(a_{\langle n,m \rangle})_{n,m}$ with $|y_m - a_{\langle n,m \rangle}| \leq 2^{-n}$, where $\langle n, m \rangle := n + (n+m) \cdot (n+m+1)/2$ denotes an integer pairing function.

Remark 12. *Note that $f(2^{-k})$ occurs within $(y_m)_m \subseteq \mathbb{R}$ (and its 2^{-1}–approximation within $(a_{\langle n,m\rangle})_{n,m} \subseteq \mathbb{Z}$) at a position exponential in k. In 'classic' TTE with infinite binary strings as codes* [38, §3]*, this requires 'skipping' over exponentially many bits in order to access $f(2^{-k})$ up to error $\frac{1}{2}$; which prevents evaluation $(f,x) \mapsto f(x)$ to be uniformly polynomial-time computable on Lip_1. In computational practice, on the other hand, such f is implemented as an (say, a C++ function) that, given integers n and m, produces a 2^{-n} approximation to $f(d_m)$ directly, that is, without generating $f(d_1), \ldots, f(d_{m-1})$ first. This has led* [18, 19] *to extend TTE with second-order representations, that is, with encodings that permit random (as opposed to sequential) access.*

For our present purpose the precise technical definition of second-order representations may be spared since we rely only on the following

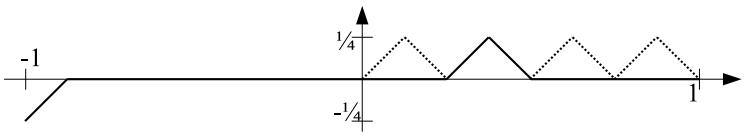
Fact 13. *There exists a (natural and canonical) second-order representation of Lip_1 allowing to recover, given $n \in \mathbb{N}$ and $d \in \mathbb{D}_n \cap [0;1]$, a 2^{-n}–approximation to $f(d)$ of length $\mathcal{O}(n)$ within time polynomial in n.*

We emphasize the similarity to Information-Based Complexity Theory [37, 16] which, based on the complementary, unit-cost model (a.k.a. *realRAM* or *Blum-Shub-Smale Machine*) of real computing, supposes exact black-box evaluation $x \mapsto f(x)$ to take constant time. And in fact, using a refinement of the standard adversary argument to finite precision queries, we can prove

Example 14. *For notational ease consider the class $\widetilde{\mathrm{Lip}_1}$ of non-expansive (a similar effect holds for appropriate smooth) functions $f : [-1;1] \to [-1;1]$;*

$$\mathcal{L} := \left\{ f \in \widetilde{\mathrm{Lip}_1} \;\middle|\; f(x) \le 0 \text{ for } x \le 0,\ f(x) = 0 \text{ for } x \ge 0 \right\}$$
$$\mathcal{K} := \left\{ f \in \widetilde{\mathrm{Lip}_1} \;\middle|\; \mathrm{Max}(f) = -f(-1) \right\}$$

and observe that $\mathcal{L} \cap \mathcal{K} = \{\equiv 0\}$, $\mathrm{Max}(f) = 0$ all $f \in \mathcal{L}$, and $\mathrm{Max}(f) = -f(-1)$ all $f \in \mathcal{K}$. In particular both restrictions $\mathrm{Max}\big|_{\mathcal{L}}$ and $\mathrm{Max}\big|_{\mathcal{K}}$ are polynomial-time computable – while, without such 2-fold advice, $\mathrm{Max}\big|_{\mathcal{L}\cup\mathcal{K}}$ is not. □



4 Conclusion and Perspectives

Recursive Analysis had traditionally focused on the continuous aspects of real number computation as opposed to classical computability theory. Still, discrete advice has over the last years been revealed as essential an ingredient to uniform computability for many natural problems over the reals. We have demonstrated

that, even for problems that are computable without such advice, its presence or absence can have a huge impact in terms of computational complexity.

While our Theorem 11 is entirely artificial, it seems that the underlying concept may be relevant to practical computations. Indeed numerical science generally considers discrete-valued functions like floor or matrix rank efficiently computable – while at the same time at least deprecating the use of tests for equality: a seemingly inherent ambiguity in the foundational semantics. The representation-based theory of real number computation (TTE and its 2nd-order extension) on the other hand regularly extends continuous with discrete information in order to assert the computability of a problem – and provides the canonical interface declaration of an actual implementation in exact real number packages [32].

Still the question remains as for natural examples of real problems where discrete advice makes a difference in complexity but not in computability: in Theorem 11 the function is artificial while its domain $[0;1]$ is easy, in Example 14 the functional is natural while its domain is artificial.

References

1. Agrawal, M.: The Isomorphism Conjecture for $\mathcal{NP}$. In: Cooper, S.B., Sorbi, A. (eds.) Computability in Context, pp. 19–48. World Scientific (2009)
2. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. Springer (1997)
3. Beyersdorff, O., Köbler, J., Müller, S.: Proof Systems that Take Advice. Proof Systems that Take Advice 209(3), 320–332 (2011)
4. Brattka, V.: Recursive Characterization of Computable Real-Valued Functions and Relations. Theoretical Computer Science 162, 45–77 (1996)
5. Brattka, V.: Computable Invariance. Theoretical Computer Science 210, 3–20 (1999)
6. Braverman, M.: On the Complexity of Real Functions. In: Proc. 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 155–164
7. Braverman, M., Cook, S.A.: Computing over the Reals: Foundations for Scientific Computing. Notices of the Americal Mathematical Society 53(3), 318–329 (2006)
8. Brattka, V., Pauly, A.M.: Computation with Advice. In: Electronic Proceedings in Theoretical Computer Science, vol. 24 (June 2010)
9. Brandt, U., Walter, H.K.-G.: Cohesiveness in Promise Problems. Presented at the 64th GI Workshop on Algorithms and Complexity (2012)
10. Even, S., Selman, A.L., Yacobi, Y.: The Complexity of Promise Problems with Applications to Public-Key Cryptography. Inform. and Control 61, 159–173 (1984)
11. Even, S., Selman, A.L., Yacobi, Y.: Hard-Core Theorems for Complexity Classes. Journal of the ACM 32(1), 205–217 (1985)
12. Goldreich, O.: On Promise Problems: A Survey. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Shimon Even Festschrift. LNCS, vol. 3895, pp. 254–290. Springer, Heidelberg (2006)
13. Goldreich, O.: Computational Complexity: A Conceptual Perspective. Cambridge University Press (2008)

14. Grzegorczyk, A.: On the Definitions of Computable Real Continuous Functions. Fundamenta Mathematicae 44, 61–77 (1957)
15. Hemaspaandra, L.A., Torenvliet, L.: Theory of Semi-Feasible Algorithms. Springer Monographs in Theoretical Computer Science (2003)
16. Hertling, P.: Topological Complexity of Zero Finding with Algebraic Operations. Journal of Complexity 18(4), 912–942 (2002)
17. Kawamura, A.: Lipschitz Continuous Ordinary Differential Equations are Polynomial-Space Complete. Computational Complexity 19(2), 305–332 (2010)
18. Kawamura, A., Cook, S.A.: Complexity Theory for Operators in Analysis. In: Proc. 42nd Ann. ACM Symp. on Theory of Computing (STOC 2010), pp. 495–502 (2010)
19. Kawamura, A., Cook, S.A.: Complexity Theory for Operators in Analysis. ACM Transactions in Computation Theory 4(2), article 5 (2012)
20. Ko, K.-I., Friedman, H.: Computational Complexity of Real Functions. Theoretical Computer Science 20, 323–352 (1982)
21. Ko, K.-I.: Complexity Theory of Real Functions. Birkhäuser (1991)
22. Ko, K.-I.: Polynomial-Time Computability in Analysis. In: Ershov, Y.L., et al. (eds.) Handbook of Recursive Mathematics, vol. 2, pp. 1271–1317 (1998)
23. Kawamura, A., Ota, H., Rösnick, C., Ziegler, M.: Computational Complexity of Smooth Differential Equations. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 578–589. Springer, Heidelberg (2012)
24. Kreisel, G., Macintyre, A.: Constructive Logic versus Algebraization I. In: Troelstra, A.S., van Dalen, D. (eds.) Proc. L.E.J. Brouwer Centenary Symposium, pp. 217–260. North-Holland (1982)
25. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer (1997)
26. Luckhardt, H.: A Fundamental Effect in Computations on Real Numbers. Theoretical Computer Science 5, 321–324 (1977)
27. Lynch, N.: On Reducibility to Complex or Sparse Sets. Journal of the ACM 22(3), 341–345 (1975)
28. Michaux, C.: $\mathcal{P} \neq \mathcal{NP}$ over the Nonstandard Reals Implies $\mathcal{P} \neq \mathcal{NP}$ over $\mathbb{R}$. Theoretical Computer Science 133, 95–104 (1994)
29. Müller, N.T.: Subpolynomial Complexity Classes of Real Functions and Real Numbers. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 284–293. Springer, Heidelberg (1986)
30. Müller, N.T.: Uniform Computational Complexity of Taylor Series. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 435–444. Springer, Heidelberg (1987)
31. Müller, N.T.: Constructive Aspects of Analytic Functions. In: Proc. Workshop on Computability and Complexity in Analysis (CCA), InformatikBerichte FernUniversität Hagen, vol. 190, pp. 105–114 (1995)
32. Müller, N.T.: 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)
33. Müller, N.T., Moiske, B.: Solving Initial Value Problems in Polynomial Time. In: Proc. 22nd JAIIO-PANEL, pp. 283–293 (1993)
34. Müller, N.T., Zhao, X.: Complexity of Operators on Compact Sets. Electronic Notes Theoretical Computer Science 202, 101–119 (2008)
35. Pour-El, M.B., Richards, J.I.: Computability in Analysis and Physics. Springer (1989)

36. Specker, E.: The Fundamental Theorem of Algebra in Recursive Analysis. In: Dejon, B., Henrici, P. (eds.) Constructive Aspects of the Fundamental Theorem of Algebra, pp. 321–329. Wiley-Interscience (1969)
37. Traub, J.F., Wasilkowski, G.W., Woźniakowski, H.: Information-Based Complexity. Academic Press (1988)
38. Weihrauch, K.: Computable Analysis. Springer (2000)
39. Ziegler, M., Brattka, V.: Computability in Linear Algebra. Theoretical Computer Science 326, 187–211 (2004)
40. Ziegler, M.: Real Computation with Least Discrete Advice: A Complexity Theory of Nonuniform Computability. Annals of Pure and Applied Logic 163(8), 1108–1113 (2012)

Computability and Computational Complexity of the Evolution of Nonlinear Dynamical Systems

Olivier Bournez[1], Daniel S. Graça[2,3], Amaury Pouly[1,2], and Ning Zhong[4]

[1] Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France
[2] CEDMES/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal
[3] SQIG/Instituto de Telecomunicações, Lisbon, Portugal
[4] DMS, University of Cincinnati, Cincinnati, OH 45221-0025, U.S.A.

Abstract. Nonlinear dynamical systems abound as models of natural phenomena. They are often characterized by highly unpredictable behaviour which is hard to analyze as it occurs, for example, in chaotic systems. A basic problem is to understand what kind of information we can realistically expect to extract from those systems, especially information concerning their long-term evolution. Here we review a few recent results which look at this problem from a computational perspective.

1 Introduction

Scientists have always been fascinated by problems involving dynamical systems. These appear in the context of many important applications, ranging from celestial mechanics to electronics. Efforts to understand those systems have led to many important insights. A relatively complete theory was developed for the case of linear systems. It was also shown that the general theory of linear systems can often be applied even for non-linear systems: near (hyperbolic) equilibrium points, a nonlinear system has the same qualitative structure as its linearization through the Hartman-Grobman theorem [1].

Despite those early successes, over the last few decades scientists have come to understand that even simple nonlinear dynamical systems can exhibit complicated behaviour. This realization can perhaps be traced back to the work of the French mathematician Henri Poincaré in the late XIXth century [2]. Poincaré studied a problem from celestial mechanics known as the three-body problem. The three-body problem is the problem of determining the motions of three bodies (e.g., two stars in a binary system plus a planet), which interact in accordance with the laws of classical mechanics (Newton's laws of motion and of universal gravitation), given their initial positions, masses and velocities. Unlike the two-body problem, which was completely solved and shown to have a very predictable behaviour (in the case of a star and a planet it yields Kepler's laws of planetary motion), Poincaré showed that orbits for the three-body problem could be very complex—in modern terms, he showed that these orbits had elements of *chaotic* behaviour. Subsequent studies of nonlinear dynamical systems were

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 12–21, 2013.

done by mathematicians such as Hadamard [3], Birkhoff [4], Kolmogorov [5], Cartwright and Littlewood [6], and S. Smale [7]. Although most of these studies involved physically inspired systems, the resulting papers were often hard to read for the non-specialist and remained within the pure mathematicians' community well into the middle of the XXth century. This situation changed with the arrival of the digital computer. Numerical simulations done by cheap, fast and widely available computers allowed the non-specialist scientist to get a grasp on the complexity of their favourite models. Edward Lorenz was one of these early scientists who stumbled upon chaos. He was a meteorologist interested in long-term weather prediction. During the course of his weather simulations, and to save time, he would sometimes start a simulation in the middle of its course, using data obtained in previous simulations. His computer used 6 decimal digits internally, but would only print 3 digits as a result. Since he only had access to those 3 digits, he restarted the computation using this truncated data. He soon realized, to his surprise, that the small error introduced by the truncation could have a huge effect in the evolution of the system, and that this effect could happen quite rapidly [8]. Nowadays this phenomenon is known as "sensitive dependence on initial conditions" or, more poetically, as the "butterfly effect"—a hurricane's formation could be contingent on whether or not a distant butterfly had flapped its wings several weeks before. However, this "butterfly effect" also raises critical questions about the reliability of computer-generated simulations of nonlinear systems. For example, the experiments suggested that the trajectories of the system which Lorenz was studying would converge over time to a butterfly shaped object, later known as the *Lorenz attractor*. But to rigorously prove that the figure drawn by a computer was not an artefact of round-off error accumulation was a much harder problem. It was the 14th problem on the list of problems proposed by the Fields medallist Smale [9] for the XXIst century. It was finally solved in 1998, following the work of Tucker [10].

The interplay between chaos (and other forms of nonlinear complex behaviour) and computers poses interesting questions to the theoretical computer science community; in particular, questions such as the following: Are chaotic and other highly complex systems necessarily computationally intractable? How do they compare to intractability in Turing Machines? Is it possible to relate the degree of computational intractability with some degree of "chaos complexity" (or other) of the system? On the other hand, since physical implementations of computers are subjected to natural laws, can chaotic/nonlinear computers be computationally more powerful than Turing machines (from a computability and/or computational complexity perspective)? Many of these interesting questions have to do with the long-term behaviour of dynamical systems. In this paper we review, from a computational perspective, a few selected results that we have obtained recently in this area.

2 Dynamical Systems: Basics

A dynamical system is a way of describing how points in a *state space* evolve (deterministically) over time. Formally it is a triple $(S, \mathbb{T}, \phi)$, where S is an open

set of the Euclidean space (the state space), $\mathbb{T}$ is a semigroup which denotes the time (usually $\mathbb{T} = \mathbb{R}$—continuous-time systems—or $\mathbb{T} = \mathbb{N}$—discrete-time systems) and $\phi : \mathbb{T} \times S \to S$ is a map (the evolution rule). The map ϕ must also have the following properties (we write $\phi(t, x) = \phi_t(x)$, where $\phi_t : S \to S$): (i) $\phi_0 : S \to S$ is the identity; (ii) $\phi_t \circ \phi_s = \phi_{t+s}$ for each $t, s \in \mathbb{T}$. It can be shown [11] that the evolution of a discrete-time system can be obtained by iterating a map ($= \phi_1$). Iterating the map t times will yield the state of the system at time t. For C^1 continuous-time systems, the evolution rule can be rewritten as a differential equation $x' = f(x)$, where $f(x) = \partial_t \phi_t(x)|_{t=0}$.

Dynamical systems theory deals with the long-term qualitative behaviour of dynamical systems, since quantitative information is usually a too-ambitious goal. Some typical questions studied in dynamical systems include: "Will the system settle down to some steady state in the long term? If yes, which properties characterize this steady state? Can one tell if there are several co-existing steady states? Which types of steady states does a particular class of dynamical systems admit?"

We remark that these questions correspond to problems in theoretical computer science and related applications (e.g., control theory—cf., e.g., [12]) which are interested in analyzing the long-term behaviour of some system. The steady states mentioned above are usually known as *attractors* or, in a broader sense, as *invariant sets*. The set of points which converge to a given attractor defines its *basin of attraction*. Up to dimension two, dynamical systems are relatively well-behaved. The Poincaré-Bendixson Theorem (cf. [13, § 8*.5] for more details) rules out chaos in two-dimensional systems—it states that, except for singularities (like homoclinic solutions [14]), attractors in the plane can only be points or closed (periodic) orbits. However, as soon as one enters three-dimensional space, complex chaotic behaviour can appear, as demonstrated by the Lorenz system (which is a three-dimensional system of ordinary differential equations defined by quadratic functions). Thus, low-dimensional systems defined by simple rules can have attractors and invariant sets with a much more complex structure than fixed points or periodic orbits.

3 Computability

Before analyzing the computational complexity of problems related to dynamical systems, it makes sense to first investigate whether these problems are computable. Since problems in dynamical systems theory deal with the long-term behaviour of systems one might stumble upon the Halting problem. Indeed, since Turing machines, considered as discrete-time dynamical systems, can be embedded into many classes of discrete-time or continuous-time dynamical systems, these latter systems often inherent the rich structures of Turing machines and, in particular, undecidability (cf., e.g., [15,16,17,18]). The embedding of a Turing machine (which is a purely discrete model) into a continuous system is often done by implicitly using discrete elements, like piecewise linear functions, or more generally piecewise defined functions or dynamics.

A more challenging task is to analyze problems which do not allow these discrete elements. Classical physics is built upon polynomials, trigonometric function, logarithms, etc. (these functions, their inverses, and all of their possible compositions are known in Analysis as *elementary functions*—not to be confused with elementary functions in computability theory, cf., e.g., [19]—or as closed-form functions) and elementary functions are analytic. Analyticity is a very strong property; for an analytic function, its global property is determined by its local behaviour. Thus, it is much harder to encode the Halting problem (or the behaviour of a given Turing machine) into an analytic dynamical system and it may not be evident whether these systems can present non-computable behaviour. For these reasons, we have focused much of our work on analytic systems, avoiding noncomputability results which could be obtained due only to the inherently discrete dynamics of the system.

Although the long term behaviour of a system can be very complicated and often chaotic, many problems associated with the system are still (algorithmically) decidable. For example, it is possible to decide whether or not some (rational) initial point will converge to the fixed point of

$$x' = Ax$$

at the origin, when A is an $n \times n$ hyperbolic (i.e., the real parts of the eigenvalues of A are nonzero) matrix [20].

3.1 Computability of Trajectories over Unbounded Domains

A basic problem concerning the computability of solutions for ordinary differential equations (ODEs)is the following: given some ODE and some initial condition, can we compute the respective solution for *arbitrary* $t \geq 0$? Or equivalently, are trajectories of a continuous-time dynamical system computable? This problem is not as trivial as it first might seem. One might suggest using any standard numerical method to solve it. The problem with numerical methods is that virtually all numerical methods use a strong hypothesis: the existence of a Lipschitz constant valid for the vector field over the *entire* domain where the solution is defined. Of course, if the vector field is C^1 and the time is restricted to a closed interval $[t_0, t_1]$, then this desired global Lipschitz constant is readily obtainable (cf., e.g., [1, p. 71]). This global Lipschitz constant is also critical for previous results concerning computability of ODEs; in fact, it is to be found in virtually any such result previously obtained (cf., e.g., [21]). Although there are several computability results established for ODEs without a global Lipschitz constant—as long as the solution is assumed to be unique (cf., [21, § 7.1]), those results however only hold for ODEs defined on a *closed interval* $[t_0, t_1]$. Another related result can be found in [22], where the author proves computability of solutions of ODEs in unbounded domains without requiring the use of Lipschitz constants. However, Ruohonen requires a very restrictive bound on the growth of f.

Thus the previous results on computability of ODEs do not address the general case where computability must be established over the time interval $[0, +\infty)$ for functions which might grow (in absolute terms) quicker than a linear function (in which case no global Lipschitz constant exists).

Classically, existence and uniqueness results for C^1 ODEs over the *maximal interval* where the solution is defined (cf., e.g., [1, § 2.4] for a precise definition of the maximal interval) is shown recursively: first establish existence and uniqueness over an interval $[a, b]$; then the interval where existence and uniqueness is guaranteed is recursively extended, and shown to converge to a maximal interval. Note that this convergence can be non-effective, as we have shown in [23]: a computable ODE with computable initial conditions can yield a non-computable maximal interval even if the ODE is analytic. The result is further refined in [24], where it is shown that the set of all initial data generating solutions with lifespans longer than k, $k \in \mathbb{N}$, is in general not computable. Although the maximal interval of existence may be non-computable, it is (lower) semi-computable and therefore, if we want to compute the solution y of the ODE at time t, it suffices to extend the interval $[a, b]$ until it includes t and then we can use a standard algorithm to compute $y(t)$ (cf., e.g., [23]). In this process, how to extend the interval $[a, b]$ is a key step and, concerning the extension, there are two approaches. The first approach (cf., e.g., [1, § 2.4] for the similar case of C^1 functions) is to use local Lipschitz constants to extend the solution recursively. This can be done for C^1 functions. In particular, we have shown in [23] that the solution of $y' = f(y), y(t_0) = y_0$ can be computed over its maximal interval of definition from (f, f', t_0, x_0) for C^1 functions.

This result was later generalized in [25], using the second classical approach to extend the interval $[a, b]$ (cf., e.g., [26, p. 12, Theorem 3.1]). In this approach we split the state space into several compacts and provide an algorithm to generate partial solutions inside these compacts (via Peano's Existence Theorem). This approach is more general then the previous one in the sense that it does not require Lipschitz constants (it is valid even for ODEs with non-unique solutions). But, computationally, it requires to know how to "glue" the partial solutions to get the whole solution over the maximal interval. In [25] we solve this problem by using an enumeration approach. The idea is to generate all possible "tubes" which cover the solution, and then check if this cover is valid within the desired accuracy. The proof is constructive, although terribly inefficient in practice. Nonetheless this technique shows that if the solution to an initial-value problem defined by an ODE is unique, then the solution must be computable over its maximal interval of existence, under the (minimal) classical conditions ensuring existence of a solution to an ODE initial-value problem (continuity).

Theorem 1 ([25]). *Consider the initial value problem $y' = f(y), y(t_0) = y_0$, where f is continuous on $\mathbb{R}^n$. Suppose that there is a unique solution y, defined on the maximal interval (α, β). Then $y(t)$ is computable from f, t_0, x_0, t, where $t \in (\alpha, \beta)$.*

An interesting problem which we shall tackle in Section 4.1 is to determine the computational complexity of computing the solution of a given ODE over its maximal interval of existence.

3.2 Computability of Attractors and Invariant Sets

We have seen in Section 2 that a trajectory may converge over time to some kind of attractor (steady state). It is an important problem both in theory and in practice to characterize these attractors. For example, one is often interested in verification problems. The purpose of verification theory is to prove (or disprove) that a system behaves as intended. A system may have safe states (invariant sets), where the system should be, and unsafe states, where undesirable behaviour may occur (e.g., turbulence over a wing, a nuclear reactor overheating, etc.). Thus many verification problems often amount to understand which kind of attractors/invariant sets a system has, and which are their respective basins of attraction.

For verification problems involving complex systems, computers are, of course, essential tools. Thus, it becomes useful to know which invariant sets are computable and which are not; for computable invariant sets, which can be computed efficiently. Many specialized results exist in the literature of control theory. Here we shall focus on more general problems in dynamical systems. One of our initial projects was to investigate computability in the planar dynamical systems. As we have mentioned, due to the Poincaré-Bendixson theorem, invariant sets in the plane can only be fixed points or periodic orbits (with the exception of singularities). Thus it is natural to study this class of systems first. In general, fixed points can be computed since they are the zeros of the function f defining the differential equation $y' = f(y)$ and isolated zeros of a computable function are also computable—cf., e.g., [27]. On the other hand, many problems related to the simple planar dynamics can be undecidable, as the following results [28] show.

Theorem 2 ([28]). *Given as input an analytic function f, the problem of deciding the number of equilibrium points of $y' = f(y)$ is undecidable, even on compact sets. However, the set formed by all equilibrium points is upper semi-computable.*

Theorem 3 ([28]). *Given as input an analytic function f, the problem of deciding the number of periodic orbits of $y' = f(y)$ is undecidable, even on compact sets. However, the set formed by all hyperbolic periodic orbits is upper semi-computable.*

In short, a hyperbolic periodic orbit is an orbit to which nearby trajectories converge exponentially fast (cf., e.g., [1] for more details). The hyperbolic property entails stronger stability properties to small perturbations of the system. This is important when computing an invariant set, since despite round-off errors, one should still be able to effectively approximate the invariant set. In this sense,

the notion of stability in dynamical systems theory seems to be intertwined with computability.

The above results show that one cannot hope for general procedures to compute invariant sets for relatively large families of dynamical systems such as planar systems. Instead, algorithms should be devised for each particular case.

As we have seen before, invariant sets need not to be fixed points or periodic orbits, but may take complex shapes such as Lorenz attractor. Are these complex shapes computable? We have analyzed in [20] the case of Smale's horseshoe (cf., e.g., [14]), which was the first example of an hyperbolic invariant set which is neither an equilibrium point nor a periodic orbit. Contrarily to what one could expect a priori, Smale's horseshoe is computable.

Theorem 4 ([20]). *The Smale Horseshoe is a computable (recursive) closed set.*

3.3 Computability of Basins of Attraction

Similar to the case of invariant sets, when a class of dynamical systems is considered, general algorithms for computing basins of attractor do not exist, even for classes of well-behaved systems; in particular, when the class is large. For instance, Zhong [29] showed that there exists a C^∞ computable dynamical system having a unique computable hyperbolic equilibrium point but the basin of attraction of this hyperbolic equilibrium point is non-computable.

This result was generalized in a paper we just submitted:

Theorem 5. *There exists a computable analytic dynamical system having a computable hyperbolic equilibrium point such that its basin of attraction is recursively enumerable, but is not computable.*

Thus finding the basin of attraction of a given attractor is, in general, a non-computable problem, although one can semi-compute this basin from the inside.

4 Computational Complexity

4.1 Computational Complexity of Trajectories over Unbounded Domains

We have seen that the trajectories of a non-linear dynamical system are computable over their maximal interval of definition. It thus becomes interesting to understand the underlying computational complexity of finding these trajectories. However, the two results [23] and [25] are not very helpful in that respect because they use an exhaustive approach—the underlying algorithm will always stop when some condition is met (and we are sure that this condition must eventually be met)—but we do not have *a priori* any clue on how much time this will take.

Recently we have been focusing on polynomial differential equations. This particular subclass has some advantages: it is well-behaved, it corresponds to a particular model of computation—Shannon's General Purpose Analog Computer [30] (thus understanding the computational complexity of polynomial differential equations is equivalent to understand the computational complexity of this computational model), and it captures more complex ODEs defined using trigonometric functions, exponentials, etc. [31].

Unfortunately, computing solutions of ODEs efficiently over unbounded domains is not easy. We can get a numerical estimate of the solution, but Lipschitz constants play a crucial role for estimating in an efficient manner the error made in the computation and are thus needed if we want to compute the solution of a polynomial ODE in the sense of computable analysis [27]. But since no global Lipschitz constant exist, in general, for polynomials in unbounded domains, we have to compute local Lipschitz constants. But these local Lipschitz constants depend on the compact set where the solution is. Thus, to be sure that the solution is in a given compact set, given only some numerical estimate of the solution, we need to know the error of this estimate, i.e., we need to know beforehand the Lipschitz constant we were trying to find in the first place.

In [32] we presented a solution to this problem: break the vicious circle using the size of the solution as one of the parameters on which the computational complexity of the solution is measured upon.

Theorem 6 ([32]). *The solution, at time T, of an initial-value problem $y' = p(y)$ with initial condition $y(t_0) = y_0 \in \mathbb{R}^d$, where p is a vector of polynomials, can be computed, in an uniform manner, with precision 2^{-n}, in time polynomial in T, n and $Y = \sup_{t_0 \leq t \leq T} \|y(t)\|$.*

Methods usually used for numerical integrations (including the basic Euler's method, Runge Kutta's methods, etc.) tend to fall in the general theory of n-order methods for some fixed $n \in \mathbb{N}$. They do compute the solution of an ODE in polynomial time in a *compact* time interval $[a, b]$, but are not guaranteed to work in polynomial time over the maximal interval of definition of the solution.

In [32] we solve this problem by using variable order methods. This is done with the help of a Taylor approximation, with the number of terms used on the approximation depending on the input. The idea of using variable order methods is not new. W. Smith mentioned it in [33], where he claimed that some classes of ODEs can be solved in polynomial time over maximal intervals, however without providing a full proof to this claim. Variable order methods have also been used in [34], [35], [36], but for the case of bounded domains.

It is not only polynomial ODEs which can be solved in polynomial time over their maximal interval of definition, many analytic ODEs can also be solved in polynomial time, as long as the function defining the ODE and its solution do not grow quicker than a very generous bound [37].

Theorem 7 ([37]). *Let $y(t)$ be the solution of the initial-value problem $y' = f(y)$, $y(t_0) = y_0$, in $\mathbb{C}^d$, where f is analytic in $\mathbb{C}^d$ and p-poly-bounded (in $\mathbb{C}^d$), f, t_0, x_0 are polynomial-time computable, and $y(t)$ admits an analytic extension*

to $\mathbb{C}^d$ and is poly-bounded over $\mathbb{C}^d$. Then the function which maps f, x_0, t_0, and t to the value $y(t)$ is polynomial-time computable.

(a function is p-poly-bounded if $\|f(x)\|$ is bounded by $2^{p(\log \|x\|)}$). This result uses previous results from Müller et al. [38], [34], [39] which say that, locally, the solution of an analytic ODE can be computed in polynomial time. We then extract (in polynomial-time) the coefficients of the Taylor series of the solution, which allow us to compute, in polynomial time, the solution of the ODE in its maximal interval of definition using the hypothesis of poly-boundedness. The hypothesis that f is analytic on the complex space (and poly-bounded there) and that the solution of the ODE admits an analytic extension to $\mathbb{C}^d$ are needed because we use the Cauchy integral formula in the proof.

Acknowledgments. D.S. Graça was partially supported by *Fundação para a Ciência e a Tecnologia* and EU FEDER POCTI/POCI via SQIG—Instituto de Telecomunicações through the FCT project PEst-OE/EEI/LA0008/2013.

References

1. Perko, L.: Differential Equations and Dynamical Systems, 3rd edn. Springer (2001)
2. Poincaré, H.: Sur le problème des trois corps et les équations de la dynamique. Acta Math. 13, 1–270 (1889)
3. Hadamard, J.: Les surfaces à courbures opposées et leurs lignes géodésiques. J. Math. Pures et Appl. 4, 27–73 (1898)
4. Birkhoff, G.D.: Dynamical Systems. American Mathematical Society Colloquium Publications, vol. 9. American Mathematical Society (1927)
5. Kolmogorov, A.N.: Preservation of conditionally periodic movements with small change in the hamiltonian function. Doklady Akademii Nauk SSSR 98, 527–530 (1954)
6. Cartwright, M.L., Littlewood, J.E.: On non-linear differential equations of the second order, i: The equation $y'' + k(1-y^2)y' + y = b\lambda k \cos(\lambda t + a)$, k large. J. Lond. Math. Soc. 20(3), 180–189 (1945)
7. Smale, S.: Differentiable dynamical systems. Bull. Amer. Math. Soc. 73, 747–817 (1967)
8. Lorenz, E.N.: Deterministic non-periodic flow. J. Atmos. Sci. 20, 130–141 (1963)
9. Smale, S.: Mathematical problems for the next century. Math. Intelligencer 20, 7–15 (1998)
10. Tucker, W.: The Lorenz attractor exists. In: Acad, C.R. (ed.) Sci. Paris. Series I - Mathematics, vol. 328, pp. 1197–1202 (1999)
11. Hirsch, M.W., Smale, S.: Differential Equations, Dynamical Systems, and Linear Algebra. Academic Press (1974)
12. Sontag, E.D.: Mathematical Control Theory, 2nd edn. Springer (1998)
13. Hubbard, J.H., West, B.H.: Differential Equations: A Dynamical Systems Approach — Higher-Dimensional Systems. Springer (1995)
14. Hirsch, M.W., Smale, S., Devaney, R.: Differential Equations, Dynamical Systems, and an Introduction to Chaos. Academic Press (2004)
15. Branicky, M.S.: Universal computation and other capabilities of hybrid and continuous dynamical systems. Theoret. Comput. Sci. 138(1), 67–100 (1995)

16. Asarin, E., Maler, O.: Achilles and the tortoise climbing up the arithmetical hierarchy. J. Comput. System Sci. 57(3), 389–398 (1998)
17. Bournez, O.: Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. Theoret. Comput. Sci. 210(1), 21–71 (1999)
18. Koiran, P., Moore, C.: Closed-form analytic maps in one and two dimensions can simulate universal Turing machines. Theoret. Comput. Sci. 210(1), 217–223 (1999)
19. Odifreddi, P.: Classical Recursion Theory, vol. 2. Elsevier (1999)
20. Graça, D., Zhong, N., Buescu, J.: Computability, noncomputability, and hyperbolic systems. Appl. Math. Comput. 219(6), 3039–3054 (2012)
21. Ko, K.I.: Computational Complexity of Real Functions. Birkhäuser (1991)
22. Ruohonen, K.: An effective Cauchy-Peano existence theorem for unique solutions. Internat. J. Found. Comput. Sci. 7(2), 151–160 (1996)
23. Graça, D., Zhong, N., Buescu, J.: Computability, noncomputability and undecidability of maximal intervals of IVPs. Trans. Amer. Math. Soc. 361(6), 2913–2927 (2009)
24. Rettinger, R., Weihrauch, K., Zhong, N.: Topological complexity of blowup problems. Journal of Universal Computer Science 15(6), 1301–1316 (2009)
25. Collins, P., Graça, D.S.: Effective computability of solutions of differential inclusions — the ten thousand monkeys approach. Journal of Universal Computer Science 15(6), 1162–1185 (2009)
26. Hartman, P.: Ordinary Differential Equations, 2nd edn. Birkhäuser (1982)
27. Weihrauch, K.: Computable Analysis: an Introduction. Springer (2000)
28. Graça, D.S., Zhong, N.: Computability in planar dynamical systems. Natural Computing 10(4), 1295–1312 (2011)
29. Zhong, N.: Computational unsolvability of domain of attractions of nonlinear systems. Proc. Amer. Math. Soc. 137, 2773–2783 (2009)
30. Graça, D.S.: Some recent developments on Shannon's General Purpose Analog Computer. Math. Log. Quart. 50(4-5), 473–485 (2004)
31. Graça, D.S., Campagnolo, M.L., Buescu, J.: Computability with polynomial differential equations. Adv. Appl. Math. 40(3), 330–349 (2008)
32. Bournez, O., Graça, D.S., Pouly, A.: On the complexity of solving polynomial initial value problems. In: 37th International Symposium on Symbolic and Algebraic Computation (ISSAC 2012) (2012)
33. Smith, W.D.: Church's thesis meets the n-body problem. Appl. Math. Comput. 178(1), 154–183 (2006)
34. Müller, N., Moiske, B.: Solving initial value problems in polynomial time. Proc. 22 JAIIO - PANEL 1993, Part 2, 283–293 (1993)
35. Werschulz, A.G.: Computational complexity of one-step methods for systems of differential equations. Math. Comput. 34, 155–174 (1980)
36. Corless, R.M.: A new view of the computational complexity of IVP for ODE. Numer. Algorithms 31, 115–124 (2002)
37. Bournez, O., Graça, D.S., Pouly, A.: Solving analytic differential equations in polynomial time over unbounded domains. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 170–181. Springer, Heidelberg (2011)
38. Müller, N.T.: Uniform computational complexity of taylor series. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 435–444. Springer, Heidelberg (1987)
39. Müller, N.T., Korovina, M.V.: Making big steps in trajectories. Electr. Proc. Theoret. Comput. Sci. 24, 106–119 (2010)

An Overview of Genomic Distances Modeled with Indels*

Marília D.V. Braga

Inmetro—Instituto Nacional de Metrologia, Qualidade e Tecnologia,
Rio de Janeiro, Brazil

Abstract. The genomic distance typically describes the minimum number of large-scale mutations that transform one genome into another. Classical approaches to compute the genomic distance are usually limited to genomes with the same content and take into consideration only rearrangements that change the organization of the genome (i.e., positions and orientation of pieces of DNA, and number of chromosomes). In order to handle genomes with distinct contents, also insertions and deletions of pieces of DNA—named *indels*—must be allowed. Some extensions of the classical approaches lead to models that allow rearrangements and indels. In this work we introduce a new graph structure that gives a unified view of these approaches, present an overview of their results and point out some open problems related to them.

1 Introduction

Genomes contain the genetic information of living organisms and are composed of one or more DNA molecules, that are called *chromosomes*. A genome analysis may be done over a high-level view of genomes, in which only "relevant" fragments of the DNA (e.g., those that code for proteins) are taken into consideration. DNA molecules are organized in two antiparallel strands, and each fragment lies in one of the strands. The orientation of the fragment indicates in which of the two strands it is. The *distance* between two genomes is given as the minimum number of steps to transform one genome into the other allowing rearrangements that change the number of chromosomes, and the positions and orientations of fragments. Often it is computed using only the common fragments, that occur on both genomes.

In 1995, a pioneering work by Hannenhalli and Pevzner [10] showed how to compute the inversion distance (that is the genomic distance in which only inversions of chromosomal segments are allowed) between two unichromosomal genomes in polynomial time. An example is given in Fig. 1 (i). Later the same authors showed how to generalize these results to the so-called *HP model*, allowing, in addition to inversions, also translocations, fusions and fissions to compute the distance between two multichromosomal genomes [9]. This approach, based on a structure called *breakpoint graph* [1], can still be computed in polynomial

* This research was supported by the agency CNPq (PROMETRO 563087/10-2).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 22–31, 2013.

time, but rely on the analysis of many particular cases and is full of technical details.

In 2005, Yancopoulos, Attie and Friedberg [14] showed that all mentioned rearrangements can be generically represented as a *double-cut-and-join* (DCJ) operation. This new DCJ model can be analyzed with the help of the *adjacency graph* [2], a structure that is very similar to the breakpoint graph. The DCJ distance, which takes into consideration only DCJ operations, can be computed in linear time, with an approach that is much simpler than the one used in the HP model, but allows circular chromosomes to be created [14].

While transforming a genome into another in the *general DCJ model*, many circular chromosomes can coexist in some intermediate genomes [2]. Due to this fact, when the compared genomes are linear, it is desirable to consider the so-called *restricted DCJ model*, in which we perform the reincorporation of a circular chromosome immediately after its creation [11,14]. These two consecutive DCJs, which create and reincorporate a circular chromosome, mimic a transposition or a block-interchange. In other words, in the restricted DCJ model most of the classical rearrangements (inversions, translocations, fusions and fissions) cost one DCJ, while transpositions and block-interchanges cost two DCJs. The restricted DCJ distance is the same as the general DCJ distance and thus can be computed in linear time [2, 14]. Fig. 1 (ii and iii) shows an example of a general and a restricted DCJ sequence transforming one genome into another.

If the genomes have unequal contents, in addition to rearrangements it is necessary to consider *insertions* and *deletions* of DNA segments, that are jointly called *indels*. Some extensions of the classical approaches lead to models that allow rearrangements and indels. In 2001, El Mabrouk [8] extended the classical sorting by inversions approach [10] and developed a method to compare unichromosomal genomes with unequal contents, considering only inversions and indels,

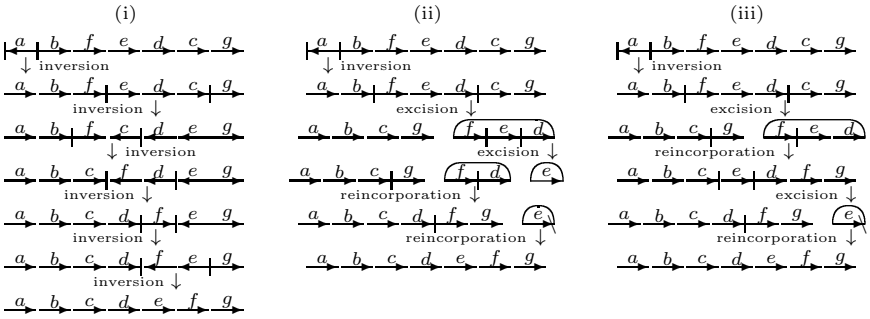


Fig. 1. Optimal sorting sequences in different models. (i) Inversion model (6 steps). (ii) General DCJ model (5 steps)—several circular chromosomes can coexist in the intermediate genomes. (ii) Restricted DCJ model (5 steps)—a circular chromosome is immediately reincorporated after its excision. Observe that the first excision-reincorporation mimics the interchange of blocks c and f. Analogously, the second excision-reincorporation mimics the transposition of blocks d and e.

both having the same cost. She provided an exact algorithm that deals with insertions and deletions asymmetrically, and a heuristic that handles the operations symmetrically. Then, in 2009, a model to sort multichromosomal genomes with unequal contents was introduced by Yancopoulos and Friedberg [15], using both DCJ and indel operations. Later, Braga *et al.* [5] presented an exact formula that can be computed in linear time handling indels symmetrically, allowing the indel cost to be distinct from and upper bounded by the DCJ cost [7].

In 2011 a more powerful operation has been introduced: a *substitution* allows a piece of DNA to be substituted by another piece of DNA [3]. It has been shown that the DCJ-substitution distance can also be computed in linear time, allowing the substitution cost to be distinct from and upper bounded by the DCJ cost [7]. Indels and substitutions are applied to pieces of DNA of any size, and a side effect of this fact is that the triangular inequality often does not hold for distances that consider these operations [4, 7, 8, 15]. However, in general it is possible to do an *a posteriori* correction, using an approach described in [4, 7].

This paper is organized as follows. In Section 2 we give definitions used in this work. In Section 3 we introduce a new graph representation that allows the analysis of different genomic distances. In Section 4 we explain the triangular inequality problem in these distances. Then, in Section 5 we present an overview of the latest results, for different models that consider both rearrangements and indels, and point out some open problems related to these models.

2 Definitions

Each marker in a genome is an oriented DNA fragment. We describe models in which duplicated markers are not allowed. The representation of a marker g in a genome A can be the symbol g, if it is read in direct orientation in A, or the symbol $\overline{g}$, if it is read in reverse orientation in A. Each one of the two extremities of a linear chromosome is called a *telomere*. Each telomere receives a unique number i and is then represented by the symbol $\textcircled{i}$. Each chromosome in a genome can be represented by a string of markers that can be circular, if the chromosome is circular, or linear and flanked by symbols $\textcircled{i}$ and $\textcircled{k}$, if the chromosome is linear. A genome is either circular (composed of circular chromosomes) or linear (composed of linear chromosomes).

2.1 Rearrangements and DCJ Operations

Genome rearrangements, such as inversions, translocations, fusions and fissions, allow us to change the number of chromosomes, the order and the orientation of the markers in a genome. In general, a rearrangement cuts a genome in two different positions, creating four open ends, and joins these open ends in a different way, and can be modeled by a *double-cut and join* or *DCJ* operation [14]. Consider, for example, a DCJ applied to genome $A = \{\textcircled{1}rb\overline{a}cest\overline{f}du\overline{g}h\textcircled{2}\}$, that cuts before and after $u\overline{g}$, creating the segments $\textcircled{1}r...d\bullet$, $\bullet u\overline{g}\bullet$ and $\bullet h\textcircled{2}$ (the symbol $\bullet$ represents the open ends). If we then join the first with the third and

the second with the fourth open end, we obtain $A' = \{①rb\overline{a}cest\overline{f}dg\overline{u}h②\}$. This DCJ corresponds to the inversion of contiguous markers $u\overline{g}$.

DCJ operations can also correspond to other rearrangements, such as translocations (if the genome is multichromosomal), circular excisions and reincorporations and, in particular cases, fusions and fissions [14]. Furthermore, as previously mentioned, a circular excision followed by a reincorporation can mimic a transposition or a block-interchange [11, 14].

2.2 Common and Unique Markers between Two Genomes

Given two genomes A and B, possibly with unequal contents, let $\mathcal{G}$, $\mathcal{A}$ and $\mathcal{B}$ be three disjoint sets, such that $\mathcal{G}$ is the set of markers that occur once in A and once in B, $\mathcal{A}$ is the set of markers that occur only in A and $\mathcal{B}$ is the set of markers that occur only in B. The markers in sets $\mathcal{A}$ and $\mathcal{B}$ are also called *unique markers between A and B*. Let $A = \{①rb\overline{a}cest\overline{f}du\overline{g}h②\}$ and $B = \{③avbcdwe\overline{x}fygzh④\}$ Then we have $\mathcal{G} = \{a, b, c, d, e, f, g, h\}$, $\mathcal{A} = \{r, s, t, u\}$ and $\mathcal{B} = \{v, w, x, y, z\}$.

2.3 Modifying the Content of a Genome

If the genomes have unequal contents, besides the rearrangements that change the organization of the genomes, we need operations that change the content of the genomes. These *content-modifying* operations, however, cannot be applied to the markers of $\mathcal{G}$ (the set of common markers of A and B).

Indels. The most classical content-modifying operations are *insertions* and *deletions* of blocks of contiguous markers [8, 15]. We refer to insertions and deletions as *indels*. In the models we consider, an indel can either delete or insert contiguous markers in a genome, with two restrictions: (i) markers of $\mathcal{G}$ cannot be deleted; and (ii) an insertion cannot produce duplicated markers [5]. At most one chromosome can be entirely deleted or inserted at once. We illustrate an indel with the following example: the deletion of markers *st* from genome $A = \{①rb\overline{a}cest\overline{f}du\overline{g}h②\}$, which results into $A' = \{①rb\overline{a}ce\overline{f}du\overline{g}h②\}$.

Substitutions. *Substitutions* are more powerful content-modifying operations, that allow blocks of contiguous markers to be substituted by other blocks of contiguous markers [3]. In other words, a deletion and a subsequent insertion that occur at the same position of the genome can be modeled as a substitution, counting together for one single step. Again, we have two restrictions: (i) markers of $\mathcal{G}$ cannot be substituted; and (ii) a substitution cannot produce duplicated markers [3]. An example is the substitution of markers *st* by $\overline{x}$ in genome $A = \{①rb\overline{a}cest\overline{f}du\overline{g}h②\}$, which results into genome $A' = \{①rb\overline{a}ce\overline{x}\overline{f}du\overline{g}h②\}$. At most one chromosome can be entirely substituted at once (but we do not allow the substitution of a linear by a circular chromosome and *vice-versa*). Observe that indels are special cases of substitutions. If a block of markers is substituted by the empty string, we have a deletion. Analogously, if the empty string is substituted by a block of markers, we have an insertion.

2.4 Sorting One Genome into Another

It is possible to transform or *sort* a genome A into a genome B with rearrangements and content-modifying operations: while sorting A into B, we delete the markers in $\mathcal{A}$, insert the markers in $\mathcal{B}$ (with indels and/or substitutions) and with rearrangements we change the organization of A so that we obtain the same markers with the same chromosome distribution, order and orientations of B. Each rearrangement or content-modifying operation is one sorting step and can be applied to a segment of contiguous markers of arbitrary size.

3 Modelling the Relation between Two Genomes

In order to find a parsimonious sequence of rearrangements (and indels) sorting one genome into the other, it is usually necessary to use some structure to represent the relation between the organization of two genomes. This task can be accomplished with the help of the *breakpoint graph*, proposed in [1] to analyze the inversion distance [10] and also used for the inversion-indel distance [8], or with the *adjacency graph*, proposed in [2] to analyze the DCJ distance, and then used for the general DCJ-indel and DCJ-substitution [7] and restricted DCJ-indel [6] distances. There is actually a clear duality between the adjacency and the breakpoint graphs, so that one is the line-graph of the other [13].

3.1 The Relational Graph

Given two genomes A and B, we introduce here the *relational graph*, denoted by $R(A,B)$, that is a generalization of both the breakpoint and the adjacency graphs. First we denote the two extremities of each $g \in \mathcal{G}$ by g^t (tail) and g^h (head). Then we create in $R(A,B)$ one *black* and one *gray vertex* for g^t, connected by a *dotted edge*, and one black and one gray vertex for g^h, also connected by a dotted edge. Furthermore, for each telomere in A we have one additional black vertex and for each telomere in B we have one additional gray vertex.

Let γ_1 and γ_2 be extremities (or telomeres) belonging to the same chromosome in A and let ℓ be the substring composed of the markers that are between γ_1 and γ_2. We have a *black edge* labeled by ℓ connecting the black vertices representing γ_1 and γ_2 if, and only if, ℓ contains no marker of $\mathcal{G}$. In the same way, let γ'_1 and γ'_2 be extremities (or telomeres) belonging to the same chromosome in B and let ℓ' be the substring composed of the markers that are between γ'_1 and γ'_2. Again, we have a *gray edge* labeled by ℓ' connecting the gray vertices representing γ'_1 and γ'_2 if, and only if, ℓ' contains no marker of $\mathcal{G}$.

Each vertex is connected by either one black or one gray edge and at most one dotted edge, thus the maximum degree of the vertices is two and the graph is a collection of paths and cycles. Telomeres are the endpoints of paths, that can be of three types: an *AB-path* (that has one endpoint in genome A and the other in B), or an *AA-path* (that has both endpoints in genome A), or a *BB-path* (that has both endpoints in B). Since the number of telomeres in each genome is even, the number of AB-paths is always even.

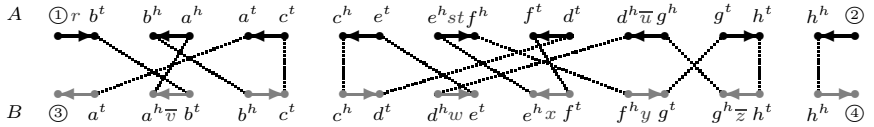


Fig. 2. For genomes $A = \{①rbacest\overline{f}du\overline{g}h②\}$ and $B = \{③avbcdwe\overline{x}fygzh④\}$, each one composed of a single linear chromosome, $R(A,B)$ has one cycle and two AB-paths

Unichromosomal linear diagram. If the two genomes are unichromosomal and linear, it is possible to represent the graph as a diagram in which the vertices of each genome are distributed in a horizontal line, and in the same order of the corresponding chromosome. By walking through each one of the components we assign a direction to the black and gray edges, that determine the direction in which the labels are read (see Fig. 2). This representation, as we shall see later, is especially useful for identifying sorting inversions, and can be extended to circular chromosomes [12] and to multichromosomal genomes [9].

Breakpoint graph. The breakpoint graph [1, 8] (Fig. 3) can be obtained from $R(A,B)$ by collapsing dotted edges and merging into a new *white vertex* their endpoints. Each new white vertex receives the same name of the corresponding merged vertices.

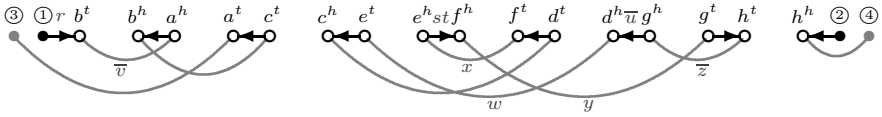


Fig. 3. Breakpoint graph obtained by collapsing dotted edges of the graph from Fig. 2 (directions of gray edges are omitted)

Adjacency graph. The adjacency graph [2,5] (Fig. 4) can be obtained by collapsing black and gray edges and merging their endpoints. Each new vertex receives a new name, that corresponds to the string obtained by concatenating the name of the endpoints, separated by the label of the original black (or gray) edge.

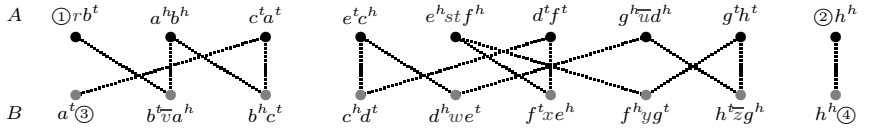


Fig. 4. Adjacency graph built collapsing gray and black edges of the graph from Fig. 2

3.2 Effect of Rearrangements on the Graph

Let c be the number of cycles and b be the number of AB-paths in the graph of A and B. It has been shown that $c + \frac{b}{2} \leq |\mathcal{G}|$. Moreover, if the common markers of A and B have the same organization (i.e., if by ignoring unique markers we obtain the same chromosome distribution, order and orientations for common markers), we have $c + \frac{b}{2} = |\mathcal{G}|$. Thus, in order to sort the common content of A into the common content of B, we need to increase either c or b. By cutting and rearranging two black edges of the graph, with a DCJ applied to genome A, we change the number of cycles by $+1$ or -1, or we change the number of AB-paths by $+2$ or -2, or the number of cycles and AB-paths remain unchanged [2]. It has been shown that rearrangements applied to two black (or two gray) edges belonging to the same cycle or path can increase the number of cycles or AB-paths [2,10]. In the DCJ model the directions of the affected edges do not matter and it is always possible to increase c or b at each step. Due to this fact, the DCJ distance is given by $d_{\text{DCJ}}(A, B) = |\mathcal{G}| - \left(c + \frac{b}{2}\right)$ [2].

In the inversion model, the genomes are unichromosomal, the number of telomeres in each genome is at most two, and, consequently, the number of paths in the graph is at most two. Without loss of generality, it is possible to transform the paths into one or two cycles [10], so that we only need to take care of cycles. Then, for finding inversions that increase the number of cycles, we first need to represent the graph as a diagram, in which an inversion can only increase c when it is applied to two black (or two gray) edges that have opposite directions (see Fig. 5). Many particular cases appear when the graph has *bad cycles*, that do not have edges with opposite directions and may require some extra steps to be sorted. Nevertheless, the inversion distance, that is lower bounded by the DCJ distance, can still be efficiently computed [10]. Observe that the diagram view also applies to the breakpoint graph, but in an asymmetric way, in which only the inversions in one of the two genomes can be identified.

3.3 Minimizing Indels and Substitutions

It is possible to combine rearrangements and indels (or substitutions) so that the overall number of operations is minimized. A rearrangement involving two

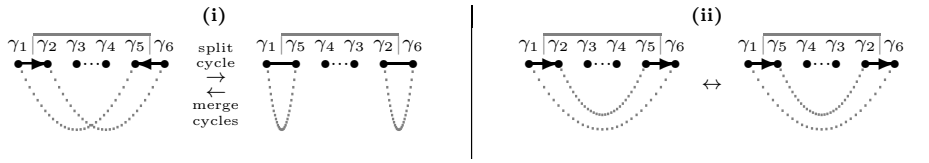


Fig. 5. Effects of an inversion in the diagram. (i) If the edges are in the same cycle and with opposite directions, the inversion splits the cycle. Inversely, if the edges are in different cycles, the inversion merges them (independently of the directions of the original edges, that are omitted). (ii) If the edges are in the same cycle with the same direction, the number of cycles remains unchanged.

labeled edges can concatenate the labels into one single edge, decreasing the number of indels. For example, take the gray edges $f^h y g^t$ and $h^t \overline{z} g^h$ from genome B (Fig. 2). A DCJ applied to these two edges could result into $h^t g^t$ and $f^h y \overline{z} g^h$, in which the label $y\overline{z}$ results from the concatenation of the labels of the two original edges. In particular, when we apply rearrangements in two edges (both black or both gray) belonging to a single component of the graph $R(A,B)$, we can concatenate labels while we create a new component [5]. Thus, the cycles and paths of our graph also establish a relation between indels and allow us to efficiently concatenate labels while increasing the number of components. This is a key idea for computing the genomic distances in the models we describe here.

4 The Triangular Inequality Problem

Since content-modifying operations can be applied to blocks of markers of arbitrary size, the triangular inequality does not hold for genomic distances that consider this type of operation. Given any three genomes A, B and C and a distance measure d, consider without loss of generality that $d(A, B) \geq d(A, C)$ and $d(A, B) \geq d(B, C)$. Then the triangular inequality is the property that guarantees that $d(A, B) \leq d(A, C) + d(B, C)$. Although this property holds for the classical models that consider only rearrangements, it does not hold for the approaches that allow content-modifying operations. Consider for example the genomes $A = \{\circ abcde\circ\}$, $B = \{\circ ac\overline{d}be\circ\}$ and $C = \{\circ ae\circ\}$ [15]. While A and B can be sorted into C with only one indel, the minimum number of inversions required to sort A into B is three. In this case we have $d(A, B) = 3$, $d(A, C) = 1$, $d(B, C) = 1$ and the triangular inequality is disrupted.

The triangular inequality must hold if one intends to use the distance to compute the *median* of three or more genomes [13] and in phylogenetic reconstructions. Fortunately, a correction can be applied *a posteriori*, as proposed in [4, 5], and consists in summing to the distance a surcharge that depends on the number of unique markers. It has been shown that, for each genomic distance d that we describe in the next section, there is a positive constant k such that, for any $k' \geq k$ the triangular inequality holds for the function $m(A, B) = d(A, B) + k'(|\mathcal{A}| + |\mathcal{B}|)$.

5 Overview of Genomic Distances with Indels

We shall now present an overview of some known results in genomic distances that allow indels, and some open problems related to these approaches. In general we assign the cost of 1 to each rearrangement and a positive cost w to each content-modifying operation. Then, given two genomes A and B, the distance of A and B is the minimum cost of a sequence of rearrangements and content-modifying operations that transforms A into B.

5.1 The DCJ-Indel Distance

In the DCJ-indel distance the genomes can be multichromosomal, the rearrangements are all generic DCJ operations and the content-modifying operations are

insertions and deletions. For any positive cost $w \leq 1$ assigned to indels, there is a formula to efficiently compute the distance [5,7]. Furthermore, for each $w \leq 1$, the constant to establish the triangular inequality is $k = \frac{w+1}{2}$ [7]. The whole family of problems in which we have $w > 1$ remains open.

The restricted version. Although there is no method to exactly compute the restricted DCJ-indel distance up to now, for any $w \leq 1$, a good upper bound was proposed in [6]. This upper bound allows the triangular inequality correction for the general DCJ-indel distance to be extended to the restricted DCJ-indel distance, that is, here we also have the constant $k = \frac{w+1}{2}$. We conjecture that the general and the restricted DCJ-indel distances are equal.

5.2 The DCJ-Substitution Distance

In the DCJ-substitution distance again the genomes can be multichromosomal and the rearrangements are all generic DCJ operations, but the content-modifying operations are substitutions. For any positive cost $w \leq 1$ assigned to substitutions, there is a formula to efficiently compute the distance [3, 7]. The constant to establish the triangular inequality is $k = \frac{w+2}{4}$ for each $w \leq 1$ [7]. Again, the whole family of problems in which we have $w > 1$ remains open.

The restricted version. The general and the restricted DCJ-substitution distances are not the same, as we can see in the example given in Fig. 6. The restricted version of the DCJ-substitution distance is a complete open problem.

5.3 The Inversion-Indel Distance

In the inversion-indel distance, that applies to unichromosomal genomes, the rearrangements can be only inversions and the content-modifying operations are indels. An exact algorithm was given for the case in which $w = 1$ (the same cost is assigned to inversions and indels) and only one indel direction is allowed (i. e. when we have only insertions or only deletions) [8]. For the general case, in which indels are handled symmetrically, a heuristic was provided [8], but obtaining an exact solution remains an open problem, as well as extending the model to allow distinct inversion and indel costs. The triangular inequality is

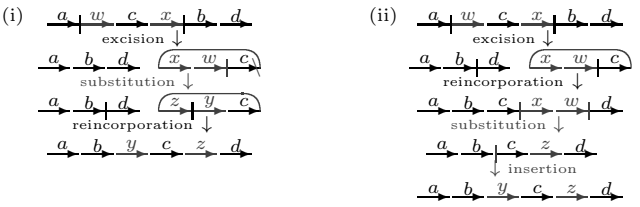


Fig. 6. (i) An optimal sorting sequence in the general DCJ-substitution model (3 steps). (ii) An optimal sorting sequence in the restricted DCJ-substitution model (4 steps).

disrupted in the inversion-indel distance and can be corrected as described in Section 4, but computing the value of the constant k is another open problem.

The difficulty in handling indels symmetrically probably derived from the complexity of integrating indels with the treatment of the many particular cases of the inversion model, and from the asymmetry of the breakpoint graph used in this analysis. The use of the symmetric relational graph may then bring some insights to the solution of this problem.

References

1. Bafna, V., Pevzner, P.: Genome rearrangements and sorting by reversals. In: Proc. of FOCS, pp. 148–157 (1993)
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
3. Braga, M.D.V., Machado, R., Ribeiro, L.C., Stoye, J.: Genomic distance under gene substitutions. BMC Bioinformatics 12(suppl. 9), S8 (2011)
4. Braga, M.D.V., Machado, R., Ribeiro, L.C., Stoye, J.: On the weight of indels in genomic distances. BMC Bioinformatics 12(suppl. 9), S13 (2011)
5. Braga, M.D.V., Willing, E., Stoye, J.: Double cut and join with insertions and deletions. J. Comp. Biol. 18(9), 1167–1184 (2011); A preliminary version appeared in Moulton, V., Singh, M. (eds.): WABI 2010. LNCS (LNBI), vol. 6293. Springer, Heidelberg (2010)
6. da Silva, P.H., Machado, R., Dantas, S., Braga, M.D.V.: Restricted DCJ-indel model: sorting linear genomes with DCJ and indels. In: Proc. of RECOMB-CG 2012, BMC Bioinformatics, vol. 13-S9, p. S14 (2012)
7. da Silva, P.H., Braga, M.D.V., Machado, R., Dantas, S.: DCJ-indel distance with distinct operation costs. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS (LNBI), vol. 7534, pp. 378–390. Springer, Heidelberg (2012)
8. El-Mabrouk, N.: Sorting signed permutations by reversals and insertions/deletions of contiguous segments. J. of Disc. Alg. 1(1), 105–122 (2001)
9. Hannenhalli, S., Pevzner, P.: Transforming men into mice (polynomial algorithm for genomic distance problem). In: Proc. of FOCS, pp. 581–592 (1995)
10. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J. of the ACM 46, 1–27 (1999); A preliminary version appeared in Proc. of STOC 1995
11. Kovác, J., Warren, R., Braga, M.D.V., Stoye, J.: Restricted DCJ model (the problem of chromosome reincorporation). J. Comp. Biol. 18(9), 1231–1241 (2011)
12. Meidanis, J., Walter, M.E.M.T., Dias, Z.: Reversal distance of signed circular chromosomes. Technical Report IC-00-23, University of Campinas (2000)
13. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics 10(120) (2009)
14. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21 (2005)
15. Yancopoulos, S., Friedberg, R.: DCJ path formulation for genome transformations which include insertions, deletions, and duplications. J. Comp. Biol. 16(10), 1311–1338 (2009)

Noise versus Computational Intractability in Dynamics

Mark Braverman*

Department of Computer Science, Princeton University, U.S.A.
mbraverm@cs.princeton.edu

Computation plays a key role in predicting and analyzing natural phenomena. There are two fundamental barriers to our ability to computationally understand the long-term behavior of a dynamical system that describes a natural process. The first one is unaccounted-for errors, which may make the system unpredictable beyond a very limited time horizon. This is especially true for chaotic systems, where a small change in the initial conditions may cause a dramatic shift in the trajectories. The second one is Turing-completeness. By the undecidability of the Halting Problem, the long-term prospects of a system that can simulate a Turing Machine cannot be determined computationally.

We shall discuss the interplay between these two forces – unaccounted-for errors and Turing-completeness. We show that the introduction of even a small amount of noise into a dynamical system is sufficient to "destroy" Turing-completeness, and to make the system's long-term behavior computationally predictable. On a more technical level, we deal with long-term statistical properties of dynamical systems, as described by invariant measures. We show that while there are simple dynamical systems for which the invariant measures are non-computable, perturbing such systems makes the invariant measures efficiently computable. Thus, noise that makes the short term behavior of the system harder to predict, may make its long term statistical behavior computationally tractable. We also obtain some insight into the computational complexity of predicting systems affected by random noise.

A significant part of this talk is based on joint work with Alexander Grigo and Christobal Rojas [1].

Reference

1. Braverman, M., Grigo, A., Rojas, C.: Noise vs computational intractability in dynamics. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 128–141. ACM (2012); Available as preprint arXiv:1201.0488

* Research supported in part by an NSERC Discovery Grant, Alfred P. Sloan Fellowship, an NSF CAREER award (CCF-1149888), and a Turing Centenary Fellowship.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, p. 32, 2013.

Cluster Editing

Sebastian Böcker[1] and Jan Baumbach[2]

[1] Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany
sebastian.boecker@uni-jena.de
[2] Computational Biology Research Group, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark
jan.baumbach@imada.sdu.dk

Abstract. The Cluster Editing problem asks to transform a graph into a disjoint union of cliques using a minimum number of edge modifications. Although the problem has been proven NP-complete several times, it has nevertheless attracted much research both from the theoretical and the applied side. The problem has been the inspiration for numerous algorithms in bioinformatics, aiming at clustering entities such as genes, proteins, phenotypes, or patients. In this paper, we review exact and heuristic methods that have been proposed for the Cluster Editing problem, and also applications of these algorithms for biological problems.

1 Introduction

Given an undirected graph G, the Cluster Editing problem asks to transform G into a vertex-disjoint union of cliques by a minimum number of edge modifications; see Fig. 1. In the corresponding Weighted Cluster Editing problem, we are given modification costs for each edge or non-edge, and we ask for a set of edge modifications with minimum total weight. To prove worst-case running times, we sometimes assume that all modification costs are integers; in this case, we additionally assume that all edges have non-zero modification cost. Cf. § 7 for Cluster Editing with zero-edges. Let n denote the number of vertices in the input graph, and let m denote the number of edges.

In application, the above task corresponds to clustering objects, that is, partitioning a set of objects into homogeneous and well-separated subsets. Similar objects are connected by an edge, and a cluster is a clique of the input graph. The

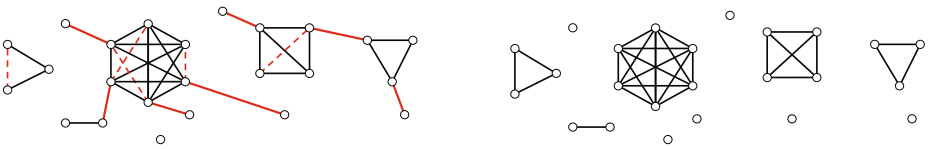


Fig. 1. Cluster Editing instance before and after editing; bogus edges in the input graph are marked in red, missing edges additionally as dashed

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 33–44, 2013.

input graph is corrupted and we have to clean (edit) the graph to reconstruct the clustering under the parsimony criterion. The Cluster Editing formulation has been successfully applied in bioinformatics for clustering genes and proteins, or to process transcription data, see § 6 for details. In addition, Cluster Editing has served as the inspiration for heuristic clustering algorithms such as CLICK [1], CAST [2], or HCS [3].

Early work on the Cluster Editing problem dates back to at least the 1960s, when Zahn [4] solved the problem for a particular type of input graphs. Another early result is due to Moon [5], who showed that almost all graphs are almost maximally far away from transitivity: Consider the set $\mathcal{G}_n$ of all undirected graphs with n vertices. Any such graph can be transformed either into the empty graph or the complete graph using at most $\binom{n}{2}/2$ edge modifications. For fixed $\epsilon > 0$ consider the set $\mathcal{G}_{n,\epsilon}$ of "hard" instances, which require at least $(1-\epsilon) \cdot \binom{n}{2}/2$ edge modifications. Then, $|\mathcal{G}_{n,\epsilon}| \,/\, |\mathcal{G}_n| \to 1$ as $n \to \infty$. In application, this result is much less disturbing than it may appear: It is only true as n approaches infinity; and, real-world instances usually belong to the "rare" instances where much less modifications are required.

During the last years, the Cluster Editing problem has received numerous names,[1] such as Transitive Approximation problem or Transitive Graph Projection problem [6]. Also, certain problems such as Clique Partitioning [7, 8] can be easily seen to be equivalent to Cluster Editing. Finally, the problem is closely related to the Consensus Clustering problem.

2 Preliminaries

For brevity, we write uv as shorthand for an unordered pair $\{u, v\} \in \binom{V}{2}$. We can encode the input of a (weighted or unweighted) Cluster Editing instance in a single *weight function* $s : \binom{V}{2} \to \mathbb{R}$: For $s(uv) > 0$ a pair uv is an edge of the graph and has deletion cost $s(uv)$, while for $s(uv) < 0$, the pair uv is not an edge (a *non-edge*) of the graph and has insertion cost $-s(uv)$. If $s(uv) = 0$, we call uv a *zero-edge*; the presence of such zero-edges makes the Cluster Editing problem considerably harder, cf. § 7. An unweighted Cluster Editing instance can be encoded as an integer-weighted instance by assigning weights $s(uv) \in \{+1, -1\}$. To solve (weighted) Cluster Editing we first identify all connected components of the input graph and calculate the best solutions for each component separately, because an optimal solution never connects disconnected components. Furthermore, if the graph is decomposed during the course of an algorithm, then we can recurse and treat each connected component individually.

Vertices uvw form a *conflict triple* if uv and vw are edges but uw is a non-edge (or a zero-edge, for the weighted case). An unweighted graph is *transitive* if it is a disjoint union of cliques; this is the case if and only if it does not contain any conflict triples. To this end, many algorithms mentioned below are based on removing all conflict triples from the input.

[1] In fact, one can argue that the name "Cluster Editing" is somewhat ill-chosen, as we are not editing clusters but edges.

We may sometimes set pairs uv to "*forbidden*" or "*permanent*": This may happen in a preprocessing step when we are sure that two vertices *cannot* or *have to* end up in the same clique/cluster; or, in a search tree algorithm where we draw this conclusion only for this particular part of the search tree, as all other cases will be taken care of elsewhere. The advantage of the weighted Cluster Editing variant is that we can immediately *merge* any permanent pair uv, replacing the vertices u and v with a single vertex u', and, for all vertices $w \in V \setminus \{u, v\}$, replacing pairs uw, vw with a single pair $u'w$. The weight of the joined pair is $s(u'w) = s(uw) + s(vw)$. In case one of the pairs is an edge while the other is a non-edge, this immediately "generates cost" and, for parameterized algorithms, reduces the cost parameter.

We encode a *forbidden pair* uv in our weight function s by setting $s(uv) = -\infty$. By definition, every forbidden pair uv is a non-edge, since $s(uv) < 0$. A forbidden pair uw can be part of a conflict triple uvw. We mention in passing that this encoding works exactly as it should when we merge vertices incident to forbidden edges.

3 Complexity of the Problem

The Cluster Editing problem is NP-hard, and several proofs of this fact have been published independently [9–11]. To the best of our knowledge, the first such proof is due to Křivánek and Morávek in 1986 [12], who call the problem "$S \,\Delta\, E$" or "best approximation of a symmetric relation by an equivalence relation". Later, stronger results were achieved: In particular, the Cluster Editing problem remains NP-hard on graphs with maximum degree six, and if at most four edge modifications incident to each vertex are allowed [13]. Under the exponential time hypothesis [14], no exact algorithm can exist with running time subexponential in n [13].

Cluster Editing can be solved in polynomial time for particular graph types: This is the case for unit interval graphs (1-dimensional vicinity graphs) since clusters in an optimal solution are consecutive sets [15]. It is also well-known that Cluster Editing can be solved in polynomial time for graphs of maximum degree two, namely paths and circles, see for example [16]. It is an open question whether Cluster Editing on graphs with maximum degree three to five is NP-hard or polynomial-time solvable [13]. For certain graphs that are "not too far" from a cluster graph, a greedy-type heuristic can find the optimal solution in polynomial time [6]. The variant of the Cluster Editing problem where we are only allowed to insert edges, is clearly polynomial-time solvable (we simply insert all edges for each connected component) and is better known as the *transitive closure* of the input graph.

A problem with input size n and parameter k is *fixed-parameter tractable* (FPT) if it can be solved in $O(f(k) \cdot \text{poly}(n))$ time where f is any computable function and $\text{poly}(\cdot)$ is a polynomial. For a general introduction we refer to [17]. In the following, let n be the number of vertices, and k the number of edge modifications. For this parameter, the Cluster Editing problem is fixed-parameter

tractable, and both kernels and search tree algorithms exist, cf. § 5. The even more interesting question is whether a parameterized algorithm with *subexponential* running time exists for Cluster Editing, as conjectured by Cao and Chen [18]. Unfortunately, Komusiewicz and Uhlmann [13] and Fomin et al. [19] independently showed that under the exponential time hypothesis, no such algorithm can exist.

Concerning the approximability of the Cluster Editing problem, Charikar et al. [20] showed that the problem is APX-hard (that is, is NP-hard to approximate within some constant factor greater than 1). There exist several (randomized and deterministic) constant-factor approximations for the problem [20–22]. We omit further details, as such approximation algorithms are outside the focus of this review: We are interested in computing solutions that are *structurally similar* to the "true" solution, and this is hopefully the case for the optimal solution. In contrast, constant-factor approximation solutions usually have no structural similarity whatsoever to the optimal solution. Well-known examples of this fact from bioinformatics include the center star 2-approximation for the Multiple Sequence Alignment problem, or the spanning tree 2-approximation for the Maximum Parsimony problem in phylogenetic tree reconstruction.

4 Integer Linear Programming

Solving Cluster Editing using Integer Linear Programming (ILP) and its Linear Programming (LP) relaxation has been in the focus of research groups in the 1980s [23–27]. Particularly noteworthy is the paper by Grötschel and Wakabayashi [7] from 1989. The basic ILP for Cluster Editing can be directly derived from the conflict triple characterization; a faster algorithm is obtained by a mathematical analysis of the corresponding clique partitioning polytope. Grötschel and Wakabayashi proposed several partition inequalities for this purpose; it turns out that their simplest form, namely 2-partition inequalities, is sufficient in practice [7, 28]. As there is an exponential number of 2-partition inequalities, one follows a *cutting plane* approach where these inequalities are added to the Linear Program only if they are violated by a current fractional solution.

In 2005, Kochenberger et al. [8] claimed that the cutting plane approach of Grötschel and Wakabayashi [7] cannot solve instances with more than 50 nodes in reasonable time. But in 2008, Böcker et al. [29] showed that this formulation can solve instances with 1000 nodes in about an hour. In part, this may be attributed to the advances in ILP solvers. Another possible explanation for the good performance in [29] is the particular structure of the used instances: These were generated by disturbing an ideal cluster graph using random edge insertions and deletions, then applying a data reduction that results in instances of the reported size, see below. In particular, the "merge operation" used as part of the data reduction, may result in uneven edge weight distributions that are advantageous for the ILP. An implementation of the ILP from [28, 29],

together with the corresponding benchmark datasets, can be downloaded from `http://bio.informatik.uni-jena.de/software/peace`.

5 Parameterized Algorithms and Data Reduction

The parameterized complexity of Cluster Editing, using the number of edge modifications as parameter k, is particularly well-studied. Parameterized algorithms require a cost limit k to be given in advance: In case a solution with cost $\leq k$ exists, the algorithm finds this solution and answers "yes"; otherwise, "no solution" is returned. If we want to find an optimal solution but do not know k, we simply call the algorithm repeatedly for $k = 0, 1, 2, \ldots$. As the running time of the last iteration dominates that of all previous calls, we end up with exactly the worst-case time complexities mentioned below. Finally, note that we are interested in the solution itself, whereas the algorithmic presentation often focuses on the decision variant of the Cluster Editing problem. In practice, all of the algorithms can easily be modified to also compute the optimal solution.

The first results on the parameterized complexity of the Cluster Editing problem were given by Gramm et al. [30] who present a simple algorithm with running time $O(3^k+n^3)$ and, using a refined branching strategy, to $O(2.27^k+n^3)$. Gramm et al. [31] later improved this to $O(1.92^k + n^3)$ by an automated and extensive case analysis; the resulting algorithm has more than 100 initial branching cases and, to the best of our knowledge, has never been implemented. By transforming the unweighted Cluster Editing problem to the integer-weighted variant, running time was advanced to $O(1.82^k+n^3)$ by Böcker et al. [32]. Later, Böcker and Damaschke used a characterization of graphs that do not contain many conflicts, to derive an algorithm with running time $O(1.76^k + m + n)$ [33]. The currently fastest algorithm solves Cluster Editing in $O(1.62^k + m + n)$ time [16].

Notably, results in [16, 33] are based on a very simple branching algorithm first proposed in [32], that works on integer-weighted instances of the Cluster Editing problem: We first search for a conflict triple uvw, that is, edges uv and vw plus a non-edge uw. We branch into two cases: Either we delete edge uv; or, we merge it as described in § 2. Some bookkeeping is then required to deal with zero-edges that may appear by merging an edge. It is somewhat surprising that an algorithm this simple can, using an involved analysis and solving certain special (simple) cases upfront, result in such a competitive running time. It will also be interesting to see in the future whether even better running time bounds can be proven for this simple branching algorithm by an improved analysis.

A *kernelization* is a polynomial-time algorithm that transforms a given instance I with parameter k, into a new instance I' with parameter $k' \leq k$, such that (i) the instance (I, k) has a solution if and only if the instance (I', k') has a solution, and (ii) the size of the instance I' is at most $f(k)$ for some computable function f. Kernelization is often achieved by applying a set of reduction rules that cut away parts of the instance that are easy to handle. The kernel size is a measure for the effectiveness of the kernel.

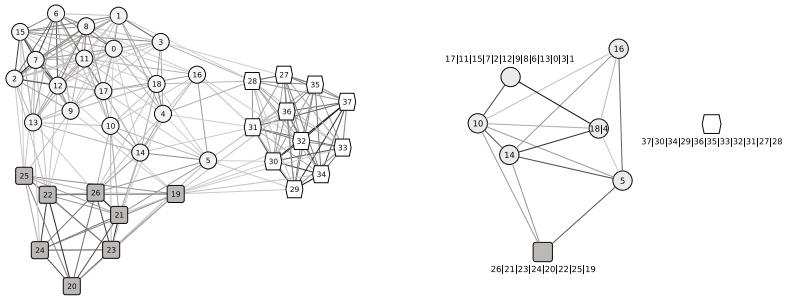


Fig. 2. Data reduction results for the "Leukemia" gene expression dataset [39]. Three different Leukemia subtypes (AML, ALL-B, ALL-T) shown as hexagons, circles, and squares, respectively. Input graph (left) and graph after data reduction (right).

Gramm et al. [30] showed how to compute a kernel for the Cluster Editing problem with $O(k^2)$ vertices in $O(n^3)$ time. A faster kernelization was given by Protti et al. [34] who computed a kernel with $2k^2 + k$ vertices in $O(n + m)$ time. At IWPEC 2006, Fellows [35] presented a polynomial-time kernel with $24k$ vertices, based on a Linear Programming formulation of the problem. In 2007, Fellows et al. [36] gave a combinatorial kernel with $6k$ vertices based on crown-type reductions. Guo [37] presented a kernel with $4k$ vertices that can be computed in $O(n^3)$ time [37], which is based on the notion of critical cliques. This was later improved to $2k$ vertices by Chen and Cao [38]. Similarly, a kernel for the integer-weighted variant with $O(k^2)$ vertices [32] was improved to $2k$ vertices [18]. It is also worth noting that the "critical clique" idea of Guo [37] allows us to directly transform an unweighted Cluster Editing instance, into an integer-weighted instance with at most $4k_{\text{opt}}$ vertices in $O(m+n)$ time, where k_{opt} are the optimal costs of the unweighted instance [32].

Kernel methods can also be viewed as a *data reduction*; in fact, we know many data reduction rules that do not result in improved kernel sizes but are still useful in practice. Parameterized rules can sometimes be made "parameter-independent" as described [28]; this allows us to apply the (polynomial-time) data reduction upfront, and solve the remaining instance using any exact or heuristic method. Data reduction can have a dramatic effect on the size of the instance, see Fig. 2 for a real-world example for gene expression data. To find exact solutions in practice, a combination of data reduction and Integer Linear Programming proved to be particularly efficient [28]. An implementation of the data reduction and parameterized algorithms from [28] can again be downloaded from `http://bio.informatik.uni-jena.de/software/peace`.

6 Cluster Editing in Practice

Clustering data still represents a key step of numerous life science problems. The weighted variant of the Cluster Editing problem has been frequently

proposed for clustering biological entities such as proteins or genes, and serves as the inspiration for numerous heuristic clustering algorithms [1–3].

The *FORCE* software [40] implements a heuristic for solving the weighted Cluster Editing problem, which is based on simulating physical forces and molecule movement coupled to simulated annealing. The more recent *Transitivity Clustering* [41] software combines the strategy behind *FORCE* with a parameterized algorithm to balance scalability and accuracy. This is decided by estimating an upper bound for the costs by using a fast but inaccurate approximation algorithm. If this bound is low, the exact fixed-parameter algorithm is applied, the force-based heuristic otherwise. The tool is available as stand-alone tool, as Cytoscape [42] plugin, as web tool for online computation, and as part of the *ClusterMaker* framework [43]. The web site (`http://transclust.mpi-inf.mpg.de`) provides corresponding information, installation instructions, workflows and test data as well as real-world project data. In the following, we give some examples of recent real-world applications of these tools to very briefly exemplify the impact of Cluster Editing in current computational biology. For all clustering settings we have given a symmetric similarity matrix, based on Euclidean distances, statistical measures (correlation coefficient, for instance) or other functions (binary local alignment scores, for instance). A reference implementation as well as evaluation data sets are available at the Transitivity Clustering web site. New methods should be compared to those.

- **Genetics and genomics:** Preprocessing of bacterial *de novo* sequencing data [44], evolution of transcription factors [45], and large-scale detection of homologous genes [46]. *Transitivity Clustering* was also used to find reasoning regarding the robustness of Cluster Editing for varying similarity thresholds and missing edge weights, at least for clustering bacterial genes and analyzing their pathogenicity [47, 48]. Similarly, the genomes of 16 methanogenic genomes were partitioned to characterize the genetic repertoire of the mesophilic hydrogenotrophic *Methanocella paludicola* [49].
- **Transcriptomics:** Comparative transcriptomics of the LexA regulon in corynebacteria [50], inter-species transfer and evolution of transcriptional gene regulatory networks between different actinobacteria [51, 52] as well as different pathogenic *E. coli* strains [53], and cancer sub-typing based on clustering gene expression data [54].
- **Metabolomics:** Metabolic profiling in ion mobility spectrometry data of human exhaled air, bacterial colonies and several more application cases [55].
- **Proteomics:** Identification of protein complexes in interactome networks [56], and unraveling of molecular factors influencing polyglutamine (polyQ) neurodegenerative diseases [57]. In another study bacterial exoproteomes were partitioned with *Transitivity Clustering* into pathogenic and non-pathogenic components [58].
- **Network analysis:** As part of *DEFOG*, functional analysis of gene sets by hierarchically organizing the genes into functionally related modules [59].

This list is far from being complete but shall rather illustrate that Cluster Editing is a nowadays very frequently utilized method for data partitioning in the life sciences.

7 Problem Variants

Above, we have stated explicitly that for the integer-weighted variant of Cluster Editing, we do not allow edges with modification costs zero in the input graph. In fact, worst-case running times considerations for the parameterized algorithms from § 5 do not hold if there are zero-edges present in the input, and much of the analysis in [16, 32] focuses on zero-edges that appear in the course of the algorithm. Recently, two groups independently showed that Cluster Editing with zero-edges ("don't care edges") is fixed-parameter tractable, too [60, 61]; Marx and Razgon [60] also provide a running time bound of $O\big(2^{O(k^3)} \cdot \mathrm{poly}(n)\big)$.

Damaschke [62] considered the enumeration of all optimal solutions of Cluster Editing and related problems. Alternative parameterizations of the Cluster Editing problem have been studied in [63]. Fomin et al. [19] studied Cluster Editing parameterized by two parameters (namely, modification costs k and a lower bound on the number of cliques p) and gave a randomized algorithm with *subexponential* running time $O\big(2^{O(\sqrt{pk})} + \mathrm{poly}(n)\big)$.

The closest relative of Cluster Editing is presumably the Cluster Deletion problem, where we are only allowed to remove edges in the input graph. This problem is also NP-hard, and parameterized algorithms exist similar to those mentioned above [30, 31, 64]. The currently fastest algorithm for this problem has worst-case running time $O(1.415^k + \mathrm{poly}(n))$ [33]. The directed variant of the Cluster Editing problem is called Transitivity Editing, and parameterized algorithms and an ILP have been proposed for this variant [65, 66]. Other problem variants include Cluster Vertex Deletion where we delete vertices instead of edges, or s-plex Editing where we do not demand the resulting graph to contain "perfect cliques". Such problems have also been studied frequently in the literature but are beyond the scope of this review.

8 Conclusion

The Cluster Editing problem and its variants have, for many decades, stimulated the creativity of numerous researchers. This rich and long-standing research comes at the prize that the same problem is known under many different names, which sometimes makes it hard to find all relevant related work. Despite its computational complexity, the problem is also highly relevant in applications, and swift algorithms (both exact and heuristic) have been developed to solve real-world instances. We assume that this process will continue in the future, and that the Cluster Editing problem "has come to stay."

Acknowledgments. We thank Anke Truss for providing Fig. 2.

References

1. Sharan, R., Maron-Katz, A., Shamir, R.: CLICK and EXPANDER: a system for clustering and visualizing gene expression data. Bioinformatics 19(14), 1787–1799 (2003)
2. Ben-Dor, A., Shamir, R., Yakhini, Z.: Clustering gene expression patterns. J. Comput. Biol. 6(3-4), 281–297 (1999)
3. Hartuv, E., Schmitt, A.O., Lange, J., Meier-Ewert, S., Lehrach, H., Shamir, R.: An algorithm for clustering cDNA fingerprints. Genomics 66(3), 249–256 (2000)
4. Zahn Jr., C.T.: Approximating symmetric relations by equivalence relations. J. Soc. Indust. Appl. Math. 12(4), 840–847 (1964)
5. Moon, J.W.: A note on approximating symmetric relations by equivalence classes. Siam J. Appl. Math. 14(2), 226–227 (1966)
6. Rahmann, S., Wittkop, T., Baumbach, J., Martin, M., Truss, A., Böcker, S.: Exact and heuristic algorithms for weighted cluster editing. In: Proc. of Computational Systems Bioinformatics (CSB 2007), vol. 6, pp. 391–401 (2007)
7. Grötschel, M., Wakabayashi, Y.: A cutting plane algorithm for a clustering problem. Math. Program. 45, 52–96 (1989)
8. Kochenberger, G.A., Glover, F., Alidaee, B., Wang, H.: Clustering of microarray data via clique partitioning. J. Comb. Optim. 10(1), 77–92 (2005)
9. Delvaux, S., Horsten, L.: On best transitive approximations to simple graphs. Acta. Inform. 40(9), 637–655 (2004)
10. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. 144(1-2), 173–182 (2004)
11. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. 56(1), 89–113 (2004)
12. Křivánek, M., Morávek, J.: NP-hard problems in hierarchical-tree clustering. Acta Inform. 23(3), 311–323 (1986)
13. Komusiewicz, C., Uhlmann, J.: Cluster editing with locally bounded modifications. Discrete Appl. Math. 160(15), 2259–2270 (2012)
14. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63(4), 512–530 (2001)
15. Mannaa, B.: Cluster editing problem for points on the real line: A polynomial time algorithm. Inform. Process. Lett. 110, 961–965 (2010)
16. Böcker, S.: A golden ratio parameterized algorithm for cluster editing. J. Discrete Algorithms 16, 79–89 (2012)
17. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999)
18. Cao, Y., Chen, J.: Cluster editing: Kernelization based on edge cuts. Algorithmica 64(1), 152–169 (2012)
19. Fomin, F.V., Kratsch, S., Pilipczuk, M., Pilipczuk, M., Villanger, Y.: Tight bounds for parameterized complexity of cluster editing. In: Proc. of Symposium on Theoretical Aspects of Computer Science (STACS 2013). LIPIcs, vol. 20, pp. 32–43. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2013)
20. Charikar, M., Guruswami, V., Wirth, A.: Clustering with qualitative information. J. Comput. System Sci. 71(3), 360–383 (2005)
21. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: Ranking and clustering. J. ACM 55(5), Article 23 (2008)
22. van Zuylen, A., Williamson, D.P.: Deterministic pivoting algorithms for constrained ranking and clustering problems. Math. Oper. Res. 34(3), 594–620 (2009)

23. Vescia, G.: Descriptive classification of cetacea: whales, porpoises and dolphins. In: Marcotorchino, J., Proth, J.M., Janssen, J. (eds.) Data Analysis in Real Life Environment: Ins and Outs of Solving Problems, pp. 7–14. Elsevier Science, North-Holland, Amsterdam (1985)
24. Vescia, G.: Automatic classification of cetaceans by similarity aggregation. In: Marcotorchino, J., Proth, J.M., Janssen, J. (eds.) Data Analysis in Real Life Environment: Ins and Outs of Solving Problems, pp. 15–24. Elsevier Science, North-Holland, Amsterdam (1985)
25. Marcotorchino, J., Michaud, P.: Heuristic approach to the similarity aggregation problem. Methods of Operations Research 43, 395–404 (1981)
26. Marcotorchino, J., Michaud, P.: Optimization in exploratory data analysis. In: Proc. of Symposium on Operations Research, Köln, Germany. Physica Verlag (1981)
27. Schader, M., Tüshaus, U.: Ein Subgradientenverfahren zur Klassifikation qualitativer Daten. Operations Research Spektrum 7, 1–5 (1985)
28. Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: Evaluation and experiments. Algorithmica 60(2), 316–334 (2011)
29. Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: Evaluation and experiments. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 289–302. Springer, Heidelberg (2008)
30. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. Theory Comput. Syst. 38(4), 373–392 (2005)
31. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. Algorithmica 39(4), 321–347 (2004)
32. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truss, A.: Going weighted: Parameterized algorithms for cluster editing. Theor. Comput. Sci. 410(52), 5467–5480 (2009)
33. Böcker, S., Damaschke, P.: Even faster parameterized cluster deletion and cluster editing. Inform. Process. Lett. 111(14), 717–721 (2011)
34. Protti, F., da Silva, M.D., Szwarcfiter, J.L.: Applying modular decomposition to parameterized cluster editing problems. Theory Comput. Syst. 44(1), 91–104 (2009)
35. Fellows, M.R.: The lost continent of polynomial time: Preprocessing and kernelization. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 276–277. Springer, Heidelberg (2006)
36. Fellows, M.R., Langston, M.A., Rosamond, F.A., Shaw, P.: Efficient parameterized preprocessing for cluster editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 312–321. Springer, Heidelberg (2007)
37. Guo, J.: A more effective linear kernelization for cluster editing. Theor. Comput. Sci. 410(8-10), 718–726 (2009)
38. Chen, J., Meng, J.: A $2k$ kernel for the cluster editing problem. J. Comput. Syst. Sci. 78(1), 211–220 (2012)
39. Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D., Lander, E.S.: Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. Science 286(5439), 531–537 (1999)
40. Wittkop, T., Baumbach, J., Lobo, F.P., Rahmann, S.: Large scale clustering of protein sequences with FORCE — a layout based heuristic for weighted cluster editing. BMC Bioinformatics 8, 396 (2007)

41. Wittkop, T., Emig, D., Lange, S., Rahmann, S., Albrecht, M., Morris, J.H., Böcker, S., Stoye, J., Baumbach, J.: Partitioning biological data with transitivity clustering. Nat. Methods 7(6), 419–420 (2010)
42. Smoot, M.E., Ono, K., Ruscheinski, J., Wang, P.L., Ideker, T.: Cytoscape 2.8: new features for data integration and network visualization. Bioinformatics 27(3), 431–432 (2011)
43. Morris, J.H., Apeltsin, L., Newman, A.M., Baumbach, J., Wittkop, T., Su, G., Bader, G.D., Ferrin, T.E.: clusterMaker: a multi-algorithm clustering plugin for Cytoscape. BMC Bioinformatics 12, 436 (2011)
44. Cerdeira, L.T., Carneiro, A.R., Ramos, R.T.J., de Almeida, S.S., D'Afonseca, V., Schneider, M.P.C., Baumbach, J., Tauch, A., McCulloch, J.A., Azevedo, V.A.C., Silva, A.: Rapid hybrid de novo assembly of a microbial genome using only short reads: Corynebacterium pseudotuberculosis I19 as a case study. J. Microbiol. Methods 86(2), 218–223 (2011)
45. Baumbach, J., Tauch, A., Rahmann, S.: Towards the integrated analysis, visualization and reconstruction of microbial gene regulatory networks. Brief Bioinform. 10(1), 75–83 (2009)
46. Baumbach, J.: On the power and limits of evolutionary conservation–unraveling bacterial gene regulatory networks. Nucleic Acids Res. 38(22), 7877–7884 (2010)
47. Röttger, R., Kalaghatgi, P., Sun, P., Soares, S.C., Azevedo, V., Wittkop, T., Baumbach, J.: Density parameter estimation for finding clusters of homologous proteins–tracing actinobacterial pathogenicity lifestyles. Bioinformatics 29(2), 215–222 (2013)
48. Röttger, R., Kreutzer, C., Vu, T.D., Wittkop, T., Baumbach, J.: Online transitivity clustering of biological data with missing values. In: Proc. of German Conference on Bioinformatics (GCB 2012), pp. 57–68 (2012)
49. Sakai, S., Takaki, Y., Shimamura, S., Sekine, M., Tajima, T., Kosugi, H., Ichikawa, N., Tasumi, E., Hiraki, A.T., Shimizu, A., Kato, Y., Nishiko, R., Mori, K., Fujita, N., Imachi, H., Takai, K.: Genome sequence of a mesophilic hydrogenotrophic methanogen methanocella paludicola, the first cultivated representative of the order methanocellales. PLoS One 6(7), e22898 (2011)
50. Jochmann, N., Kurze, A.K., Czaja, L.F., Brinkrolf, K., Brune, I., Hüser, A.T., Hansmeier, N., Pühler, A., Borovok, I., Tauch, A.: Genetic makeup of the Corynebacterium glutamicum LexA regulon deduced from comparative transcriptomics and in vitro DNA band shift assays. Microbiology 155(pt. 5), 1459–1477 (2009)
51. Baumbach, J., Rahmann, S., Tauch, A.: Reliable transfer of transcriptional gene regulatory networks between taxonomically related organisms. BMC Syst. Biol. 3, 8 (2009)
52. Pauling, J., Röttger, R., Tauch, A., Azevedo, V., Baumbach, J.: CoryneRegNet 6.0—updated database content, new analysis methods and novel features focusing on community demands. Nucleic Acids Res. 40(Database issue), D610–D614 (2012)
53. Pauling, J., Röttger, R., Neuner, A., Salgado, H., Collado-Vides, J., Kalaghatgi, P., Azevedo, V., Tauch, A., Pühler, A., Baumbach, J.: On the trail of EHEC/EAEC—unraveling the gene regulatory networks of human pathogenic Escherichia coli bacteria. Integr. Biol. (Camb.) 4(7), 728–733 (2012)
54. Wittkop, T., Emig, D., Truss, A., Albrecht, M., Böcker, S., Baumbach, J.: Comprehensive cluster analysis with transitivity clustering. Nat. Protocols 6, 285–295 (2011)

55. Hauschild, A.C., Schneider, T., Pauling, J., Rupp, K., Jang, M., Baumbach, J., Baumbach, J.: Computational methods for metabolomics data analysis of ion mobility spectrometry data — reviewing the state of the art. Metabolites 2(4), 733–755 (2012)
56. Wittkop, T., Rahmann, S., Böcker, S., Baumbach, J.: Extension and robustness of transitivity clustering for protein-protein interaction network analysis. Internet Math. 7(4), 255–273 (2011)
57. Robertson, A.L., Bate, M.A., Buckle, A.M., Bottomley, S.P.: The rate of polyQ-mediated aggregation is dramatically affected by the number and location of surrounding domains. J. Mol. Biol. 413(4), 879–887 (2011)
58. Pacheco, L.G.C., Slade, S.E., Seyffert, N., Santos, A.R., Castro, T.L.P., Silva, W.M., Santos, A.V., Santos, S.G., Farias, L.M., Carvalho, M.A.R., Pimenta, A.M.C., Meyer, R., Silva, A., Scrivens, J.H., Oliveira, S.C., Miyoshi, A., Dowson, C.G., Azevedo, V.: A combined approach for comparative exoproteome analysis of Corynebacterium pseudotuberculosis. BMC Microbiol. 11(1), 12 (2011)
59. Wittkop, T., Berman, A.E., Fleisch, K.M., Mooney, S.D.: DEFOG: discrete enrichment of functionally organized genes. Integr. Biol (Camb) 4(7), 795–804 (2012)
60. Marx, D., Razgon, I.: Fixed-parameter tractability of multicut parameterized by the size of the cutset. In: Proc. of ACM Symposium on Theory of Computing (STOC 2011), pp. 469–478. ACM press, New York (2011)
61. Bousquet, N., Daligault, J., Thomassé, S.: Multicut is FPT. In: Proc. of ACM Symposium on Theory of Computing (STOC 2011), pp. 459–468. ACM press, New York (2011)
62. Damaschke, P.: Fixed-parameter enumerability of cluster editing and related problems. Theory Comput. Syst. 46(2), 261–283 (2010)
63. Komusiewicz, C., Uhlmann, J.: Alternative parameterizations for cluster editing. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 344–355. Springer, Heidelberg (2011)
64. Damaschke, P.: Bounded-degree techniques accelerate some parameterized graph algorithms. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 98–109. Springer, Heidelberg (2009)
65. Weller, M., Komusiewicz, C., Niedermeier, R., Uhlmann, J.: On making directed graphs transitive. J. Comput. Syst. Sci. 78(2), 559–574 (2012)
66. Böcker, S., Briesemeister, S., Klau, G.W.: On optimal comparability editing with applications to molecular diagnostics. BMC Bioinformatics 10(suppl. 1), S61 (2009); Proc. of Asia-Pacific Bioinformatics Conference (APBC 2009)

Beyond Rogers' Non-constructively Computable Function*

John Case and Michael Ralston

Computer and Information Sciences Department, University of Delaware,
Newark, DE 19716, U.S.A.
{case,ralston}@udel.edu

Abstract. On page 9 of Rogers' computability book he presents two functions each based on eventual, currently unknown patterns in the decimal expansion of π. One of them is easily (classically) seen to be computable, but the proof is highly non-constructive and, conceptually interestingly, there is no *known* example algorithm for it. For the other, it is unknown as to whether it is computable. In the future, though, these unknown patterns in the decimal expansion of π may be sufficiently resolved, so that, for the one, we shall know a particular algorithm for it, and/or, for the other whether it's computable. The present paper provides a "safer" real to replace π so that the associated one function retains its trivial computability but has unprovability of the correctness of any particular program for it. Re the other function, a real $\mathbf{r}$ to replace π is given with each bit of this $\mathbf{r}$ uniformly linear time computable in the *length* of its position and so that the Rogers' other function associated with $\mathbf{r}$ is provably uncomputable.

1 Introduction

Rogers [1, p. 9] defines functions f and, then, g—each based on eventual patterns in the decimal expansion of π.[1] We'll discuss each of these in turn (in Sections 1.1 and 1.2), but opposite the order in which Rogers discusses them.[2]

1.1 Our f and Variants

As will be seen, the second of these Rogers' functions, which we're calling f, is clearly computable *but not constructively so—and* there is currently apparently *no known* way to compute it.

For each $x \in \mathbb{N} = \{0, 1, 2, \ldots\}$, let $f(x) \stackrel{\text{def}}{=} 1$, if there are at least x consecutive 5's in π's decimal expansion; 0, otherwise.

Clearly, f is either constantly 1—in case (i) π's decimal expansion has arbitrary long, finite runs of consecutive 5's—*or* f steps exactly once from 1 to

* We are grateful for anonymous referees' helpful corrections and suggestions.

[1] Brouwer first similarly employed patterns in π's digits, e.g., [2].

[2] To preserve alphabetical order in *our* discussion, we'll call his g, f, and his f, g.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 45–54, 2013.

0—(ii) otherwise. In each possibility f is trivially computable: in case (i), employing Church's Lambda Notation [1], $f = \lambda x.(1)$; in case (ii), if n is the maximum length of any consecutive sequences of 5s in the decimal expansion of π, $f = \lambda x.(1 \text{ if } x \leq n; 0 \text{ otherwise})$. Hence, in any case, f is trivially computable.

It's *un*known *which* of cases (i) and (ii) just above holds, *and*, if (ii) became known, it might still be unknown the exact size of a largest consecutive run of 5's in π's decimal expansion. Then we still wouldn't know exactly where f steps from 1 to 0—and we still wouldn't know how to compute f.

This Rogers' example very nicely illustrates the (classical) conceptual difference between, *there is an algorithm for f* and *the human race knows an algorithm for f*: the former is true, but the latter is currently false. *However*, in the future, the above *un*knowns about the decimal expansion of π may be suitably resolved, so that, then, Rogers' example will no longer illustrate this just above pleasant conceptual difference.[3]

As will be seen, our Theorems 3 and 9 below provide *"safer"* replacements for f—so as to preserve an interesting version of the nice (classical) conceptual difference above.

Suppose **T** is any fixed, *computably axiomatized* extension of first order Peano Arithmetic (**PA**) [3, 1]. E.g., **T** might be: **PA** itself, the also first order variant with variables for both natural numbers and for sets thereof and with induction expressed for sets (e.g., from [1]), or **ZFC**. We *also need* that (certain of) **T**'s theorems which are *expressible in* **PA** are *true* (essentially those featured in the proofs of Theorems 3 and 9 below). The theory **T** being computably axiomatized makes its set of theorems computably enumerable, so **T** plays the role of an *algorithmic extractor of (relevant) true information.*

Except for π, from now on, we'll restrict our attention to reals $\mathbf{r}$ in the interval $[0, 1]$ represented as an infinite expansion in *binary* (not in base 10).

We say that such a real $\mathbf{r} = .r_0 r_1 r_2 \ldots$, where the r_js are its successive binary digits, is *computable* iff the function $\lambda j.(r_j)$ is computable.

For a real $\mathbf{r}$, for each $x \in \mathbb{N}$, we let

$$f_{\mathbf{r}}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if there are } \textit{at least } x \textit{ consecutive } 1\text{'s in } \mathbf{r}\text{'s } \textit{binary} \text{ expansion;} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Clearly, $f_{\mathbf{r}}$, just as is f, is trivially computable—even if $\mathbf{r}$ is not computable. Again, even for *computable* $\mathbf{r}$, it may, in some cases, be hard to know that some particular program q *is* a program for $f_{\mathbf{r}}$.

We can and do consider the *acceptable programming systems* (synonym: *acceptable numberings*) [4, 1, 5–8] as those programming systems for the 1-argument partial computable functions: $\mathbb{N} \to \mathbb{N}$ which are intercompilable with naturally occurring general purpose programming formalisms such as a full Turing machine formalism or a LISP programming language. Typically, for

[3] A future proof as to *which* function is f may still be non-constructive, but, importantly for the present paper, re the nice conceptual difference above, we would have the *less* interesting (and more normal) situation that both *there is an algorithm for f* and *the human race knows an algorithm for f* are true—of course, classically.

theoretical work, one works with *numerical* names for the programs in these systems—whence the term 'numbering.' Let φ be any fixed acceptable programming system. By φ_p we denote the 1-argument partial computable function from $\mathbb{N}$ to $\mathbb{N}$ computed by program (number) p in the φ-system.

If E is an expression such as 'φ_p is total', where p is a particular element of $\mathbb{N}$, we shall write $\ll \mathsf{E} \gg$ to denote a *naturally* corresponding, fixed standard cwff (closed well-formed formula) of **PA** which (semantically) expresses E. We assume that

> if E' is obtained from E by changing some numerical values, then $\ll \mathsf{E}' \gg$ can be *algorithmically* obtained from those changed numerical values and $\ll \mathsf{E} \gg$.

It is well known that cwffs extensionally equivalent may not be intensionally or even provably equivalent [9]. In what follows, when we use the $\ll \mathsf{E} \gg$ notation, it will always be for E that are easily and *naturally* (semantically) expressible in **PA** as $\ll \mathsf{E} \gg$.

'$\vdash$' denotes the provability relation, and '$\nvdash$' is its negation.

Definition 1. In the following, we say that s ($\in \mathbb{N}$) *computes* (say, a $\{0,1\}$-valued total function) h iff $\varphi_s = h$.

Our Theorems 3 and 9 below each imply that there are *computable* reals $\mathbf{r}$ such that, for *any* p, q which compute $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively,

$$\mathbf{T} \nvdash \ll q \text{ computes } f_{\varphi_p} \gg . \tag{2}$$

N.B. The word *any* just above is important, since, then, (2) is not the result of only pathological choices of p, q for computing $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively; (2) holds for *any* such choices of p, q. Hence, for *such* computable $\mathbf{r}$, *there is an algorithm for* $f_{\mathbf{r}}$ is trivially true, but $\mathbf{T}$ *knows an algorithm for* $f_{\mathbf{r}}$ is false.[4]

Below $\downarrow$ means converges or is defined, and $\uparrow$ means diverges or is undefined.

In our proofs of Theorems 3 and 9, in view of Definition 1 above and the nature of computable reals $\mathbf{r}$ and corresponding $f_{\mathbf{r}}$, we expand (2) above to: $\mathbf{T} \nvdash \ll (\forall x)[\varphi_q(x) \downarrow \leq 1 \ \wedge \ [\varphi_q(x) = 1 \rightarrow (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z) \downarrow = 1]] \ \wedge \ [\varphi_q(x) = 0 \rightarrow (\forall y)(\exists z \mid y \leq z < y + x)[\varphi_p(z) \downarrow = 0]]] \gg$.

More importantly, in the light of (2) above, the following is perhaps surprising. For Theorem 3, $f_{\mathbf{r}}$ is made $= \lambda x.(1)$. For Theorem 9, for each $n > 0$, a corresponding $f_{\mathbf{r}}$ is made $= \lambda x.(1$ if $x \leq n; 0$ otherwise). The solution to the *apparent* paradoxes, one for each theorem, is that $\mathbf{T}$ does not and cannot know that theorem's information about *which* function $f_{\mathbf{r}}$ is!

1.2 Our g and Variants

The first of Rogers' functions [1, Page 9], which we're calling g, is defined thus. For each $x \in \mathbb{N}$, let $g(x) \stackrel{\text{def}}{=} 1$, if there are *exactly* x consecutive 5's in π's decimal expansion; 0, otherwise.

[4] Here $\mathbf{T}$ knows something means $\mathbf{T}$ proves it.

Rogers [1, Page 9] points out that it is *un*known whether g just above is computable. Of course in the future enough may become known about the distributions of runs of consecutive 5s in the decimal expansion of π for whether g is computable to be resolved. Again we'll consider only reals $\mathbf{r}$ (except for π), as in Section 1.1 above, which are restricted to the interval $[0, 1]$ and represented as an infinite expansion in *binary* (not in base 10).

For a such a real $\mathbf{r}$, for each $x \in \mathbb{N}$, we let

$$g_{\mathbf{r}}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if there are } \textit{exactly } x \textit{ consecutive } 1\text{'s in } \mathbf{r}\text{'s } \textit{binary} \text{ expansion;} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

For clarity as to what is meant by 'exactly' just above, we define a *consecutive run of exactly* x $1s$ to be one that is bounded on both sides by a 0.

Trivially, there are many examples, such as $\mathbf{r} = .00000\ldots$, so that $g_{\mathbf{r}}$ *is* computable. The first author has given the next theorem (Theorem 2) as a course exercise with hints.

Theorem 2. *There is a primitive recursive* $\mathbf{r}$ *such that* $g_{\mathbf{r}}$ *is not computable.*

We omit the hints for lack of space, and, anyhow, instead describe next a considerable improvement.

In [10] there is a nice conjecture about reals $\mathbf{r} = \lambda j.(r_j)$ computable in linear time in j. Our last theorem below (Theorem 10) is about reals $\mathbf{r} = \lambda j.(r_j)$ computable in linear time in $|j|$, the *length of* j (and *not* requiring prior computing of r_i, for any $i < j$). This theorem says that, for each computably enumerable set A, there is real $\mathbf{r} = \lambda j.(r_j)$ computable in linear time in $|j|$ such that $g_{\mathbf{r}}$ is the characteristic function of $(A \cup \{0\})$. If we choose $A = K$, where K is the diagonal halting problem set from [1], then $g_{\mathbf{r}}$ is not computable.

2 Preliminaries

The material in this section (Section 2) is largely important for our machine-*dependent*, *natural*-complexity theorem (Theorem 10) and its proof.

φ^{TM} is the specific fixed acceptable programming system from [11, Chapter 3] for the partial computable functions: $\mathbb{N} \to \mathbb{N}$; it is a system based on deterministic, *multi-tape* Turing machines (TMs). In this system the φ^{TM}-programs are *efficiently* given numerical names or codes. This efficient numerical coding guarantees that many simple operations run in linear time in the *length* of input.

Φ^{TM} denotes the natural TM step counting complexity measure (also in [11, Chapter 3]) and associated with φ^{TM}. In the present paper, we employ a number of complexity bound results from [11, Chapters 3 & 4] regarding $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$. These results will be clearly referenced as we use them.

Herein *lintime computable* means computable in linear time in the *length* of inputs—of course as measured by Φ^{TM} in the φ^{TM} system.

We let $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the fixed, 1-1, onto, *lintime computable* pairing function from [11, Section 2.3]. Its respective left and right inverses, π_1 and

π_2, are also *lintime computable* [11, Section 2.3]. The function $\langle \cdot, \cdot \rangle$ enables us to restrict our attention to one-argument partial computable functions and still handle, with iterated coding by $\langle \cdot, \cdot \rangle$, multiple argument cases. The lintime computable functions π_1 and π_2 are employed below in the proof of our last theorem (Theorem 10).

We employ the convenient discrete log function from [11, Page 22]: for each $x \in \mathbb{N}$, $\log(x) \stackrel{\text{def}}{=} (\lfloor \log_2(x) \rfloor, \text{ if } x > 1; 1, \text{ if } x \leq 1)$.

We let $\Phi^{\text{SlowedDownTM}}$ be the special, *slowed down* step-counting measure associated with the acceptable φ^{TM}-system from [11, Theorem 3.20]. In the proof of [11, Theorem 3.20], for the case of $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$, $\Phi^{\text{SlowedDownTM}}$ is obtained from the standard Φ^{TM} measure associated with φ^{TM}—in part by a standard log-factors slow down trick. The measure $\Phi^{\text{SlowedDownTM}}$ has the nice property (among others) that the predicate

$$T \stackrel{\text{def}}{=} \lambda p, x, t.(1, \text{ if } \Phi_p^{\text{SlowedDownTM}}(x) \leq t; 0, \text{ otherwise}) \tag{4}$$

is *lintime computable*!

In the proof of Claim 11 below (part of the proof of our last theorem, Theorem 10), we'll make use of the following lemma from [11] concerning a very efficiently implemented *conditional* (i.e., *if-then-else*) control structure for $(\varphi^{\text{TM}}, \Phi^{\text{TM}})$. In particular we'll employ the remark immediately following this lemma (Remark 1).

Lemma 1 (Lemma 3.14, [11]). *There is a $k > 0$ and a lintime computable function* cond *such that, for all $a, b, c, x \in \mathbb{N}$,*

$$\varphi^{\text{TM}}_{\text{cond}(\langle a,b,c \rangle)}(x) = \begin{cases} \varphi^{\text{TM}}_b(x), & \text{if } \varphi^{\text{TM}}_a(x) \downarrow > 0; \\ \varphi^{\text{TM}}_c(x), & \text{if } \varphi^{\text{TM}}_a(x) \downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases} \tag{5}$$

and

$$\Phi^{\text{TM}}_{\text{cond}(\langle a,b,c \rangle)}(x) \leq k \cdot \begin{cases} (\Phi^{\text{TM}}_a + \Phi^{\text{TM}}_b(x)) + 1, & \text{if } \varphi^{\text{TM}}_a(x) \downarrow > 0; \\ (\Phi^{\text{TM}}_a + \Phi^{\text{TM}}_c(x)) + 1, & \text{if } \varphi^{\text{TM}}_a(x) \downarrow = 0; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{6}$$

Remark 1. Note that from (6) just above, that, if φ^{TM}-programs a, b, c each run in time linear in input length, then so does φ^{TM}-program $\text{cond}(\langle a, b, c \rangle)$.

3 Results

The notations, basic assumptions, and some employed technical observations are from the Introduction and Preliminaries (Sections 1 and 2) above. Recall from there that, for *any* choice of real $\mathbf{r}$, $f_{\mathbf{r}}$ is *always computable*.

Theorem 3. *There is a computable real $\mathbf{r}$ such that, for any p, q which compute $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively, $\mathbf{T} \not\vdash \ll q$ computes $f_{\varphi_p} \gg$. Furthermore, $f_{\mathbf{r}} = \lambda x.(1)$.*

Proof. As above in Section 1, our expansion of $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$ is: $\mathbf{T} \vdash \ll (\forall x)[\varphi_q(x) \downarrow \leq 1 \ \wedge \ [\varphi_q(x) = 1 \rightarrow (\exists y)(\forall z \mid y \leq z < y + x)[\varphi_p(z) \downarrow = 1]] \ \wedge \ [\varphi_q(x) = 0 \rightarrow (\forall y)(\exists z \mid y \leq z < y + x)[\varphi_p(z) \downarrow = 0]]] \gg$.

Define program list as follows: it runs a theorem prover for $\mathbf{T}$ and algorithmically lists, in some order, all the (p, q) such that $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$.

The global variable seq has value at any point a finite sequence of bits with known canonical index, and its intial value is empty. We reconceptualize this initial value equivalently as an infinite sequence of known-to-be-undefined values.

Define no-input subroutine longest_run (with global variable seq) as follows, and let longest_run with no arguments stand for the value returned by a call to this subroutine: longest_run scans seq to find the longest subsequence of contiguous 1s in seq and returns the length of that subsequence.

Define subroutine outputs_ones (with global variable seq) as follows. On input x, outputs_ones scans seq for the first x contiguous known-undefined values in seq and defines them all to be 1.

Define subroutine setup_seq (with global variable seq) as follows.

> On input (p, q), setup_seq runs q on all values between 0 and longest_run inclusive. If φ_q returns 0 on any of these values, setup_seq returns and does nothing.
>
> Otherwise, let x be longest_run + 1, and setup_seq runs q on x. If it returns a 0, then setup_seq calls outputs_ones on x and returns.
>
> If $\varphi_q(x)$ returns a non-zero value, then setup_seq dovetails an enumeration of the values of φ_p, and compares them against the *corresponding* entries of seq. If any entries are defined as something different from φ_p's values, it returns and does nothing. If it finds a value v such that $\varphi_p(v) \downarrow$ and seq(v) is still known-undefined, it defines seq(v) to be $1 \dot{-} \varphi_p(v)$ and returns.
>
> If none of the previously-listed stopping conditions are met, or if setup_seq runs a program that goes undefined, it runs forever.

Define program define_real (with global variable seq) as follows.

> On input x, define_real does the following:
>
> Set i to 0.
>
> For $x + 1$ iterations, do:
>
> Run another step of list, and if, by the end of that step, it outputs some (p, q), pass (p, q) to setup_seq.[5]
>
> If seq(i) is known-undefined after that, define it equal to 0.
>
> Increment i by 1.
>
> Output the value of seq(x).

Claim 4. *If* setup_seq *is passed a* (p, q) *such that* $\mathbf{T} \vdash \ll q$ *computes* $f_{\varphi_p} \gg$, *then it will terminate.*

[5] list can output either nothing or exactly one ordered pair on a given step.

Proof. Since $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$, and $\mathbf{T}$ does not prove false things (of this sort), it follows that $(\forall x)[\varphi_q(x) \downarrow \leq 1]$. Thus, running q on a finite set of values must terminate. As only finitely many elements of seq can be defined during a call to setup_seq, all calls to outputs_ones must terminate. The only remaining case is that $(\forall x \mid x \leq \text{longest\_run} + 1)[\varphi_q(x) = 1]$, from which it follows that $(\exists y)(\forall z \mid y \leq z < y + \text{longest\_run} + 1)[\varphi_p(z) \downarrow = 1]$, which provides a sequence of consecutive 1s that is longer than the longest sequence of consecutive 1s in seq, which means there is at least one value of φ_p which is defined equal to 1 whose corresponding value of seq is not defined to be equal to 1, and thus setup_seq will terminate in this case as well.

Claim 5. *For any p, q, if $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$, then passing (p, q) to* setup_seq *results in a* seq *such that the set of reals compatible with the defined values of* seq *does not include φ_p.*

Proof. Case one: for some x less than or equal to the value of longest_run, $\varphi_q(x) = 0$, in which case seq is already incompatible with φ_p as seq has a run of 1s which are longer than the longest such run in φ_p. Case two: $\varphi_q(\text{longest\_run}+1) = 0$, in which case setup_seq calls outputs_ones and makes seq incompatible with φ_p for the same reason as case one. Case three: $(\forall x \mid x \leq \text{longest\_run}+1)[\varphi_q(x) = 1]$. In this case, setup_seq will search for a value of φ_p whose corresponding entry in seq either doesn't match or is known-undefined and will be set to not match. By Claim 4, such a value will be found, after which seq will not be compatible with φ_p.

Claim 6. *The program* define_real *does, in fact, define a real.*

Proof. outputs_ones clearly terminates, by Claim 4 and the definition of list, all setup_seq calls will terminate, and, on any input x to define_real, list is only run for a finite number of steps. Thus, define_real will always output a value from seq. The loop in define_real includes a step that ensures that the element of seq output by define_real will be defined, and all times when elements of seq are assigned values, they are assigned a value of either 0 or 1. So, define_real computes a total $\{0, 1\}$-valued function, a real.

Claim 7. *For any p, q, if $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$, then $\varphi_p \neq \varphi_{\text{define\_real}}$.*

Proof. By Claim 6, $\varphi_{\text{define\_real}}$ defines a real, and thus, if φ_p does not define a real, they are not equal. By Claim 5 and the fact that define_real's output is always compatible with seq, if φ_p does define a real, the real it defines is not equal to $\varphi_{\text{define\_real}}$. Therefore, $\varphi_p \neq \varphi_{\text{define\_real}}$.

Claim 8. $f_{\varphi_{\text{define\_real}}} = \lambda x.(1)$.

Proof. For all positive n, by provable in **PA** (hence, in **T**) padding for the φ-system, there's an infinite set of ps & qs such that q computes $\lambda x.(1$ if $x \leq n$; 0 otherwise) *and* $\mathbf{T} \vdash \ll q$ computes $f_{\varphi_p} \gg$, so outputs_ones is called infinitely often. For such calls, it provides a sequence of consecutive 1s longer than prior such.

By Claims 6, 7, and 8, the statement of the theorem follows with $\mathbf{r} = \varphi_{\text{define\_real}}$. □ (**Theorem 3**)

In general, $\lambda x.(1)$ is only one possibility for $f_{\mathbf{r}}$, so the question naturally arises: does there exist computable real $\mathbf{r}$ such that $(\forall p, q)[\varphi_p = \mathbf{r} \rightarrow \mathbf{T} \not\vdash \ll q$ computes $f_{\varphi_p} \gg$, where $f_{\mathbf{r}} \neq \lambda x.(1)$?

For *all* $f_{\mathbf{r}}$ of the form $\lambda x.(1$ if $x \leq n; 0$ otherwise), except for the case where $n = 0$, the answer is, *Yes*. For $f_{\mathbf{r}} = \lambda x.(1$ if $x = 0; 0$ otherwise), there is precisely one such real, $\lambda x.(0)$, and there exists p, q such that $\varphi_p = \lambda x.(0) \ \wedge \ \mathbf{PA} \vdash \ll q$ computes $f_{\varphi_p} \gg$.

Theorem 9. *For each $n > 0$, there exists a corresponding computable real $\mathbf{r}$ such that, for any p, q which compute $\mathbf{r}$ and $f_{\mathbf{r}}$, respectively, $\mathbf{T} \not\vdash \ll q$ computes $f_{\varphi_p} \gg$. Furthermore, $f_{\mathbf{r}} = \lambda x.(1$ if $x \leq n; 0$ otherwise).*

We omit Theorem 9's proof for lack of space, but it partly proceeds like the proof above of Theorem 3.

Next we present our strong, promised theorem regarding $g_{\mathbf{r}}$s.

Theorem 10. *For* any *computably enumerable (c.e.) set A containing* 0*, there exists a lintime computable real $\mathbf{r}$ such that $g_{\mathbf{r}}$ is the characteristic function of A.*[6]

Proof. Since set A is c.e., there exists a φ^{TM}-program a which half-decides A, terminating after possibly arbitrarily much computation for inputs in A, and going undefined for inputs not in A.

We informally define φ^{TM}-program r, which computes the corresponding, desired real $\mathbf{r}$, as follows. It is to be understood, that this r makes *implicit* use of the efficient conditional control structure from Lemma 1 and Remark 1 above. In (7) just below the T predicate is the one defined in (4) above. For each $x \in \mathbb{N}$, let

$$\varphi_r^{\text{TM}}(x) = \begin{cases} 0, & \text{if } (x < 4 \vee \\ & T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x))) = 0 \vee \\ & x = 2^{\log(x)} \vee \\ & x > \pi_1(\log\log(x)) + 2^{\log(x)}); \\ 1, & \text{otherwise.} \end{cases} \tag{7}$$

Claim 11. *φ^{TM}-program r runs in time linear in the length of its input.*

Proof. Clearly, comparing x to a constant can be done in constant time. Computing $\log(x)$ can be done in time linear in the length of x [11, Lemma 3.2(k)], as can computing $2^{\log(x)}$.[7] As noted in Section 2 above, calculation of each of

[6] For technical reasons, it is difficult, in the proof just below, to produce a real that does not contain two consecutive 0s. A real $\mathbf{r}$ *with* two consecutive 0s always satisfies $g_{\mathbf{r}}(0) = 1$.

[7] An algorithm for this latter, with I/O in binary number representation, is to scan x (as usual left to right), and, on the output tape *from right to left*, writing a 0 onto an output tape for each symbol in x, then replacing the last-written 0 with a 1 when the end of x is reached. The φ^{TM}-system is based on dyadic I/O [11], but, by [11, Lemma 3.2(b)] passing between binary and dyadic is lintime computable.

π_1 and π_2 take time linear in the length of their input. Then, from (4) and line following (4) (above), $T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x)))$ can be computed in time linear in the length of $\log\log(x)$. Adding two numbers can be done in time linear in the length of the longer, and comparing the result to another number can once again be done in time linear in the length of the longer number [11, Lemma 3.2(f, g)]. By [11, Lemma 3.18], lintime computable predicates are closed under Boolean operations. Thus, from r's implicit employment of cond from Lemma 1 above, and by Remark 1 above, r itself executes in time linear in the length of its input x.

Claim 12. *φ^{TM}-program r defines a real.*

Proof. From Claim 11 it follows that r is total. Since r returns only values in the range $\{0, 1\}$, it follows that it defines a real.

Claim 13. *For all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, it is the case that for all y such that $2^{\log(x)} < y \leq x$, $\varphi_r^{\mathrm{TM}}(y) = 1$.*

Proof. Let x be any value such that $\varphi_r^{\mathrm{TM}}(x) = 1$. Clearly, x is not a power of 2, $T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x))) = 1$, and $x \leq \pi_1(\log\log(x)) + 2^{\log(x)}$. If $x - 1$ is not a power of 2, then it follows that $\log(x) = \log(x - 1)$. From these facts, it follows that $T(a, \pi_1(\log\log(x - 1)), \pi_2(\log\log(x - 1))) = 1$ and $x - 1 \leq \pi_1(\log\log(x - 1)) + 2^{\log(x-1)}$, and thus it follows that $\varphi_r^{\mathrm{TM}}(x - 1) = 1$, if and only if $x - 1$ is not a power of 2. By downwards induction on x, the claim follows.

Claim 14. *For all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, it is the case that for all y such that $x \leq y \leq \pi_1(\log\log(x)) + 2^{\log(x)}$, $\varphi_r^{\mathrm{TM}}(y) = 1$, and also that $\varphi_r^{\mathrm{TM}}(1 + \pi_1(\log\log(x)) + 2^{\log(x)}) = 0$.*

Proof. Let x be any value such that $\varphi_r^{\mathrm{TM}}(x) = 1$. Clearly, x is not a power of 2, $T(a, \pi_1(\log\log(x)), \pi_2(\log\log(x))) = 1$, and $x \leq \pi_1(\log\log(x)) + 2^{\log(x)}$. If $x + 1$ is a power of 2, then $\varphi_r^{\mathrm{TM}}(x + 1) = 0$. If $x + 1$ is not a power of 2, then it follows that $\log(x) = \log(x + 1)$, from which it follows that $T(a, \pi_1(\log\log(x + 1)), \pi_2(\log\log(x + 1))) = 1$, and thus $\varphi_r^{\mathrm{TM}}(x + 1) = 1$ if and only if $x + 1$ is not a power of 2 and $x + 1 \leq \pi_1(\log\log(x + 1)) + 2^{\log(x+1)}$. By induction on x and the fact that $2^{\log(x)} + \pi_1(\log\log(x)) < 2^{\log(x)+1}$, the claim follows.

Claim 15. *For all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, there is a run of exactly $\pi_1(\log\log(x))$ 1s in the real $\mathbf{r}$, and the position x is in one such run.*

Proof. From Claims 13 and 14, as well as the fact that, for all x, $\varphi_r^{\mathrm{TM}}(2^{\log(x)}) = 0$, it follows that, for all x such that $\varphi_r^{\mathrm{TM}}(x) = 1$, there is a run of 1s from $2^{\log(x)} + 1$ to $\pi_1(\log\log(x)) + 2^{\log(x)}$, bounded on both sides by 0s. The claim follows immediately by use of basic algebra and arithmetic.

Claim 16. *For all $y > 0$ and for all t, if $\varphi_a^{\mathrm{TM}}(y)$ terminates within t steps, then $\varphi_r^{\mathrm{TM}}(2^{2^{<y,t>}} + 1) = 1$.*

Proof. The claim follows directly from the definitions of a and r.

Claim 17. *$g_{\mathbf{r}}$ is the characteristic function of A.*

Proof. By Claim 12 and the definition of $g_{\mathbf{r}}$, it follows $g_{\mathbf{r}}$ is a well-defined total $\{0,1\}$-valued function. From Claims 15 and 16, as well as the fact that, for all y, for all but finitely many t, for some xs, $< y, t >= \log\log(x)$, it follows that, for all $y > 0$ such that $\varphi_a^{\mathrm{TM}}(y) \downarrow$, there is a run of exactly y 1s in the binary representation of r, and thus $g_{\mathbf{r}}(y) = 1$. Assume by way of contradiction that there is some $y > 0$ such that $\varphi_a^{\mathrm{TM}}(y) \uparrow$ but $g_{\mathbf{r}}(y) = 1$. Then, there must be a run of exactly y 1s in r. Let x be the least number such that position x is in such a run. Then, by Claim 15, $\pi_1(\log\log(x)) = y$. From the definition of r, it follows that $T(a, y, \pi_2(\log\log(x))) = 1$, a contradiction to the assumption that $\varphi_a^{\mathrm{TM}}(y) \uparrow$. The previous shows that the claim holds for all inputs > 0. It is clear that $\varphi_r^{\mathrm{TM}}(0) = \varphi_r^{\mathrm{TM}}(1) = 0$, therefore $g_{\mathbf{r}}(0) = 1$, and thus, by the assumption that 0 is in A, the claim is proven.

The theorem follows from Claims 11, 12, and 17. □ (**Theorem 10**)

References

1. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York (1967); Reprinted. MIT Press (1987)
2. Brouwer, L., In van Dalen, D. (eds.): Brouwer's Cambridge Lectures on Intuitionism. Cambridge University Press (1981)
3. Mendelson, E.: Introduction to Mathematical Logic, 5th edn. Chapman & Hall, London (2009)
4. Rogers, H.: Gödel numberings of partial recursive functions. Journal of Symbolic Logic 23, 331–341 (1958)
5. Machtey, M., Young, P.: An Introduction to the General Theory of Algorithms. North Holland, New York (1978)
6. Riccardi, G.: The Independence of Control Structures in Abstract Programming Systems. PhD thesis, SUNY Buffalo (1980)
7. Riccardi, G.: The independence of control structures in abstract programming systems. Journal of Computer and System Sciences 22, 107–143 (1981)
8. Royer, J.S.: A Connotational Theory of Program Structure. LNCS, vol. 273. Springer, Heidelberg (1987)
9. Feferman, S.: Arithmetization of metamathematics in a general setting. Fundamenta Mathematicae 49, 35–92 (1960)
10. Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
11. Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. Research monograph in Progress in Theoretical Computer Science. Birkhäuser, Boston (1994), `www.cis.udel.edu/~case/RC94Errata.pdf` (errata)

Constructing Continuous Systems from Discrete Cellular Automata*

Julien Cervelle

Laboratoire d'Algorithmique, Complexité et Logique (LACL), University Paris-Est, 94010 Créteil cedex, France
julien.cervelle@univ-paris-est.fr

Abstract. This paper studies a way of transforming discrete time and discrete space cellular automata into systems described by partial differential equations with a similar behavior. The goal is to find new kinds of chaotic behaviors for systems ruled by partial differential equations.

1 Introduction

Deterministic cellular automata (CA for short) are a simple dynamical system defined on the *Cantor space* $A^{\mathbb{Z}}$ for a finite set of states A. The model consists in an infinite number of finite state automata placed on cells labeled by $\mathbb{Z}$. The time is discrete and at each step, each cell's state is updated according to the state of the cell and the states of its left and right neighboring cells. One of the main interest of CA is that, despite their simplicity, they can have numerous different behaviors, from very simple ones like the *shift* automaton where each cell simply copies the value of its right neighbor, to more complex ones like, for instance, the one which performs multiplication by 3 on numbers represented in base 6 (cf. [1,2]).

However classical time-continuous and space-continuous models described by partial differential equations (PDE for short) are quite often diffusion systems which means that, asymptotically, for energy invariant systems all the positions in the space ends at the same energy level. Other systems which are driven by wave propagation are also quite simple and similar to a Cartesian product of shifts of various speeds.

Our goal is to find simple systems which are natural in some sense but with complex and unpredictable behavior. For instance, we would like to have a positively expansive system which means that any modification to the initial condition is observable at any place at some time (cf. [2] for the formal definition). This notion, defined for CA using the Cantor topology, can simply be extended to continuous systems. For instance, in the simpler case where $A = \{0, 1\}$, let us represent the Cantor topology by the distance d, defined, for $x, y \in \{0, 1\}^{\mathbb{Z}}$, by $d(x, y) = \sum_{i \in \mathbb{Z}} (1 - \delta_{x_i, y_i}) 2^{-|i|}$ where δ is the usual Kronecker delta. The

* This work is supported by projects EQINOCS (ANR 11 BS02 004 03) and TARMAC (ANR 12 BS02 007 01).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 55–64, 2013.

distance d is naturally extended to continuous system defining, for $x, y \in [0,1]^{\mathbb{R}}$, $d(x,y) = \int_{-\infty}^{\infty} |x_i - y_i| 2^{-|i|} di$.

Wave propagation based models are not a good answer since if the change in the initial condition only modifies, creates or deletes a wave which goes to the left, nothing will be seen to the right.

To this purpose, we try to extend CA to a space-continuous, time-continuous and state-continuous system. In [3,4], a state-continuous version of interacting particle systems is studied. In this model, each cell has a probability distribution over A which evolution is governed by an ordinary differential equation (ODE for short) system which is derived from a local rule close to the local rules of *probabilistic CA* (PCA for short). A radius 1 PCA rule associates a probability distribution over A to the states of a cell and its neighbors: The value of $P[s_i^{t+1} = q | s_{i-1}^t = l \wedge s_i^t = c \wedge s_{i+1}^t = r]$ for all possible states q, l, c and r. For instance, a slower stochastic version of the shift could be that with probability p, the cell copies the state from the right neighbor and with probability $1-p$ keeps its state (cf. [5] for more about PCA). *Interacting particle systems* (IPS for short) are a time-continuous version of PCA whose behavior is analogous. However, the local rule is different since, instead of a probability of reaching a state given the states in the neighborhood, the rule gives a hint about the average time after which the change is made. The model described in [3,4] chooses a simplified deterministic view: cell states are probability distributions over A. Though easier to be dealt with, this view implies that the states of the cells are supposed independent. This is not the case when dealing with PCA or IPS. We describe the impact of this feature in Subsection 2.2 together with the formal definitions.

However, in order to converge to a space-continuous system, the authors of [3,4] have to add a supplementary behavior called *fast stirring* which mixes the states of near cells linearly (quadratically in 2D) faster than the regular application of the CA rule. This ensures that the states of near cells are identically distributed at the limit and allows to compute a limit system ruled by a PDE. This supplementary behavior make the resulting system a diffusion system which does not suit our needs: The fast-stirring effect makes that only CA rules whose transition function depends only on the multiset of states in the neighborhood are to be considered. Though Turing complete (Conway's game of life for instance), these rules perturbed by fast stirring are unlikely to produce complex dynamical system: The gadgets used to simulated a Turing machine are difficult to be made stable by the action of fast-stirring. In this paper, we give a sufficient and necessary condition to have a space-continuous limit system without the fast stirring behavior.

The paper is organized as follows. Section 2 gives the definition of the models. Section 3 and 4 are devoted to finding the limit systems of this model and Section 5 gives conclusions and further work.

2 Definitions and Systems

2.1 Deterministic Cellular Automata

Formally, a one-dimensional CA is a tuple $\langle A, r, \mu \rangle$. The finite set A is the set of *states*. Cells of the CA are indexed by $\mathbb{Z}$ and each has a state taken from A. The *radius* of the CA, r, is the number of cells to the left and to the right that are looked up for a cell update. This update is controlled by the *local rule* $\mu : A^{2r+1} \to A$. At each time step, each cell changes its state synchronously to $\mu(s_{\leftarrow}, s, s_{\rightarrow})$ where s is the state of the cell, $s_{\leftarrow}$ are the states of its r left neighboring cells and $s_{\rightarrow}$ are the states of its r right neighboring cells.

This local rule induces a global behavior. The state of the whole CA, that is, the states of all the cells is completely given by a *configuration*. A configuration is a member c of $A^{\mathbb{Z}}$ such that $c(j)$ is the state of the jth cell. For better readability, the state $c(j)$ is noted c_j, as it is for other types of configurations in the rest of the paper. The *global rule* of the CA is the function $f : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ which, given the current configuration, returns the next configuration. The function f is defined by $f(c)_j = \mu(c_{j-r}, \ldots, c_{j+r})$.

2.2 Interacting Stochastic Particle Systems

A one-dimensional *interacting stochastic particle system* (ISPS for short) is a tuple $\langle A, r, \lambda \rangle$ where A and r play the same role as for CA. The local rule is a function $\lambda : A \times A^{2r+1} \to [0, \infty)$ where the value of $\lambda(s, n)$ is the *rate* at which a cell whose neighborhood (including itself) is n changes its state to s. The higher the rate, the faster the cell is performing the change: The speed of the change is proportional to the rate.

For IPS (not ISPS), a configuration is a probability distribution over the configurations of a CA. In the definition of ISPS, we made the assumption that the distribution makes the probabilities of the values of each cells to be independent one another. This makes the configurations members of $(P_A)^{\mathbb{Z}}$ where P_A is the set of probability distributions over the finite set A. This clearly is not true for PCA and IPS.

Here are the reasons why we think this model is worth studying. First, the model of ISPS is sound and well defined by itself as if the state of a cell was not a member of A but a member of P_A which we call *stochastic particles* (e.g., the proportions of predators and preys). Second, this model was successfully used for modeling biological and physical behaviors (cf. [6] for many examples). Third, the dependency radius increases of r each time a rule is applied in a PCA. When looking at the limit system, we make the distance between two cells tends to 0. However, the time is already continuous so we do not make the time step tends to 0, only the space step. Hence the dependency radius is 0 in the limit system. Finally, the goal of this study is to find good candidate PDE for complex behavior trying to mimic CA behavior. We hope that the independency hypothesis will not prevent us from finding interesting systems.

A configuration is a member of $(P_A)^{\mathbb{Z}}$. If p is a member of P_A we denote by $P[p = s]$ the probability that the cell is in state s. If c is a configuration of $(P_A)^{\mathbb{Z}}$,

we note $P[c_j = s]$ the probability that the cell j of configuration c is in state s. The evolution of the ISPS is described by the ODE (infinitely many ones): For $j \in \mathbb{Z}$, $\frac{dP[c_j=s]}{dt} = i - o$ where i is the rate for entering state s from any other state and o is the rate for leaving state s. More precisely, one has

$$i = \sum_{n \in A^{2r+1}, n_0 \neq s} \lambda(s, n) \prod_{v=-r}^{r} P[c_{j+v} = n_v] \tag{1}$$

and

$$o = \sum_{g \in A \setminus \{s\}} \sum_{n \in A^{2r+1}, n_0 = s} \lambda(g, n) \prod_{v=-r}^{r} P[c_{j+v} = n_v], \tag{2}$$

where n is indexed as $n = (n_{-r}, \ldots, n_r)$. The system is deterministic on configurations of $(P_A)^{\mathbb{Z}}$. Remark that the values of $\lambda(s, n)$ for $s = n_0$ are not taken into account.

Note that the system may diverge at some time, when one of the probabilities leaves $[0, 1]$. However, the Cauchy-Lipschitz theorem on the Banach space $((\mathbb{R}^{|A|})^{\mathbb{Z}}, \|\cdot\|_\infty)$ tells us that, given an initial condition c, there is some greatest t such that the system is defined from time 0 to t. We call t the *diverge time* of c for the ISPS. When we consider the evolution of an ISPS $\mathcal{I}$, we only consider the maximal solution of the ODE and initial condition problem: we denote this solution by $\mathcal{I}_c : [0, t] \to (P_A)^{\mathbb{Z}}$.

3 Limit Systems for ISPS

In this section, we are interested in computing a *limit system* for an ISPS when we make the discrete grid $\mathbb{Z}$ tends to the continuous line $\mathbb{R}$: We want that the behavior of the ISPS tends to the behavior of the limit system for a suitable convergence definition. The target models are PDE since these are the most widely used for continuous space and time modeling and the model in which we would like to find complex behaviors.

3.1 Differential Normal Form for Multivariate Polynomials

In this section, we elaborate a tool that allows us to see a possible limit system for an ISPS.

We consider polynomial rings $\mathbb{K}[\mathbb{X}]$ and $\mathbb{K}[\partial\mathbb{X}]$ over a ring $\mathbb{K}$ using variables from the ordered set $\mathbb{X} = X_0, \ldots, X_n$ and over the ordered set $\partial\mathbb{X} = X^{(0)}, X^{(1)}, \ldots, X^{(n)}$ respectively. The link with multivariate polynomials can be seen in the writing of i and o in Equation (1) and (2). If we fix $j = 0$ (as every cell behaves the same way, we only need to study one cell), both i and o are polynomials using variables $P[c_v = s]$ for $v \in \{-r, \ldots, r\}$ and $s \in S$.

We only consider two states ISPS, both for sake of simplicity and because we expect to be able to find chaotic ISPS even with $A = \{0, 1\}$. Using the fact that for all $j \in \mathbb{Z}$, $P[c_j = s] + P[c_j \neq s] = 1$, we only need one of these two variables.

It only remains $2r+1$ variables which is the reason why the variables of $\mathbb{X}$ have one-dimensional index. The extension to other set of states is identical but leads to more intricate formulas.

We introduce the following notation, defined recursively. For all integer $j \leqslant n$, denote $X_j^{(0)} = X_j$. For all integers $k < n$ and $j < n-k$, define $X_j^{(k+1)} = X_{j+1}^{(k)} - X_j^{(k)}$. The idea behind this notation is that if $X_j = u(j\varepsilon)$, for a $\mathcal{C}^\infty$ function $u : \mathbb{R} \to \mathbb{R}$, $\varepsilon^{-k} X_j^{(k)}$ is the discrete approximation of step ε of the value of the kth derivative of u at point $(j + \frac{k}{2})\varepsilon$ as done using the finite difference method (FDM for short) to numerically solve a PDE.

Proposition 1. *Let $\mathbb{K}$ be a ring. For all polynomial P in $\mathbb{K}[\mathbb{X}]$, one can compute a polynomial ∂P of $\mathbb{K}[\partial\mathbb{X}]$ such that if, for all integer $k \leqslant n$, one replaces the variable $X^{(k)}$ in ∂P by $X_0^{(k)}$, one get P. We call ∂P the* differential normal form *(DNF for short) of P.*

Proof. The idea is simple. Let P be a polynomial in $\mathbb{K}[\mathbb{X}]$. Perform the following replacement:

For i from n down to 1
  For k from 0 to $n-i$
    Replace $X_i^{(k)}$ in P by $X_{i-1}^{(k)} + X_{i-1}^{(k+1)}$
Finally, replace $X_0^{(k)}$ in P by $X^{(k)}$ for all k. □

3.2 PDE Systems as a Limit of ISPS

Now we use the previous proposition in order to give a limit system for ISPS. This idea is to do the reverse as in FDM. In FDM, when one wants to simulate a PDE of the form $\frac{\partial u}{\partial t} = f\left(\left(\frac{\partial^n u}{\partial x^n}\right)_{n\in\mathbb{N}}\right)$, one starts from a discretized version of the initial condition at $t = 0$ and computes the values of u at $t = \varepsilon$ approximating the space derivatives as previously and repeats the process for $t = 2\varepsilon$, etc. For more information about FDM, cf., e.g., [7].

Here we are looking for ISPS rules which are performing a computation close to the FDM simulation of a PDE. By close, we mean that the least negligible terms in the ISPS computation are equal to the FDM simulation of the PDE. What links this work to FDM field is that when one defines a FDM scheme, one has to prove the this scheme tends to behave as the simulated system when ε tends to 0. Unfortunately, no general result exists which can be applied for the PDE we consider.

The sequel of the paper is devoted to determining which ISPS have such a property and computing the associated PDE.

As stated in the previous section, we consider the case with two states $\{0, 1\}$ for sake of simplicity. However, the reasoning can be applied to the general case which only leads to more complicated formulas.

We consider that $n = 2r$ in the definition of $\mathbb{X}$ and $\partial\mathbb{X}$ and all their subsequent definitions. We first introduce some new notions.

Definition 1 (Rank, reduction and π). *Consider the polynomial ring* $\mathbb{R}[\partial\mathbb{X}]$. *We define the* rank *of the variable* $X^{(p)}$ *as* p *and the rank of a monomial of* $\mathbb{R}[\partial\mathbb{X}]$ *as the sum of the ranks of its variables. Finally, the rank of a polynomial* P *of* $\mathbb{R}[\mathbb{X}]$, *denoted by* $\mathfrak{R}P$, *is the minimal rank of the monomials of its DNF.*

The reduction of a polynomial P, *denoted* $\mathrm{red}(P)$, *is the sum of the monomials of rank* $\mathfrak{R}P$ *of its DNF.*

Let P *be a polynomial from* $\mathbb{R}[\partial\mathbb{X}]$ *and* u *be a functional symbol. We denote* $\pi_u(P)$ *the result of the replacement of all variable* $X^{(p)}$ *in* $\mathrm{red}(P)$ *by* $\frac{\partial^p u}{\partial x^p}$, *the* p*th spatial derivative of* u. *We call* $\frac{\partial u}{\partial t} = \pi_u(P)$ *the* equation associated to P.

For instance, if one considers the polynomial

$$P = 2X_1^3 - 4X_1^2X_2 + 2X_1X_2^2 + X_0^2 - X_0X_1 - 2X_1^2 + X_0X_2 + X_1X_2 \ .$$

Its DNF is:

$$\partial P = 2XX^{(1)^2} + 2X^{(1)^3} + 4XX^{(1)}X^{(2)} + 4X^{(1)^2}X^{(2)} + 2XX^{(2)^2} + 2X^{(1)}X^{(2)^2} + 2XX^{(2)} + X^{(1)}X^{(2)} \ .$$

Its rank is 2, $\mathrm{red}(P) = 2XX^{(1)^2} + 2XX^{(2)}$ and its associated equation is

$$\frac{\partial u}{\partial t} = 2u\left(\frac{\partial u}{\partial x}\right)^2 + 2u\frac{\partial^2 u}{\partial x^2} \ .$$

The higher the rank of a monomial, the more negligible it is. The reduction of a polynomial only retains the less negligible terms. This is due to the fact that the discrete kth spatial derivative of step ε is $\varepsilon^{-k}X^{(k)}$. When ε tends to 0, this term is supposed to converge since we are dealing with a discretization of a $\mathcal{C}^\infty$ function. Hence, $X^{(k)}$ is $O(\varepsilon^k)$. As these terms are divided by ε^ℓ where ℓ is the smallest rank, $O(\varepsilon^{\ell+1})$ terms lead to null limit. Hence, we only have to keep the monomials with rank ℓ. This fact is formally described after Theorem 1.

Now we can find a limit system for an ISPS $\langle\{0,1\}, r, \lambda\rangle$. For $v \in \{0,\ldots,2r\}$, denote $Y_1^v = X_{v+r}$ and $Y_0^v = 1 - X_{v+r}$. The probability that cell v is in the state $s \in \{0,1\}$ is represented by the expression Y_s^v. This idea is classical and has been exploited in the definition of *fuzzy cellular automata* [8].

Definition 2 (Rank and limit system of an ISPS). *Let* $\langle\{0,1\}, r, \lambda\rangle$ *be an ISPS.*

***Step** 1. Let* P *be the polynomial of* $\mathbb{R}[\mathbb{X}]$ *defined by* $P = I - O$ *where*

$$I = \sum_{n \in A^{2r+1}, n_0 = 0} \lambda(1,n) \prod_{v=-r}^{r} Y_{n_v}^v$$

and

$$O = \sum_{n \in A^{2r+1}, n_0 = 1} \lambda(0,n) \prod_{v=-r}^{r} Y_{n_v}^v \ .$$

***Step** 2. Build ∂P the DNF of P.*
***Step** 3. Let $\ell = \mathfrak{R}P$ and consider the PDE $E : \frac{\partial u}{\partial t} = \pi_u(P)$.*

We call ℓ the rank *of the ISPS and E its* limit system.

The justification for this name is given by the following theorem. If λ is an ISPS rule, denote $\varepsilon\lambda$ the ISPS rule λ whose all rates are multiplied by ε.

Theorem 1. *Let $\mathcal{I} = \langle \{0,1\}, r, \lambda \rangle$ be an ISPS of rank $\ell > 0$. Let f be a $\mathcal{C}^\infty$ function.*

For a given positive real ε, let $\mathcal{I}^{(\varepsilon)} = \langle \{0,1\}, r, \varepsilon^{-\ell}\lambda \rangle$ and c^ε be the configuration of $\mathcal{I}^{(\varepsilon)}$ such that $P[c^\varepsilon_j = 1] = f(j\varepsilon)$. Let $t^{(\varepsilon)}$ be the diverge time of c^ε for $\mathcal{I}^{(\varepsilon)}$.

Finally, let u be the maximal solution of the limit system of the ISPS using f as its initial condition for $t = 0$ and t_u its diverge time.

Then, one has $t_u \leqslant \liminf_{\varepsilon\to\infty} t^{(\varepsilon)}$ and $\mathcal{I}^{(\varepsilon)}_{c^\varepsilon}$ pointwise converges to u when ε tends to 0.

Note that the increase of the rates in $\mathcal{I}^{(\varepsilon)}$ is not artificial. As f and its derivatives are continuous, when ε gets smaller, the values tend to become closer. The rates are modified to take this into account.

Also, note that if the rank is 0, the limit system of the ISPS has no spatial derivative and is of little interest.

The key idea of the proof of the theorem is the following lemma.

Lemma 1. *Let P be a polynomial and f be a $\mathcal{C}^\infty$ function. Let $\ell = \mathfrak{R}P$. For $\varepsilon > 0$, define $c^\varepsilon \in \mathbb{R}^{\mathbb{Z}}$ by $c^\varepsilon_j = f(\varepsilon j)$. Finally, consider P as a function from $\mathbb{R}^{\mathbb{X}}$ to $\mathbb{R}$. Then,*

$$\lim_{\varepsilon\to 0} \varepsilon^{-\ell} P(c^\varepsilon_0, \ldots, c^\varepsilon_{2r}) = \pi_f(P)(0) \ .$$

Proof. We consider P in its DNF. Note that from the linearity of the limit and π, we only have to prove this lemma for a monomial of rank ℓ and prove that the limit is 0 for a monomial of greater rank. Let $M = X^{(m_0)} \cdots X^{(m_l)}$ be a monomial of rank $h \geqslant \ell$. Note that $h = \sum_{a=0}^{l} m_a$. By simple induction on m, one can prove that $\lim_{\varepsilon\to 0} \varepsilon^{-m} X^{(m)}(c^\varepsilon_0, \ldots, c^\varepsilon_{2r}) = f^{(m)}(\varepsilon\frac{m}{2})$ when $f^{(m)}$ is the mth derivative of f. Multiplying these finite limits together, one gets that $\lim_{\varepsilon\to 0} \varepsilon^{-h} M(c^\varepsilon_0, \ldots, c^\varepsilon_{2r}) = \pi_f(M)(0)$ which proves the both cases $h = \ell$ and $h > \ell$ since, for the latter, $\varepsilon^{h-\ell}$ tends to 0. □

Proof (of Theorem 1). Using this lemma, one has that the right member of the ODE ruling the ISPS $\mathcal{I}^{(\varepsilon)}$ tends to the right member of the PDE ruling the limit system when ε tends to 0, the left member being the time derivative for all systems. The existence of u and t_u comes from the Cauchy-Lipschitz theorem on Banach space $(\mathbb{R}^{\mathbb{R}}, \|\cdot\|_*)$ (for a suitable $\|\cdot\|_*$) and u is defined on $\mathbb{R} \times [0, t_u)$. The time t_u can be much smaller than the $t^{(\varepsilon)}$ because the spatial derivatives could diverge in the continuous model. The proof of Theorem 1 can be ended by straightforward calculus. □

4 Limit System of Positive Rank

In this section, we design a method to compute the positively ranked local rules and give a conjecture about their general forms.

First, note that a polynomial P has a positive rank if and only if the univariate polynomial $\tilde{P}$ obtained replacing all variables of P by Z is 0. This can be easily seen since, once put into DNF, if one replaces all variables by Z, all $X^{(i)}$ for $i > 0$ become 0. It only remains the $X^{(0)}$ variable whose powers form the rank 0 monomials we want to avoid.

We consider i and o with $j = 0$ in Equations (1) and (2) as polynomial as we did at the beginning of Subsection 3.1: As above, we consider $i - o$ as a polynomial in variables $X_v = P[C_v = 1]$ for $v \in \{-r, \dots, r\}$ since we replace $P[C_v = 0]$ by $1 - P[C_v = 1]$. We also consider the $\Lambda_{s,n} = \lambda(s, n)$ terms as variables. Then $i - o$ becomes a polynomial of $\mathbb{Z}[\Lambda_{s,n}][X_v]$. We want that, when we replace the X_v variables by the single variable Z, this polynomial becomes 0. Let P be the obtained polynomial of $\mathbb{Z}[\Lambda_{s,n}][Z]$. A rule λ has positive rank if and only if all the coefficients of P, which are polynomials over $\mathbb{Z}[\Lambda_{s,n}]$, are 0.

For a fixed r, the polynomial P can be easily computed[1]. Looking at the results for small r, we make the following conjecture:

Conjecture 1. If $n \in \{0,1\}^{2r+1}$, denote $|n| = |\{i | n_i = 1\}|$, the number of 1 in n.

The ISPS of radius r and positive rank have their local rule λ verify:

$$\begin{cases} \forall v \in \{1, \dots, 2r\}, \sum_{\substack{n \in \{0,1\}^{2r+1} \\ n_0 = 0, |n| = v}} \lambda(1, n) = \sum_{\substack{n \in \{0,1\}^{2r+1} \\ n_0 = 1, |n| = v}} \lambda(0, n) \\ \lambda(1, 0 \dots 0) = \lambda(0, 1 \dots 1) = 0 \end{cases}$$

To better understand the previous formulas, we study the special case where λ mimics a CA rule μ: given the states n of the cell and its neighbors, the rate to change the state of a cell is 0 for all state values but $\mu(n)$. More formally, if μ is the local rule of a CA, we define λ as the ISPS local rule defined by

$$\lambda(s, n) = \begin{cases} 1 & \text{if } \mu(n) = s \\ 0 & \text{otherwise} \end{cases} .$$

On alphabet $\{0, 1\}$, the polynomial P defined above is transformed replacing $\Lambda_{1,n}$ by M_n ($\mu(n) = 1$ if and only if $\lambda(1, n) = 1$) and $\Lambda_{0,n}$ by $1 - M_n$ ($\mu(n) = 0$ if and only if $\lambda(0, n) = 1$).

For this case, the conjecture's condition becomes:

$$\forall v \in \{0, 2r + 1\}, \sum_{n \in \{0,1\}^{2r+1}, |n| = v} \mu(n) = \binom{v - 1}{2r} \quad \text{with } \tbinom{-1}{2r} = 0 \ .$$

To improve our understanding, one can compute the rank and the limit system for small well-known CA rules. For $r = 1$, one has 256 such CA. Nevertheless,

[1] Computation done with Sage (`http://www.sagemath.org`).

one can only find 9 CA of positive rank: The identity CA, the two shift CA (to the left and to the right), the two traffic CA (cars going to the left and to the right) and four CA which are equivalent, up to swapping 0 and 1 and/or left and right neighbors, to elementary rule 172. The shift has for limit system the PDE $\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}$ which is as expected. For the traffic CA where cars go to the right, the PDE is $\frac{\partial u}{\partial t} = (2u - 1)\frac{\partial u}{\partial x}$.

About $r = 2$, peeking at random a rule sampled Fig 1, one get the PDE:

$$\frac{\partial u}{\partial t} = 18u \left(\frac{\partial u}{\partial x}\right)^2 - 5u^2 \frac{\partial^2 u}{\partial x^2} - 9 \left(\frac{\partial u}{\partial x}\right)^2 + 5u \frac{\partial^2 u}{\partial x^2} \ .$$

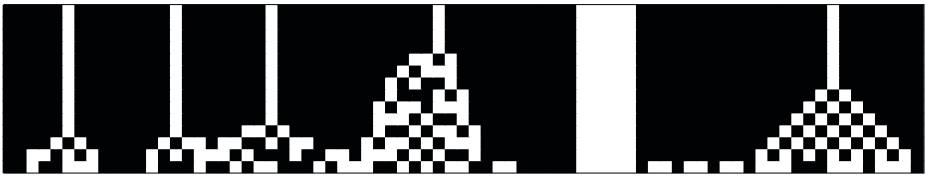


Fig. 1. Space-time diagram (stack of configurations at $t = 0, 1, \ldots$) of the random positively ranked rule. State 0 is in black. Time goes upward.

5 Conclusion

In this paper, we proposed a way to produce a space and time continuous limit system for ISPS and consequently to associate such a system to some CA.

We gave the enumeration of these limit systems for elementary CA (radius 1 and two-letter alphabet) having a perfect match for the shift automaton and a reasonable one for the traffic automaton. We also gave a conjecture about the condition a rule must respect in order to converge to a PDE.

However, but proving the conjecture, this work leaves open many paths of research described in the following sections.

5.1 CA Corresponding to an Equation

The reverse process could be interesting in order to find the cellular automaton which best matches a given PDE. Finding an ISPS is easy simply applying the FDM. To do such, one only has to pay attention to ranks. If the PDE contains terms of different ranks (e.g., $\frac{\partial u}{\partial x} + u\frac{\partial^2 u}{\partial x^2}$ is the sum of a rank 1 and a rank 2 term), the ISPS does not exists. Yet, the ISPS model is too complicated to give better insights than the equation itself.

However, if all the terms have the same rank, one can try to find if the ISPS has a local rule derived from a CA. However, it could be of any radius so an exhaustive search is only a semi-decision procedure. We also plan to study which equations can be obtained from a CA rule.

5.2 Links between Models

The construction of the limit continuous system requires to consider ISPS as an intermediary step between CA and PDE. We have shown in Section 3 a link between the ISPS and the limit system but there are still to see what dynamical properties (expansivity, sensitivity, entropy, etc.) transfers from one another. The fact that the limit system is the limit of the ISPS tends to make us think that this is true. However we do not think that there is such a connection between the CA and the limit system since discrete systems deals with many singularities which are erased by continuous models. We are interested in looking under which conditions the CA behaves the same as its parented PDE.

5.3 Removing the Independence Simplification

Finally, we plan to use the DNF notion about ISPS in which the independence assumption is removed. We still have to consider polynomials but countably many polynomials with countably many variables.

References

1. Kari, J.: Universal pattern generation by cellular automata. Theor. Comput. Sci. 429, 180–184 (2012)
2. Kurka, P.: Topological and symbolic dynamics. Société Mathématique de France (2003)
3. Masi, A.D., Ferrari, P., Lebowitz, J.: Reaction-diffusion equations for interacting particle systems. Journal of Statistical Physics 44, 589–644 (1986)
4. Durret, R., Neuhauser, C.: Particle systems and reaction-diffusion equations. Annals of Probability 22, 289–333 (1994)
5. Dobrushin, R., Kriukov, V., Toom, A.: Stochastic Cellular Systems: Ergodicity, Memory, Morphogenesis. Nonlinear science. Manchester University Press (1990)
6. Durrett, R.: Stochastic spatial models. SIAM Review 41(4), 677–718 (1999)
7. Thomas, J.: Numerical Partial Differential Equations: Finite Difference Methods. Texts in Applied Mathematics. Springer (2010)
8. Cattaneo, G., Flocchini, P., Mauri, G., Quaranta Vogliotti, C., Santoro, N.: Cellular automata in fuzzy backgrounds. Physica D: Nonlinear Phenomena 105, 105–120 (1997)

Latency-Bounded Target Set Selection in Social Networks

Ferdinando Cicalese[1], Gennaro Cordasco[2], Luisa Gargano[1], Martin Milanič[3], and Ugo Vaccaro[1]

[1] Department of Computer Science, University of Salerno, Italy
{cicalese,lg,uv}@dia.unisa.it
[2] Department of Psychology, Second University of Naples, Italy
gennaro.cordasco@unina2.it
[3] University of Primorska, UP IAM and UP FAMNIT, SI 6000 Koper, Slovenia
martin.milanic@upr.si

Abstract. We study variants of the Target Set Selection problem, first proposed by Kempe *et al.* In our scenario one is given a graph $G = (V, E)$, integer values $t(v)$ for each vertex v, and the objective is to determine a small set of vertices (*target set*) that activates a given number (or a given subset) of vertices of G *within* a prescribed number of rounds. The activation process in G proceeds as follows: initially, at round 0, all vertices in the target set are activated; subsequently at each round $r \geq 1$ every vertex of G becomes activated if at least $t(v)$ of its neighbors are active by round $r-1$. It is known that the problem of finding a minimum cardinality Target Set that eventually activates the whole graph G is hard to approximate to a factor better than $O(2^{\log^{1-\epsilon}|V|})$. In this paper we give *exact* polynomial time algorithms to find minimum cardinality Target Sets in graphs of bounded clique-width, and *exact* linear time algorithms for trees.

1 Introduction

Let $G = (V, E)$ be a graph, $S \subseteq V$, and let $t : V \longrightarrow \mathbb{N} = \{1, 2, \ldots\}$ be a function assigning integer thresholds to the vertices of G. An *activation process in G starting at S* is a sequence $\mathsf{Active}[S, 0] \subseteq \mathsf{Active}[S, 1] \subseteq \ldots \subseteq \mathsf{Active}[S, i] \subseteq \ldots \subseteq V$ of vertex subsets, with $\mathsf{Active}[S, 0] = S$, and such that for all $i > 0$,

$$\mathsf{Active}[S, i] = \mathsf{Active}[S, i-1] \cup \left\{ u \,:\, \left| N(u) \cap \mathsf{Active}[S, i-1] \right| \geq t(u) \right\}$$

where $N(u)$ is the set of neighbors of u.

The central problem we introduce and study in this paper is defined as follows:

(λ, β, α)-Target Set Selection ((λ, β, α)-TSS).
Instance: A graph $G = (V, E)$, thresholds $t : V \longrightarrow \mathbb{N}$, a latency bound $\lambda \in \mathbb{N}$, a budget $\beta \in \mathbb{N}$ and an activation requirement $\alpha \in \mathbb{N}$.
Problem: Find $S \subseteq V$ s.t. $|S| \leq \beta$ and $|\mathsf{Active}[S, \lambda]| \geq \alpha$ (or determine that no such a set exists).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 65–77, 2013.

We will be also interested in the case in which *a set of nodes that need to be activated* (within the given latency bound) is explicitly given as part of the input.

(λ, β, A)-Target Set Selection ((λ, β, A)-TSS).
Instance: A graph $G = (V, E)$, thresholds $t : V \longrightarrow \mathbb{N}$, a latency bound $\lambda \in \mathbb{N}$, a budget $\beta \in \mathbb{N}$ and a set to be activated $A \subseteq V$.
Problem: Find a set $S \subseteq V$ such that $|S| \leq \beta$ and $A \subseteq \mathsf{Active}[S, \lambda]$ (or determine that such a set does not exist).

Eliminating any one of the parameters λ and β, one obtains two natural minimization problems. For instance, eliminating β, one obtains the following problem:

(λ, A)-Target Set Selection ((λ, A)-TSS).
Instance: A graph $G = (V, E)$, thresholds $t : V \longrightarrow \mathbb{N}$, a latency bound $\lambda \in \mathbb{N}$ and a set $A \subseteq V$.
Problem: Find a set $S \subseteq V$ of minimum size such that $A \subseteq \mathsf{Active}[S, \lambda]$.

The above algorithmic problems have roots in the general study of the *spread of influence* in Social Networks [11]. For instance, in the area of viral marketing [10] companies wanting to promote products or behaviors might try initially to target and convince a few individuals which, by word-of-mouth effects, can trigger a cascade of influence in the network, leading to an adoption of the products by a much larger number of individuals. It is clear that the (λ, β, α)-TSS problem represents an abstraction of that scenario, once one makes the reasonable assumption that an individual decides to adopt the products if a certain number of his/her friends have adopted said products. Analogously, the (λ, β, α)-TSS problem can describe other diffusion problems arising in sociological, economical and biological networks, again see [11]. Therefore, it comes as no surprise that special cases of our problem (or variants thereof) have recently attracted much attention by the algorithmic community. The first authors to study problems of spread of influence in networks from an algorithmic point of view were Kempe *et al.* [13,14]. However, they were mostly interested in networks with randomly chosen thresholds. Chen [4] studied the following minimization problem: Given a graph G and fixed thresholds $t(v)$, find a target set of minimum size that eventually activates all (or a fixed fraction of) vertices of G. He proved a strong inapproximability result that makes unlikely the existence of an algorithm with approximation factor better than $O(2^{\log^{1-\epsilon}|V|})$. This motivated the work [1,2,5] that isolated interesting cases in which the problems become efficiently tractable.

All the above mentioned papers did not consider the issue of the number of rounds necessary for the activation of the required number of vertices. However, this is a relevant question: In viral marketing, for instance, it is quite important to spread information quickly.

For general graphs, Chen's [4] inapproximability result still holds if one demands that the activation process ends in a bounded number of rounds; this

motivates our first result. We show that the general (λ, β, α)-TSS problem is polynomially solvable in graph of bounded clique-width and constant latency bound λ (see Theorem 1 in Section 2). Since graphs of bounded treewidth are also of bounded clique-width [8], this result implies a polynomial solution of the (λ, β, α)-TSS problem with constant λ also for graphs of bounded treewidth, complementing the result of [2] showing that for bounded-treewidth graphs, the TSS problem without the latency bound (equivalently, with $\lambda = |V| - 1$) is polynomially solvable. Moreover, the result settles the status of the computational complexity of the VECTOR DOMINATION problem for graphs of bounded tree- or clique-width, which was posed as an open question in [6].

We also consider the instance when G is a tree. For this special case we give an *exact linear time* algorithm for the (λ, A)-TSS problem, *for any* λ and $A \subseteq V$. When $\lambda = |V| - 1$ and $A = V$ our result is equivalent to the (optimal) linear time algorithm for the classical TSS problem (i.e., without the latency bound) on trees proposed in [4].

Because of space constraints, most proofs are omitted and presented in [7].

2 TSS Problems on Bounded Clique-Width Graphs

In this section, we give an algorithm for the (λ, β, α)-TARGET SET SELECTION problem on graphs G of clique-width at most k given by an irredundant k-expression σ. For the sake of self-containment we recall here some basic notions about clique-width.

The Clique-Width of a Graph. A *labeled graph* is a graph in which every vertex has a label from $\mathbb{N}$. A labeled graph is a *k-labeled graph* if every label is from $[k] := \{1, 2, \ldots, k\}$. The *clique-width* of a graph G is the minimum number of labels needed to construct G using the following four operations: (i) Creation of a new vertex v with label a (denoted by $a(v)$); (ii) disjoint union of two labeled graphs G and H (denoted by $G \oplus H$); (iii) Joining by an edge each vertex with label a to each vertex with label b ($a \neq b$, denoted by $\eta_{a,b}$); (iv) renaming label a to b (denoted by $\rho_{a \to b}$). Every graph can be defined by an algebraic expression using these four operations. For instance, a chordless path on five consecutive vertices u, v, x, y, z can be defined as follows:
$\eta_{3,2}(3(z) \oplus \rho_{3\to 2}(\rho_{2\to 1}(\eta_{3,2}(3(y) \oplus \rho_{3\to 2}(\rho_{2\to 1}(\eta_{3,2}(3(x) \oplus \eta_{2,1}(2(v) \oplus 1(u)))))))))$.
Such an expression is called a *k-expression* if it uses at most k different labels. The clique-width of G, denoted $\mathrm{cw}(G)$, is the minimum k for which there exists a k-expression defining G. If a graph G has a clique-width at most k, then a $(2^{k+1} - 1)$-expression for it can be computed in time $O(|V(G)|^3)$ using the rank-width [12,15].

Every graph of clique-width at most k admits an *irredundant* k-expression, that is, a k-expression such that before any operation of the form $\eta_{a,b}$ is applied, the graph contains no edges between vertices with label a and vertices with label b [9]. In particular, this means that every operation $\eta_{a,b}$ adds at least one edge to the graph G. Each expression σ defines a rooted tree $T(\sigma)$, that we also call a *clique-width tree*.

Our Result on Graphs with Bounded Clique-Width. We describe an algorithm for the (λ, β, α)-TSS problem on graphs G of clique-width at most k given by an irredundant k-expression σ. Denoting by n the number of vertices of the input graph G, the running time of the algorithm is bounded by $O(\lambda k|\sigma|(n+1)^{(3\lambda+2)k})$, where $|\sigma|$ denotes the encoding length of σ. For fixed k and λ, this is polynomial in the size of the input. We will first solve the following naturally associated *decision problem:*

(λ, β, α)-Target Set Decision ((λ, β, α)-TSD).
Instance: A graph $G = (V, E)$, thresholds $t : V \longrightarrow \mathbb{N}$, a latency bound $\lambda \in \mathbb{N}$, a budget $\beta \in \mathbb{N}$ and an activation requirement $\alpha \in \mathbb{N}$.
Problem: Determine whether there exists a set $S \subseteq V$ such that $|S| \leq \beta$ and $|\mathsf{Active}[S, \lambda]| \geq \alpha$.

Consider an instance $(G, t, \lambda, \beta, \alpha)$ to the (λ, β, α)-Target Set Decision problem, where $G = (V, E)$ is a graph of clique-width at most k given by an irredundant k-expression σ. We will develop a dynamic programming algorithm that will traverse the clique-width tree bottom up and simulate the activation process for the corresponding induced subgraphs of G, keeping track only of the minimal necessary information, that is, of how many vertices of each label become active in each round. For a bounded number of rounds λ, it will be possible to store and analyze the information in polynomial time. In order to compute these values recursively with respect to all the operations in the definition of the clique-width—including operations of the form $\eta_{a,b}$—we need to consider not only the original thresholds, but also reduced ones. This is formalized in Definition 1 below. We view G as a k-labeled graph defined by σ. Given a k-labeled graph H and a label $\ell \in [k]$, we denote by $V_\ell(H)$ the set of vertices of H with label ℓ.

Definition 1. *Given a k-labeled subgraph H of G and a pair of matrices with non-negative integer entries $(\boldsymbol{\alpha}, \boldsymbol{r})$ such that $\boldsymbol{\alpha} \in (\mathbb{Z}_+)^{[0,\lambda]\times[k]}$ (where $[0, \lambda] := \{0, 1, \ldots, \lambda\}$) and $\boldsymbol{r} \in (\mathbb{Z}_+)^{[\lambda]\times[k]}$, an* $(\boldsymbol{\alpha}, \boldsymbol{r})$-activation process for H *is a non-decreasing sequence of vertex subsets $S[0] \subseteq \ldots \subseteq S[\lambda] \subseteq V(H)$ such that the following conditions hold:*

(1) For every round $i \in [\lambda]$ and for every label $\ell \in [k]$, the set of all vertices with label ℓ activated at round i is obtained with respect to the activation process starting at $S[0]$ with thresholds $t(u)$ reduced by $r[i, \ell]$ for all vertices with label ℓ. Formally, for all $\ell \in [k]$ and all $i \in [\lambda]$,

$$(S[i]\backslash S[i-1]) \cap V_\ell(H) = \Big\{u \in V_\ell(H)\backslash S[i-1] \,:\, \big|N_H(u) \cap S[i-1]\big| \geq t(u) - r[i, \ell]\Big\}.$$

(2) For every label $\ell \in [k]$, there are exactly $\alpha[0, \ell]$ initially activated vertices with label ℓ: $|S[0] \cap V_\ell(H)| = \alpha[0, \ell]$.

(3) For every label $\ell \in [k]$ and for every round $i \in [\lambda]$, there are exactly $\alpha[i, \ell]$ vertices with label ℓ activated at round i: $|(S[i] \setminus S[i-1]) \cap V_\ell(H)| = \alpha[i, \ell]$.

Let $\mathcal{A}$ denote the set of all matrices of the form $\boldsymbol{\alpha} = (\alpha[i,\ell] : 0 \leq i \leq \lambda, 1 \leq \ell \leq k)$ where $\alpha[i,\ell] \in [0,\alpha]$ for all $0 \leq i \leq \lambda$ and all $1 \leq \ell \leq k$. Notice that $|\mathcal{A}| = (\alpha+1)^{(\lambda+1)k} = O((n+1)^{(\lambda+1)k})$. Similarly, let $\mathcal{R}$ denote the set of all matrices of the form $\boldsymbol{r} = (r[i,\ell] : 1 \leq i \leq \lambda, 1 \leq \ell \leq k)$, where $r[i,\ell] \in [0,n]$ for all $1 \leq i \leq \lambda$ and all $1 \leq \ell \leq k$. Then $|\mathcal{R}| = (n+1)^{\lambda k}$.

Every node of the clique-width tree $T := T(\sigma)$ of the input graph G corresponds to a k-labeled subgraph H of G. To every node of T (and the corresponding k-labeled subgraph H of G), we associate a Boolean-valued function $\gamma_H : \mathcal{A} \times \mathcal{R} \longrightarrow \{0,1\}$ where $\gamma_H(\boldsymbol{\alpha},\boldsymbol{r}) = 1$ if and only if there exists an $(\boldsymbol{\alpha},\boldsymbol{r})$-activation process for H. Each matrix pair $(\boldsymbol{\alpha},\boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}$ can be described with $O(\lambda k)$ numbers. Hence, the function γ_H can be represented by storing the set of all triples $\{(\boldsymbol{\alpha},\boldsymbol{r},\gamma_H(\boldsymbol{\alpha},\boldsymbol{r})) : (\boldsymbol{\alpha},\boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}\}$, requiring, in total, space $O(\lambda k)\cdot|\mathcal{A}\times\mathcal{R}| = O(\lambda k)\cdot O((n+1)^{(\lambda+1)k})\cdot O((n+1)^{\lambda k}) = O(\lambda k(n+1)^{(2\lambda+1)k})$.

Below we will describe how to compute all functions γ_H for all subgraphs H corresponding to the nodes of the tree T. Assuming all these functions have been computed, we can extract the solution to the (λ,β,α)-Target Set Decision problem on G from the root of T as follows.

Proposition 1. *There exists a set $S \subseteq V(G)$ such that $|\mathsf{Active}[S,\lambda]| \geq \alpha$ and $|S| \leq \beta$ if and only if there exists a matrix $\boldsymbol{\alpha} \in \mathcal{A}$ with $\gamma_G(\boldsymbol{\alpha},\mathbf{0}) = 1$ (where $\mathbf{0} \in \mathcal{R}$ denotes the all zero matrix) such that $\sum_{\ell=1}^{k}\alpha[0,\ell] \leq \beta$ and $\sum_{i=0}^{\lambda}\sum_{\ell=1}^{k}\alpha[i,\ell] \geq \alpha$.*

Let us now describe how to compute the functions γ_H by traversing the tree T bottom up. We consider four cases according to the type of a node v of T.

Case 1: v is a leaf. In this case, the labeled subgraph H of G associated to v is of the form $H = a(u)$ for some vertex $u \in V(G)$ and some label $a \in [k]$. That is, a new vertex u is introduced with label a. Let us denote by $(\mathcal{A}\times\mathcal{R})^*$ the set

$$(\mathcal{A}\times\mathcal{R})^* = \Bigg\{(\boldsymbol{\alpha},\boldsymbol{r}) \in \mathcal{A}\times\mathcal{R} : (\forall \ell \neq a)(\alpha[i,\ell] = 0) \wedge \Bigg(\sum_{i=0}^{\lambda}\alpha[i,a] \leq 1\Bigg)$$

$$\wedge \Bigg[\Bigg(\sum_{i=0}^{\lambda}\alpha[i,a] = 0\Bigg) \Rightarrow \Big((\forall i)\big(r[i,a] < t(u)\big)\Big)\Bigg]$$

$$\wedge \Bigg[\Big((\exists i^*)(\alpha[i^*,a] = 1)\Big) \Rightarrow \Big(i^* = 0 \vee i^* = \min\{i \geq 1 : r[i,a] \geq t(u)\}\Big)\Bigg]\Bigg\}.$$

In this case for every $(\boldsymbol{\alpha},\boldsymbol{r}) \in \mathcal{A}\times\mathcal{R}$, we set $\gamma_H(\boldsymbol{\alpha},\boldsymbol{r}) = \begin{cases} 1, \text{ if } (\boldsymbol{\alpha},\boldsymbol{r}) \in (\mathcal{A}\times\mathcal{R})^*; \\ 0, \text{ otherwise.} \end{cases}$

Case 2: v has exactly two children in T. In this case, the labeled subgraph H of G associated to v is the disjoint union $H = H_1 \oplus H_2$, where H_1 and H_2 are the labeled subgraphs of G associated to the two children of v in T. In this case, for every $(\boldsymbol{\alpha},\boldsymbol{r}) \in \mathcal{A}\times\mathcal{R}$ we set

$$\gamma_H(\boldsymbol{\alpha},\boldsymbol{r}) = \begin{cases} 1, \text{ if } (\exists \boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2 \in \mathcal{A})(\boldsymbol{\alpha}=\boldsymbol{\alpha}_1+\boldsymbol{\alpha}_2 \text{ and } \gamma_{H_1}(\boldsymbol{\alpha}_1,\boldsymbol{r})=\gamma_{H_2}(\boldsymbol{\alpha}_2,\boldsymbol{r})=1); \\ 0, \text{ otherwise.} \end{cases}$$

Case 3: v has exactly one child in T and the labeled subgraph H of G associated to v is of the form $H = \eta_{a,b}(H_1)$. In this case, graph H is obtained from H_1 by adding all edges between vertices labeled a and vertices labeled b. Since the k-expression is irredundant, in H_1 there are no edges between vertices labeled a and vertices labeled b.

For every $(\boldsymbol{\alpha}, \boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}$ we define the integer-valued matrix $\boldsymbol{r}_1$ by setting

$$r_1[i,\ell] = \begin{cases} \min\{n, r[i,a] + \sum_{j<i} \alpha[j,b]\}, \text{ if } \ell = a; \\ \min\{n, r[i,b] + \sum_{j<i} \alpha[j,a]\}, \text{ if } \ell = b; \\ r[i,\ell], \qquad\qquad\qquad \text{otherwise}, \end{cases}$$

for every $i \in [0,\lambda]$ and for every label $\ell \in [k]$. Then, we set, $\gamma_H(\boldsymbol{\alpha}, \boldsymbol{r}) = \gamma_{H_1}(\boldsymbol{\alpha}, \boldsymbol{r}_1)$.

Case 4: v has exactly one child in T and the labeled subgraph H of G associated to v is of the form $H = \rho_{a\to b}(H_1)$. For every $(\boldsymbol{\alpha}, \boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}$ we set $\gamma_H(\boldsymbol{\alpha}, \boldsymbol{r}) = 1$ if and only if there exists $(\boldsymbol{\alpha}_1, \boldsymbol{r}_1) \in \mathcal{A} \times \mathcal{R}$ such that $\gamma_{H_1}(\boldsymbol{\alpha}_1, \boldsymbol{r}_1) = 1$, where for every round i and for every label $\ell \in [k]$, we have

$$r_1[i,\ell] = \begin{cases} r[i,b], \text{ if } \ell = a; \\ r[i,\ell], \text{ otherwise.} \end{cases} \quad \text{and} \quad \alpha[i,\ell] = \begin{cases} 0, & \text{if } \ell = a; \\ \alpha_1[i,a] + \alpha_1[i,b], & \text{if } \ell = b; \\ \alpha_1[i,\ell], & \text{otherwise.} \end{cases}$$

This completes the description of the four cases and the description of the algorithm.

Correctness and Time Complexity. Correctness of the algorithm follows from the derivation of recursive formulas (see [7]). We now analyze the algorithm's time complexity. Given an irredundant k-expression σ of G, the clique-width tree T can be computed from σ in linear time. The algorithm computes the sets $\mathcal{A}$ and $\mathcal{R}$ in time $|\mathcal{A}| = O((n+1)^{(\lambda+1)k})$ and $|\mathcal{R}| = O((n+1)^{\lambda k})$, respectively.

The algorithm then traverses the clique-width tree bottom-up. At each leaf of T and for each $(\boldsymbol{\alpha}, \boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}$, it can be verified in time $O(\lambda k)$ whether $(\boldsymbol{\alpha}, \boldsymbol{r}) \in (\mathcal{A} \times \mathcal{R})^*$. Hence, the function γ_H at each leaf can be computed in time $O(\lambda k (n+1)^{(2\lambda+1)k})$.

At an internal node corresponding to Case 2, the value of $\gamma_H(\boldsymbol{\alpha}, \boldsymbol{r})$ for a given $(\boldsymbol{\alpha}, \boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}$ can be computed in time $O(|\mathcal{A}|\lambda k)$ by iterating over all $\boldsymbol{\alpha}_1 \in \mathcal{A}$, verifying whether $\boldsymbol{\alpha}_2 := \boldsymbol{\alpha} - \boldsymbol{\alpha}_1 \in \mathcal{A}$ and looking up the values of $\gamma_{H_1}(\boldsymbol{\alpha}_1, \boldsymbol{r})$ and $\gamma_{H_2}(\boldsymbol{\alpha}_2, \boldsymbol{r})$. Hence, the total time spent at an internal node corresponding to Case 2 is $O(|\mathcal{A}|\lambda k) \cdot O((n+1)^{(2\lambda+1)k}) = O(\lambda k (n+1)^{(3\lambda+2)k})$.

At an internal node corresponding to Case 3 or Case 4, the value of $\gamma_H(\boldsymbol{\alpha}, \boldsymbol{r})$ for a given $(\boldsymbol{\alpha}, \boldsymbol{r}) \in \mathcal{A} \times \mathcal{R}$ can be computed in time $O(\lambda k)$. Hence, the total time spent at any such node is $O(\lambda k (n+1)^{(2\lambda+1)k})$.

The overall time complexity is $O(\lambda k |\sigma| (n+1)^{(3\lambda+2)k})$. For fixed k and λ, this is polynomial in the size of the input.

The above algorithm for the (λ, β, α)-Target Set Decision problem on graphs of bounded clique-width can be easily modified so that it also finds a solution to the (λ, β, α)-Target Set Selection problem. Hence, we have the following theorem.

Theorem 1. *For every fixed k and λ, the* (λ, β, α)-Target Set Selection *problem can be solved in polynomial time on graphs of clique-width at most k.*

When $\lambda = 1$ and $\alpha = |V(G)|$, the (λ, β, α)-Target Set Selection problem coincides with the Vector Domination problem (see, e.g, [6]). Hence, Theorem 1 answers a question from [6] regarding the complexity status of Vector Domination for graphs of bounded treewidth or bounded clique-width.

The (λ, β, A)-TSS Problem on Graphs of Small Clique-Width. The approach to solve the (λ, β, A)-Target Set Selection problem on graphs of bounded clique-width is similar to the one above. First, we consider the decision problem naturally associated with the (λ, β, A)-TSS problem, the (λ, β, A)-Target Set Decision problem ((λ, β, A)-TDS for short). Consider an instance $(G, t, \lambda, \beta, A)$ to the (λ, β, A)-TSD problem, where $G = (V, E)$ is a graph of clique-width at most k given by an irredundant k-expression σ. First, we construct a $2k$-expression σ' in such a way that every labeled vertex $a(u)$ with $u \in A$ changes to $(a+k)(u)$. Moreover, every operation of the form $\eta_{i,j}$ is replaced with a sequence of four composed operations $\eta_{i,j} \circ \eta_{i,j+k} \circ \eta_{i+k,j} \circ \eta_{i+k,j+k}$, and every operation of the form $\rho_{i \to j}$ is replaced with a sequence of two composed operations $\rho_{i,j} \circ \rho_{i+k,j+k}$. The so defined expression σ' can be obtained from σ in linear time, and defines a labeled graph isomorphic to G such that the set A contains precisely the vertices with labels strictly greater than k. Using the same notation as above (with respect to σ'), we obtain the following

Proposition 2. *There exists a set $S \subseteq V(G)$ such that $A \subseteq |\mathsf{Active}[S, \lambda]|$ and $|S| \leq \beta$ if and only if there exists a matrix $\boldsymbol{\alpha} \in \mathcal{A}$ with $\gamma_G(\boldsymbol{\alpha}, \mathbf{0}) = 1$ such that $\sum_{\ell=1}^{k} \alpha[0, \ell] \leq \beta$ and $\sum_{i=0}^{\lambda} \sum_{\ell=k+1}^{2k} \alpha[i, \ell] = |A|$.*

Hence, the same approach as above can be used to solve first the (λ, β, A)-Target Set Decision problem, and then the (λ, β, A)-Target Set Selection problem itself.

Theorem 2. *For every fixed k and λ, the* (λ, β, A)-Target Set Selection *problem can be solved in polynomial time on graphs of clique-width at most k.*

3 (λ, A)-TSS on Trees

Since trees are graphs of clique-width at most 3, results of Section 2 imply that the (λ, β, α)- and (λ, β, A)-TSS problems are solvable in polynomial time on trees when λ is constant. In this section we improve on this latter result by giving a *linear time* algorithm for the (λ, A)-TSS problem, for *arbitrary* values of λ. Our result also extends the linear time solution for the classical TSS problem (i.e., without the latency bound) on trees proposed in [4]. Like the solution in [4], we will assume that the tree is rooted at some node r. Then, once such rooting is fixed, for any node v we will denote by $T(v)$ the subtree rooted at v, by $C(v)$ the set of children of v and, for $v \neq r$, by $p(v)$ the parent of v. In the following we assume that $\forall v \in V, 1 \leq t(v) \leq d(v)$.

Next Algorithm (λ, A)-**TSS on Trees** considers each node for being included in the target set S in a bottom-up fashion. Each node is considered after all its children. Leaves are never added to S because there is always an optimal solution in which the target set consists of internal nodes only. Indeed, since all leaves have thresholds equal to 1, starting from any target set containing some leaves we can get a solution of at most the same size by substituting each targeted leaf by its parent.

Thereafter, for each non-leaf node v, the algorithm checks whether the partial solution S constructed so far allows to activate all the nodes in $T(v) \cap A$ (where A is the set of nodes which must be activated) within round λ: the algorithm computes the round $\tau = \lambda - \text{maxPath}(v)$ by which v has to be activated (line 12 of the pseudocode), where maxPath(v) denotes the maximum length of a path from v to one of its descendants which requires v's influence to become active by round λ. Notice that $\tau < \lambda$ when there exists a vertex in the subtree $T(v)$ which has to be activated by time λ, and this can happen only if v is activated by time τ. Then the algorithm computes the set $Act(v)$ consisting of those v's children which are activated at round $\tau - 1$ (line 13). The algorithm is based on the following three observations **(a)**, **(b)**, and **(c)** (assuming that v is in the set of nodes which must be activated): **(a)** v must be included in S whenever the nodes belonging to $Act(v) \cup \{p(v)\}$ do not suffice to activate v, i.e., the current partial solution is such that at most $t(v) - 2$ children of v can be active at round $\tau - 1$; **(b)** v must be included in S if $\tau = 0$ (i.e., $\lambda = \text{maxPath}(v)$). Indeed, in this case, there exists a vertex in $T(v)$, at distance λ from v, which requires v's influence to be activated, and this can only happen if v is activated at round 0 (line 19-21).

If neither **(a)** nor **(b)** is verified, then v is not activated. However, it might be that the algorithm has to guarantee the activation of some other node in the subtree $T(v)$. To deal with such a case, when **(c)** the size of the set $Act(v)$ is $t(v) - 1$, then the algorithm puts $p(v)$ in the set A of nodes to be activated; moreover, the value of the parameter path(v) is updated coherently in such a way to correctly compute the value of maxPath$(p(v))$ which assures that $p(v)$ gets active within round $\lambda - \text{maxPath}(p(v))$ (see lines 22-24).

In order to keep track of the above cases while traversing the tree bottom-up, the algorithm uses the following parameters:

round(v) assumes value equal to the round (of the activation process with target set S) in which v would be activated only thanks to its children and irrespectively of the status of its parent. Namely, round$(v) = \infty$ if v is a leaf, round$(v) = 0$ if $v \in S$, and round$(v) = 1 + \min^{t(v)}\{\text{round}(u) \mid u \in C(v)\}$ otherwise. Here $\min^{t(v)} C$ denotes the $t(v)$-th smallest element in the set C.

path(v) assumes value equal to -1 in case v's parent is not among the activators of v; otherwise, assume value equal to the maximum length of a path from v to one of its descendants which (during the activation process with target set S) requires v's influence in order to become active. It will be shown that during the activation process with target set S, for each node $v \in A$ we

have $v \in \mathsf{Active}[S, \min\{\lambda - \max_{u \in C(v)} \mathrm{path}(u) - 1, \mathrm{round}(v)\}]$, for each node $v \in A$.

Moreover, the algorithm maintains a set $A' \supseteq A$ of nodes to be activated. Initially $A' = A$, the set A' can be enlarged when the algorithm decides not to include in S the node v under consideration but to use $p(v)$ for v's activation, like in the case **(c)** above. In the rest of the section, we prove the following Theorem 3.

Theorem 3. *Algorithm* (λ, A)***-TSS on Tree*** *computes, in time* $O(|V|)$*, an optimal solution for the* (λ, A)-Target Set Selection *problem on a tree.*

Time Complexity. The initialization (line 1-10) requires time $O(|V|)$. The order in which nodes have to be considered is determined using a BFS which requires time $O(|V|)$ on a tree. The forall (line 11) considers all the internal nodes: the algorithm analyzes each node v in time $O(|C(v)|)$. We notice that the computation in line 15 can be executed in $O(|C(v)|)$ by using an algorithm that solve the selection problem in linear time. Overall the complexity of the algorithm is $O(|V|) + \sum_{v \in V} O(|C(v)|) = O(|V|)$.

Lemma 1. *Algorithm* (λ, A)**-TSS on Tree** *outputs a solution for the* (λ, A)-Target Set Selection *problem on* $T = (V, E)$.

Let $T(r) = (V, E)$ be a tree rooted at a $r \in V$, and let $X \subseteq V$ be a target set such that $Active[X, \lambda] \supseteq A$. Let $T(v)$ be the subtree of $T(r)$ rooted at a node v. Henceforth let $\mathsf{Active}[X, i, T(v)]$ be the set of nodes that is active at round i by targeting $X \cap T(v)$ in the subtree $T(v)$. Notice that while X is a target set for $T(r)$ this not necessarily means that $X \cap T(v)$ is a target set for $T(v)$.

$$\mathrm{round}_X(v) = \begin{cases} i & \text{if } v \in \mathsf{Active}[X, i, T(v)] \setminus \mathsf{Active}[X, i-1, T(v)] \\ 0 & \text{if } v \in X \\ \infty & \text{otherwise} \end{cases}$$

$$\mathrm{path}_X(v) = \begin{cases} 0 & \text{if } v \in A \text{ is a leaf AND } v \notin X \\ i & \text{if } (\mathrm{maxPath}_X(v) = i < \lambda) \\ & \quad \text{AND } (|\mathsf{Active}[X, \lambda - i - 1, T(v)] \cap C(v)| = t(v) - 1) \\ & \quad \text{AND } (v \in A \text{ OR } \mathrm{maxPath}_X(v) > 0) \text{ AND } (v \notin X) \end{cases}$$

$$\mathrm{maxPath}_X(v) = \begin{cases} 1 + \max_{u \in C(v)} \mathrm{path}_X(u) & \text{if } v \text{ is an internal node} \\ 0 & \text{if } v \text{ is a leaf.} \end{cases}$$

Let X be a target set solution (i.e., $\mathsf{Active}[X, \lambda] \supseteq A$). For an edge (v, u) we say that v activates u and write $v \to u$ if $v \in \mathsf{Active}[X, i-1]$ and $u \in \mathsf{Active}[X, i] \setminus \mathsf{Active}[X, i-1]$, for some $1 \leq i \leq \lambda$. An activation path $v \rightsquigarrow u$ from v to u is a path in T such that $v = x_0 \to x_1 \to \ldots \to x_k = u$ with $x_j \in \mathsf{Active}[X, i_j] \setminus \mathsf{Active}[X, i_j - 1]$ for $0 \leq i_1 < i_2 < \ldots < i_k \leq \lambda$. In other words x_i is activated before x_{i+1}, for $i = 0, \ldots, k-1$.

Lemma 2. *Let* X *be a target set solutions (i.e.,* $\mathsf{Active}[X, \lambda] \supseteq A$*) and* $v \in V$*. If* $\mathrm{maxPath}_X(v) = i$ *then there is an activation path of length* i *in* $T(v)$ *starting at* v *and ending at a node* $u \in A$*.*

Algorithm 1. (λ, A)-**TSS on Trees**

Input: A tree $T = (V, E)$, thresholds function $t : V \to \mathbb{N}$, a latency bound $\lambda \in \mathbb{N}$ and a set to be activated $A \subseteq V$.
Output: $S \subseteq V$ of minimum size such that $A \subseteq Active[S, \lambda]$.

```
1  S = ∅;
2  A' = A;
3  Fix a root r ∈ V                          // T(r) denotes the tree T rooted at r
4  forall v in the set of T(r) leaves do
5      round(v) = ∞
6      if v ∈ A' then                        // v belongs to the set of nodes to be activated
7          A' = A' ∪ {p(v)}                  // p(v) denotes v's parent
8          path(v) = 0
9      else
10         path(v) = −1
11 forall v in the set of T(r) internal nodes, listed in reverse order with respect
   to the time they are visited by a breadth-first traversal from r do
12     maxPath(v) = 1 + max_{u∈C(v)} path(u)        // C(v) is the set of v'children
13     Act(v) = {u ∈ C(v) | round(u) < λ − maxPath(v)}
14     path(v) = −1
15     round(v) = 1 + min^{t(v)}{round(u) | u ∈ C(v)}
16     if v ∈ A' then                              // v has to be activated
17         if v ≠ r then
18             switch do
19                 case (|Act(v)|≤t(v)−2) OR (maxPath(v) = λ)
20                     S = S ∪ {v}                 // v has to be in the target set
21                     round(v) = 0
22                 case (|Act(v)| = t(v)−1) AND (maxPath(v) < λ)
23                     A' = A' ∪ {p(v)}  // v will be activated thanks to its parent
                       p(v)
24                     path(v) = maxPath(v)
25         else                                    // v is the root
26             if (|Act(v)| ≤ t(v)−1) then
27                 S = S ∪ {v}
28                 round(v) = 0
29 return (S)
```

Lemma 3. *If* $X = S$ *then for each node* $v \in V$, $\text{round}(v) = \text{round}_S(v)$, $\text{path}(v) = \text{path}_S(v)$ *and* $\text{Act}(v) = \mathsf{Active}[S, \lambda - \text{maxPath}_S(v) - 1, T(v)] \cap C(v)$.

Let X and Y be two target set solutions and $v \in V$. The following properties hold:

Property 4. *If* $\text{round}_X(v) > \text{round}_Y(v)$ *and* $v \notin Y$ *then there exists* $u \in C(v)$ *such that* $\text{round}_X(u) > \text{round}_Y(u)$.

Property 5. *If* $v \notin X$ *then* $|\mathsf{Active}[X, \lambda - \text{maxPath}_X(v) - 1, T(v)] \cap C(v)| > t(v) - 2$ *AND* $\text{maxPath}_X(v) < \lambda$.

Lemma 6. *Algorithm* (λ, A)***-TSS on Tree*** *outputs an optimal solution for the* (λ, A)-Target Set Selection *problem on* $T = (V, E)$.

Proof. Let S and O be respectively the solutions found by the algorithm (λ, A)**-TSS on Tree** and an optimal solution. For each $v \in V$ let $S(v) = S \cap T(v)$ (resp. $O(v) = O \cap T(v)$) be the set of target nodes in S (resp. O) which belong to $T(v)$. Let $s(u) = |S(u)|$ and $o(u) = |O(u)|$ be the cardinality of such sets. We will use the following claim whose proof is omitted due to space constraints.

Claim 7. *If either* $\text{path}_S(v) > \text{path}_O(v)$ *or* $\text{round}_S(v) > \text{round}_O(v)$ *then* $s(v) < o(v)$.

We show by induction on the height of the node that $s(v) \leq o(v)$, for each $v \in V$.

The inequality trivially holds when v is a leaf. Since our algorithm does not target any leaf (i.e. $v \notin S$), we have $s(v) = 0$. Since we have $o(v) = 0$ or $o(v) = 1$ according to whether v belongs to the optimal solution, the inequality is always satisfied.

Now consider any internal node v. By induction, $s(u) \leq o(u)$ for each $u \in C(v)$; hence

$$\sum_{u \in C(v)} s(u) \leq \sum_{u \in C(v)} o(u). \tag{1}$$

It is not hard to see that if $v \in O$ by (1) we immediately have $s(v) \leq o(v)$. The same result follows from (1) for the case where v is neither in O nor in S.

We are left with the case $v \in S$ and $v \notin O$ In this case eq. (1) only gives $s(v) - 1 \leq o(v)$. In order to obtain the desired result we need to find a child u of v such that $s(u) < o(u)$. We distinguish the following two cases:

Case ($v \notin A$ and $\text{maxPath}_O(v) = 0$): Since $v \in S$ we have $v \in A'$ (that is $v \in A$ or $\text{maxPath}_S(v) > 0$). Since $v \in A' \setminus A$ then $\text{maxPath}_S(v) > 0$ and there is a children u of v such that $\text{path}_S(u) \geq 0 > -1 = \text{path}_O(u) = -1$ and we have found the desired vertex because by the Claim above we have $s(u) < o(u)$.

Case ($v \in A$ or $\text{maxPath}_O(v) > 0$): Since $v \in S$ we have that either $|\text{Act}(v)| = |\mathsf{Active}[S, \lambda - i - 1, T(v)] \cap C(v)| \leq t(v) - 2$ or where $i = \text{maxPath}_S(v)$. We consider the two subcases separately:

Subcase ($\text{maxPath}_S(v) = \lambda$)**:** Since $v \notin O$, by Property 5, we have $\text{maxPath}_O(v) < \lambda$. Hence, there is a vertex $u \in C(v)$ such that $\text{path}_S(u) = \lambda - 1 > \text{path}_O(u)$ and we have found the desired vertex because, by the Claim we have $o(u) > s(u)$.

Subcase ($|\mathsf{Active}[S, \lambda - \text{maxPath}_S(v) - 1, T(v)] \cap C(v)| \leq t(v) - 2$)**:** Since $v \notin O$ by Property 5 we have $|\mathsf{Active}[O, \lambda - j - 1, T(v)] \cap C(v)| > |\mathsf{Active}[S, \lambda - i - 1, T(v)] \cap C(v)|$ where $i = \text{maxPath}_S(v)$ and $j = \text{maxPath}_O(v)$. If $i > j$ (i.e., $1 + \max_{u \in C(v)} \text{path}_S(u) > 1 + \max_{u \in C(v)} \text{path}_O(u)$) then there is a child u of v such that $\text{path}_S(u) > \text{path}_O(u)$ and by the Claim we have $o(u) > s(u)$. If $i = j$, then there exists a child $u \in C(v)$ such that, $u \in \mathsf{Active}[O, \lambda - j - 1, T(u)] \setminus \mathsf{Active}[O, \lambda - j - 2, T(u)]$ and $u \notin \mathsf{Active}[S, \lambda - i - 1, T(u)]$. Hence $\text{round}_O(u) < \text{round}_S(u)$. By the Claim we have $o(u) > s(u)$.

In all cases we have that there is $u \in C(v)$ with $s(u) < o(u)$. Hence $s(v) \leq o(v)$.

References

1. Ackerman, E., Ben-Zwi, O., Wolfovitz, G.: Combinatorial model and bounds for target set selection. Theoretical Computer Science 411, 4017–4022 (2010)
2. Ben-Zwi, O., Hermelin, D., Lokshtanov, D., Newman, I.: Treewidth governs the complexity of target set selection. Discrete Optimization 8, 87–96 (2011)
3. Brunetti, S., Cordasco, G., Gargano, L., Lodi, E., Quattrociocchi, W.: Minimum Weight Dynamo and Fast Opinion Spreading. In: Golumbic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) WG 2012. LNCS, vol. 7551, pp. 249–261. Springer, Heidelberg (2012)
4. Chen, N.: On the approximability of influence in social networks. SIAM J. Discrete Math. 23, 1400–1415 (2009)
5. Chopin, M., Nichterlein, A., Niedermeier, R., Weller, M.: Constant Thresholds Can Make Target Set Selection Tractable. In: Even, G., Rawitz, D. (eds.) MedAlg 2012. LNCS, vol. 7659, pp. 120–133. Springer, Heidelberg (2012)
6. Cicalese, F., Milanič, M., Vaccaro, U.: On the approximability and exact algorithms for vector domination and related problems in graphs. Disc. Appl. Math. 161, 750–767 (2013)
7. Cicalese, F., Cordasco, G., Gargano, L., Milanič, M., Vaccaro, U.: Latency-Bounded Target Set Selection in Social Networks. ArXiV:1303.6785 (2013)
8. Corneil, D.G., Rotics, U.: On the relationship between clique-width and treewidth. SIAM J. Comput. 34, 825–847 (2005)
9. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Math. 101, 77–114 (2000)
10. Domingos, P., Richardson, M.: Mining the network value of customers. In: Proc. of the 7-th ACM SIGKDD International Conference, pp. 57–66 (2001)
11. Easley, D., Kleinberg, J.M.: Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press (2010)
12. Hlinený, P., Oum, S.-I.: Finding branch-decompositions and rank-decompositions. SIAM J. Comput. 38, 1012–1032 (2008)

13. Kempe, D., Kleinberg, J.M., Tardos, E.: Maximizing the spread of influence through a social network. In: Proc. of the 9-th ACM SIGKDD International Conference, pp. 137–146 (2003)
14. Kempe, D., Kleinberg, J.M., Tardos, É.: Influential Nodes in a Diffusion Model for Social Networks. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1127–1138. Springer, Heidelberg (2005)
15. Oum, S.-I., Seymour, P.: Approximating clique–width and branch–width. J. Combin. Theory Ser. B 96, 514–528 (2006)

Summary Data Structures for Massive Data

Graham Cormode

Department of Computer Science, University of Warwick, United Kingdom
graham@cormode.org

Abstract. Prompted by the need to compute holistic properties of increasingly large data sets, the notion of the "summary" data structure has emerged in recent years as an important concept. Summary structures can be built over large, distributed data, and provide guaranteed performance for a variety of data summarization tasks. Various types of summaries are known: summaries based on random sampling; summaries formed as linear sketches of the input data; and other summaries designed for a specific problem at hand.

1 Introduction

It is widely acknowledged in the business and scientific communities that "big data" holds both tremendous promise, and substantial challenges [10]. There is much potential for extracting useful intelligence and actionable information from the large quantities of data generated and captured by modern information processing systems. A chief problem presented by this scenario is the scale in terms of the so-called "three V's": volume, variety, and velocity. That is, big data challenges involve not only the sheer volume of the data, but the fact that it can represent a complex variety of entities and interactions between them, and new observations arrive, often across multiple locations, at high velocity.

Such sources of big data are becoming increasingly common, while the resources to deal with big data (chiefly, processor speed, fast memory and slower disk) are growing at a slower pace. The consequence of this trend is that we need more effort directed towards capturing and processing data in many critical applications. Careful planning and scalable architectures are needed to fulfill the requirements of analysis and information extraction on big data.

Some examples of applications that generate big data include:

Physical Data. The growing development of sensors and sensor deployments have led to settings where measurements of the physical world are available at very high dimensionality and at a great rate. Scientific measurements are the cutting edge of this trend. Astronomy data gathered from modern telescopes can easily generate terabytes of data in a single night. Aggregating large quantities of astronomical data provides a substantial big data challenge to support the study and discovery of new phenomena. Big data from particle physics experiments is also enormous: each experiment can generate many terabytes of readings, which can dwarf what is economically feasible to store for later comparison and investigation.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 78–86, 2013.

Medical Data. It is increasingly feasible to sequence entire genomes. A single human genome is not so large—it can be represented in under a gigabyte—but considering the entire genetic data of a large population represents a big data challenge. This may be accompanied by increasing growth in other forms of medical data, based on monitoring multiple vital signs for many patients at fine granularity. Collectively, this leads to the area of data-driven medicine, seeking better understanding of disease, and leading to new treatments and interventions, personalized for each individual patient.

Activity Data. Human activity data is increasingly being captured and stored. Online social networks record not just friendship relations but interactions, messages, photos and interests. Location data is also more available, due to mobile devices which can obtain GPS information. Other electronic activities, such as patterns of website visits, email messages and phone calls can be collected and analyzed. Collectively, this provides ever-larger collections of activity information. Service providers who can collect this data seek to make sense of it in order to identify patterns of behavior or signals of behavioral change, and opportunities for advertising and marketing.

Business Data. Businesses are increasingly able to capture more and complex data about their customers. Online stores can track millions of customers as they explore their site, and seek patterns in purchasing and interest, with the aim of providing better service, and anticipating future needs. The detail level of data is getting finer and finer. Previously, data would be limited to just the items purchased, but now extends to more detailed shopping and comparison activity, tracking the whole path to purchase.

Across all of these disparate settings, certain common themes emerge. The data in question is large, and growing. The applications seek to extract patterns, trends or descriptions of the data. Scalability and timeliness of response are vital in many of these applications.

In response to these needs, new computational paradigms are being adopted to deal with the challenge of big data. Large scale distributed computation is a central piece: the scope of the computation can exceed what is feasible on a single machine, and so clusters of machines work together in parallel. On top of these architectures, parallel algorithms are designed that can take the complex task and break it into independent pieces suitable for distribution over multiple machines.

A central challenge within any such system is how to compute and represent complex features of big data in a way that can be processed by many single machines in parallel. One answer is to be able to build and manipulate a compact summary of a large amount of data. This notion of a small summary is the subject of study of this article. The idea of a summary is a natural and familiar one. It should represent something large and complex in a compact fashion. Inevitably, a summary must dispense with some of the detail and nuance of the object which it is summarizing. However, it should also preserve some key features of the object in an accurate fashion.

There is no single summary which accurately captures all properties of a data set, even approximately. Thus, at the heart of the study of small summaries are the questions of *what should be preserved?* and *how accurately can it be preserved?*. The answer to the first question determines which of many different possible summary types may be appropriate, or indeed whether any compact summary even exists. The answer to the second question can determine the size and processing cost of working with the summary in question.

Another important question about summaries for big data is how they can be constructed and maintained as new data arrives. Given that it is typically not feasible to load all the data into memory on one machine, then we need summaries which can be constructed incrementally. That is, we seek summaries that can be built by observing each data item in turn, and updating the partial summary. Or, more strongly, we seek summaries such that summaries of different subsets of data built on different machines can be combined together to obtain a single summary that accurately represents the full data set.

Note that the notion of summarization is distinct from traditional ideas of *compression*. Lossless compression is concerned with identifying regularity and redundancy in datasets to provide a more compact exact representation of the data. This is done for the purpose of archiving, or reducing transmission time. However, in general, there is no guarantee of significant size reduction from compression. The compressed form is also typically difficult to utilize, and decompression is required in order to work with the data. In contrast, summarization is intended to provide a very significant reduction in the size of the data (sometimes several orders of magnitude), but does not promise to reconstruct the original data, only to capture certain key properties. Lossy compression methods fall in between, as they can provide guaranteed size reductions. They also aim to allow an approximate reconstruction of the original data with only small loss of fidelity based on some measure of loss: typically, human perception of multimedia data, such as audio or video. Summarization aims to provide only small loss of fidelity, but on other dimensions; summaries do not necessarily provide a way to make even an approximate reconstruction of the original input.

As a first example of summarization, consider a data set consisting of a large collection of temperature readings over time. A suitable summary might be to keep the sum of all the temperatures seen, and the count. From this summary, we can extract the average temperature. This summary is easy to update incrementally, and can also be combined with a corresponding summary by computing the overall sum and count. A different summary retains only the maximum and minimum temperature observed so far. From this, we can extract the range of temperatures observed. This too is straightforward to maintain under updates, and to merge across multiple subsets. However, neither summary is good at retrieving the median temperature, or some other properties of the statistical distribution of temperatures. Instead, more complex summaries and maintenance procedures are required.

2 Operations on Summaries

While different summaries capture different functions and properties of the data, we abstract a set of four basic operations that summaries should support. These basic operations are Initialize, Update, Merge and Query.

Initialize. The Initialize operation for a summary is to initialize a new instance of the summary. Typically, this is quite simple, just creating empty data structures for the summary to use. For summaries that use randomization, this can also involve drawing the random values that will be used throughout the operation of the summary.

Update. The Update operation takes a new data item, and updates the summary to reflect this. The time to do this Update should be quite fast, since we want to process a large input. Ideally, this is faster than reading the whole summary. Since Update takes a single item at a time, the summary can process a stream of items one at a time, and only retain the current state of the summary at each step.

Merge. When faced with a large amount of data to summarize, we would like to distribute the computation over multiple machines. Performing a sequence of Update operations does not guarantee that we can parallelize the action of the summary, so we also need the ability to Merge together a pair of summaries to obtain a summary of the union of their inputs. This is possible in the majority of cases, although a few summaries only provide an Update operation and not a Merge. Merge is often a generalization of Update: applying Merge when one of the input summaries consists of just a single item will reduce to the Update operation. In general a Merge operation is slower than Update, since it requires reading through both summaries in full.

Query. At various points we want to use the summary to learn something about the data that is summarized. We abstract this as Query, with the understanding that the meaning of Query depends on the summary: different summaries capture different properties of the data. In some cases, Query takes parameters, while for other summaries, there is a single Query operation. Some summaries can be used to answer several different types of query. In this presentation, we pick one primary query to answer with the Query operation.

3 Three Examples of Summaries

This section describes three examples of different constructions of summaries, and some of their applications and uses. The aim of the presentation is to convey the basic ideas; for further details the interested reader is directed to more detailed surveys [8,18].

3.1 Samples

The notion of random sampling is a familiar and convenient one. Given a large number of data items, we randomly pick a subset, so that the probability of any subset being picked as the sample is uniform. Then, we can estimate the answer to a query by evaluating the query over the sample, and scaling appropriately. For certain queries, strong error guarantees are known for this process.

Working with a sample of size k can fit neatly into the summary framework. To INITIALIZE a sample, we create an empty set. To UPDATE a sample, we determine (randomly) whether to include the new item in the sample, and to MERGE two samples, we shall determine which subset of items to maintain. One natural way to accomplish these steps is to maintain the "weight" of each sample, as the number of input items represented by the sample. To MERGE two samples of weight n_1 and n_2 respectively, we pick items from the first with probability $n_1/(n_1 + n_2)$, and from the second with probability $n_2/(n_1 + n_2)$. An UPDATE operation is then a MERGE where one of the input samples has size 1. Lastly, to QUERY a sample to estimate the fraction of input items satisfying a property P, we can report the fraction of items in the sample satisfying P.

Tools such as the Chernoff-Hoeffding inequality [16] can be brought to bear to analyze the accuracy of answering a query from samples. First, we observe that the result of a combination of MERGE and UPDATE operations generates a uniform random sample of size k—the probability of any item being included in the sample is uniform, and independent of the inclusion probabilities of other items. Then we can consider the random variable corresponding to the fraction of items in the sample that satisfy property P. Let X_i be the random variable indicating if the ith sampled item satisfies P: we have $\Pr[X_i = 1] = p$, where p is the global proportion we are trying to estimate. Applying the Chernoff-Hoeffding inequality, we have

$$\Pr[|\frac{1}{k}\sum_{i=1}^{k} X_i - p| > \varepsilon] \leq 2\exp(-2\varepsilon^2 k)$$

Consequently, setting the sample size $k = O(\frac{1}{\varepsilon^2}\ln 1/\delta)$ is sufficient to ensure that the error is at most ε with probability at least $1 - \delta$.

Samples: pros and cons. Due to their flexibility and simplicity, samples have been used in a variety of applications. For example, many routers in the internet sample information about flows for network traffic monitoring [17]. It is often convenient to work with samples, since we just have to apply the desired computation on the sample instead of on the full data (in contrast, other summaries require different, complex procedures to be performed on the summary data). However, they have some limitations. Samples do not work well for problems not based on raw counts—for example, samples are not helpful for estimating the number of distinct items [5]. The accuracy bounds in terms of $O(1/\varepsilon^2)$ can be high in some applications where ε is very small. In some cases, other summary techniques achieve a smaller cost, proportional to $O(1/\varepsilon)$, which can make a big difference in practice.

3.2 Sketches

The term 'sketches' refers generally to a class of summary that can be expressed as a linear transformation of the input data. That is, if we can express the input as a vector, then the sketch is the result of multiplying the vector by some matrix S. We are most interested in cases where this matrix S has a compact, implicit representation, so that we do not have to explicitly store it in full. A simple example is the Count-Min sketch, which summarizes a large vector so that individual entries of the vector can be estimated [9]. Entries of the vector are mapped down to a smaller domain by a hash function, and the sum of entries mapping to each location are maintained; this is repeated for a small number of different hash functions with the same range.

Using the terminology of summaries, to INITIALIZE a Count-Min sketch, we first determine the parameters: d, the number of hash functions to apply, and w, the range of the hash functions. We then create an array C of size $d \times w$ to summarize the data, and pick d hash functions h_j. To UPDATE the sketch with a positive increment of u to vector location i, we map i into C by the hash functions, and compute $C[j, h_j(i)] \leftarrow C[j, h_j(i)] + u$ for all $1 \leq j \leq d$. This is a linear update function. To MERGE two sketches that share the same parameters (d, w and h_j), we sum the two arrays. Lastly, to QUERY the estimated value of a location i, we find $\min_j C[j, h_j(i)]$, the smallest value in the array of the locations where location i was mapped to: assuming positive weights for all item, this indicates the estimate with the least error from colliding items.

From this description it is reasonably straightforward to see that the sketch is a linear transform of its input vector x, so it is the same, irrespective of the ordering of the UPDATE and MERGE operations. It is less immediate to see why the sketch should be accurate, given that w is typically chosen to be much smaller than the size of the vector being summarized. The intuition behind the sketch is that the hash functions work to spread out the different items, so on average the error in the estimates cannot be too high. Then the use of different hash functions ensures that the chance of getting an estimate that is much more inaccurate than average is quite small. In particular, in any given row, the estimate for i is expected to have error proportional to $\|x\|_1/w$, where $\|x\|_1$ denotes the L_1 norm of the input vector x. Consequently, there is constant probability ($\frac{1}{2}$) that the error is at most $2\|x\|_1/w$, by the Markov inequality. Taking the minimum of d repetitions amplifies this to a probability of 2^{-d}. Equivalently, to guarantee an error of at most $\varepsilon\|x\|_1$ with probability $1 - \delta$, we set the parameters to $w = 2/\varepsilon$ and $d = \log_2 1/\delta$.

Sketches: pros and cons. Sketch techniques have been used widely for log data analysis in large systems, such as Sawzall [19] and CMON [22]. Their linearity offers extra flexibility: it means that we can obtain a sketch of the sums, differences and scaled versions of inputs by applying the corresponding transformation to the sketch data structure. This makes them useful for applying linear forecasting models to large data [7]. They have also been used as the basis for building machine learning models over very large feature domains [23]. Sketch constructions

are known for a variety of input types, including count distinct and set sizes [11], set membership [4], vector operations such as dot-product [3] and matrix computations [6]. However, they have their limitations: some sketch structures can be large and very slow to update, limiting their practicality. Sketch ideas do not naturally apply to settings with unbounded domains, such as sets of strings. Lastly, they do not support arbitrary complex operations: one sketch addresses only one or two defined problems, and its is hard to compose sketches to address complex aggregates.

3.3 Special-Purpose Summaries

Other summaries take the form of various special-purpose structures, which capture various aspects of the data. The properties of these vary instance by instance, and require special analysis to accompany their implementation. However, they can offer very good performance in terms of their space and time costs.

A simple example is the MG summary, named for the initials of its inventors. This was originally proposed in the context of streams of data [15], and later generalized to allow merging of summaries [1] for summarizing a collection of weights. The MG summary stores a collection of pairs: items drawn from the input x and associated weights w_x. To INITIALIZE an empty MG summary, we create an empty set of tuples, with space for k pairs. The MERGE of two MG summaries takes two summaries constructed using the same parameter k. We first merge the component tuples in the natural way. If x is stored in both summaries, its merged weight is the sum of the weights in each input summary. If x is stored in only one of the summaries, it is also placed in the merged summary with the same weight. We then sort the tuples by weight, and find the $k+1$st largest weight, w_{k+1}. This weight is subtracted from *all* tuples, and only those with positive weight are retained. At most k tuples can now have weight above zero: the tuple with $k+1$st largest weight, and all tuples with smaller weight, will now have weight 0 or below, and so can be discarded from the summary. The UPDATE procedure is the special case of MERGE where one of the summaries contains just a single item. To QUERY for the estimated weight of an item x, we look up whether x is stored in the summary. If so, QUERY reports the associated weight w_x as the estimate, otherwise the weight is assumed to be 0.

Comparing the approximate answer given by QUERY, and the true weight of x (the sum of all weights associated with x in the input), the approximate answer is never more than the true answer. This is because the weight associated with x in the summary is the sum of all weights for x in the input, less the various decreases due to MERGE and UPDATE operations. The MG summary also ensures that this estimated weight is not more than εW below the true answer, where W is the sum of all input weights.

Special-purpose summaries: pros and cons. Summaries such as the MG summary have been used for quite a wide variety of purposes, such as web clickstream mining [14]. They tend to work very well for their target domain but, as indicated

by their name, do not obviously generalize to other problems. Other special-purpose summaries include summaries for order statistics of distributions, such as the Greenwald-Khanna summary [12], and the q-digest [21]; ε-approximations and ε-nets for geometric data [13]; and summaries for graph distances and connectivity [2].

4 Summaries Summary

As noted in the preceding discussion, there are numerous, growing applications for summary data structures, from scaling up machine learning and data mining, to applications in privacy and security (e.g., [20]). For more information, see various tutorials and surveys [8]. Many directions for further research are open.

Natural questions surround improving on the current summaries, either in terms of their parameters (reducing the space and update time), or extending their generality (allowing them to be applied in more general update models). While summaries for computing over data that can be thought of in terms of vectors are by now quite well-understood, the situation for other forms of data—in particular, matrices and graphs—is less complete. New styles of summary are required in the context of verification of computation: these are summaries that allow a "verifier" to check the computation performed by a more powerful "prover" on a stream of data. A list of specific open problems is maintained at `http://sublinear.info/`.

References

1. Agarwal, P., Cormode, G., Huang, Z., Phillips, J., Wei, Z., Yi, K.: Mergeable summaries. In: ACM Principles of Database Systems (2012)
2. Ahn, K.J., Guha, S., McGregor, A.: Analyzing graph structure via linear measurements. In: ACM-SIAM Symposium on Discrete Algorithms (2012)
3. Alon, N., Gibbons, P., Matias, Y., Szegedy, M.: Tracking join and self-join sizes in limited storage. In: ACM Principles of Database Systems, pp. 10–20 (1999)
4. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
5. Charikar, M., Chaudhuri, S., Motwani, R., Narasayya, V.R.: Towards estimation error guarantees for distinct values. In: ACM Principles of Database Systems, pp. 268–279 (2000)
6. Clarkson, K.L., Woodruff, D.P.: Numerical linear algebra in the streaming model. In: ACM Symposium on Theory of Computing, pp. 205–214 (2009)
7. Cormode, G., Garofalakis, M.: Sketching streams through the net: Distributed approximate query tracking. In: International Conference on Very Large Data Bases (2005)
8. Cormode, G., Garofalakis, M., Haas, P., Jermaine, C.: Synposes for Massive Data: Samples, Histograms, Wavelets and Sketches. Foundations and Trends in Databases. NOW publishers (2012)
9. Cormode, G., Muthukrishnan, S.: An improved data stream summary: The Count-Min sketch and its applications. Journal of Algorithms 55(1), 58–75 (2005)

10. Cukier, K.: Data, data everywhere. The Economist (February 2010)
11. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for database applications. Journal of Computer and System Sciences 31, 182–209 (1985)
12. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: ACM SIGMOD International Conference on Management of Data (2001)
13. Har-Peled, S., Mazumdar, S.: Coresets for k-means and k-median clustering and their applications. In: ACM Symposium on Theory of Computing, pp. 291–300 (2004)
14. Metwally, A., Agrawal, D.P., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 398–412. Springer, Heidelberg (2005)
15. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming 2, 143–152 (1982)
16. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
17. Cisco NetFlow, More details at `http://www.cisco.com/warp/public/732/Tech/netflow/`
18. Olken, F.: Random Sampling from Databases. PhD thesis, Berkeley (1997)
19. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the data: Parallel analysis with sawzall. Dynamic Grids and Worldwide Computing 13(4), 277–298 (2005)
20. Schechter, S., Herley, C., Mitzenmacher, M.: Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In: Proceedings of HotNets (2010)
21. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: New aggregation techniques for sensor networks. In: ACM SenSys (2004)
22. To, K., Ye, T., Bhattacharyya, S.: CMON: A general purpose continuous IP backbone traffic analysis platform. Technical Report RR04-ATL-110309, Sprint ATL (2004)
23. Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A.J., Attenberg, J.: Feature hashing for large scale multitask learning. In: International Conference on Machine Learning (ICML) (2009)

Determinant versus Permanent: Salvation via Generalization?

Nicolas de Rugy-Altherre

Institut de Mathématiques de Jussieu, Université Paris Diderot, Sorbonne Paris Cité, UMR 7586 CNRS, 75205 Paris, France
nderugy@math.univ-paris-diderot.fr

Abstract. The fermionant $\mathrm{Perm}_n^k(\bar{x}) = \sum_{\sigma \in S_n} (-k)^{c(\pi)} \prod_{i=1}^n x_{i,j}$ can be seen as a generalization of both the permanent (for $k = -1$) and the determinant (for $k = 1$). We demonstrate that it is VNP-complete for any rational $k \neq 1$. Furthermore it is $\#P$-complete for the same values of k. The immanant is also a generalization of the permanent (for a Young diagram with a single line) and of the determinant (when the Young diagram is a column). We demonstrate that the immanant of any family of Young diagrams with bounded width and at least n^ϵ boxes at the right of the first column is VNP-complete.

1 Introduction

In algebraic complexity (more specifically Valiant's model[2]) one of the main question is to know whether VP = VNP or not. Answering this is considered to be a very good step towards the resolution of $P = NP$. This question is very close to the question per vs. det, where we ask if the permanent can be computed in polynomial time in the size of the matrix, as is the determinant.

The main idea of this paper is to find a generalization of both the permanent and the determinant in order to study exactly where the difference between them lies. A generalization is here understood as a parameter, let us say t, and a function $f(t, \bar{x})$ such that for example $f(0, \bar{x}) = \det(\bar{x})$ and $f(1, \bar{x}) = \mathrm{per}(\bar{x})$. If we have a complete classification of the complexity of $f(t, \bar{x})$ for any t (with t fixed), we should be able to see where we step from VP to VNP and maybe understand a little bit more why the permanent is hard and not the determinant.

Here we study two different generalizations. First the fermionant, secondly the immanant. The fermionant was introduced by Chandrasekharan and Wiese [3] in 2011 in a context of quantum physics. It is defined with a real parameter k such that for $k = 1$ it is the determinant and for $k = -1$ it is the permanent. Mertens and Moore [7] have demonstrated its hardness for $k \geq 3$ (and with a weaker hardness for $k = 2$), in the framework of counting complexity.

Likewise, but in a different framework and with a complete different proof, we demonstrate the hardness of the fermionant seen as a polynomial for any rational $k \neq 1$ (and of course for $k \neq 0$). This give a interesting point of view on where the hardness of the permanent lies. We also get a bonus: we use a

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 87–96, 2013.

technique developed by Valiant to demonstrate the hardness of the fermionant in the counting complexity framework for $k \neq 1$. We thus extend the results of Mertens and Moore [7], in particular to the case $k = 2$, which is, from what I understand, the most interesting case for physicists.

The second generalization is more classical and comes from the field of group representation. It is the immanant, introduced by Littlewood [6] in 1940. Immanants are families of polynomials indexed by Young diagrams. If the Young diagrams are a single column with n boxes, the immanant is the determinant. At the opposite end, if it is a single line of n boxes, the immanant is the permanent. The main question is: for which Young diagrams do we step from VP to VNP?

We know that if there are only a finite number of boxes on the right of the first column, the immanant is still in VP (cf [2]). On the other hand, a few hardness results have been found, fundamentally for Young diagrams in which the permanent is hidden. For example, the hook (a line of n boxes and a column of any number of boxes) and the rectangle (any number of lines each with n boxes) are hard (cf [2]), or more generally if the maximal difference between the size of two consecutive lines is as big as a power of n (cf [1]).

Here we shall demonstrate that for Young diagrams with only two columns, each with n boxes, the immanant is hard, which was an open question (cf [2] Problem 7.1). As each line of these Young diagrams has length no more than two, the permanent is not hidden in there. More generally for any family of Young diagrams with a bounded number of columns and with at least n^ϵ boxes at the right of the first column, the immanant is hard. It has been conjecture that it is still hard if we remove the bounded condition(cf [7] for example).

For a complete classification of the immanant in algebraic complexity, one "just" has to determine the complexity of the ziggurat: the Young diagrams where the first line has n boxes, the second $n-1$, the third $n-2$ etc. and the last 1 box. This immanant is most probably also hard. The complexity of the immanant with a logarithmic number of boxes to the right of the first column is also unknown.

2 Definitions

We work within Valiant's algebraic framework. Here is a brief introduction to this complexity theory. For a more complete overview, see [2].

An *arithmetic circuit* over $\mathbb{Q}$ is a labeled directed acyclic connected graph with vertices of indegree 0 or 2 and only one sink. The vertices with indegree 0 are called *input gates* and are labeled with variables or constants from $\mathbb{Q}$. The vertices with indegree 2 are called *computation gates* and are labeled with $\times$ or $+$. The sink of the circuit is called the *output gate*.

The polynomial computed by a gate of an arithmetic circuit is defined by induction: an input gate computes its label; a computation gate computes the product or the sum of its children's values. The polynomial computed by an arithmetic circuit is the polynomial computed by the sink of the circuit.

A *p-family* is a sequence (f_n) of polynomials such that the number of variables as well as the degree of f_n is polynomially bounded in n. The *complexity* $L(f)$ of a polynomial $f \in \mathbb{Q}[x_1, \ldots, x_n]$ is the minimal number of computational gates of an arithmetic circuit computing f from variables $x_1, \ldots, x_n$ and constants in $\mathbb{Q}$.

Two of the main classes in this theory are: the analog of P, VP, which contains of every p-family (f_n) such that $L(f_n)$ is a function polynomially bounded in n; and the analog of NP, VNP. A p-family (f_n) is in VNP iff there exists a VP family (g_n) such that for all n,

$$f_n(x_1, \ldots, x_n) = \sum_{\bar{\epsilon} \in \{0,1\}^n} g_n(x_1, \ldots, x_n, \epsilon_1, \ldots, \epsilon_n)$$

As in most complexity theory we have a notion of reduction, the c-reduction: the oracle complexity $L^g(f)$ of a polynomial f with oracle access to g is the minimum number of computation gates and evaluations of g over previously computed values that are sufficient to compute f from the variables $x_1, \ldots x_n$ and constants from $\mathbb{Q}$. A p-family (f_n) *c-reduces* to (g_n) if there exists a polynomially bounded function p such that $L^{g_{p(n)}}(f_n)$ is a polynomially bounded function.

VNP is closed under c-reductions (See [8] for an idea of the proof). However this reduction does not distinguish lower classes. For example, 0 is VP-complete for c-reductions. In this paper we shall demonstrate hardness results, a smallest notion of reduction (as projection) is thus not needed.

The determinant is in VP. The permanent is VNP-complete for c-reductions ([2]).

3 The Fermionant

Let A be an $n \times n$ matrix. The *fermionant* of A, with parameter k is defined as

$$\mathrm{Ferm}^k A = (-1)^n \sum_{\pi \in S_n} (-k)^{c(\pi)} \prod_{i=1}^{n} A_{i,\pi(i)}$$

where S_n denotes the symmetric group of n objects and, for any permutation $\pi \in S_n$, $c(\pi)$ denotes the number of cycles of π. To study the complexity of such a function, we work within the algebraic complexity framework. The algebraic equivalent of the fermionant is the polynomial obtain where we compute the fermionant on the matrix $(x_{i,j})_{1 \le i,j \le n}$. If we write Ferm^k the p-family $(\mathrm{Ferm}^k_n)_{n \in \mathbb{N}}$, we have a complete classification of the algebraic complexity of those polynomials.

Theorem 1. *Let k be a rational.*

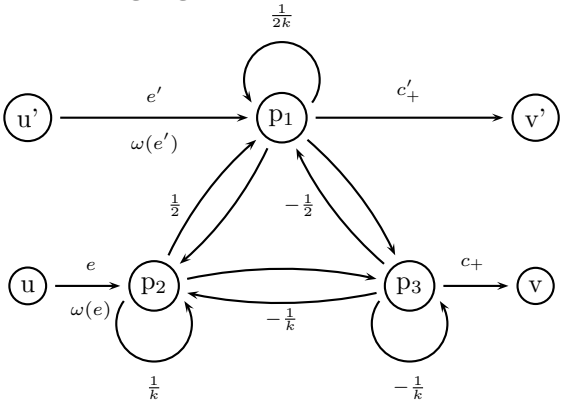
- *$Ferm^0 = 0$.*
- *$Ferm^1$ is in* VP
- *for other values of k $Ferm^k$ is* VNP*-complete for c-reductions.*

Similarly to the permanent we can see the fermionant as a computation on a graph G with n vertices and the edge between the vertices i and j is labeled with the variable $x_{i,j}$. A permutation $\pi \in S_n$ can be seen as a cycle cover on this graph. A *cycle cover* of G is a subset of its edges that covers all vertices of G and that form cycles. The weight of a cycle cover π is $\omega(\pi) = \prod_{e\in\pi} x_e$ and we write $c(\pi)$ its number of cycles, then

$$\mathrm{Ferm}^k(\bar{x}) = \sum_{\pi\in CC(G)} (-k)^{c(\pi)} \prod_{e\in\pi} x_e$$

where $CC(G)$ is the set of all cycle covers of G. We shall use a so call iff-gadget, which is the labeled graph draw above. The idea of this gadget is when placed between two edges e and e' on G, any cycle cover containing exactly one of the edges e and e' will not contribute to the fermionant computed on the resulting graph.

iff-gadget for the fermionant



Lemma 1. *Let G be a graph with n vertices and $(e_1^i, e_2^i)_{1\leq i\leq l}$ be a set of pairs of edges of G such that no two edges in this set are equal. Let G' be the same graph but where we place an iff-gadget between every pair (e_1^i, e_2^i). Let π be a cycle cover of G, $\Pi(\pi)$ be the set of cycle covers of G' that match π on $E(G)$.*

- *If there is a pair (e_1^i, e_2^i) of edges such that $e_1^i \in \pi$ and $e_2^i \notin \pi$, or vice versa, then*
$$\sum_{\pi'\in\Pi(\pi)} (-k)^{c(\pi')}\omega(\pi') = 0$$
- *Else, let $d(\pi)$ be the number of pair (e_1^i, e_2^i) of edges such that $e_1^i \notin \pi$ and $e_2^i \notin \pi$. Then*
$$\sum_{\pi'\in\Pi(\pi)} (-k)^{c(\pi')}\omega(\pi') = \left(\frac{1}{2}(1-k)\right)^{d(\pi)} (-k)^{c(\pi)}\omega(\pi)$$

The proof is not given here, it is in color in the full version of the paper. Now here is the main tool of our demonstration, that allows us to interpolate the fermionant and compute the permanent.

Lemma 2. *Let G be a graph with n vertices. We make l copies of G and name them $G_1, \ldots, G_l$. Let $\tilde{F}^l$ be the disjoint union of those copies in which we label the edges of G_1 with the same weight as those of G and the edges of G_i for $i \geq 2$ with 1. If e is an edge of G, we call e_i the corresponding edge in G_i. We name F^l the graph $\tilde{F}^l$ where for any edge $e \in E(G)$ and any $1 \leq i \leq l$, we have placed an iff-gadget between e_i and e_{i+1}. Let π be a cycle cover of G and $\Pi(\pi)$ be the set of cycle covers of F^l that match π on $E(G_1)$. Then*

$$\sum_{\pi' \in \Pi(\pi)} (-k)^{c(\pi')} \omega(\pi') = \left(\frac{1}{2}(1-k)\right)^{(|E(G)|-n)(l-1)} (-k)^{l \times c(\pi)}\, \omega(\pi)$$

Proof. The idea is, with the help of the iff-gadget, to copy a cycle cover from G_1 to every other copies of G, without changing the weight of this cycle cover, just multiplying the number of cycles. The demonstration is by induction on l.

If $l = 2$, then we simultaneously add $|E(G)|$ iff-gadgets, but only one on each edge. By design, a cycle cover π on G_1 is repeated on G_2 (i.e., if e_1 is in π then e_2 is also in π as there is a iff-gadget between e_1 and e_2. see Lemma 1). The edges of G_2 are labeled with 1 and therefore do not contribute to the weight of the cycle cover. The number of cycles of $\pi' \in \Pi(\pi)$ is twice the number of cycles of π. There is $|E(G)|$ iff-gadgets in F^2. A cycle cover of G passes through n edges and therefore activates exactly n iff-gadgets. The other iff-gadgets are not activated and thus each contribute $\frac{1}{2}(1-k)$ to the sum.

Suppose the lemma true for $l-1$ copies. Let F^{l-1} be the disjoint union of $l-1$ copies of G with iff-gadgets. We add a new copy G_l of G linked to F^{l-1} with iff-gadgets to obtain F^l. Let π be a cycle cover of G, $\Pi^l(\pi)$ the set of every cycle covers of F^l that match π on $E(G_1)$ and $\Pi^{l-1}(\pi)$ the same but on F^{l-1}. By induction,

$$\sum_{\pi' \in \Pi^{l-1}(\pi)} (-k)^{c(\pi')} \omega(\pi') = \left(\frac{1}{2}(1-k)\right)^{(|E(g)|-n)(l-2)} (-k)^{(l-1)\times c(\pi)} \omega(\pi)$$

Let $\hat{F}^l$ be the disjoint union of F^{l-1} and G_l. To obtain F^l from this graph, one has just to add a iff-gadget between every edge e_{l-1} and e_l. We can apply then Lemma 1 to this graph. If π'' is a cycle cover of $\hat{F}^l$ that match π on G_1, let $\Lambda(\pi'')$ be the set of cycle covers of F^l that match π'' on $E(\hat{F}^l)$. Then, if we call $d(\pi'')$ the number of pairs (e_{l-1}, e_l) that are not in π'',

$$\sum_{\lambda \in \Lambda(\pi'')} (-k)^{c(\lambda)} \omega(\lambda) = \left(\frac{1}{2}(1-k)\right)^{d(\pi'')} (-k)^{c(\pi'')} \omega(\pi'')$$

Let us study a little bit more π''. It is a cycle cover of two disjoint graphs, F^{l-1} and G_l. Therefore it is composed of two sub cycle covers: σ' a cycle cover of F^{l-1} which by induction is in a $\mathit{\Pi}^{l-1}(\pi)$ and a cycle cover λ of G_l. However, as every edge of G_l is linked with an iff-gadget to its image in G_{l-1} in F^l, the cycle cover π'' will contribute to the last sum if and only if it contain both e_{l-1} and e_l, or neither e_{l-1} and e_l. Thus, λ must be the copy of π in G_l, which we write λ_π and $c(\pi'') = c(\sigma) + c(\lambda_\pi) = c(\sigma) + c(\pi)$.

There are n edges in the last image G_l that are passed through by π''. Therefore, there are $(|E(G)| - n)$ iff-gadgets that are not activated by π'' (i.e., $d(\pi'') = |E(G)| - n$). Thus,

$$\begin{aligned}
\sum_{\pi' \in \mathit{\Pi}(\pi)} (-k)^{c(\pi')}\omega(\pi') &= \sum_{\pi'' \in \mathit{\Pi}^l(\pi)} \sum_{\lambda \in \mathit{\Lambda}(\pi'')} (-k)^{c(\lambda)}\omega(\lambda) \\
&= \sum_{\pi'' \in \mathit{\Pi}^l(\pi)} \left(\frac{1}{2}(1-k)\right)^{|E(G)|-n} (-k)^{c(\pi'')}\omega(\pi'') \\
&= \left(\frac{1}{2}(1-k)\right)^{|E(G)|-n} (-k)^{c(\lambda_\pi)} \sum_{\sigma \in \mathit{\Pi}^{l-1}(\pi)} (-k)^{c(\sigma)}\omega(\sigma) \\
&= \left(\frac{1}{2}(1-k)\right)^{(|E(G)-n)\times(l-1)} (-k)^{l \times c(\pi)}\omega(\pi)
\end{aligned}$$

Where $\mathit{\Pi}(\pi)$ is the set of cycle covers of F^l that match π on $E(G)$; $\mathit{\Pi}^l(\pi)$ the set of cycle covers of $\tilde{F}^l$ that match π on $E(G)$ and for $\pi'' \in \mathit{\Pi}^l(\pi)$, $\mathit{\Lambda}(\pi'')$ the set of cycle covers that match π'' on $E(\tilde{F}^l)$. We have $\mathit{\Pi}(\pi) = \bigcup_{\pi'' \in \mathit{\Pi}^l(\pi)} \mathit{\Lambda}(\pi'')$ which completes our demonstration.

□

Proof of Theorem 1. The first case is trivial. For the second, it is a well known result, as $\mathrm{Ferm}_n^1(\bar{x}) = \det_n(\bar{x})$. Now, let k be a rational different than 0 and 1.

Let us write $(P_l G)$ the graph obtained in the previous lemma, when we duplicate l times G and add iff-gadgets to repeat every cycle cover l times. We have seen that

$$\mathrm{Ferm}_{ln}^k(P_l G)(\bar{x}) = \sum_{\pi \in \mathrm{CC}(G)} (-k)^{l \times c(\pi)} \prod_{e \in \pi} \omega(e) \left(\frac{1}{2}(1-k)\right)^{(l-1)\times(|E(G)|-n)}$$

Let us write $c_m = \sum_{\pi \in \mathrm{CC}(G) | c(\pi) = m} \prod_{e \in \pi} \omega(e)$, $\alpha = \left(\frac{1}{2}(1-k)\right)^{|E(G)|-n}$, $f_l = \mathrm{Ferm}_{ln}^k(P_l(G))$ and $\omega_l = (-k)^l$, then

$$\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} \alpha & 0 & \dots & 0 \\ 0 & \alpha^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \alpha^n \end{pmatrix} \begin{pmatrix} \omega_1 & \omega_1^2 & \dots & \omega_1^n \\ \omega_2 & \omega_2^2 & \dots & \omega_2^n \\ \vdots & \ddots & & \vdots \\ \omega_n & \omega_n^2 & \dots & \omega_n^n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}$$

This system of equation is a Vandermonde system and therefore is invertible (if $k \neq 1$ and $k \neq -1$, because in these cases, some ω_i are equal and the matrix is not invertible): there exists some rationals $\omega^*_{l,m}$ such that for any m, $c_m = \sum_{l=1}^{n} \omega^*_{l,m} f_l(\bar{x})$.

Therefore, for any m, we have a c-reduction from c_m to the fermionant, $(c_m) \leq_c (\text{Ferm}^k)$. But, $c_1 := \sum_{\pi \in S_n | c(\pi)=1} \prod_{i+1}^{n} x_{i,\pi(i)} = \text{Ham}_n(\bar{x})$, where Ham_n is the Hamiltonian, which is known to be VNP-complete ([2], Corollary 3.19). □

The fermionant can be expressed as a linear combination of polynomial size of the Hamiltonian. From that we have concluded that the fermionant is VNP-complete. However, the Hamiltonian is also $\#P$-complete, when considered as a counting problem. This gives us a Turing reduction from the Hamiltonian to the fermionant and thus it is also $\#P$-complete, but only when computed on rational matrix; the Turing reductions requires rationals ($\frac{1}{2}$, $-\frac{1}{k}$, etc). We can adapt the proof of Valiant for the $\#P$-completeness of the permanent to replace those rationals by some gadgets only using 0 and 1. And thus we have the following non trivial corollary. The proof is in the full version of the paper.

Corollary 1. *For every $k \neq 1$ and $k \neq 0$, Ferm^k is $\#P$-complete for matrices over $\{0, 1\}$.*

4 Immanant with Constant Length

Immanants are defined with characters of representations of S_n. Such characters can be indexed by Young diagrams of n boxes (i.e., collections of boxes arranged in left-adjusted rows with a decreasing row length). As all the work of representation theory has already be done (Lemma 3), I will not define more those characters. We shall only work on Young diagrams.

The *immanant* associated with a Young diagram Y (and its associate character χ_Y) is

$$\text{im}_\chi(\bar{x}) = \sum_{\pi \in S_n} \chi_Y(\pi) \prod_{i=1}^{n} x_{i,\pi(i)}$$

For example, if the Young diagram is a single row of n boxes, then for any $\sigma \in S_n$, $\chi_Y(\sigma) = 1$ and thus $\text{im}_Y = \text{per}$. At the opposite end, if Y is a single column with n boxes, $\chi_Y(\sigma) = sg(\sigma)$ and $\text{im}_Y = \det$. For more details (and for a nice demonstration of the Murnaghan-Nakayama rule, one of the main parts of our demonstration), see [5].

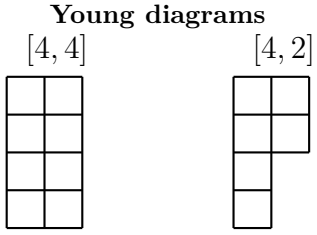
A classical theorem states that the irreducible characters of the symmetric group form a basis for the class functions on S_n. Class functions are real functions defined on S_n and stabled under conjugation (i.e., $\forall \pi, \sigma \in S_n, f(\pi\sigma\pi^{-1}) = f(\sigma)$). The function $\pi \longmapsto (-k)^{c(\pi)}$ is such a class function and thus is a linear combination of characters. Mertens and Moore [7] have computed those characters, and applied to the immanant we get:

Lemma 3. *For any integers k and n, if we write Λ_k^n for the set of every Young diagram with n boxes and at most k columns, then there exists some constants d_Y^k such that for any matrix A:*

$$Ferm_n^k(A) = \sum_{Y \in \Lambda_k^n} d_Y^k \, im_Y(A)$$

Intuitively this suggests that the family of every immanants of bounded width is VNP-complete. In algebraic complexity this is not that interesting, as this family is very large. But if we prove that with a certain family of immanant we can compute every immanants of width less than a certain k, then this family will be VNP-complete. It is exactly what we are going to do for the demonstration of the following proposition.

Proposition 1. *Let $[n, n]$ be the square Young diagram with two columns, each with n rows. Then $(im_{[n,n]})_{n \in \ltimes}$ is* VNP*-complete for c-reductions.*



Proof. More generally, let $[l_1, l_2]$ be the two columns Young diagram with l_1 boxes in the first column and l_2 in the second. More specifically, the Young diagrams of width a most 2 and of n boxes are $([l, n-l])_{l \in [n/2, n]}$. Each of them can be obtained from the square diagram $[l, l]$ by removing a skew hook of size $\delta = (l - (n - l)) = 2l - n$. A skew hook in a Young diagram is a connected collection of boxes in the border of the diagram such that if you remove this hook it is still a Young diagram (i.e., the row sizes are still decreasing). Furthermore, if you remove a skew hook of size δ to $[l, l]$, you can obtain only $[l, n-l]$ and $[l-1, n-l+1]$. The Murnaghan-Nakayama rule (c.f. [2] chap. 7.2 for more details) states that:

$$\mathrm{im}_{[l,l]}(\bar{x}, \boldsymbol{l}) = (-1)^{l-1} \mathrm{im}_{[l,n-l]}(\bar{x}) + (-1)^{l} \mathrm{im}_{[l-1,n-l+1]}(\bar{x})$$

Where $\boldsymbol{l}$ is an encoding of a cycle of length l. We know that, from Lemma 3:

$$\mathrm{Ferm}_n^2(\bar{x}) = \sum_{Y \in \Lambda_2^n} d_Y^2 \mathrm{im}_Y(\bar{x}) = \sum_{l=n/2}^{n} d_{[l,n-l]}^2 \mathrm{im}_{[l,n-l]}(\bar{x})$$

From those two facts, we can compute the fermionant from the square immanant. We just have to take new constants: let $\alpha_{[n-1,1]} = d_{[n-1,1]}(-1)^n$ and for any $2 \leq l \leq \frac{n}{2}$, $\alpha_{[n-l,l]} = (-1)^l (d_{[l,n-1]} - \alpha_{[l+1,n-l-1]}(-a)^{l+1})$. For simplicity, we write $\alpha_{\boldsymbol{l}} = \alpha_{[l,n-l]}$. If n is even

$$\sum_{l=n/2}^{n-1} \alpha_{\mathbf{l}} \mathrm{im}_{[l,l]}(\bar{x}, \boldsymbol{2l-n})$$

$$= \sum_{l=n/2}^{n-1} \alpha_{\mathbf{l}} \left((-1)^{l-1} \mathrm{im}_{[l,n-l]}(\bar{x}) + (-1)^{l} \mathrm{im}_{[l-1,n-l+1]}(\bar{x}) \right)$$

$$= d_{[n-1,1]} \mathrm{im}_{[n-1,1]}(\bar{x}) + \sum_{l=n/2}^{n-2} d_{\mathbf{l}} \mathrm{im}_{[l,n-l]}(\bar{x})$$

$$- \sum_{l=n/2}^{n-2} \alpha_{\mathbf{l+1}} (-1)^{l+1} \mathrm{im}_{[l,n-l]}(\bar{x}) + \sum_{l=n/2}^{n-1} \alpha_{\mathbf{l}} (-1)^{l} \mathrm{im}_{[l-1,n-l+1]}(\bar{x})$$

$$= \sum_{l=n/2}^{n-1} d_{\mathbf{l}} \mathrm{im}_{[l,n-l]}(\bar{x}) - \sum_{l=n/2+1}^{n-1} \alpha_{\mathbf{l}} (-1)^{l} \mathrm{im}_{[l-1,n-l+1]}(\bar{x})$$

$$+ \sum_{l=n/2}^{n-1} \alpha_{\mathbf{l}} (-1)^{l} \mathrm{im}_{[l-1,n-l+1]}(\bar{x})$$

$$= \sum_{l=n/2}^{n-1} d_{\mathbf{l}} \mathrm{im}_{[l,n-l]}(\bar{x}) + \alpha_{\mathbf{n/2+1}} (-1)^{n/2} \mathrm{im}_{[n/2,n/2]}(\bar{x})$$

Furthermore, $\mathrm{im}_{[n,0]}(\bar{x}) = \det_n(\bar{x})$ and then can be computed with only a polynomial number of arithmetic operations. Thus,

$$\sum_{l=n/2}^{n-1} \alpha_{\mathbf{l}} \mathrm{im}_{[l,l]}(\bar{x}, \boldsymbol{2l-n}) + \det_n(\bar{x}) + (-1)^{\frac{n}{2}} \alpha_{\frac{n}{2}+1} \mathrm{im}_{[\frac{n}{2},\frac{n}{2}]}(\bar{x})$$

$$= \sum_{l=n/2}^{n} d_{[l,n-l]} \mathrm{im}_{[l,n-l]} = \mathrm{Ferm}_n^2(\bar{x})$$

We obtain an arithmetic circuit of polynomial size that compute Ferm_n^2 with $n/2$ oracles that can compute $\mathrm{im}_{[l,l]}$ for $l \in [n/2, n]$. To obtain a c-reduction from the fermionant to the immant, we just have to notice that $\mathrm{im}_{[l,l]} \leq_p \mathrm{im}_{[l',l']}$ as soon as $l' \geq l$. Indeed, we just have to erase the first $l' - l$-th rows, which can be done by Corollary 3.2 of [1].

The demonstration for n odd works the same, the border cases must just be studied a little bit more closer. □

We can generalize this result to almost every family of bounded width. The proof is similar and is in the annex.

Theorem 2. *Let (Y_n) be a family of Young diagrams of length bounded by $k \geq 2$ such that $|Y_n| = \Omega(n)$. Then*

- *if the number of boxes in the right of the first column if bounded by a constant c, then* (im_{Y_n}) *is in* VP.
- *otherwise, if there is an* $\epsilon > 0$ *and at least* n^ϵ *boxes at the right of the first column,* (im_{Y_n}) *is* VNP*-complete for c-reductions.*

5 Conclusion and Perspectives

The generalization via the fermionant tell us that the determinant is really special: the coefficients 1 and -1 allows us, in a simplify way, to cancel some monomials and not to have to compute everything. The k in the fermionant, even thinly different than 1, separates these monomials and prevents the cancelations.

As for the immanant, the interpretation of the result is harder. Especially as our theorem does not completely classify immanants of constant width, what about the immanant of $[n, \log n]$? Bürgisser's algorithm gives a subexponentiel upper bound, but does not put it in VP. Howerver, under the extended Valiant hypothesis (end of chapter 2 in [2]), it can not be VNP-complete. Is it a good candidate to be neither VP nor VNP-complete? Or even VP-complete? Or is it as hard as the determinant? This is unknown.

Other generalizations also can be imagined. For example generating functions of a graph property are polynomials that generalize the permanent and some of them can be computed as fast as the determinant. This framework allows us to use our knowledge on graph theory to understand where we step from VP to VNP. There is no classification of these generation functions, but some results have been found [2,4].

I thank to both of my doctoral advisors, A. Durand and G. Malod as well to M. Casula and E. Boulat who try to explain to me the physic behind the fermionant.

References

1. Brylinski, J.-L., Brylinski, R.: Complexity and Completeness of Immanants. CoRR, cs.CC/0301024 (2003)
2. Bürgisser, P.: Completeness and Reduction in Algebraic Complexity Theory. Springer (2000)
3. Chandrasekharan, S., Wiese, U.-J.: Partition Functions of Strongly Correlated Electron Systems as "Fermionants". ArXiv e-prints (August 2011)
4. de Rugy-Altherre, N.: A dichotomy theorem for homomorphism polynomials, pp. 308–322 (2012)
5. Greene, C.: A rational-function identity related to the Murnaghan-Nakayama formula for the characters of Sn. J. Algebraic Comb. 1(3), 235–255 (1992)
6. Littlewood, D.E.: The theory of group characters and matrix representations of groups. The Clarendon Press (1940)
7. Mertens, S., Moore, C.: The complexity of the fermionant, and immanants of constant width. ArXiv e-prints (October 2011)
8. Poizat, B.: À la recherche de la définition de la complexité d'espace pour le calcul des polynômes à la manière de valiant. Journal of Symbolic Logic 73 (2008)

Aligning and Labeling Genomes under the Duplication-Loss Model

Riccardo Dondi[1] and Nadia El-Mabrouk[2]

[1] Dipartimento di Scienze Umane e Sociali,
Università degli Studi di Bergamo, Bergamo, Italy
[2] Département d'Informatique et de Recherche Opérationnelle,
Université de Montréal, Montréal, Québec, Canada
riccardo.dondi@unibg.it, mabrouk@iro.umontreal.ca

Abstract. We investigate the complexity of two combinatorial problems related to pairwise genome alignment under the duplication-loss model of evolution. Under this model, the unaligned parts of the genomes (gaps) are interpreted as duplications and losses. The first, and most general, combinatorial problem that we consider is the Duplication-Loss Alignment problem, which is to find an alignment of minimum duplication-loss cost. Defining the cost as the number of segmental duplications and individual losses, the problem has been recently shown NP-hard. Here, we improve this result by showing that the Duplication-Loss Alignment (DLA) problem is APX-hard even if the number of occurrences of a gene inside a genome is bounded by 2. We then consider a more constrained version, the Feasible Relabeling Alignment problem, involved in a general methodology for solving DLA, that aims to infer a feasible (in term of evolutionary history) most parsimonious relabeling of an initial best candidate labeled alignment which is potentially cyclic. We show that it is equivalent to Minimum Feedback Vertex Set on Directed Graph, hence implying that the problem is APX-hard, is fixed-parameter tractable and approximable within factor $O(\log |\mathcal{X}| \log \log |\mathcal{X}|)$, where $\mathcal{X}$ is the aligned genome considered by Feasible Relabeling Alignment.

1 Introduction

The comparison of complete genomes requires considering two kinds of macro-evolutionary mutations modifying their overall gene (or other building blocks) content and organization: (1) rearrangements, such as inversions, transpositions and translocations, affecting the order of genes and (2) content modifying operations, such as duplications, insertions and losses affecting the number and type of genes. Preliminary to such global comparison of genomes is the identification of homologs and the representation of genomes as strings of symbols over an alphabet Σ of gene families. In the last decades, genome comparisons have been largely based on rearrangement events [3, 4, 7, 9, 10, 13–16]. Contrariwise, we

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 97–107, 2013.

introduced in [11] an evolutionary model restricted to content-modifying operations, namely duplications and losses. We showed that this model is effective in studying the evolution of certain gene families, such as Transfer RNAs. From a combinatorial point of view, when rearrangements are ignored gene orders are preserved, hence allowing to reformulate the comparison of two genomes as a Duplication-Loss Alignment problem: find a pairwise alignment minimizing a duplication and loss cost. As in [2, 5, 11], we consider here the cost of an alignment to be the number of underlying segmental duplications (duplication of a string of adjacent genes) and single losses (loss of a single gene). Under this cost, Duplication-Loss Alignment has been recently shown to be NP-hard, even when each gene has at most five occurrences in each genome [5]. In this paper (Section 3), we improve this result by showing that the Duplication-Loss Alignment problem is APX-hard, even if the number of occurrences of a gene inside a genome is bounded by 2. As most of the genes in a genome are present in a limited number of paralogous copies, understanding how the complexity of the problem is influenced by the number of gene copies is important to orient appropriate algorithmic developments.

Two integer linear-programming approaches have been developed in [5, 11] for the Duplication-Loss Alignment problem. Although the algorithm in [5] has a very reasonable running time in general, they are both exponential in the worst case, preventing from being applicable to large genomic datasets. Therefore, a time-efficient heuristic, exhibiting a high degree of accuracy on simulated datasets, has been developed in [2]. It is based on two steps: (1) compute a best candidate, possibly cyclic, labeled alignment using a dynamic-programming approach; (2) (Feasible Relabeling Alignment) find a most parsimonious correction of the cycles, allowing to get a non-cyclic (feasible) labeled alignment, which corresponds to a duplication-loss history. In [2], a more general problem called Minimum Labeling Alignment, asking for a most parsimonious feasible labeling of an unlabeled alignment, has been proved APX-hard. In this paper, we prove (Section 4) that Feasible Relabeling Alignment is equivalent to Minimum Feedback Vertex Set on Directed Graph, hence showing that this problem is also APX-hard and that: (1) it is fixed-parameter tractable, when the parameter is the cost of the relabeling, (2) it is approximable within factor $O(\log |\mathcal{X}| \log \log |\mathcal{X}|)$, where $\mathcal{X}$ is the aligned genome considered by Feasible Relabeling Alignment. Due to space limitation, some of the proofs are omitted.

2 Evolutionary Model and Problem Statement

Strings: We consider single chromosomal (circular or linear) genomes. Hence, given an alphabet Σ, each symbol representing a specific gene family, a *genome* or *string* is a sequence of symbols from Σ, where each symbol may have many occurrences (paralogs). For example, X in Figure 1 is a genome on the alphabet $\Sigma = \{a, b, c, d, e, f\}$, with three copies from the gene family identified by a.

Given a string Z, we denote by $|Z|$ its length, by $Z[i]$, $1 \leq i \leq |Z|$, the i-th symbol of Z, and by $Z[i,j]$, $1 \leq i \leq j \leq |Z|$, the substring of Z that starts at position i and ends at position j. Finally, we say that two substrings $Z[i_1, i_2]$ and $Z[j_1, j_2]$, with $1 \leq i_2 \leq j_2 \leq |Z|$, overlap if $j_1 \leq i_2$.

The Duplication-Loss Model of Evolution: We assume that present-day genomes have evolved from an ancestral string through (segmental) duplications and (single) losses, where, given a genome X: (i) A *duplication* D of size $|D| = z$ is an operation that copies a substring of size z of X somewhere else in the genome. Given two identical non overlapping substrings $X[i, i+z-1]$ and $X[j, j+z-1]$ of X, we denote by $D = (X[i, i+z-1], X[j, j+z-1])$ a *duplication* from $X[i, i+z-1]$ to $X[j, j+z-1]$; $X[i, i+z-1]$ is the *source*, and $X[j, j+z-1]$ is the *target* of the duplication D; (ii) A *loss* is an operation $L = (X[i])$ that removes a symbol $X[i]$ from X. Notice that the model of evolution we consider assumes that events represented in the evolution from an ancestral genome to a present-day genome are not obscured by subsequent events (see [11] for a formal definition of "visible history").

The Duplication-Loss Alignment Problem: In the rest of the paper, we consider two genomes X and Y on an alphabet Σ. We introduced in [11] the concept of "Feasible" Labeled Alignment of two genomes X and Y. Definitions on alignments are given below, and illustrated in Figure 1.

Denote by $\Sigma^- = \Sigma \cup \{-\}$ the alphabet Σ augmented with a symbol '$-$' (called a *gap*) not in Σ.

Definition 1. *An* alignment *of X and Y, denoted by $\mathcal{A}(\mathcal{X}, \mathcal{Y})$, consists of a pair $(\mathcal{X}, \mathcal{Y})$ of strings on $\Sigma^- \times \Sigma^-$ obtained by filling X and Y respectively with gaps, such that (1) the* aligned genomes *$\mathcal{X}$ and $\mathcal{Y}$ are equal length and (2) for each position i, $1 \leq i \leq |\mathcal{X}|$, it holds that either $\mathcal{X}[i] = \mathcal{Y}[i] \neq -$ (i is called a* Match*), or exactly one of $\mathcal{X}[i]$, $\mathcal{Y}[i]$ is equal to a gap (i is called a* Mismatch*).*

To explain an alignment $\mathcal{A}(\mathcal{X}, \mathcal{Y})$ as a duplication-loss history leading to X and Y from a common ancestor, we need to label the mismatched positions of the aligned genomes $\mathcal{X}$ and $\mathcal{Y}$ in terms of *duplications* and *losses*.

Definition 2. *A* labeling *$\mathcal{L}(\mathcal{X})$ of an aligned genome $\mathcal{X}$ is a set of losses and duplications, such that for each mismatched position j, $1 \leq j \leq |\mathcal{X}|$, $\mathcal{L}(\mathcal{X})$ contains either a loss $L = (\mathcal{X}[j])$, in which case we say that position j is labeled as a loss, or exactly one duplication $D = (\mathcal{X}[i_1, i_2], \mathcal{X}[j_1, j_2])$ with $1 \leq j_1 \leq j \leq j_2 \leq |\mathcal{X}|$, in which case we say that position j is labeled as a duplication. A* Labeled alignment *$\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ is a* labeling *of the two aligned genomes $\mathcal{X}$ and $\mathcal{Y}$.*

The cost of a labeling $\mathcal{L}(\mathcal{X})$, denoted by $c(\mathcal{L}(\mathcal{X}))$, is the number of underlying operations (duplications and losses). The cost of a labeled alignment $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ is the sum of $c(\mathcal{L}(\mathcal{X}))$ and $c(\mathcal{L}(\mathcal{Y}))$.

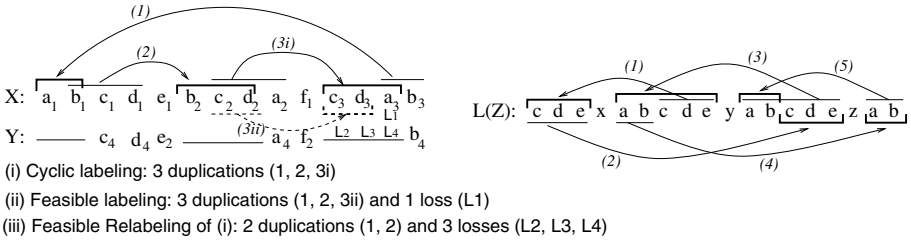


Fig. 1. Left: An alignment for the strings $X = abcdebcdafcdab$ and $Y = cdeafb$ with one cyclic labeling (i) and two feasible labeling (ii) and (iii) for the aligned genome $\mathcal{X}$. Losses are denoted by "L" and duplications by arrows from source (indicated by lines) to target (bold brackets). In (i), the set of duplications $(1, 2, 3i)$ is a cycle. Right: A cyclic labeling of an aligned genome $\mathcal{Z}$. Symbols x, y and z refer to matches in the alignment.

The above definition is not sufficient to unsure a correct interpretation of an alignment in term of a duplication-loss history, as it may induce duplication cycles (see Figure 1), rigorously defined as follows.

Definition 3. *Consider a set of duplications $\mathcal{D}$. $\mathcal{D}$ induces a* duplication cycle *if there is a permutation $D_1 = (\mathcal{X}[i_1, r_1], \mathcal{X}[j_1, s_1])$, $D_2 = (\mathcal{X}[i_2, r_2], \mathcal{X}[j_2, s_2])$, ..., $D_h = (\mathcal{X}[i_h, r_h], \mathcal{X}[j_h, s_h])$ of the duplications in $\mathcal{D}$, such that (1) the substrings $\mathcal{X}[j_p, s_p]$ and $\mathcal{X}[i_{p+1}, r_{p+1}]$ overlap, for each $1 \leq p \leq h-1$, and (2) the substrings $\mathcal{X}[j_h, s_h]$ and $\mathcal{X}[i_1, r_1]$ overlap.*

A labeling $\mathcal{L}(\mathcal{X})$ is called *feasible* if it contains no subset of duplications that induces a duplication cycle (in Figure 1, (ii) and (iii) are feasible labeling). A *feasible labeled alignment* $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ is a feasible labeling of an alignment of X and Y where $\mathcal{L}(\mathcal{X})$ and $\mathcal{L}(\mathcal{Y})$ are feasible labeling. As demonstrated in [11], a *feasible labeled alignment* of two genomes X and Y leads to a unique duplication-loss history from a unique ancestral genome A to X and Y.

We are now ready to give the main optimization problem introduced in [11]. Studying the complexity of this problem is the purpose of Section 3.

Problem 1 ***Duplication-Loss Alignment****[DLA]*
Input*: Two genomes X and Y over alphabet Σ.*
Output*: A Feasible Labeled Alignment $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ of minimum cost.*

Feasible Relabeling Alignment: A natural heuristic to DLA, suggested in [11] and shown very accurate and time efficient in [2], proceeds in two steps. First, based on a dynamic programming approach, compute a (possibly cyclic) labeled alignment $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ of minimum cost. Such a labeled alignment is not necessarily feasible as it may contain cycles. Then, in a second step, find a feasible relabeling $\mathcal{L}'(\mathcal{X})$ (respec. $\mathcal{L}'(\mathcal{Y})$) of $\mathcal{X}$ (respec. $\mathcal{Y}$) by correcting cycles of $\mathcal{L}(\mathcal{X})$ (respec. $\mathcal{L}(\mathcal{Y})$). Notice that once the genomes X and Y are aligned, each of $\mathcal{L}(\mathcal{X})$ and $\mathcal{L}(\mathcal{Y})$ can be treated separately. As matched versus mismatched positions

cannot be modified, correcting cycles of $\mathcal{L}(\mathcal{X})$ (respec. $\mathcal{L}(\mathcal{Y})$) can only be done by interpreting some positions that are covered by the target of a duplication rather as losses. For example, the labeling (ii) in Figure 1 is obtained by correcting the cycle $(1, 2, 3i)$: symbol a_3, covered by the target of duplication (3i), is rather interpreted as a loss. In this paper we show that, even an apparently simple strategy that consists in replacing complete duplications by losses (all symbols of a duplication target are interpreted as losses) is hard. A rigorous definition of such "allowed" relabeling follows.

Definition 4. *Given an aligned genome $\mathcal{X}$ and a labeling $\mathcal{L}(\mathcal{X})$, a* feasible relabeling *$\mathcal{L}'(\mathcal{X})$ of $\mathcal{L}(\mathcal{X})$ is a labeling of $\mathcal{X}$ obtained by replacing a set $D_1, \cdots D_n$ of duplications of $\mathcal{L}(\mathcal{X})$ with losses, so that $\mathcal{L}'(\mathcal{X})$ is a feasible labeling of $\mathcal{X}$.*

The labeling $\mathcal{L}'(\mathcal{X})$ has cost $c(\mathcal{L}'(\mathcal{X})) = c(\mathcal{L}(\mathcal{X})) + \sum_{1 \leq i \leq n}(|D_i| - 1)$; hence we define the relabeling cost of *$\mathcal{L}'(\mathcal{X})$ with respect to $\mathcal{L}(\mathcal{X})$, denoted as $c_r(\mathcal{L}'(\mathcal{X}))$, as: $c_r(\mathcal{L}'(\mathcal{X})) = c(\mathcal{L}'(\mathcal{X})) - c(\mathcal{L}(\mathcal{X})) = \sum_{1 \leq i \leq n}(|D_i| - 1)$.*

With this, the Feasible Relabeling Alignment problem can be defined as follows:

Problem 2 ***Feasible Relabeling Alignment****[FRA]*
Input*: An aligned genome $\mathcal{X}$ over alphabet Σ and a labeling $\mathcal{L}(\mathcal{X})$.*
Output*: A feasible relabeling $\mathcal{L}'(\mathcal{X})$ of $\mathcal{L}(\mathcal{X})$ of minimum relabeling cost.*

In Figure 1, (iii) is a feasible relabeling of (i) obtained by removing duplication (3i). For an intuition on the difficulty of the FRA problem, see the more involved example of a labeling with interleaving cycles in Figure 1, right.

3 Complexity of Duplication-Loss Alignment

In this section we consider undirected graphs, and we recall that an undirected graph is *cubic* when each of its vertex has degree 3. We investigate the complexity of Duplication-Loss Alignment, and we show that the problem is APX-hard even when each gene appears at most twice in the genome (we denote this restriction as 2-DLA). We prove the APX-hardness of 2-DLA by giving a reduction from Minimum Vertex Cover on Cubic Graphs (MVCC), which is known to be APX-hard [1]. Given an undirected cubic graph $G = (V, E)$, MVCC asks for a subset $V' \subseteq V$ of minimum cardinality, such that for each edge $\{u, v\} \in E$, at least one of u, v is in V'.

Let $G = (V, E)$ be a cubic graph, in the following we define an instance (X, Y) of DLA. Given a vertex v_i and its incident edges $\{v_i, v_j\}$, $\{v_i, v_q\}$, $\{v_i, v_r\}$, with $j < q < r$, we say that $\{v_i, v_j\}$ ($\{v_i, v_q\}$, $\{v_i, v_r\}$ respectively) is the first (second, third respectively) edge incident in v_i.

First, set $t = 4|V| + |E| + 1$. We define the alphabet Σ over which X and Y range: $\Sigma = \{\alpha_{i,j} : v_i \in V \wedge 1 \leq j \leq 6\} \cup \Gamma \cup \{\beta_{i,j} : v_i \in V \wedge 1 \leq j \leq 4\} \cup \Lambda$, where $\Gamma = \{\gamma_{i,j} : v_i \in V \wedge 1 \leq j \leq t\}$, $\Lambda = \{\lambda_{i,j,h} : \{v_i, v_j\} \in E \wedge 1 \leq h \leq t\}$. For each $v_i \in V$, define two substrings $B_X(v_i)$, $B_Y(v_i)$ (substrings of X, Y respectively),

as follows: $B_X(v_i) = \alpha_{i,1} \dots \alpha_{i,6} \beta_{i,1} \dots \beta_{i,4}$; $B_Y(v_i) = \beta_{i,1} \dots \beta_{i,4} \alpha_{i,1} \dots \alpha_{i,6}$. Now, consider the edge $\{v_i, v_j\} \in E$ and assume that $\{v_i, v_j\}$ is the h-th edges incident in v_i, $1 \leq h \leq 3$, and the k-th edge incident in v_j, $1 \leq k \leq 3$. Define two substrings $B_X(e_{i,j})$, $B_Y(e_{i,j})$ of X, Y respectively, associated with $\{v_i, v_j\}$, as follows: $B_X(e_{i,j}) = \alpha_{i,2h-1} \alpha_{i,2h} \alpha_{j,2k-1} \alpha_{j,2k}$; $B_Y(e_{i,j}) = \alpha_{j,2k-1} \alpha_{j,2k} \alpha_{i,2h-1} \alpha_{i,2h}$.

Based on this, we are able to define the two genomes X, Y (where $\{v_1, v_w\}$, $\{v_q, v_r\}$ are two edges of E):

$$X = \gamma_{1,1} \dots \gamma_{1,t} B_X(v_1) \gamma_{2,1} \dots \gamma_{2,t} B_X(v_2) \dots \gamma_{n,1} \dots \gamma_{n,t} B_X(v_n) \cdot \lambda_{1,w,1} \dots \lambda_{1,w,t} B_X(e_{1,w}) \dots \lambda_{q,r,1} \dots \lambda_{q,r,t} B_X(e_{q,r})$$

$$Y = \gamma_{1,1} \dots \gamma_{1,t} B_Y(v_1) \gamma_{2,1} \dots \gamma_{2,t} B_Y(v_2) \dots \gamma_{n,1} \dots \gamma_{n,t} B_Y(v_n) \cdot \lambda_{1,w,1} \dots \lambda_{1,w,t} B_Y(e_{1,w}) \dots \lambda_{q,r,1} \dots \lambda_{q,r,t} B_Y(e_{q,r})$$

It is easy to see that (X, Y) is an instance of 2-DLA, as each symbol of Σ has at most two occurrences in each of X, Y.

In order to prove the main properties of the reduction we have to show some intermediate results. First, in Proposition 1, we show that all the positions containing symbols in $\Gamma \cup \Lambda$ are aligned.

Proposition 1. *Given a labeled alignment $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ of cost less than $2t$, we can compute in polynomial time a labeled alignment $\mathcal{A}'(\mathcal{L}'(\mathcal{X}), \mathcal{L}'(\mathcal{Y}))$ that (1) aligns each position of X and Y containing a symbol in $\Gamma \cup \Lambda$ and (2) has a cost not greater than that of $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$.*

As a consequence of Proposition 1, we can assume that if two positions containing symbols $\Sigma \setminus (\Gamma \cup \Lambda)$ of X, Y are aligned, then either they both belong to substrings $B_X(v_i)$, $B_Y(v_i)$, with $v_i \in V$, or they both belong to substrings $B_X(e_{i,j})$, $B_Y(e_{i,j})$, with $\{v_i, v_j\} \in E$.

Now, we are ready to prove the main results of the reduction. The idea of the reduction is that for each pair of substrings $B_X(v_i)$, $B_Y(v_i)$ we have two possible cases: (1) the substrings $\alpha_{i,1} \dots \alpha_{i,6}$ are aligned, each position of value $\beta_{i,x}$, $1 \leq x \leq 4$, is labeled as a loss, and duplications from $B_X(v_i)$ ($B_Y(v_i)$ respectively) to some of $B_X(e_{i,j})$, $B_X(e_{i,q})$, $B_X(e_{i,r})$ ($B_Y(e_{i,j})$, $B_Y(e_{i,q})$, $B_Y(e_{i,r})$ respectively) are defined (this case corresponds to vertex v_i in the vertex cover of G and has cost 8); (2) the substrings $\beta_{i,1} \dots \beta_{i,4}$ are aligned and duplications from each of $B_X(e_{i,j})$, $B_X(e_{i,q})$, $B_X(e_{i,r})$ ($B_Y(e_{i,j})$, $B_Y(e_{i,q})$, $B_Y(e_{i,r})$ respectively) to $B_X(v_i)$ ($B_Y(v_i)$ respectively) are defined (this case corresponds to vertex v_i not in the vertex cover of G and has cost 6).

Lemma 1. *Let $G = (V, E)$ be a cubic graph and let (X, Y) be the corresponding instance of 2-DLA. Then, given a vertex cover $V' \subseteq V$ of G, we can compute in polynomial time a feasible labeled alignment $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ of cost $8|V'| + 6|V \setminus V'| + 2|E|$.*

Lemma 2. *Let $G = (V, E)$ be a cubic graph and let (X, Y) be the corresponding instance of 2-DLA. Then, given an alignment $\mathcal{A}(\mathcal{L}(\mathcal{X}), \mathcal{L}(\mathcal{Y}))$ of cost $8p + 6(|V| - p) + 2|E|$ we can compute in polynomial time a vertex cover of G of size at most p.*

The APX-hardness of 2-DLA is a direct consequence of Lemmas 1, 2, and of the APX-hardness of MVCC [1].

4 Complexity of Feasible Relabeling Alignment

In this section we show that FRA is equivalent to the Minimum Directed Feedback Vertex Set (DFVS) problem. Given a directed graph $G = (V, A)$, DFVS asks for a feedback vertex set $V' \subseteq V$ of minimum cardinality. A *feedback vertex set* (FVS) of G is a subset $V' \subseteq V$ such that V' contains at least one vertex from every directed cycle in G. First, in Section 4.1 we give an L-reduction from DFVS to FRA. As a consequence, we prove that FRA is APX-hard. Then, in Section 4.2, we give a reduction from DFVS to FRA.

4.1 Hardness of FRA

In this section we give an L-reduction from DFVS to FRA. Given a directed graph $G = (V, A)$, with $V = \{v_1, \ldots, v_n\}$, in what follows we define the corresponding genome $\mathcal{X}$ and labeling $\mathcal{L}(\mathcal{X})$. Given a substring s of $\mathcal{X}$, we denote with s^a the fact that s is aligned in $\mathcal{X}$ (hence it does not need any labeling). In the definition of $\mathcal{L}(\mathcal{X})$, first we define the aligned genome $\mathcal{X}$, then we define the labeling of $\mathcal{X}$.

Before giving the details of the construction we give an overview of the construction of $\mathcal{X}$ and $\mathcal{L}(\mathcal{X})$. For each vertex $v_i \in V$ we define a substring $F(v_i)$ obtained by concatenating four substrings $s_{i,\text{IN}}$, $s_{i,1}$, $s_{i,2}$, $s_{i,\text{OUT}}$. The reduction defines two kinds of duplications: (1) duplications between substrings of $F(v_i)$ (one duplication between $s_{i,\text{IN}}$, $s_{i,1}$, one duplication between $s_{i,1}$, $s_{i,2}$, one duplication between $s_{i,2}$, $s_{i,\text{OUT}}$); (2) duplications between substrings of different $F(v_i)$, $F(v_j)$. The latter kind of duplications encodes arcs of the graph G. Furthermore, notice that $s_{i,\text{IN}}$ is used to encode arcs incoming in v_i, while $s_{i,\text{OUT}}$ is used to encode arcs outgoing from v_i.

Now, we define formally the instance of FRA. Define the alphabet $\Sigma = \{w_{i,j,t} : (v_i, v_j) \in A, 1 \leq t \leq n+2\} \cup \{x_i, y_i : v_i \in V\}$. Given an arc $(v_i, v_j) \in A$, define the string $e_{i,j} := w_{i,j,1} w_{i,j,2} \ldots w_{i,j,n+2}$. We define the strings $s_{i,\text{IN}}$, $s_{i,1}$, $s_{i,2}$, $s_{i,\text{OUT}}$ (notice that we assume that the vertices with an arc (v_h, v_i) are $\{v_{h_1}, \ldots, v_{h_z}\}$, and that $h_1 < h_2 \cdots < h_z$):

$$
\begin{aligned}
s_{i,\text{IN}} &= e_{i,h_1}\ e_{i,h_2}\ \ldots\ e_{i,h_z}\ x_i^a; \\
s_{i,1} &= e_{i,h_1}\ e_{i,h_2}\ \ldots\ e_{i,h_z}\ x_i\ y_i^a; \\
s_{i,2} &= e_{i,h_1}^a\ e_{i,h_2}^a\ \ldots\ e_{i,h_z}^a\ x_i\ y_i; \\
s_{i,\text{OUT}} &= e_{i,h_1}\ e_{i,h_2}\ \ldots\ e_{i,h_z}\ x_i.
\end{aligned}
$$

Define $F(v_i) = s_{i,\text{IN}} \cdot s_{i,1} \cdot s_{i,2} \cdot s_{i,\text{OUT}}$. The aligned genome $\mathcal{X}$ is defined as follows: $\mathcal{X} = F(v_1) \cdot F(v_2) \cdot \cdots \cdot F(v_n)$.

Now, we define the labeling $\mathcal{L}(\mathcal{X})$ of $\mathcal{X}$. $\mathcal{L}(\mathcal{X})$ consists of two kinds of duplications: duplications between two substrings of the same $F(v_i)$ and duplications between substrings belonging to different $F(v_i)$, $F(v_j)$. We start by defining the labeling of the strings in $F(v_i)$ (for the not aligned positions), which is used to encode the vertex $v_i \in V$:

1. a duplication from the substring $e_{i,h_1}\ e_{i,h_2}\ \ldots\ e_{i,h_z}x_i^a$ of $s_{i,\text{IN}}$ to the substring $e_{i,h_1}\ e_{i,h_2}\ \ldots\ e_{i,h_z}x_i$ of $s_{i,1}$;
2. a duplication from the substring $x_i\ y_i^a$ of $s_{i,1}$ to the substring $x_i\ y_i$ of $s_{i,2}$;
3. a duplication from the substring $e_{i,h_1}^a\ e_{i,h_2}^a\ \ldots\ e_{i,h_z}^a x_i$ of $s_{i,2}$ to the substring $e_{i,h_1}\ e_{i,h_2}\ \ldots\ e_{i,h_z}x_i$ of $s_{i,OUT}$.

Now, we define the duplications between substrings of $\mathcal{X}$ that belong to different $F(v_i)$. Those duplications are used to encode the arcs in A. Given an arc $(v_i, v_j) \in A$, define a duplication from the substring $e_{i,j}$ of $s_{i,\text{OUT}}$ to the substring $e_{i,j}$ of $s_{j,\text{IN}}$.

The duplication from the substring $s_{i,1}$ to the substring $s_{i,2}$, $1 \leq i \leq n$, both belonging to $F(v_i)$ (notice that the duplicated string is x_iy_i), is called a *candidate duplication*, and it is denoted by D_i. Each other duplication is called a *non candidate duplication*. Informally, the reduction is based on the following properties. Since each non candidate duplication has a relabeling cost of at least $n+1$ (it includes the duplication of substring $e_{i,j}$), it follows that: (1) a feasible relabeling $\mathcal{L}'(\mathcal{X})$ is computed by relabeling only candidate duplications; (2) a vertex v_i in a solution V' of DFVS corresponds to the relabeling of a candidate duplication D_i.

Now, we present the two main properties of the reduction.

Lemma 3. *Let $G = (V, A)$ be a directed graph, and let $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ be the corresponding instance of FRA. Then, given a feedback vertex set V' of G, we can compute in polynomial time a feasible relabeling $\mathcal{L}'(\mathcal{X})$ of $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ of relabeling cost $|V'|$.*

Lemma 4. *Let $G = (V, A)$ be a graph, and let $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ be the corresponding instance of FRA. Then, given a feasible relabeling $\mathcal{L}'(\mathcal{X})$ of $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ of relabeling cost c, we can compute in polynomial time a feedback vertex set V' of G, with $|V'| \leq c$.*

The APX-hardness of FRA is a direct consequence of Lemmas 3, 4 and of the APX-hardness of DFVS [12].

4.2 Tractability of FRA

In this section we give give a reduction from FRA to DFVS. The reduction we present is both a parametrized and an approximation preserving reduction, hence it follows that: (1) FRA is fixed-parameter tractable, when parametrized by the relabeling cost of the solution; (2) FRA can be approximated within factor $O(\log |\mathcal{X}| \log\log |\mathcal{X}|)$.

Now, let $\mathcal{X}$ be a labeled genome associated with a labeling $\mathcal{L}(\mathcal{X})$. In what follows, we define the directed graph $G = (V, A)$ (input of DFVS) associated with $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$. Consider the set $\mathcal{D}$ of duplications induced by $\mathcal{L}(\mathcal{X})$. First, notice that we assume that each duplication $D \in \mathcal{D}$ has size at least 2, otherwise we can relabel such a duplication with relabeling cost 0.

We can now define $G = (V, A)$ by letting $V := \bigcup_{D \in \mathcal{D}} V(D)$, where $V(D)$ is a set of vertices associated with duplication $D \in \mathcal{D}$, defined by $V(D) := \{v_{D,i} : 1 \leq i \leq |D| - 1\}$. The set of arcs A is defined as $A := \{(v_{D_i,p}, v_{D_j,q}) : D_i = (\mathcal{X}[i_1, i_2], \mathcal{X}[i_3, i_4]), D_j = (\mathcal{X}[j_1, j_2], \mathcal{X}[j_3, j_4]), \mathcal{X}[i_3, i_4], \mathcal{X}[j_1, j_2]$ overlap, $1 \leq p \leq |D_i| - 1, 1 \leq j \leq |D_j| - 1\}$.

Informally, given two duplications D_i, D_j, such that the target of D_i and the source of D_j overlap, we have an arc from each vertex of $V(D_i)$ to each vertex of $V(D_j)$.

Next, we show how to relate a feedback vertex set V' of G and a solution of FRA having size $|V'|$. The idea is that a set $V(D_i)$ of nodes in the feedback vertex set of G corresponds to a duplication D_i relabeled as loss. Notice that a feedback vertex set V' of G is *minimal* if there exists no vertex $v \in V'$ such that $V' \setminus \{v\}$ is a feedback vertex set of G. Next, we prove some properties of a minimal FVS of G.

Lemma 5. *Let V' be a minimal feedback vertex set of the graph $G = (V, E)$ associated with $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$. Then, given a duplication D_i of $\mathcal{D}$, either all the vertices of $V(D_i)$ belong to V' or none of the vertices of $V(D_i)$ belongs to V'.*

Now, we are ready to prove the main properties of the reduction.

Lemma 6. *Let $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ be an instance of FRA and let G be the corresponding instance of DFVS. Then, given a feasible relabeling $\mathcal{L}'(\mathcal{X})$ of $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ of relabeling cost c, we can compute in polynomial time a feedback vertex set of G of size c.*

Lemma 7. *Let $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ be an instance of FRA and let G be the corresponding instance of DFVS. Then, given a minimal feedback vertex set V' of G, we can compute in polynomial time a feasible relabeling $\mathcal{L}'(\mathcal{X})$ of $(\mathcal{X}, \mathcal{L}(\mathcal{X}))$ of relabeling cost $|V'|$.*

Theorem 2 is a consequence of Lemma 6, Lemma 7, and of the fact that DFVS admits a fixed-parameter algorithm of time complexity $O(4^k k! poly - time(|\mathcal{X}|))$ [6], and it is approximable within factor $O(\log |V| \log \log |V|)$ [8, 17].

Theorem 2 *The FRA problem: (1) admits a fixed-parameter algorithm of time complexity $O(4^k k! poly - time(|\mathcal{X}|))$, where k is the value of the relabeling cost; (2) admits an approximation algorithm of factor $O(\log |\mathcal{X}| \log \log |\mathcal{X}|)$.*

5 Conclusion

Although ancestral genome inference is a classical problem, developed methods stand on evolutionary models involving rearrangements. Surprisingly the case of an evolution with only content-modifying operations has only been considered very recently. From a combinatorial point of view, the benefit is the ability to reformulate the problem of comparing two gene orders as an alignment problem, which is *a priori* simpler to handle than rearrangements. However, the complexity results given in this paper show that there is no direct and simple way for inferring optimal alignments. Even the apparently simple FRA problem of correcting a cyclic alignment by changing duplications into losses has been proved APX-hard (although it is fixed-parameter tractable). Interesting future work include the investigation of approximation and parametrized complexity of DLA, together with the more relaxed version of FRA allowing for individual symbols (rather than complete duplications) to be interpreted as losses.

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoretical Computer Science 237(1-2), 123–134 (2000)
2. Benzaid, B., Dondi, R., El-Mabrouk, N.: Duplication-loss genome alignment: Complexity and algorithm. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 116–127. Springer, Heidelberg (2013)
3. Bergeron, A.: A very elementary presentation of the hannenhalli-pevzner theory. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 106–117. Springer, Heidelberg (2001)
4. Bourque, G., Pevzner, P.: Genome-scale evolution: Reconstructing gene orders in the ancestral species. Genome Research 12, 26–36 (2002)
5. Canzar, S., Andreotti, S.: A branch-and-cut algorithm for the 2-species duplication-loss phylogeny problem. CoRR abs/1208.2698 (2012)
6. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. J. ACM 55(5) (2008)
7. El-Mabrouk, N.: Genome rearrangement with gene families. In: Mathematics of Evolution and Phylogeny, pp. 291–320. Oxford University Press, Oxford (2005)
8. Even, G., Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multicuts in directed graphs. Algorithmica 20(2), 151–174 (1998)
9. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of genome rearrangements. The MIT Press, Cambridge (2009)
10. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). J. ACM 48, 1–27 (1999)
11. Holloway, P., Swenson, K.M., Ardell, D.H., El-Mabrouk, N.: Evolution of genome organization by duplication and loss: An alignment approach. In: Chor, B. (ed.) RECOMB 2012. LNCS, vol. 7262, pp. 94–112. Springer, Heidelberg (2012)
12. Kann, V.: On the Approximability of NP-complete Optimization Problems. Ph.D. thesis, Royal Institute of Technology of Stockholm (1992)
13. Ma, J., Zhang, L., Suh, B., Raney, B., Burhans, R., Kent, W., Blanchette, M., Haussler, D., Miller, W.: Reconstructing contiguous regions of an ancestral genome. Genome Research 16, 1557–1565 (2007)

14. Marron, M., Swenson, K.M., Moret, B.M.E.: Genomic distances under deletions and insertions. In: Warnow, T.J., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 537–547. Springer, Heidelberg (2003)
15. Moret, B., Wang, L., Warnow, T., Wyman, S.: New approaches for reconstructing phylogenies from gene order data. Bioinformatics 17, S165–S173 (2001)
16. Sankoff, D., Blanchette, M.: The median problem for breakpoints in comparative genomics. In: Jiang, T., Lee, D.T. (eds.) COCOON 1997. LNCS, vol. 1276, pp. 251–264. Springer, Heidelberg (1997)
17. Seymour, P.D.: Packing directed circuits fractionally. Combinatorica 15(2), 281–288 (1995)

Irrationality Is Needed to Compute with Signal Machines with Only Three Speeds

Jérôme Durand-Lose*

Laboratoire d'Informatique Fondamentale d'Orléans (LIFO), Université d'Orléans,
B.P. 6759, 45067 ORLÉANS Cedex 2, France
Jerome.Durand-Lose@univ-orleans.fr

Abstract. Space-time diagrams of signal machines on finite configurations are composed of interconnected line segments in the Euclidean plane. As the system runs, a network emerges. If segments extend only in one or two directions, the dynamics is finite and simplistic. With four directions, it is known that fractal generation, accumulation and any Turing computation are possible.

This communication deals with the three directions/speeds case. If there is no irrational ratio (between initial distances between signals or between speeds) then the network follows a mesh preventing accumulation and forcing a cyclic behavior. With an irrational ratio (here, the Golden ratio) between initial distances, it becomes possible to provoke an accumulation that generates infinitely many interacting signals in a bounded portion of the Euclidean plane. This behavior is then controlled and used in order to simulate a Turing machine and generate a 25-state 3-speed Turing-universal signal machine.

1 Introduction

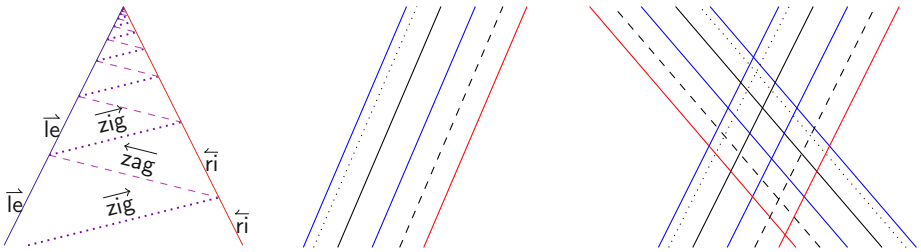
Imagine yourself with some color pencils and a sheet of paper together with ruler and compass. Some colored line segments are drawn and you are given rules so as to extend the drawing. According to the rules and the initial drawing/configuration you might stop soon, have to extend the paper indefinitely or draw forever in a bounded part of the paper as on top of Fig. 1(a). Could such a system compute?

This communication concentrates on the case where the dynamical system is a signal machine. In this setting, one drawing direction is distinguished and used as time axis. Line segments are enlarged synchronously until they intersect one another and get replaced. This goes on until no more collision can happen.

The line segments are the traces of *signals* and their intersections are *collisions*. Each signal corresponds to some *meta-signal*. During a collision, in-coming signals are removed and new ones are emitted according to the meta-signals associated to the incoming signals. This is called a *collision rule*. Signals that

* This work was partially funded by the ANR project AGAPE, ANR-09-BLAN-0159-03.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 108–119, 2013.



(a) Most basic accumulation (b) One speed:no collision (c) Two speeds: finitely many collisions

Fig. 1. Basic cases

correspond to the same meta-signal must travel at the same speed (or directions on the drawing) thus the resulting traces are parallel. There are finitely many meta-signals so there are finitely many collision rules.

The signals move on a one dimensional Euclidean space orthogonal to the temporal axis. Considering the traces leads to two dimensional drawings called *space-time diagrams* (as illustrated throughout the communication). Space and time are continuous ($\mathbb{R} \times \mathbb{R}^+$). Signals as well as collisions are dimensionless points. Computations are exact, there is no noise nor approximation.

Problematics. Signal machines are very powerful and colorful complex systems capable of computing in the classical Turing understanding [6]. This communication is in the line of minimality thresholds in order to have Turing-computability capability (like [19,20] for Turing machines and [3,18] for cellular automata; a more general picture is presented in [14]). This communication extends [10] on small Turing-universal signal machines (few meta-signals but 4+ speeds) and [1] that addresses only accumulation (i.e. not computation) with 3-speed signal machines.

Accumulations are easy to produce and are the cornerstone to hyper-computation in the model [9]. The present communication investigates the minimal number of speeds so that accumulations or Turing computations are possible (starting from a finite configuration). Four meta-signals of different speeds are enough to make an accumulation as depicted in Fig. 1(a). Four speeds and 15 meta-signals are enough to compute [10].

In this communication, only the number of different speeds is considered and in particular the case of three speeds. One speed does not allow any collision (see Fig. 1(b)). With two speeds, the number of collisions is finite and signals have to follow a regular grid which has no accumulation (see Fig. 1(c)). Three-speed signal machines with rational speeds and rational initial positions always enter a cyclic behavior with no accumulation and limited computing capability. But if an irrational ratio between distances in the initial configuration is allowed, then accumulations are possible as well as any Turing computation.

State of the Art. Signal machines are one of the unconventional models of computation dealing with Euclidean geometry together with Euclidean abstract machines [12,17], Piecewise Constant Derivatives systems [2] and colored universes [13].

Signal machines were originally introduced as a continuous counterpart of (discrete) cellular automata to provide a context for the underlying Euclidean (continuous) reasoning often found in the literature as well as to propose an abstract formalization of the concept of *signal* [15,16,8].

Accumulations provide a powerful tool to accelerate a computation, to do hyper-computation and analog computation [9]. Fractals can be generated and their construction modified so as to achieve massive parallelism and the capability to solve efficiently NSPACE-complete problems (Q-SAT in [4]).

In [1], the present author and colleagues already proved that irrational ratio between speeds or between initial distances is needed in order to have an accumulation with three speeds. They exhibit a geometrical implantation of the Euclid algorithm inside the computation. If all ratios are rational then this algorithm stops (generating a non-accumulating mesh) otherwise it goes on indefinitely provoking the accumulation. They also cover the case of an irrational ratio between speeds which is not addressed here.

Contribution. If all the ratios are rational, then some global regular *mesh* emerges. The signals have to be on that mesh which does not have any accumulation point. Moreover, being on this mesh ensures that the collisions have to be ultimately periodic so that the dynamics is ultimately cyclic and computing capability is limited. (The behavior is called *cyclic* and not *periodic* since at the configuration level, there is no periodicity.)

Using the Golden ratio between distances makes it possible to draw a fractal which accumulates. Because of self-similarity, ensuring one repetition step is enough.

With three speeds, it is straightforward to simulate a Turing machine on a bounded tape. But even though the tape remains finite, it cannot be bounded in the general case. From the fractal construction, a scheme is extracted to extend the tape *on demand* and get the full Turing computing power.

Outline. Section 2 provides all the definitions. Section 3 shows that in the rational case the dynamics is trapped into a mesh that does not allow any accumulation and restrain computing capability. Section 4 shows how to get an accumulation with non-rational ratio between distances. Section 5 provides the simulation of any Turing machine in such a case. Conclusion, remarks and perspectives are gathered in Sect. 6.

2 Definitions

This communication deals only with finite configurations so all definitions are restrained to this case.

A *signal machine* collects the definitions of available meta-signals, their speeds (positive for rightward signals and negative for leftward ones) and their collision rules. For example, the machine to generate Fig. 1(a) is composed of the following meta-signals (with speed): $\overrightarrow{\mathsf{le}}$ ($\frac{1}{2}$), $\overrightarrow{\mathsf{zig}}$ (4), $\overleftarrow{\mathsf{zag}}$ (-4), and $\overleftarrow{\mathsf{ri}}$ ($-\frac{1}{2}$). There are only two collision rules:

$$\left\{\overrightarrow{\mathsf{le}}, \overleftarrow{\mathsf{zag}}\right\} \longrightarrow \left\{\overrightarrow{\mathsf{le}}, \overrightarrow{\mathsf{zig}}\right\} \quad \text{and} \quad \left\{\overrightarrow{\mathsf{zig}}, \overleftarrow{\mathsf{ri}}\right\} \longrightarrow \left\{\overleftarrow{\mathsf{zag}}, \overleftarrow{\mathsf{ri}}\right\} \ .$$

It might happen that exactly three (or more) meta-signals meet. In such a case, collision rules involving three (or more) meta-signals are used. There can be any number of meta-signals in the range of a collision rule, as long as their speeds differ (i.e. they are not parallel).

Definition 1 (Signal machine). A *Signal machine*, (M, S, R), is defined by: M is a finite set of *meta-signals*, S is a function from meta-signals to real numbers, assigning *speeds*, and R is a deterministic set of *collision rules*. A collision rule is written $\rho = \rho^- \to \rho^+$ where ρ^- and ρ^+ are sets of meta-signals of different speeds, and ρ^- must have at least two meta-signals. The set of collision, rules, R is deterministic: $\rho \neq \rho'$ implies that $\rho^- \neq \rho'^-$.

A *configuration* is a function from the real line (space) into the set of meta-signals and collision rules plus one extra value $\oslash$ (nothing there). There should be finitely many non-$\oslash$ locations.

Definition 2 (Configuration). A *configuration*, c, is a function from the real line into meta-signals, rules, and the value $\oslash$ (let $V = M \cup R \cup \{\oslash\}$ so that $c : \mathbb{R} \to V$) such that $|c^{-1}(M \cup R)| < \infty$.

If there is a signal of speed s at x, then, unless it enters a collision before, after a duration Δt, its position is $x + s{\cdot}\Delta t$. At a collision, all incoming signals are immediately replaced by outgoing signals in the following configurations according to collision rules.

Definition 3 (Sequence of collision times). Considering a configuration, c, the *time to the next collision*, $\Delta(c)$, is equal to the minimum of the positive real numbers d such that:

$$\exists x_1, x_2 \in \mathbb{R}, \exists \mu_1, \mu_2 \in M \begin{cases} x_1 + d{\cdot}S(\mu_1) = x_2 + d{\cdot}S(\mu_2) \ , \\ c(x_1) = \mu_1 \vee (c(x_1) = \rho^- \to \rho^+ \wedge \mu_1 \in \rho^+) \ , \\ c(x_2) = \mu_2 \vee (c(x_2) = \rho^- \to \rho^+ \wedge \mu_2 \in \rho^+) \ . \end{cases}$$

It is $+\infty$ if there is no such d. The *sequence of collision times* is defined by: $t_0 = 0$, $t_{n+1} = t_n + \Delta(c_{t_n})$.

This sequence is finite if there is an n such that $\Delta(c_{t_n}) = +\infty$. Otherwise, since it is non-decreasing, it admits a limit. If the sequence is finite or its limit is infinite, then the whole space-time diagram is defined. Otherwise, there is an *accumulation* and the limit configuration is left undefined. Let c_t denote the configuration at time t.

Definition 4 (Dynamics between collisions). For t' between t and $t+\Delta(c_t)$, the configuration at t' is defined as follows. Signals are set: $c_{t'}(x') = \mu$ iff $c_t(x) =$

$\mu \vee (c_t(x) = \rho^- \to \rho^+ \wedge \mu \in \rho^+)$ where $x = x' + (t - t') \cdot S(\mu)$. (There is no collision to set.) It is $\oslash$ everywhere else.

Definition 5 (Dynamics at a collision time). For the configuration at $t' = t + \Delta(c_t)$, collisions are set first: $c_{t'}(x') = \rho^- \to \rho^+$ iff for all $\mu \in \rho^-$, $c_t(x_\mu) = \mu \vee (c_t(x_\mu) = \rho^- \to \rho^+ \wedge \mu \in \rho^+)$ where $x_\mu = x' + (t - t') \cdot S(\mu)$. Then meta-signals are set (where there is not already a collision). It is $\oslash$ everywhere else.

A *space-time diagram* is the collection of consecutive configurations which forms a two dimensional picture. In the space-time diagram in Fig. 1(a) the sequence of collision times is given by the collisions of $\overrightarrow{\mathsf{zig}}$ on $\overleftarrow{\mathsf{ri}}$, then $\overleftarrow{\mathsf{zag}}$ on $\overrightarrow{\mathsf{le}}$, then $\overrightarrow{\mathsf{zig}}$ on $\overleftarrow{\mathsf{ri}}$... This sequence accumulates on top of the space-time diagram.

Definition 6 (Rational signal machine). A signal machine is *rational* if all speeds are rational numbers and any non-$\oslash$ position in any initial configuration must also be a rational number.

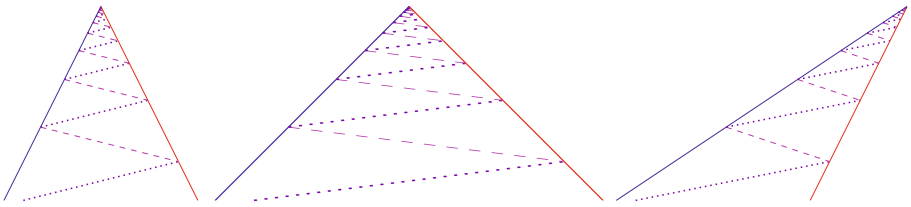
Since the position of collisions are solutions of systems of rational linear equations, they are rational. (Coordinates of accumulation may be non-rational [11].)

Definition 7 (Rational-like). A *signal machine* is *rational-like* if its speeds are rational up to a coefficient. (There is no restriction on possible initial configuration.) A *configuration* is *rational-like* if all the ratios between distances between signals are rational.

Linear Transformation. It is possible to linearly change the speed of the meta-signals or the positions in the initial configuration. As long as the coefficient is positive, the dynamics is not changed. This comes from the absence of any absolute origin or scale. This is not formally proved (this was done, e.g., in [5, Chap. 5]), but exemplified by the space-time diagrams in Fig. 2.

A linear transformation with a positive ratio of all the positions in the initial configuration only corresponds to changing the spatial origin and the space scale (Fig. 2(a)), time scale has to be changed accordingly.

A linear transformation with a positive ratio of all the speeds of the signal machine results in a change in the scale of time (but not of space, Fig. 2(b)). The term added corresponds to a drift, i.e., a slant in the time axis (Fig. 2(c)).



(a) Initial positions divided by 2 (b) Speeds multiplied by 2 (c) Speeds multiplied by 2 plus 1

Fig. 2. Linear transformations on the space-time diagram of Fig. 1(a)

Only locations are affected. The existence of a collision as well as the computing capability does not depend on linear transformations with positive ratio.

Normalization. When considering a finite set of real numbers, there is always a unique linear transformation with positive ratio that maps the two lowest values in the set to 0 and 1. We call this a *0-1-normalization.*

If all ratios between distances in a configuration are rational then its 0-1-normalization has only signals at rational coordinates: with O and I the signals with new coordinate 0 and 1, a signal at position M have coordinate $\overline{OM}/\overline{OI}$ which is rational since non-degenerated linear operators preserve ratio.

That ratios between distances are rational is both a sufficient and necessary condition to rescale into $\mathbb{Q}$. Up to normalization, the dynamics of a rational-like signal machine on a rational-like configuration is the one of a rational signal machine.

3 Rational 3-Speed Signal Machines

If speeds are rational up to a coefficient, then their 0-1-normalization results in rational speeds. The 0-1-normalized speeds are 0, 1, and (a rational value greater than 1) $1+p/q$ (with $p, q \in \mathbb{N}$, $1 \leq p, q$ and relatively prime). They are linearly transformed into $-q$, 0 and p.

The 0-1-normalized initial configuration is scaled so that all (signal) positions are natural numbers. The extreme positions are 0 and n.

Definition 8 ((p, q, n)-mesh). The (p, q, n)-*mesh* corresponds to the union of the following half-lines of $\mathbb{R} \times \mathbb{R}^+$:

- V_v: $x = v/(p+q)$ where $v \in \{0, 1, 2, \ldots, n(p+q)\}$,
- L_l: $x \leq n$, and $x + q.t = l/p$ where $r \in \{0, 1, 2, \ldots\}$ and
- R_r: $0 \leq x$ and $x - p.t = r/q$ where $l \in \{\ldots, -1, 0, 1, 2, \ldots, (n.q)\}$.

Figure 3 shows the display of such a mesh. Half-lines V_v are vertical, L_l have a negative slope and are dotted and R_r have a positive slope and are dashed.

A mesh has only L_l half-lines when $x < 0$ and only R_r half-lines when $n < x$. When $0 \leq x \leq n$, then all three kinds of lines are present and each every point

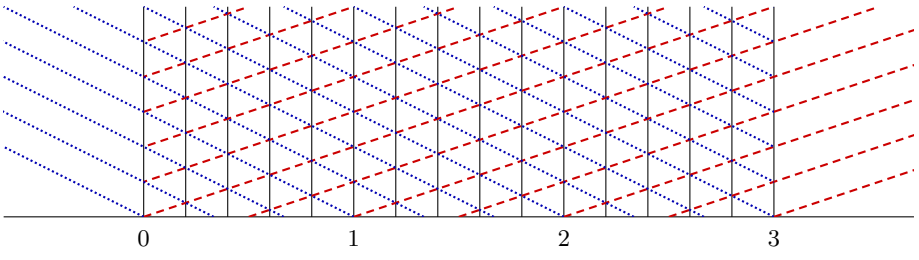


Fig. 3. $(3, 2, 3)$-mesh

of intersection is incident upon a line of each type. Moreover, there is a line of each kind starting at positions $(x, 0)$ when $x \in \{0, 1, 2, ..., n\}$.

Lemma 1. *During the computation of a rational machine with speeds $-q$, 0 and p from an initial configuration with no signal outside of $\{0, 1, 2, ..., n\}$, there is no signal outside the mesh (p, q, n).*

This is true because: initially signals are on the mesh on a half-line with the same speed/slope (all are issued from coordinates in $\{0, 1, 2, ..., n\}$); if a signal is on such half-lines, it remains upon it until it participates in a collision; and new signals appear only in collisions, collisions can only happen at line crossings and all three kinds of lines go on from there.

If the computation would lead to an accumulation, then this accumulation should also be on the mesh. But the mesh has no accumulation.

The mesh is time periodic in the region $0 \leq x \leq n$ and the computation in this part does not receive anything from the outside. When considering the configurations at time where lines intersect on $x = 0$, the mesh is exactly the same. On $0 \leq x \leq n$, these configurations can be described by a fixed length string on meta-signals and collision rules. Since this alphabet is finite (and the system is deterministic), this eventually enters a cycle.

Outside of the region $0 \leq x \leq n$, the mesh is also ultimately time-periodic. No collision happens there and the output signals also eventually enter a cycle. Rational 3-speed signal machines always enter a cyclic behavior. Transient time and cycle duration can be computed from the initial configuration and machine. These bounds prevent Turing-universality.

Lemma 2. *Rational 3-speed signal machines cannot produce accumulation and are not Turing-universal.*

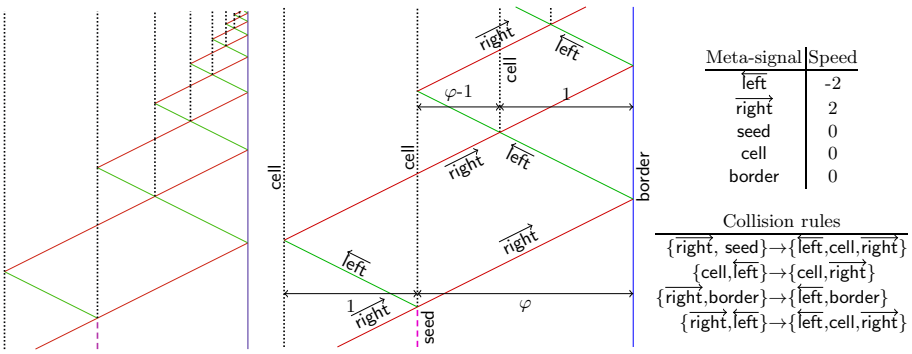
This also holds for rational-like machines on rational-like initial configurations.

4 Accumulation with the Golden Ratio

Using the Golden ratio, it is possible to generate the accumulating fractal in Fig. 4(a). The signal machine is directly read from the picture as depicted in Fig. 4(b). The signal machine is rational-like but not rational since the initial configuration is not.

The initial configuration is not part of the fractal cycle since we prefer to have all signals but one parallel. The distance between cell and seed is taken to be one. The exact position of $\overrightarrow{\text{right}}$ between cell and seed is not important. To position border, φ has to be given a value so that the fractal is generated, i.e., the ratio of the distance of the three last signals is preserved.

As indicated in Fig. 4(b), the first ratio is $\frac{\varphi}{1}$. To compute the next ratio, the distance between the third cell and border has to be 1 because of the presence of the parallelogram (with $\overleftarrow{\text{left}}$ and $\overrightarrow{\text{right}}$ sides) and the parallelism of the cell and border signals. Thus the other distance has to be $\varphi - 1$. The second ratio is $\frac{1}{\varphi - 1}$. By equaling these ratios, φ is $\frac{1+\sqrt{5}}{2}$, the Golden ratio.



(a) Fractal (b) Fractal construction

Fig. 4. Accumulating with three speeds and an irrational position

The construction then repeats forever. The ratio is preserved and the fractal is generated.

Lemma 3. *A 3-speed signal machine started on an irrational configuration can generate an accumulation, even when all speeds are rational.*

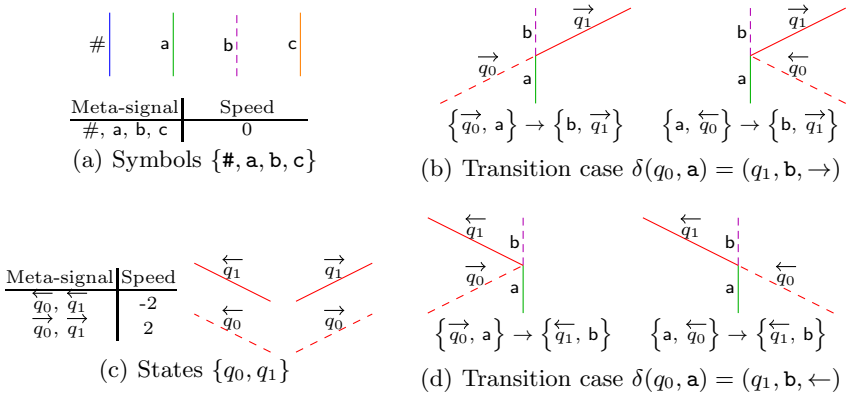
In fact, this is for any non-rational ratio. Moreover, any non-rational ratio between speeds can also be "transferred" into distances so that an accumulation can be produced [1].

5 Computing with the Golden Ratio

It is possible to start simulating a Turing machine with a rational 3-speed signal machine. Null speed signals are used to encode the cells of the tape. Each one encodes a symbol of the tape (# is the blank symbol). The head is encoded by a sequence of signals that record the state and move left or right (with the two other speeds). Each time it collides on a "symbol" signal, it updates the cell and the state and then goes left or right. This is done according to the transition table of the Turing machine. The meta-signals and collision rules are given on Fig. 5. An example of this simulation is given on the lower half of Fig. 6.

The tape is finite but not bounded. Starting from a finite configuration, the representation of the tape should be enlargeable. The fractal presented in previous section allows one to generate an unbounded sequence of sites (purposely named cell) to simulate the tape of a Turing machine. It is not fully generated to avoid an accumulation. A new cell is generated only when the tape needs enlargement.

An enlargement can be seen on the upper half of Fig. 6(b). It is started when a rightward state-encoding signal ($\overrightarrow{q_0}$ in the figure) reaches the border signal, this means that the head is looking for a cell that does not exist yet. Signal



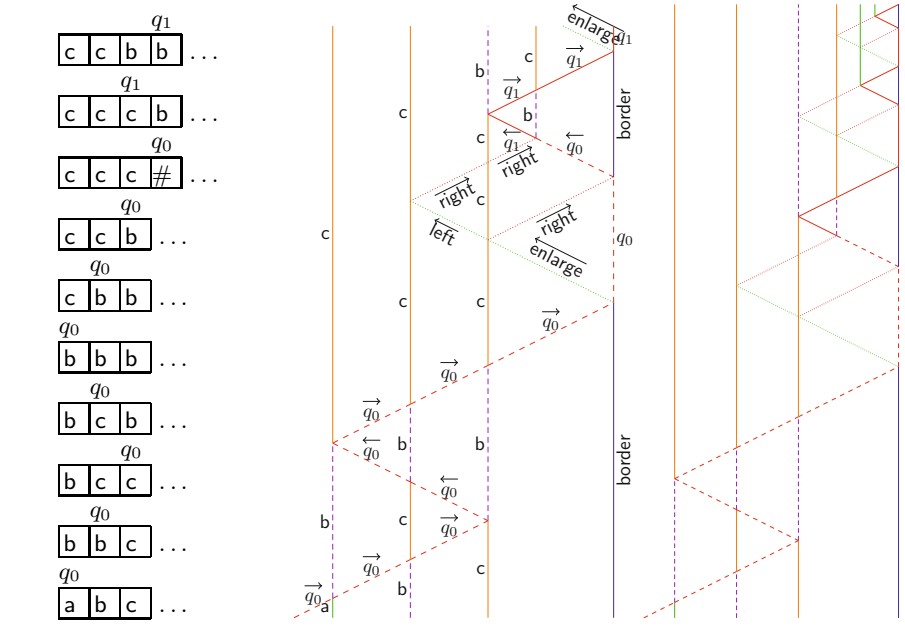
(a) Symbols $\{\#, \mathsf{a}, \mathsf{b}, \mathsf{c}\}$

(b) Transition case $\delta(q_0, \mathsf{a}) = (q_1, \mathsf{b}, \rightarrow)$

(c) States $\{q_0, q_1\}$

(d) Transition case $\delta(q_0, \mathsf{a}) = (q_1, \mathsf{b}, \leftarrow)$

Fig. 5. Basic encoding of a Turing machine



(a) Turing machine run

(b) Labeled simulation

(c) Longer simulation

Fig. 6. Computing with three speeds and an irrational position

border is replaced by a motionless signal encoding the state (q_0 in the figure) and preserving the position of border. Simultaneously, an $\overleftarrow{\text{enlarge}}$ signal is sent on the left to create the new cell.

This creation uses the same parallelogram construction as before (dotted signals $\overleftarrow{\mathsf{left}}$ and $\overrightarrow{\mathsf{right}}$) and closed on top by $\overleftarrow{q_0}$. The lower $\overrightarrow{\mathsf{right}}$ is used to restore border and set $\overleftarrow{q_0}$ on movement. The latter, instead of colliding with a symbol signal, collides with the upper $\overrightarrow{\mathsf{right}}$. The collision happens as if it were on the blank symbol. The needed meta-signals and collision rules are given in Fig. 7.

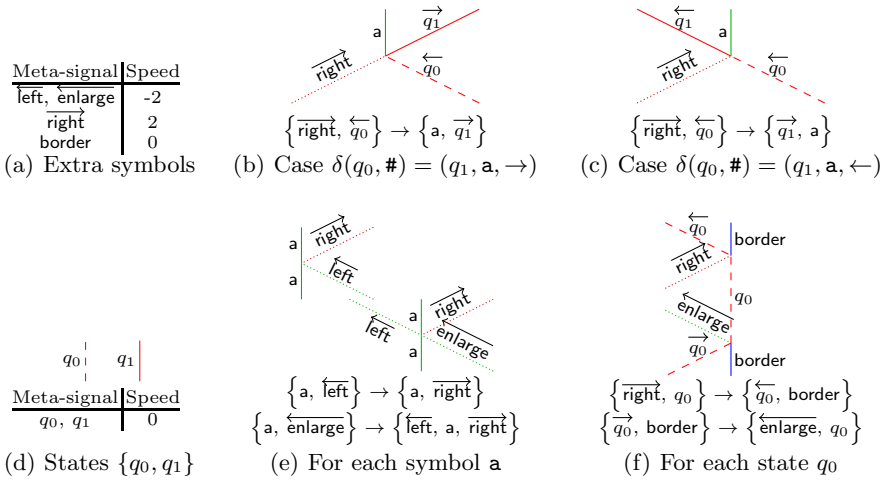


Fig. 7. Extra encoding to enlarge the simulation

6 Conclusion and Perspectives

With three speeds, it is possible to accumulate only if there is an irrational ratio between initial positions or between speeds (see [1] for speeds).

The number of meta-signals for the Turing-machine simulation with the Golden ratio is 1 for each symbol plus 3 for each state plus 4. With the small universal Turing machines listed in [20], a Turing-universal 25-meta-signal 3-speed signal machine using a Golden ratio distance can be constructed.

With a Golden ratio location, undecidable problems arise. Directly from classical computability theory, to have only finitely many collisions (or to enter a cyclic behavior) is not decidable. In the simulation of a Turing machine, if the head goes on the right each time, then an accumulation is generated. For an accumulation to happen, the head would have to explore all the cells on the right which is also not decidable. (In the general case, forecasting an accumulation is Σ_2^0-complete [7].)

An irrational ratio is an important piece of information (it could encode the halting problem). We conjecture that it is possible to use it as an oracle.

In [9], accumulations are used in order to hyper-compute, with an irrational ratio, computations and accumulations become possible, we conjecture that it is possible to solve the halting problem and even to climb the finite levels of the arithmetic hierarchy with three speeds and the Golden ratio.

Signal machines are know to be able to do analog computations [9] with real numbers encoded in distances. Up to what extent is analog computation possible with three speeds?

References

1. Becker, F., Chapelle, M., Durand-Lose, J., Levorato, V., Senot, M.: Abstract geometrical computation 8: Small machines, accumulations & rationality. Draft (2013)
2. Bournez, O.: Some bounds on the computational power of piecewise constant derivative systems (extended abstract). In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 143–153. Springer, Heidelberg (1997)
3. Cook, M.: Universality in elementary cellular automata. Complex Systems 15, 1–40 (2004)
4. Duchier, D., Durand-Lose, J., Senot, M.: Computing in the fractal cloud: modular generic solvers for SAT and Q-SAT variants. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 435–447. Springer, Heidelberg (2012)
5. Durand-Lose, J.: Calculer géométriquement sur le plan – machines à signaux. Habilitation à Diriger des Recherches, École Doctorale STIC, Université de Nice-Sophia Antipolis (2003) (in French)
6. Durand-Lose, J.: Abstract geometrical computation: Turing-computing ability and undecidability. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 106–116. Springer, Heidelberg (2005)
7. 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)
8. Durand-Lose, J.: The signal point of view: from cellular automata to signal machines. In: Durand, B. (ed.) Journées Automates Cellulaires (JAC 2008), pp. 238–249 (2008)
9. Durand-Lose, J.: Abstract geometrical computation 3: Black holes for classical and analog computing. Nat. Comput. 8(3), 455–472 (2009)
10. Durand-Lose, J.: Abstract geometrical computation 4: small Turing universal signal machines. Theoret. Comp. Sci. 412, 57–67 (2011)
11. Durand-Lose, J.: Abstract geometrical computation 7: Geometrical accumulations and computably enumerable real numbers. Nat. Comput. 11(4), 609–622 (2012), Special issue on Unconv. Comp. 2011
12. Huckenbeck, U.: Euclidian geometry in terms of automata theory. Theoret. Comp. Sci. 68(1), 71–87 (1989)
13. Jacopini, G., Sontacchi, G.: Reversible parallel computation: an evolving space-model. Theoret. Comp. Sci. 73(1), 1–46 (1990)
14. Margenstern, M.: Frontier between decidability and undecidability: A survey. Theor. Comput. Sci. 231(2), 217–251 (2000)
15. Mazoyer, J.: Computations on one dimensional cellular automata. Ann. Math. Artif. Intell. 16, 285–309 (1996)

16. Mazoyer, J., Terrier, V.: Signals in one-dimensional cellular automata. Theoret. Comp. Sci. 217(1), 53–80 (1999)
17. Mycka, J., Coelho, F., Costa, J.F.: The euclid abstract machine: Trisection of the angle and the halting problem. In: Calude, C.S., Dinneen, M.J., Păun, G., Rozenberg, G., Stepney, S. (eds.) UC 2006. LNCS, vol. 4135, pp. 195–206. Springer, Heidelberg (2006)
18. Ollinger, N., Richard, G.: Four states are enough? Theoret. Comp. Sci. 412(1-2), 22–32 (2011)
19. Rogozhin, Y.V.: Small universal Turing machines. Theoret. Comp. Sci. 168(2), 215–240 (1996)
20. Woods, D., Neary, T.: The complexity of small universal Turing machines: A survey. Theoret. Comp. Sci. 410(4-5), 443–450 (2009)

Processes Inspired by the Functioning of Living Cells: Natural Computing Approach

Andrzej Ehrenfeucht[1] and Grzegorz Rozenberg[1,2]

[1] Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309, U.S.A.
[2] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Niels Bohrweg 1, 2300 RA Leiden, The Netherlands
rozenber@liacs.nl

Natural Computing (cf., e.g., [12,13]) is concerned with human-designed computing inspired by nature as well as with computation taking place in nature, i.e., it investigates models, computational techniques, and computational technologies inspired by nature as well as it investigates, in terms of information processing, phenomena/processes taking place in nature.

Examples of the first strand are evolutionary, neural, molecular, and quantum computation, while examples of the second strand are investigations into the computational nature of self-assembly, the computational nature of developmental processes and the computational nature of biochemical reactions. Obviously, the two research strands are not disjoint.

A computational understanding of the functioning of the living cell is one of the research topics from the second strand. A motivation for this research is nicely formulated by Richard Dawkins, a world leading expert in evolutionary biology: "If you want to understand life, don't think about vibrant throbbing gels and oozes, think about information technology", cf. [4].

We view this functioning in terms of formal processes resulting from interactions between individual reactions, where these interactions are driven by two mechanisms, facilitation and inhibition: reactions may (through their products) facilitate or inhibit each other.

We present a formal framework for the investigation of processes resulting from these interactions. We provide the motivation by explicitly stating a number of assumptions that hold for these interactive processes, and we point out that these assumptions are very different from assumptions underlying traditional models of computation.

The central formal model of our framework, reaction systems (cf. [1,5,10]), follows the philosophy of processes outlined above, and moreover:

(1) it takes into account the basic bioenergetics (flow of energy) of the living cell,
(2) it abstracts from various technicalities of biochemical reactions to the extent that it becomes a qualitative rather than a quantitative model, and

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 120–122, 2013.

(3) it takes into account the fact that the living cell is an open system and so its behavior (expressed by formal processes) is influenced by its environment.

Our full formal framework (cf. [1,5]) contains also models that are extensions of the basic model of reaction systems. The research themes investigated within this framework are motivated either by biological considerations or by the need to understand the underlying computations. Some examples of these themes are:

- the notion of time in reaction systems, cf. [11],
- formation of modules in biological systems, cf. [9,16],
- understanding decay and its influence on interactive processes, cf. [3],
- how to include in our framework quantitative aspects of processes in living cells, cf. [1,5,9,11],
- static and dynamic causalities, cf. [2],
- the nature of state transitions in reaction systems, cf. [6,14,8,15].

We (hope to) demonstrate that the framework of reaction systems is:

(i) well motivated by and relevant for biological considerations, and
(ii) novel and attractive from the theory of computation point of view.

References

1. Brijder, R., Ehrenfeucht, A., Main, M.G., Rozenberg, G.: A tour of reaction systems. International Journal of Foundations of Computer Science 22(7), 1499–1517 (2011)
2. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: A note on Causalities in Reaction Systems. Electronic Communications of ECASST 30 (2010)
3. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction Systems with Duration. In: Kelemen, J., Kelemenová, A. (eds.) Păun Festschrift. LNCS, vol. 6610, pp. 191–202. Springer, Heidelberg (2011)
4. Dawkins, R.: The Blind Watchmaker. Penguin, Harmondsworth (1986)
5. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Qualitative and Quantitative Aspects of a Model for Processes Inspired by the Functioning of the Living Cell. In: Katz, E. (ed.) Biomolecular Information Processing. From Logic Systems to Smart Sensors and Actuators, pp. 303–322. Wiley-VCH Verlag, Weinheim (2012)
6. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Minimal reaction systems. In: Priami, C., Petre, I., de Vink, E. (eds.) Transactions on Computational Systems Biology XIV. LNCS (LNBI), vol. 7625, pp. 102–122. Springer, Heidelberg (2012)
7. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Combinatorics of life and death for reaction systems. International Journal of Foundations of Computer Science 21(3), 345–356 (2010)
8. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Functions defined by reaction systems. International Journal of Foundations of Computer Science 21(1), 167–178 (2011)
9. Ehrenfeucht, A., Rozenberg, G.: Events and Modules in Reaction Systems. Theoretical Computer Science 376(1-2), 3–16 (2007)
10. Ehrenfeucht, A., Rozenberg, G.: Reaction Systems. Fundamenta Informaticae 75(1-4), 263–280 (2007)

11. Ehrenfeucht, A., Rozenberg, G.: Introducing Time in Reaction Systems. Theoretical Computer Science 410(4-5), 310–322 (2009)
12. Kari, L., Rozenberg, G.: The many facets of natural computing. Communications of the ACM 51(10), 72–83 (2008)
13. Rozenberg, G., Bäck, T., Kok, J. (eds.): Handbook of Natural Computing. Springer (2012)
14. Salomaa, A.: On state sequences defined by reaction systems. In: Constable, R.L., Silva, A. (eds.) Kozen Festschrift. LNCS, vol. 7230, pp. 271–282. Springer, Heidelberg (2012)
15. Salomaa, A.: Functions and sequences generated by reaction systems. Theoretical Computer Science 466, 87–96 (2012)
16. Schlosser, G., Wagner, G.P. (eds.): Modularity in Development and Evolution. The University of Chicago Press, Chicago (2004)

Recent Developments in Collective Decision Making in Combinatorial Domains

Ulle Endriss

Institute for Logic, Language and Computation
University of Amsterdam
The Netherlands

Collective decision making is the process of mapping the individual views of several individual agents into a joint decision. The need for collective decision making mechanisms is abundant, not just in the realm of politics, but also for a wide range of scientific and technological applications. These include, for instance, logistics, grid computing, and recommender systems. The alternatives to be decided upon often have a combinatorial structure: an alternative is characterised by a tuple of variables, each ranging over a finite domain. Classical approaches to collective decision making, developed in social choice theory, do not take the computational limitations induced by the combinatorial nature of the problem into account. For instance, if we are asked to elect a committee consisting of k representatives, choosing from a pool of n candidates, then we are in fact faced with a social choice problem with $\binom{n}{k}$ alternatives. Asking each voter for their full preferences over these $\binom{n}{k}$ alternatives may not be feasible in practice. Similarly, if we have to choose between accepting or rejecting each of n different propositions, then we are dealing with a social choice problem with 2^n possible outcomes.

In this talk I will report on a number of recent developments in the area of collective decision making in combinatorial domains. This includes the study of languages for the compact representation of preferences [2,4], the design of voting rules for multi-issue elections [1], and the analysis of the computational complexity of decision problems arising in judgment aggregation [3].

References

1. Airiau, S., Endriss, U., Grandi, U., Porello, D., Uckelman, J.: Aggregating dependency graphs into voting agendas in multi-issue elections. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011 (2011)
2. Bouveret, S., Endriss, U., Lang, J.: Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009 (2009)
3. Endriss, U., Grandi, U., Porello, D.: Complexity of judgment aggregation. Journal of Artificial Intelligence Research 45, 481–514 (2012)
4. Uckelman, J., Chevaleyre, Y., Endriss, U., Lang, J.: Representing utility functions via weighted goals. Mathematical Logic Quarterly 55(4), 341–361 (2009)

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, p. 123, 2013.

Software Streams: Big Data Challenges in Dynamic Program Analysis

Irene Finocchi

Computer Science Department, *Sapienza* University of Rome,
Via Salaria, 113, 00198 Rome, Italy
finocchi@di.uniroma1.it

Abstract. Dynamic program analysis encompasses the development of techniques and tools for analyzing computer software by exploiting information gathered from a program at runtime. The impressive amounts of data collected by dynamic analysis tools require efficient indexing and compression schemes, as well as on-line algorithmic techniques for mining relevant information on-the-fly in order to identify frequent events, hidden software patterns, or undesirable behaviors corresponding to bugs, malware, or intrusions. The paper explores how recent results in algorithmic theory for data-intensive scenarios can be applied to the design and implementation of dynamic program analysis tools, focusing on two important techniques: sampling and streaming.

1 Introduction

In our modern society, software has become ubiquitous in many branches of human activities and has gained an unprecedented level of complexity. This poses many challenges regarding reliability, performance, and scalability on contemporary computing platforms, thus calling for a much deeper understanding of what happens inside a software program than the conventional visibility offered by the program's output. Dynamic program analysis, defined in [1] as "*the analysis of the properties of a running software system*", encompasses the development of techniques and tools for analyzing computer software by exploiting information gathered at runtime. It can be used for a variety of tasks [2], including optimization (profiling, tracing, self-configuration), error detection (testing, assertion checking, type checking, memory safety, leak detection), error correction (runtime data structure repair, protections against security attacks), and program understanding (coverage, call graph construction, invariant detection, software visualization).

Over the past few years, dynamic analysis has emerged as a focused subject aimed at bridging the gap between the complexity-haunted field of formal verification and the ad-hoc field of testing. Being run-time information precise and sensitive to the input data, dynamic analysis can complement and reinforce traditional static analysis techniques, which might be inaccurate in modern object-oriented software systems: since software is often deployed as a collection of dynamically linked libraries or as Java bytecode that is delivered dynamically

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 124–134, 2013.

and on demand, compilers and other programming tools know less and less of the finally executing program. The use of static analysis in such programming tools requires conservative assumptions, which yield analysis results that may be too imprecise to be useful for either program optimization or program understanding tasks. In these contexts, dynamic analysis can enable new powerful techniques that would be impossible to achieve otherwise.

The development of dynamic analysis tools that can successfully assist programmers and software engineers raises issues in a variety of areas, including operating systems, algorithm design, software engineering, and programming languages. In particular, optimizing the performance of dynamic analysis tools is of crucial importance for their effective deployment and usability. For instance, tools that analyze the patterns of memory accesses of a running program, such as memory profilers, debuggers, or invariant checkers, must be able to deal with huge streams of data generated on-the-fly by monitoring traffic on the address bus and data bus at typical rates of several megabytes per second. Two main problems arise in this context. Firstly, since dynamic program analysis routines are inlined with program execution, they can substantially impact system performance, greatly reducing their practical applicability: the cost of collecting runtime information must be therefore appropriately lowered by means of available hardware/software support. Secondly, the sheer size of data collected by a dynamic analysis tool requires on-line techniques for mining relevant information on-the-fly, as well as efficient indexing and compression schemes for storing the data for post-mortem examination.

While optimizing the costs of instrumentation and analysis can largely boost the performance of dynamic analysis tools, it is a very difficult task: modern computer systems must deal with billions of events per second, such as instruction executions, accesses to main memory and caches, or packet forward operations. Hence, execution traces generated from real applications, even from very short runs, can be overwhelmingly large and processing them is very time-consuming. Exploiting advanced algorithmic techniques to cope with the sheer size of data collected throughout execution is thus regarded as a key challenge in this field [3, 4]. In the last few years the design of algorithms and data structures for handling massive data sets has sparked a lot interest in the algorithmic community, but this wealth of novel algorithmic techniques has been explored only to a very little extent in dynamic program analysis. In this paper we shall discuss a few relevant examples where big-data algorithmics has provided valuable insights in the design and implementation of dynamic program analysis tools, addressing two important techniques: sampling (Section 3) and streaming (Section 4).

2 Execution Traces

Information collected by dynamic analysis tools is typically expressed in the form of execution traces. Traces can be recorded via different instrumentation techniques at the source code, binary code, or execution environment level [5, 6],

Table 1. Data obtained from execution traces of routine invocations. The number of nodes in the call tree is proportional to the trace length.

Application	|Call graph|	|Call sites|	|Call tree|
amarok	13 754	113 362	991 112 563
ark	9 933	76 547	216 881 324
audacity	6 895	79 656	924 534 168
firefox	6 756	145 883	625 133 218
gedit	5 063	57 774	407 906 721
gimp	5 146	93 372	805 947 134
sudoku	5 340	49 885	325 944 813
inkscape	6 454	89 590	675 915 815
ooimpress	16 980	256 848	730 115 446
oowriter	17 012	253 713	563 763 684
pidgin	7 195	80 028	404 787 763
quanta	13 263	113 850	602 409 403

and can contain a variety of information related to, e.g., routine invocations, executions of program statements or basic blocks, memory accesses, and thread operations. The information recorded in a trace clearly depends on software properties that need to be analyzed. With respect to static analysis, execution traces are incomplete, since they capture only a small fraction of all possible execution paths. However, they have the advantage of being extremely precise and sensitive to the input data.

Even traces obtained from short runs of real applications can be extremely large and complex, affecting not only performance and storage of dynamic analysis tools, but also the cognitive load humans can deal with. Consider, as an example, traces of routine invocations, which are especially useful for performance profiling. These traces can be naturally regarded as a stream of tuples containing routine name, call site, event type (i.e., routine enter or exit), and possibly timing information. Table 1 is excerpted from [7] and analyzes a variety of prominent Linux applications, for which traces were obtained from short running sessions of just a few minutes. The table reports the number of distinct routines in a trace (i.e., the number of nodes of the call graph), the number of distinct call sites (i.e., the number of code lines which call a routine), and the number of nodes in the call tree. The call graph and the call tree are fundamental data structures that maintain information about interprocedural control flow: in a call graph, nodes represent routines and arcs caller-callee relationships, while each node of a call tree represents a different routine invocation. The number of call tree nodes is thus proportional to the stream length. Table 1 shows that the number of call tree nodes can be very large, even when compared with call graph nodes and call sites: execution traces obtained from short runs of real applications produce a few hundred millions of events, which result in a few GigaBytes of memory under the optimistic assumption that each stream tuple can be stored using only ten bytes. To mitigate this size explosion issue, many trace simplification and abstraction techniques have been proposed in the literature,

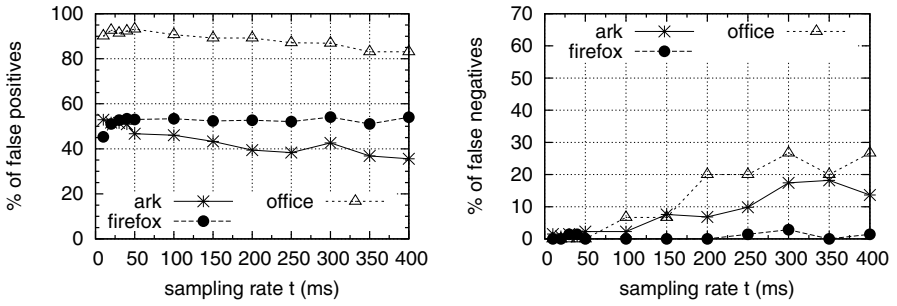
aimed at extracting high-level views and relevant data from long raw traces: execution traces can indeed contain several repetitions, either contiguous or not, and a very large number of patterns. Each pattern, in turn, can have thousands of occurrences, which makes data mining and pattern detection techniques quite useful to understand the characteristics of program traces [4]. Redundancies can be also reduced by compression techniques as proposed, e.g., in [8–10]. In the rest of this paper we shall describe some relevant trace analysis techniques based on sampling and data stream algorithmics.

3 Sampling

Sampling is used in statistics to estimate the characteristics of a large population by analyzing only a small portion of its members. A sample typically represents a subset of the population of manageable size, thus allowing faster data collection and smaller analysis costs. Many previous works, such as [11–16], have explored the use of sampling to reduce the size of execution traces and/or the runtime overhead of dynamic analysis tools. Overall, sampling appears to be a valuable tool in dynamic analysis, although sampled traces are not always representative of the original ones, and the results often heavily depend on manual tuning of a variety of parameters. Furthermore, it has been observed that the same sampling parameters might work well for one trace, while being inappropriate for a different trace [12].

Fixed Rate Sampling. A widespread approach consists of selecting sample points at fixed intervals, e.g., one point out of n trace items or every t milliseconds. This technique might be easily biased when the original trace exhibits regular patterns. Consider, for instance, the following scenario: the execution trace stores memory accesses, the sampling distance n is set to 10, and a memory location ℓ is accessed exactly every 10 memory operations. Then, depending on where the sampling starts, the traced sample might never contain ℓ or might contain exclusively operations on ℓ. Though this is a worst-case example, such regularities in execution traces are far from being rare.

A Case Study in Performance Profiling. Fixed rate sampling can be naturally implemented using periodic timer interrupts, ranging from process level interrupts to processor hardware performance counters. Hence, it has become very popular in the design of performance profilers. In accordance with the well-known Pareto principle (also known as the $80-20$ rule), more than 80% of running time is indeed spent in only 20% of routines: "hot" routines must appear frequently in the trace, and therefore are likely to be sampled. Unfortunately, even for such skewed distributions, fixed rate sampling might not work properly if the sampling parameter (n or t) is not appropriately tuned. As a concrete example, we report the outcome of an experiment discussed in [17], aimed at comparing the set of hot routines returned by a sampling-based profiler with the set of hot routines obtained by full instrumentation (i.e., computed on the full execution



(a) (b)

Fig. 1. False positives and false negatives generated by fixed rate sampling. (a) A routine is a false positive if it appears to be hot with respect to a sampled trace, but is not hot with respect to the full trace; (b) a routine is a false negative if is not reported as hot when using sampling, but is hot with respect to the full trace. Graphs are excerpted from [17].

trace). Let τ be a trace of routine invocations and $T(\tau)$ be the total time required by all the routines appearing in τ. We define the set $H(\tau)$ of routines that are hot with respect to trace τ as follows: all routines appearing in τ are sorted by decreasing running time and are then progressively added to $H(\tau)$, according to the precomputed order, until the total time required by routines in $H(\tau)$ becomes larger than $0.9\,T(\tau)$. Intuitively, $H(\tau)$ is the minimal set of the most costly routines that account for at least 90% of the overall running time. When τ is a sampled trace, some of the routines that are hot with respect to τ could not be hot with respect to the full trace, yielding false positives. Symmetrically, it may happen that routines that are hot with respect to the full trace are not hot with respect to the sampled trace τ, yielding false negatives. Figure 1 reports the percentage of false positives and false negatives as a function of the sampling rate t (larger values of t imply less frequent sampling and thus smaller sampled traces). The outcome of the experiment is exemplified on some of the benchmarks listed in Table 1. Both quantities are considerably large: in some applications, false positives account for up to 90% of the total number of reported hot routines. The quantity of false negatives is smaller, but remains non-negligible and is badly affected by larger sampling rates. Further details are given in [17].

Random Sampling. Overall, the experiments reported in [17] confirm that fixed rate sampling can yield unrealistic results, even when data distributions are very skewed. A valuable tool to mitigate these issues is random sampling, which consists of selecting sample points with a fixed probability. Mytkowicz *et al.* [18] observe that collecting samples randomly is a fundamental requirement to obtain accurate profiles, but is often violated by commonly-used Java profilers. Unfortunately, random sampling might be difficult to implement on-line (i.e., during trace generation) and, if not done properly, might result in samples of

unbounded size for long running applications. To overcome these issues, Coppa *et al.* [17] advocate the use of reservoir sampling [19]: the experimental evaluation of reservoir-based profiling shows that, while maintaining uniform sampling probability, this technique yields much better and more stable profiling results than fixed rate sampling, even when the stored sample is very small.

Adjusting Sampling Probabilities. A variety of works propose different strategies for controlling sampling probabilities. For instance, Marino, Musuvathi, and Narayanasamy apply sampling to the problem of detecting data races in concurrent applications [20]. Apparently, a sampling-based data-race detector may seem unlikely to succeed, because most memory accesses do not participate in data races and sampling approaches, in general, are not well suited at capturing rare events. Hence, [20] proposes an adaptive approach, adjusting sampling probabilities during the execution so that infrequently accessed regions of code progressively become more likely to be sampled. Experimental results show that this adaptive strategy achieves both a high detection rate and small slowdown on the running time. In [15], Pirzadeh *et al.* use stratified sampling to create samples representative of different characteristics of the entire execution. Stratified sampling turns out to be useful on heterogeneous populations that can be divided into homogeneous sub-populations, known as strata: this is often the case in execution traces, where the sequence of trace events can be typically partitioned into subsequences representing specific tasks performed by the software system. In this scenario, strata correspond to execution phases, which can be automatically identified using k-means clustering algorithms, and different sampling parameters can be then used in each stratum.

4 Streaming

The data streaming model has gained increasing popularity in the last few years as an effective paradigm for processing massive data sets. Streaming algorithms are well suited in application domains where input data come at a very high rate, cannot be stored entirely due to their huge (possibly unbounded) size, and need to be continuously monitored in order to support exploratory analyses and to detect correlations, frequent or rare events, fraud, intrusion, and anomalous activities. Relevant examples include monitoring network traffic, online auctions, transaction logs, telephone call records, automated bank machine operations, atmospheric and astronomical events. For instance, in IP traffic analysis we may want to monitor the packet log over a given link in order to estimate how many distinct IP addresses used that link in a given period of time: since the number of packets may be very large and stream items (source-destination IP address pairs) are drawn from a large universe, a space-efficient data streaming algorithm maintains a compact data structure supporting both dynamic updates upon arrival of new packets and distinct items queries. Approximate answers are allowed when it is impossible to obtain an exact solution using only one sequential pass over the data and limited space.

One-pass streaming algorithms are typically designed to optimize four main performance measures: space required to store the data structure, update time (i.e., per-item processing time), query time, and guaranteed solution quality. Starting from early papers appeared in the late 1970s (cf., e.g., [21, 22]), a wide range of results have been obtained in the last decade, mainly for statistics and data sketching problems such as the computation of frequency moments [23], histograms and quantiles [24], norm estimation [25], wavelet decomposition [24], most frequent items [26], and clustering [27]. In this section we discuss two applications of streaming algorithms to dynamic program analysis, focusing on the problem of performance profiling.

Range Adaptive Profiling. In [28], Mysore *et al.* address a problem called profiling with adaptive precision: they devise a profiling methodology capable of hierarchically classifying items in an execution trace into increasingly precise categories based on the frequency with which they occur. Differently from traditional profiles, which produce a flat list of items together with their performance metrics, adaptive profiling outputs profile data into a hierarchical fashion, striving for higher precision on most frequent events. Assume, as an example, that items of interest are lines of code: if 90% of the running time is spent on the top half of the code, according to Amdahl's law fine-grained profile data on the bottom half would not be very useful. Hence, it makes sense to summarize the behavior of the bottom half using a single performance counter, exploring in more detail possible optimization targets in the top half: the top half could be divided in turn into a top and a bottom quarter, refining data on the quarter that consumes most of the time. In summary, in [28] profile data is grouped into ranges: the most frequently occurring ranges are broken down into more precise subranges, while the least frequently occurring events are kept as larger ranges.

Ranges can be naturally stored in a tree, together with their associated counters. The tree can be easily updated by incrementing the appropriate counter to keep track of stream events. However, since relative range frequencies can dynamically change over time, the tree structure must be also adaptively changed to resemble the hottest ranges. The solution proposed in [28] exploits a streaming algorithm for adaptive spatial partitioning of multidimensional data streams described in [29]. When a range becomes sufficiently hot, the corresponding tree node is split into subranges. Symmetrically, ranges that get colder are merged together, pruning the tree in order to maintain the least number of relevant counters. Tree update, split, and merge operations can be performed on-line and are designed so as to guarantee worst case bounds on precision and space usage.

Let $[0, R]$ be the largest range to be considered (in the example above, R might denote the number of code lines), let $\varepsilon \in (0, 1)$ be a user defined constant, and let n be the number of stream items at any time during the execution. The streaming data structure of [29] splits nodes whenever their counter becomes larger than the threshold $\varepsilon \cdot n / \log(R)$. It can be proved that this guarantees $O(\log R/\varepsilon)$ size, independently of the length of the stream, and maximum error upper bounded by $\varepsilon \cdot n$. We notice that the error for any given range is relative to the entire input stream, and not to the actual counter of that range.

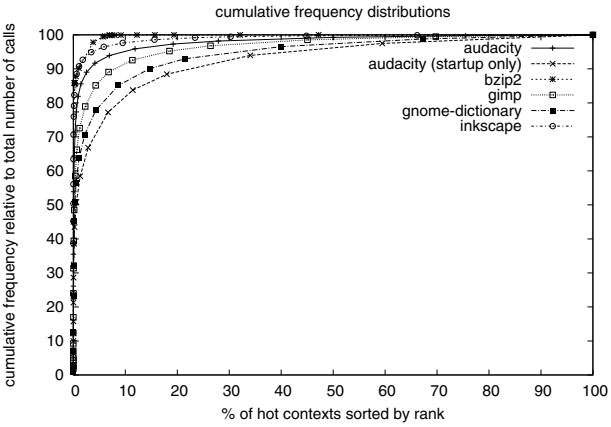


Fig. 2. Skewness of calling context distribution (graphs excerpted from [7])

In [28], the authors show that this streaming approach can be efficiently implemented via specialized hardware. Experimental results indicate that, using just a few kilobytes of memory, it is possible to maintain range profiles with an average accuracy of 98%. We remark that range adaptive profiling is not a fully general technique, but can be nevertheless applied to a variety of scenarios, such as profiling segments of code, blocks of data and IP addresses, or ranges of memory addresses in order to quantify cache locality issues.

Mining Hot Calling Contexts. In [7], D'Elia *et al.* show an efficient and accurate solution for context sensitive profiling based on the computation of frequent items in the data stream model. Calling contexts are typically stored in a data structure called *calling context tree*, which is substantially smaller than standard call trees, but may still be very large and difficult to analyze in real applications. However, only the most frequent contexts are of interest, since they represent the hot spots to which optimizations must be directed. Context frequency distribution satisfies the well-known Pareto principle: Figure 2, excerpted from [7], shows on a variety of real applications that only a small fraction of contexts are hot, and typically more than 90% of routine calls take place in only 10% of calling contexts. This skewness suggests that space could be greatly reduced by keeping information about hot contexts only, discarding on the fly contexts that are likely to be cold. This is the approach taken in [7], where the problem of identifying the most frequent contexts on the fly is cast into a data streaming setting, exploiting fast and space-efficient algorithms for mining frequent items.

Given a frequency threshold $\phi \in [0, 1]$ and a stream of lenght N, the frequent items problem is to find all items that appear in the stream at least $\lfloor \phi N \rfloor$ times. Since computing an exact solution requires $\Omega(N)$ bits, even using randomization [30], research focused on solving an approximate version of the problem, called (ϕ, ε)-*heavy hitters*: given two parameters $\phi, \varepsilon \in [0, 1]$, with $\varepsilon < \phi$, return

all items with frequency $\geq \lfloor \phi N \rfloor$ and no item with frequency $\leq \lfloor (\phi - \varepsilon)N \rfloor$. In the approximate solution, false negatives cannot exist, i.e., all frequent items must be returned. Instead, false positives are allowed, but their real frequency must be guaranteed to be at most εN-far from the threshold $\lfloor \phi N \rfloor$. Different algorithms for computing (ϕ, ε)-heavy hitters are known in the literature. Among them, *Space Saving* [31] and *Sticky Sampling* [26] are counter-based algorithms that track a subset of the input items, monitoring counts associated with them. For each new arrival, they decide whether to store the item or not, and, if so, what counts to associate with it. Sticky Sampling is probabilistic: it fails to produce the correct answer with a minuscule probability, say δ, and uses at most $\frac{2}{\varepsilon} \log(\phi^{-1}\delta^{-1})$ entries in its data structure [26]. Space Saving [31] is instead deterministic and uses $\frac{1}{\varepsilon}$ entries.

Frequent items algorithms can be naturally adapted to context sensitive profiling: during the computation the profiler maintains a subtree of the full calling context tree, called *Hot Calling Context Tree (HCCT)*, storing only hot contexts and their ancestors. More formally, the HCCT is defined as the (unique) subtree of the calling context tree obtained by pruning all cold nodes that are not ancestors of a hot node: by definition all hot nodes are included in the HCCT, whose leaves are necessarily hot (the converse, however, is not true). The frequent items algorithms decide which hot nodes must be monitored, and the profiler updates the HCCT accordingly so as to maintain also their ancestors. The space used by the HCCT includes both monitored hot contexts, and their (possibly cold) ancestors. The former quantity can be analyzed theoretically, as in [26, 31], while the latter depends on properties of the execution trace and on the structure of the calling context tree. In practice, experiments show that this amount is negligible with respect to the number of hot nodes. Hence, the HCCT represents the hot portions of the full calling context tree very well using only an extremely small percentage of space: even when the peak memory usage of the stream-based profiler of [7] is only 1% of standard context-sensitive profilers, all the hottest calling contexts are always identified correctly (no false negatives), the number of false positives (cold contexts that are considered as hot) is very small, and frequency counters are very close to the true values.

5 Concluding Remarks

In this paper we have discussed how recent results in algorithmic theory for data-intensive scenarios can be applied to the design and implementation of dynamic program analysis tools. We have focused on sampling and data stream algorithmics. The examples illustrated in this work should be considered as a non-exhaustive starting point: many other techniques (e.g., data mining or compression) may prove to be valuable in dynamic program analysis. This is indeed a fresh area, and we believe that it represents a novel fertile ground for fundamental algorithmic research: not only dynamic program analysis can inspire many novel, interesting algorithmic questions (or genuinely new variants of well understood problems), but the transfer of algorithmic knowledge in the implementation of program analysis tools can also have a significant practical impact.

Furthermore, algorithm engineering techniques for developing fast, robust, and scalable implementations can play a major role in this scenario, where architectural aspects, such as the presence of memory hierarchies and of multiple cores, can be successfully exploited in order to leverage the runtime impact of dynamic analysis tools.

References

1. Ball, T.: The concept of dynamic analysis. In: Wang, J., Lemoine, M. (eds.) ESEC 1999 and ESEC-FSE 1999. LNCS, vol. 1687, pp. 216–234. Springer, Heidelberg (1999)
2. Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R.: A systematic survey of program comprehension through dynamic analysis. IEEE Transactions on Software Engineering 35(5), 684–702 (2009)
3. Finkbeiner, B., Havelund, K., Rosu, G., Sokolsky, O.: Runtime verification, dagstuhl sem. 07011 executive summary. Technical report (2007)
4. Hamou-Lhadj, A., Lethbridge, T.: Measuring various properties of execution traces to help build better trace analysis tools. In: 10th IEEE Int. Conference on Engineering of Complex Computer Systems, pp. 559–568 (2005)
5. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005), pp. 190–200 (2005)
6. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2007), pp. 89–100 (2007)
7. D'Elia, D.C., Demetrescu, C., Finocchi, I.: Mining hot calling contexts in small space. In: Proc. 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011), pp. 516–527. ACM (2011)
8. Larus, J.R.: Whole program paths. In: ACM SIGPLAN Conference on Programming language design and implementation (PLDI 1999), pp. 259–269. ACM (1999)
9. Nevill-Manning, C.G., Witten, I.H.: Compression and explanation using hierarchical grammars. The Computer Journal 40(2/3), 103–116 (1997)
10. Nevill-Manning, C.G., Witten, I.H.: Linear-time, incremental hierarchy inference for compression. In: 7th Data Compression Conference (DCC 1997), pp. 3–11. IEEE Computer Society (1997)
11. Arnold, M., Ryder, B.G.: A framework for reducing the cost of instrumented code. SIGPLAN Not 36(5), 168–179 (2001)
12. Chan, A., Holmes, R., Murphy, G.C., Ying, A.T.T.: Scaling an object-oriented system execution visualizer through sampling. In: 11th Int. Workshop on Program Comprehension (IWPC 2003), pp. 237–244. IEEE Computer Society (2003)
13. Dugerdil, P.: Using trace sampling techniques to identify dynamic clusters of classes. In: Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2007), pp. 306–314. IBM Corporation (2007)
14. Liblit, B., Aiken, A., Zheng, A.X., Jordan, M.I.: Bug isolation via remote program sampling. In: ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2003), pp. 141–154. ACM (2003)

15. Pirzadeh, H., Shanian, S., Hamou-Lhadj, A., Alawneh, L., Shafiee, A.: Stratified sampling of execution traces: Execution phases serving as strata. Science of Computer Programming (2012) (in press)
16. Zhuang, X., Serrano, M.J., Cain, H.W., Choi, J.D.: Accurate, efficient, and adaptive calling context profiling. In: ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2006), pp. 263–271. ACM (2006)
17. Coppa, E., Finocchi, I., Lo Re, D.: Reservoir profiling. Unpublished Manuscript (January 2013)
18. Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F.: Evaluating the accuracy of Java profilers. In: Proc. 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2010), pp. 187–197 (2010)
19. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. 11(1), 37–57 (1985)
20. Marino, D., Musuvathi, M., Narayanasamy, S.: Literace: effective sampling for lightweight data-race detection. In: Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2009), pp. 134–143 (2009)
21. Morris, R.: Counting large numbers of events in small registers. Comm. ACM 21(10), 840–842 (1978)
22. Munro, J., Paterson, M.: Selection and sorting with limited storage. Theoretical Computer Science 12(3), 315–323 (1980)
23. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. Journal of Computer and System Sciences 58(1), 137–147 (1999)
24. Gilbert, A.C., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.J.: Fast, small-space algorithms for approximate histogram maintenance. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 389–398 (2002)
25. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computation. J. ACM 53(3), 307–323 (2006)
26. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 346–357 (2002)
27. Charikar, M., O'Callaghan, L., Panigrahy, R.: Better streaming algorithms for clustering problems. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003), pp. 30–39 (2003)
28. Mysore, S., Agrawal, B., Sherwood, T., Shrivastava, N., Suri, S.: Profiling over adaptive ranges. In: IEEE/ACM Int. Symposium on Code Generation and Optimization (CGO 2006), pp. 147–158. IEEE Computer Society (2006)
29. Hershberger, J., Shrivastava, N., Suri, S., Tóth, C.D.: Adaptive spatial partitioning for multidimensional data streams. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 522–533. Springer, Heidelberg (2004)
30. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science 1(2) (2005)
31. Metwally, A., Agrawal, D., Abbadi, A.E.: An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Trans. Database Syst. 31(3), 1095–1133 (2006)

On λ-Definable Functions on Ordinals

Tim Fischbach and Benjamin Seyfferth

Mathematisches Institut, Rheinische Friedrich-Wilhelms-Universität Bonn,
Endenicher Allee 60, 53115 Bonn, Germany
seyfferth@math.uni-bonn.de

Abstract. We are going to generalize classical λ-calculus to the ordinal domain. Our reasoning is centered around a generalization of Church numerals, i.e., terms that define the n-fold application of their first argument to the second, to numerals for transfinite ordinals. Once the new class of ordinal λ-terms is established, we define a transfinite procedure to assign to a given ordinal λ-term a normal form if one exists. This normal form procedure is compatible with the classical case, i.e., will find normal forms for classical terms whenever they exist. We go on to prove a confluence property for our procedure. The calculus thus defined is tied into the existing framework of ordinal computability: Using our terms to define a class of functions on the ordinals, we show that this class is identical with the class of $\mathbf{\Sigma}_1(L)$ definable functions on Ord. This paper takes the form of an 'extended abstract': The technical details of the main definition, detailed examples, as well as proofs of the theorems are omitted for brevity.

1 Introduction

Ordinal computability is the study of models of classical computability lifted to the ordinal domain. Particular attention is paid to the elementary computational steps of thus defined transfinite computations and the subtle differences of the different models and their liftings. In his Master's thesis [1], the first author compared ordinal Turing machines (OTMs) [2] and ordinal register machines (ORMs) [3] to existing liftings of classical recursion schemes, namely Kripke's equation calculus and ordinal min-recursive functions [4]. He showed the equivalence of all approaches on admissible ordinals. This lifting of the most common approaches to classical, finite computability theory is clearly missing a λ-calculus variant, which we shall define here. We prove its equivalence to the aforementioned models, further strengthening the idea of an ordinal Church-Turing thesis.

2 Notation from Classical λ-Calculus

As a reference on classical λ-calculus the authors used the monograph by Barendregt [5]. Terms in classical λ-calculus are formed over the alphabet $\{\lambda, .,), (\} \cup \{v_k \mid k \in \omega\}$ by the following rules (we allow lower case letters to stand in for variables such as v_k):

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 135–146, 2013.

(1) Every variable x is a term.

(2) If M is a term and x is a variable, then $\lambda x.M$ is a term.

(3) If M and N are terms, then so is (M, N).

We abbreviate terms of the form $\lambda x.\lambda y.M$ as $\lambda xy.M$. The *subterm* relation $S \subseteq T$ is the transitive closure of the relation $M \subseteq M$, $M \subseteq \lambda x.M$, and $M, N \subseteq (M, N)$. With respect to the expression $\lambda x.$, the notion of x as a *bound* or *free* variable has its intended meaning.

We want to identify terms that arise from each other by renaming of bound variables. By $M\frac{N}{x}$ we denote the syntactic substitution of every occurrence of the free variable x in M by the term N. If we want to replace a single instance S of some subterm of T by some term R, we write the result as $T\left[\begin{smallmatrix} R \\ S \end{smallmatrix}\right]$. Whenever we write such a substitution or replacement, adequate renaming of bound variables is implied to avoid variable conflicts.

The implied interpretation of λ-terms is the following. A term $(\lambda x.M, N)$ is to be interpreted as 'the application of the function $M(x)$ to N'. Accordingly, a *rule of conversion* is defined. The above term may be transformed into $M\frac{N}{x}$. More generally, for any term T, any subterm of T of the form $(\lambda x.M, N)$ may be replaced by $M\frac{N}{x}$, i.e., we may transform T into $T\left[\begin{smallmatrix} M\frac{N}{x} \\ (\lambda x.M,N) \end{smallmatrix}\right]$. We call such a transformation an application of $\boldsymbol{\beta}$*-conversion* or of the $\boldsymbol{\beta}$*-rule* on T. A subterm of T of the form $(\lambda x.M, N)$ is called a *redex (reducible expression)* of T.

A term S is in *normal form*, if $\boldsymbol{\beta}$-conversion cannot be applied to it. S is *a normal form of* some term T, if S is in normal form and can be obtained from T by possibly repeated applications of the $\boldsymbol{\beta}$-rule. There are terms without normal forms, e.g., $(\lambda x.(x, x), \lambda x.(x, x))$. The classical theory proves that the normal form of a term is uniquely determined if it exists and can be found by a certain pattern of applications of $\boldsymbol{\beta}$-conversion. This was first proved in [6]. We denote the normal form of a term T as $\overline{T}$.

The calculus can be fitted with various semantics. From the perspective of computability theory, maybe one of the most important ones is the λ-definability of functions on the natural numbers. There are several ways of modeling natural numbers as λ-terms; consider the following:

$$\begin{aligned}
\underline{\underline{0}} &= \lambda f x.x \\
\underline{\underline{1}} &= \lambda f x.(f, x) \\
&\vdots \\
\underline{\underline{n}} &= \lambda f x.\underbrace{(f, (f, (\ldots (f}_{n\text{-times}}, x) \ldots) \\
&\vdots
\end{aligned}$$

The terms thus defined are referred to as *Church numerals*.

A partial function $f : \mathbb{N} \supset \operatorname{dom} f \to \mathbb{N}$ is called *λ-definable* if there is a λ-term F such that for all $n \in \operatorname{dom} f$:

$$\overline{(F, \underline{\underline{n}})} = \underline{\underline{f(n)}}$$

The class of λ-definable functions is identical to the class of Turing-computable functions. By virtue of the Church-Turing thesis we also speak of the class of computable functions.

3 λI-Calculus

The basis for our generalization of λ-calculus shall be given by the λ**I**-calculus as described in [5, Chapter 9]. The λ**I**-terms form a subset of the λ-terms and are formed by replacing formation rule (2) by the following:

(2) If M is a term and x is a variable that appears free in M, then $\lambda x.M$ is a term.

With λ**I**, trivial applications ('forget the argument') are impossible, as terms of the form $\mathbf{K} = \lambda xy.x$ are illegal. So, in general, case distinctions (returning one of several arguments depending on the situation) or constant functions cannot be defined in λ**I**. For functions on numerals, however, this can be circumvented, exploiting the syntactic structure of Church numerals.

Definition 1 ([5]). *Set* $\mathbf{I} = \lambda y.y$. *This is a* λ**I***-term defining the unary identity function.*

As an example, the numeral $\underline{\underline{0}}$ could be replaced by $\underline{\underline{0'}} = \lambda fx.(((f, \mathbf{I}), \mathbf{I}), x)$. Let $\underline{\underline{n'}} = \underline{\underline{n}}$ for all $n > 0$. Note that, since $((\underline{\underline{n'}}, \mathbf{I}), \mathbf{I})$ reduces to $\mathbf{I}$ for every n', the normal form of $((\underline{\underline{0'}}, \underline{\underline{n'}}), \underline{\underline{m'}})$ is $\underline{\underline{m'}}$. So, on numerals, the meaning '0-fold application of the first argument to the second' is retained.

In contrast to λ**I**-terms, the full set of λ-terms is sometimes referred to as λ**K**-terms. Omitting further details which can be found in [5], we state the following:

Fact 1 ([5]). *The* λ**I***-definable functions on natural numbers coincide with the* λ**K***-definable ones.*

4 Ordinal λ-Terms

Our approach revolves around the idea of generalizing Church numerals from 'the n-fold application of f to x' to 'the α-fold application of f to x'. The intent behind that idea is that terms defining the successor function or arithmetic should generalize to the successor function on ordinals or ordinal arithmetic.

Note 1. In developing our theory, we briefly considered introducing terms of transfinite length, but the asymmetry of ordinals—a limit ordinal has a right neighbor (a least larger ordinal) but no left neighbor (largest smaller ordinal)—limits the intuitive use of syntactical operations on such strings. Instead, we introduce symbols for ordinals on term level and we propose the following generalization of λ-terms to ordinal λ-terms.

Definition 2. *Over the alphabet* $\Sigma_{\mathrm{Ord}} = \{\lambda, .,), (\} \cup \{v_k \mid k \in \omega\} \cup \{{}^{\alpha} \mid \alpha \in \mathrm{Ord}\}$ *define the set* $\mathrm{Term}_{\mathrm{Ord}}$ *of* ordinal λ-terms *by*

(1) *Every variable x is an ordinal λ-term.*
(2) *If M is an ordinal λ-term and x appears free in M, then $\lambda x.M$ is an ordinal λ-term.*
(3) *If α is an ordinal and M and N are ordinal λ-terms, then so is ${}^{\alpha}(M, N)$.*

We often write (M, N) instead of ${}^{1}(M, N)$.

Informally, we refer to ordinal λ-terms just as *terms*.

We introduce an equivalence relation $\simeq_v$ on terms, identifying all terms that can be obtained from each other by renaming of bound variables. If V is a finite set of variables with largest element v_i, define for every equivalence its *v-minimal term over V* as the one where all bound variables are named v_{i+1}, v_{i+2}, v_{i+3}, etc. from left to right. For a term T, we denote by T_v^V its v-minimal term over V. We simply write v-minimal and T_v if $V = \emptyset$.

Definition 3. *The* (ordinal) Church numerals *take the form $\underline{\alpha} = \lambda f x.{}^{\alpha}(f, x)$ for $\alpha \in \mathrm{Ord}$. More generally, we refer to all terms $\simeq_v$-equivalent to some $\underline{\alpha}$ as Church numerals.*

The intended meaning of terms like ${}^{\beta}(M, {}^{\alpha}(M, N))$ and ${}^{\alpha+\beta}(M, N)$ is the same. So, we want to identify all the terms of the form

$$ {}^{\alpha_{k-1}}(M, {}^{\alpha_{k-2}}(M, \ldots {}^{\alpha_0}(M, N)\ldots)) $$

with

$$ {}^{\alpha_0+\alpha_1+\ldots+\alpha_{k-1}}(M, N). $$

Let T be a term. We call a replacement of all subterms of T that are of the former form by their equivalent terms of the latter form a *contraction of applications of T*. We define an equivalence relation $\simeq_a$ by identifying every term T with the terms resulting from contractions of its applications and closing transitively. For a given term T we define its *a-minimal term T_a* as the shortest term a-equivalent to T.

In order to define an equivalent to the $\boldsymbol{\beta}$-normal form, we define a transfinite procedure that for every term either finds a term we shall call its normal form or diverges, the latter which we shall interpret as the term not having a normal form.

5 Normal Form Derivation

The normal form derivation will be a transfinite procedure. We declare a kind of limit convergence for our ordinal λ-terms.

Definition 4. *Consider a term M as a finite sequence of symbols in Σ_{Ord}, i.e., $M : n \to \Sigma_{\mathrm{Ord}}$ for some natural number $n \in \omega$. Let $(j_i \mid 0 \leq i < k)$ be the increasing sequence of those $j < n$ with $M(j) \in \{ {}^{\alpha} \mid \alpha \in \mathrm{Ord}\}$. Let $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_{k-1})$ be some sequence of ordinals. Define*

(i) the flesh *of M as* $\mathrm{fl}(M) = (M(j_i) \mid i < k)$
(ii) the skeleton *of M as* $\mathrm{sk}(M) : n \to \mathrm{ran}(M)$ *with*

$$\mathrm{sk}(M)(j) := \begin{cases} M(j), & \text{if } j \notin \{j_i \mid 0 \leq i < k\} \\ {}^{1}, & \text{if } j \in \{j_i \mid 0 \leq i < k\}. \end{cases}$$

(iii) the insertion of $\boldsymbol{\alpha}$ in M *as*

$$M[\boldsymbol{\alpha}] = \begin{cases} M(j), & \text{if } j \notin \{j_i \mid 0 \leq i < k\} \\ {}^{\alpha_i}, & \text{if } j = j_i \text{ for some } i < k. \end{cases}$$

Note that $\mathrm{sk}(M)[\mathrm{fl}(M)] = M$.

Definition 5. *Let α be a limit ordinal.*

(i) Let $s : \alpha \to \mathrm{Ord}^n$ be a sequence of n-tuples of ordinals for some $n < \omega$. Define the pointwise limes inferior *by*

$$\liminf_{\beta \to \alpha} s(\beta) := (\liminf_{\beta \to \alpha} s(\beta)_0, \ldots, \liminf_{\beta \to \alpha} s(\beta)_{n-1}).$$

(ii) Let $s : \alpha \to \mathrm{Term}_{\mathrm{Ord}}$ be a sequence of terms. We say that the skeletons of s converge, *if there is an a-minimal and v-minimal skeleton S such that there is a $\gamma < \alpha$ and for all $\gamma < \beta < \alpha$ we have $\mathrm{sk}(s(\beta))_a \simeq_v S$. We shall call S the* limit skeleton for s. *If V is a finite set of variables, S may be chosen v-minimal over V; we then speak of the limit skeleton* over V.
(iii) Let $s : \alpha \to \Sigma^*_{\mathrm{Ord}}$ be a sequence of terms whose skeletons converge to its limit skeleton S. Let γ be minimal such that $\mathrm{sk}(s(\beta)_a) = S$ for $\gamma < \beta < \alpha$. Then the syntactical limes inferior *of s exists and is defined by*

$$\liminf_{\beta \to \alpha} s(\beta) := \mathrm{sk}(S)[\liminf_{\beta \to \alpha, \beta > \gamma} \mathrm{fl}(s(\beta)_a)]$$

If S is the limit skeleton over some finite set of variables V, we speak of the syntactical limes inferior over V, written $\liminf^V_{\beta \to \alpha} s(\beta)$

Note 2. In the following we give a deterministic procedure to arrive at a given ordinal λ-term's normal form. Every step can be seen to correspond to one application of the classical $\boldsymbol{\beta}$-rule. In the classical $\boldsymbol{\lambda I}$-calculus, any pattern of

iterated application of the β-rule eventually yields a normal form. The idea of α-fold application of one term to another implies a transfinite length of applications of a β-rule, and we want to make use of the limit notions for terms we just defined. We chose to give up the nondeterministic freedom of the finite case to produce stabilizing and natural behavior at limits. In turn, we get some of that freedom back by proving a weak confluence property in Section 6. There, we also conjecture a stronger property that would enable us to perform finitely many arbitrary deviations from the algorithm.

In this extended abstract, we shall not give a rigorous technical definition of the normal form derivation. Instead, we shall describe the main ideas behind the process for arriving at a normal form that we have in mind. The algorithm will maintain a stack. Each stack element is a term whose normal form is to be determined. The bottom element of the stack is the original term T we wish to reduce to its normal form by some generalizations of the β-rule. The next element shall be the *leftmost redex* of T, i.e., a leftmost subterm S of the form $S = {}^{\alpha}(\lambda x.M, N)$ where $\alpha > 0$. A redex is called leftmost if its operating λ, i.e., the λ that is spelled out in the representation of S above, appears to the left of all other operating λ's of redexes of T. So, S is put on the second stack level. Recursively, the algorithm will determine the normal form of S. In the mean time, the stack will get built up and torn down again and as soon as S's normal form $\overline{S}$ is found, our stack will contain exactly two elements: T as the bottom one with $\overline{S}$ on top. The next step will be to remove $\overline{S}$ from the stack, replace S in T with $\overline{S}$ and start the procedure over for the resulting term. Eventually, the bottom element will contain no more redexes and a normal form is found.

So how does the algorithm proceed to determine the normal form of some redex $S = {}^{\alpha}(\lambda x.M, N)$? We retain the intuition of 'the α-fold application of M to N' by the following procedure: Determine, by putting on the stack consecutively, the normal forms of the approximations ${}^{1}(\lambda x.M, N)$, ${}^{2}(\lambda x.M, N)$, etc. At limit times, syntactical inferior limits are taken (if they exist, otherwise the normal form procedure breaks down). More precisely, instead of ${}^{\gamma+1}(\lambda x.M, N)$, we evaluate the a-equivalent term $(\lambda x.M, {}^{\gamma}(\lambda x.M, N))$, substituting the term N' we determined in the previous steps as the normal form of ${}^{\gamma}(\lambda x.M, N)$, which in the end gives us $(\lambda x.M, N')$ to evaluate. We can now rely on an application of what is known as the β-rule in the finite case to end up with $M\frac{N'}{x}$ which is what is being put on the stack instead of ${}^{\gamma+1}(\lambda x.M, N)$.

Finally, we have to deal with our stack length becoming infinite. This might happen via the use of terms that work like $(\lambda x.(x, x), \lambda x.(x, x))$ and may be used as so-called fixed-point combinators in recursive definitions. For instance, one usually implements unbounded search by a term describing the following function $Q(\alpha)$: 'if condition P holds on α then return α, else evaluate and return $Q(\alpha + 1)$'. A normal form procedure for $Q(0)$ will build up a stack of height β if β is the least ordinal such that P holds. If the stack length approaches a limit ξ, we shall define the stack content at level ξ in the following way: First, identify the first (from the bottom) term on the stack whose skeleton appears cofinally often below ξ as skeleton of terms on the stack. Now, set as the term

on level ξ the syntactical $\liminf$ over all terms on the stack with this skeleton. In the above example, we shall 'try α' (i.e., put $Q(\alpha)$ on the stack), then do some steps to determine whether P holds for α and if not 'try $\alpha+1$'. In the next limit we want to 'try the first limit after α', i.e., put $Q(\alpha+\omega)$ on the stack. The term $Q(0)$ representing 'try 0' obviously is the first whose skeleton appears cofinally often. Theorem 3 will confirm this behavior. To avoid problems with backtracking downwards in the ordinals, we shall keep track of the stack height on which the first term that lends its skeleton to the limit appears. That way, as soon as a normal form is found (in our example the least β such that P holds) and has been handed down step by step until the stack is torn down to some limit height, we can propagate this normal form directly downwards to whenever $Q(0)$ was put on the stack.

For a term T, we denote the normal form thus defined by $\overline{T}$. If the normal form procedure for T breaks down or diverges (i.e. has length Ord), we write $\overline{T}\uparrow$.

Note 3. Terms of the form ${}^0(\lambda x.M, N)$ are not treated as reducible; they are unchanged by the algorithm save for internal modifications of M and N and may vanish only through contractions of applications. One could argue that the intended meaning behind such a term is simply N (the 0-fold application of M to N) but such transformations would reintroduce terms of the form $\lambda xy.x$, violating the boundaries of $\lambda\mathbf{I}$-calculus.

For our purposes, there is an added benefit of not resolving 0-fold applications: The numeral $\underline{0} = \lambda fx.{}^0(f,x)$ is of the same syntactical form as the other numerals, enabling $\underline{0}$ to be a possible value of a syntactical $\liminf$ of numerals. However, not setting $\underline{0}$ apart from the other numerals introduces a difficulty with arithmetic: Several classical algorithms for arithmetic (predecessor of natural numbers, subtraction, etc.) rely heavily on a term that recognizes the numeral for 0 among all the other numerals. We resolve this issue by expanding our calculus by a term that defines equality on ordinals in Definition 7.

Definition 6. *Let $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_{n-1})$ be a finite sequence of ordinal numbers. A function $f : \mathrm{Ord}^k \to \mathrm{Ord}$ is* ordinal λ-definable in parameters $\boldsymbol{\alpha}$ *if there is an ordinal λ-term T in which all applications are of the form ${}^\beta(\cdot,\cdot)$ where $\beta \in \omega \cup \{\alpha_0, \ldots, \alpha_{n-1}\}$ such that for all $(\gamma_0, \ldots, \gamma_{k-1}) \in \mathrm{Ord}^k$ we have*

$$\overline{(\ldots(T, \underline{\gamma_0}), \underline{\gamma_1}), \ldots), \underline{\gamma_{k-1}})} \simeq_v \underline{f(\gamma_0, \ldots, \gamma_{k-1})}.$$

If f is a partial function, we call f ordinal λ-definable in $\boldsymbol{\alpha}$ if $f \restriction \mathrm{dom}\, f$ is ordinal λ-definable in $\boldsymbol{\alpha}$ and $\overline{(T, \underline{\gamma})}\uparrow$ on $\gamma \notin \mathrm{dom}\, f$. If $\boldsymbol{\alpha} = \emptyset$, we simply speak of ordinal λ-definability.

As explained in Note 3, we would like to add the capability of defining equality on ordinals to our calculus:

Definition 7. *Let us add a constant symbol E to our alphabet and consider the terms formed over $\Sigma_{\mathrm{Ord}} \cup \{E\}$ with the additional rule 'E is a term'*

as the ordinal $\lambda + E$-terms. *We extend our definition by a case for terms of the form* ${}^1({}^1(E, \underline{\alpha}), \underline{\beta})$*: The algorithm is to replace* ${}^1({}^1(E, \underline{\alpha}), \underline{\beta})$ *with the normal form* $\mathbf{T}_I = \lambda xy.(((y, \mathbf{I}), \mathbf{I}), x)$ *if* $\alpha = \beta$ *and with the normal form* $\mathbf{F}_I = \lambda x.(((x, \mathbf{I}), \mathbf{I}), \mathbf{I})$ *else. In all other cases,* E *is to be treated like a variable symbol. The resulting notion of definability for functions on the ordinals is that of* ordinal $\lambda + E$-definable *functions.*

The terms $\mathbf{T}_I$ and $\mathbf{F}_I$ can be used to define case distinctions in the following manner: Suppose $\overline{((P, \mathbf{I}), \mathbf{I})} \simeq_v \overline{((Q, \mathbf{I}), \mathbf{I})} \simeq_v \mathbf{I}$ which is the case, e.g., for P, Q Church numerals. Then

$$\overline{((B, P), Q)} \simeq_v \begin{cases} \overline{P} & , \text{if } \overline{B} = \mathbf{T}_I \\ \overline{Q} & , \text{if } \overline{B} = \mathbf{F}_I. \end{cases}$$

The term $((B, P), Q)$ hence may be read as $\texttt{if}\ B\ \texttt{then}\ P\ \texttt{else}\ Q$.

Classical $\lambda\mathbf{I}$-terms are ordinal λ-terms. On the other hand, if we have an ordinal λ-term where for all applications ${}^\alpha(M, N)$ we have that α is finite, we can convert it to a classical $\lambda\mathbf{I}$-term by a map ϕ given by:

- replacing any subterm of the form ${}^n(M, N)$ by $\underbrace{(M, (M, (\ldots (M}_{n\text{-times}}, N) \ldots)$ if $n > 0$,
- replacing any subterm of the form ${}^0(M, N)$ by $(((M, \mathbf{I}), \mathbf{I}), N)$. In particular, this maps $\underline{0}$ to $\underline{0}' = \lambda fx.(((f, \mathbf{I}), \mathbf{I}), x)$, the term [5] uses in the treatment of $\lambda\mathbf{I}$-calculus.

Proposition 1. *If* M *is a classical* $\lambda\mathbf{I}$*-term with classical normal form* M' *and* M'' *is the output of our algorithm on input* M*, then* $\phi(M'') \simeq_v M'_a$.

Proof. In $\lambda\mathbf{I}$-calculus, every reduction strategy (pattern of applying the $\boldsymbol{\beta}$-rule to various subterms until no redex is left) is normalizing, i.e., eventually yields normal forms. Our algorithm, although working with a-minimal terms, will simply run a finite number of applications of the $\boldsymbol{\beta}$-rule before halting with a term M'' without redexes. Converting this term to a classical $\lambda\mathbf{I}$-term does not introduce any redexes, so the resulting term is also classically in normal form. Since classically normal forms are unique up to renaming of variables, we have indeed found a term $\simeq_v$-equivalent to M'.

6 A Confluence Property for Our Algorithm

The classical Church-Rosser result establishes that for any two terms Q and Q' that are obtained from the same term P via $\boldsymbol{\beta}$-reduction, there is a term R that can be obtained from Q and Q' by $\boldsymbol{\beta}$-reduction. This is known as the Church-Rosser property. It ensures that normal forms are unique. In the $\lambda\mathbf{K}$-calculus, a term's unique normal form is obtained by a certain pattern of applications of the $\boldsymbol{\beta}$-rule, whereas in $\lambda\mathbf{I}$, any pattern of applications of the $\boldsymbol{\beta}$-rule leads to the

term's normal form, given that one exists. In our situation, where we restrict ourselves from applying the $\boldsymbol{\beta}$-rule freely for the sake of convergence at limits, we propose the following as the correct lifting of the Church-Rosser theorem:

Conjecture 1. Let T be a term with normal form and let $S \subseteq T$ be a subterm with normal form. Then $\overline{T} \simeq_v \overline{T\left[\begin{smallmatrix}\overline{S}\\ S\end{smallmatrix}\right]}$.

For the purposes of this thesis, the following result is sufficient, as it will establish that the composition of two λ-definable functions is λ-definable (results for $\lambda{+}E$ follow analogously).

Theorem 1. *Let T be a term with normal form and let $S \subseteq T$ be a subterm that has a normal form and is of the form $S = {}^{\alpha}(M, N)$ such that all free variables in S are not bound in T. Then $\overline{T} \simeq_v \overline{T\left[\begin{smallmatrix}\overline{S}\\ S\end{smallmatrix}\right]}$.*

The proof is a rather straightforward application of the technique of keeping track of certain *residuals* as in the classical paper [6].

7 Ordinal λ-Definable Functions and Ordinal Computability

We shall now explore which functions on the ordinals are ordinal λ-definable. In his Diploma thesis [1], the first author showed how various existing notions of ordinal computability coincide in strength. We tie in our proposed model of ordinal λ-definability into this framework to state our main result at the end of this section.

7.1 Primitive Recursive Set and Ordinal Functions

A generalization of primitive recursive functions on natural numbers, operating on the universe of sets, has been used in the study of the constructible hierarchy [4,7]. In [4], Jensen and Karp gave a definition for Prim_O, the class of primitive recursive functions mapping ordinals to ordinals, which is compatible with their notion Prim of primitive recursiveness of functions mapping sets to sets defined alongside in their paper. We use a slight modification of their definition that is equivalent in strength:

Definition 8. *Let $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_{k-1})$ be a finite sequence of ordinals for $0 \leq i < k$. The symbol* $\mathrm{Prim}_O(\boldsymbol{\alpha})$ *(*primitive recursive ordinal functions in $\boldsymbol{\alpha}$*) denotes the collection of all functions of type (1) to (5) closed under the schemes for substitution (a) and (b) and recursion (R).*

(1) $f(\xi) = \alpha_i$ *for* $0 \leq i < k$
(2) $\mathrm{pr}_{n,i}(\boldsymbol{\xi}) = \xi_i$*, for all* $n \in \omega$*,* $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)$ *and* $1 \leq i < n$*.*
(3) $f(\xi) = 0$
(4) $f(\xi) = \xi + 1$

(5) $e(\xi,\zeta) = \begin{cases} 1 \text{ if } \xi = \zeta \\ 0 \text{ else} \end{cases}$
(a) $f(\boldsymbol{\xi},\boldsymbol{\zeta}) = g(\boldsymbol{\xi}, h(\boldsymbol{\xi}), \boldsymbol{\zeta})$
(b) $f(\boldsymbol{\xi},\boldsymbol{\zeta}) = g(h(\boldsymbol{\xi}), \boldsymbol{\zeta})$
(R) *If g and h are given, define f by*

$$\begin{aligned} f(0,\boldsymbol{\zeta}) &= g(\boldsymbol{\zeta}) \\ f(\xi+1,\boldsymbol{\zeta}) &= h(f(\xi),\xi,\boldsymbol{\zeta}) \\ f(\xi,\boldsymbol{\zeta}) &= \liminf_{\eta<\xi} f(\eta,\boldsymbol{\zeta}) \text{ if } \xi \text{ is a limit ordinal} \end{aligned}$$

We write Prim_O *for* $\mathrm{Prim}_O(\emptyset)$.

We can now show that the Prim_O functions are $\lambda + E$-definable, the first major step towards our main theorem.

Theorem 2. *Every* $\mathrm{Prim}_O(\boldsymbol{\alpha})$ *function is ordinal* $\lambda + E$*-definable in* $\boldsymbol{\alpha}$.

The proof makes extensive use of the predicate E.

7.2 Minimization

An *ordinal* λ*-definable predicate* on the ordinals is given by a term P such that $(P, \underline{\alpha})$ takes $\mathbf{T}_I$ as normal form for α in some subset or subclass of the ordinals and $\mathbf{F}_I$ for α in the complement.

In this section, we shall see that for every ordinal λ-definable predicate, there is a function defining its least witness. The proof is a generalization of [5, Chapter 9, §2] and Barendregt credits Kleene for the construction. In [5], for any classically λ-definable predicate P on ω, a cleverly constructed term H_P is given that has the least witness of the predicate P as normal form. It turns out that the same term yields least witnesses for ordinal λ-definable predicates on Ord under our algorithm.

We give the central result of this subsection, the second ingredient to our main result:

Theorem 3. *Let P be an ordinal λ-definable predicate. Then* $\overline{(H_P, \underline{0})} \simeq_v \underline{\gamma}$ *where* $\gamma = \min_{\gamma \in \mathrm{Ord}}(\overline{(P,\gamma)} \simeq_v \mathbf{T}_I)$ *if such a γ exists. Otherwise* $\overline{(H_P, \underline{0})}\uparrow$.

The proof is done by analyzing the normal form derivation step-by-step. Note that the stack will reach height γ in the process, as mentioned before Note 3.

7.3 Main Result

By combining Theorems 2 and 3 we get:

Theorem 4. *A partial function* $F : \mathrm{Ord} \rightharpoonup \mathrm{Ord}$ *on the ordinals is $\lambda + E$-definable in finitely many ordinal parameters if and only if it is $\boldsymbol{\Sigma}_1$-definable over L.*

We can restrict both the length of the normal form derivation and the stack height in our definition to some admissible ordinal α: If either reaches α, we say that the normal form derivation diverges. With the resulting notion of α-*normal form derivation* we can define the α-$\lambda + E$-definable (partial) functions on α (possibly in parameters $< \alpha$).

Corollary 1. *Let α be admissible. A function partial $F : \alpha \rightharpoonup \alpha$ is α-$\lambda + E$-definable in a finite set of parameters $< \alpha$ if and only if it is $\mathbf{\Sigma}_1$-definable over L_α.*

8 Open Questions

8.1 A Stronger Confluence Property

We already stated Conjecture 1 and hope that this stronger confluence property holds for our calculus.

8.2 $\lambda + Z$-Definability

Another open problem is whether we can prove our main result for calculi seemingly weaker than $\lambda + E$. We added a predicate for equality of ordinals to our calculus and defined the $\lambda + E$-definable functions for the sake of Theorem 2. All other arguments from Section 7 go through also for λ-definable functions. We conjecture that we can replace the predicate E by a predicate Z that tests numerals for being zero, in the same fashion as E tests for equality.

Due to Note 3, it seems unlikely that we are able to go even weaker than $\lambda + Z$. Since, syntactically, the numeral for 0 is indistinguishable from the numerals for non-zero ordinals, it appears doubtful to obtain a test for zero by syntactical tricks. Also, its arithmetical properties cannot be validated without a means to talk about equality of ordinals.

8.3 Variations of the Model

As with the other models of ordinal computation, there are interesting variations imaginable. While λ-calculus does not come with a canonical distinction between time and space, our normal form algorithm can easily be restricted in both runtime or stack height. Asymmetric models such as ITTMs or the restriction of OTMs in [8] have interesting theories, so these two paths, i.e., restricting stack height but not runtime and restricting input complexity but neither runtime nor stack height, should be explored. In an early stage of the development, the authors conjectured that the present calculus restricted to finite stacks would be equivalent in strength to the Prim_O functions. While plausible from interpreting terms of the form ${}^{\alpha}(M, N)$ as some kind of `for`-loops, it turned out that this is false: Due to its un-typed nature, the thus defined generalization of λ-calculus is capable of giving 'primitive recursive' definitions for functionals such as the

Ackermann function, while the calculus of primitive recursive functions is limited to defining only functions on ordinals in a primitive recursive manner.

Moving away from looking at the calculus solely as means of defining functions on the ordinals, the work done to generalize λ-calculus may perhaps be used to extend other calculi that are centered on the re-writing of terms to the transfinite.

References

1. Fischbach, T.: The Church-Turing Thesis for Ordinal Computable Functions. Master's thesis, University of Bonn (October 2010)
2. Koepke, P.: Turing computations on ordinals. The Bulletin of Symbolic Logic 11, 377–397 (2005)
3. Koepke, P., Siders, R.: Register computations on ordinals. Submitted to: Archive for Mathematical Logic, 14 pages (2006)
4. Jensen, R.B., Karp, C.: Primitive recursive set functions. In: Scott, D.S. (ed.) Axiomatic Set Theory. Volume XIII (Part 1) of Proceedings of Symposia in Pure Mathematics, pp. 143–176. American Mathematical Society, Providence (1971)
5. Barendregt, H.P.: The lambda calculus. Its syntax and semantics. In: Studies in Logic and the Foundations of Mathematics. North/Holland Publishing Company, Amsterdam (1981)
6. Church, A., Rosser, J.B.: Some properties of conversion. Transactions of the American Mathematical Society 39(3), 472–482 (1936)
7. Devlin, K.J.: Aspects of constructibility. Lecture Notes in Mathematics. Springer, Heidelberg (1973)
8. Schlicht, P., Seyfferth, B.: Tree representations and ordinal machines. Computability 1(1), 45–57 (2012)

A Personal View of the P versus NP Problem

Lance Fortnow

Georgia Institute of Techology, Atlanta, Georgia 30309, U.S.A.
fortnow@cc.gatech.edu

I recently completed a general audience book on the P versus NP problem [1]. Writing the book has forced me to step back and take a fresh look at the question from a non-technical point of view. There are really two different P versus NP problems. One is the formal mathematical question, first formulated by Steve Cook in 1971 [2] and listed as one of the six unresolved millennium problems by the Clay Mathematics Institute[1]. The other P versus NP problem is the one that interests physicists, biologists, economists and the mathematically-curious general public. This talk will explore both faces of the P versus NP problem and what it means for mathematics and computer science moving forward.

Most of this audience is familiar with P versus NP as a mathematical challenge. Let Σ^* be the set of finite length sequences ("strings") over an alphabet Σ. Typically we take $\Sigma = \{0, 1\}$. Let M be a computational procedure that takes as input a finite string and either "accepts" or "rejects" the string. Typically we describe M by a Turing machine, a computational device developed by Alan Turing in his classic paper [3]. A machine M computes a language $L \subseteq \Sigma^*$ if for all x in L, M on input x accepts and for all x not in L, M on input x does not accept. The machine M runs in polynomial time if there is a fixed polynomial p such that for all inputs x, M on input x accepts in time at most $p(|x|)$ where $|x|$ is the number of alphabet symbols in the string x.

By default M is deterministic, the next action of the machine is uniquely defined by its current configuration. We can also consider nondeterministic Turing machines M that have many possible legal future actions. We say a nondeterministic machine M accepts if there exists a series of legal actions leading to M starting on input x to an accept state.

The class P is the set of languages L that are computable in polynomial time by some (deterministic) Turing machine. The class NP consists of those languages accepted by nondeterministic Turing machines in polynomial time. The P versus NP questions asks simply if P = NP.

The question immediately became central to theoretical computer science for several reasons. First of all, the question is quite robust to the various models. It doesn't matter which model of Turing machine we use, or we can use any other reasonable model of computation, the answer to the P versus NP question will not change. More importantly Cook [2], Karp [4] and Levin [5] identified a number of logical and combinatorial problems that are NP-complete,

[1] http://www.claymath.org/millennium

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 147–148, 2013.

computationally as hard as any problem in NP, and thus have polynomial-time algorithms if and only if P = NP.

This talk will explore some of the approaches researchers have used to attack the P versus NP problem, though we don't hold out much hope for a quick resolution.

We will also explore the other less mathematical side of P versus NP question. Computational power has dramatically increased through the years since Cook and Levin first formulated the P versus NP problem in 1971 allow us to solve via clever algorithms and brute force search a number of moderate-sized NP-complete problems. Paradoxically, these successes have only bolstered interest in the P versus NP problem, as we now wish to tackle larger computational problems where the exponential running times of our best algorithms for NP-complete problems really kick in. The huge growth in data has also made the P versus NP problem an issue for scientists in many fields such as biology, physics and economics.

These scientists and the general public don't care so much about the fine technical details of the P versus NP problem but more about the spirit of what we can or cannot solve on computers in a reasonable amount of time. In this talk we look at a "beautiful world" (with hidden dangers) that arise if P = NP (in a strong way).

While most computer scientists believe P $\neq$ NP, that can't be the end of the story as we need to tackle these difficult problems. The talk will give a high-level survey of some approaches to dealing with NP-complete problems such as heuristical and approximate algorithms.

The P versus NP problem is both an incredibly challenging mathematical puzzle but simply understanding the P versus NP problem and it challenges is now of critical importance for any scientist. Our challenge is to tackle the mathematical challenge while keeping the practical issues at heart.

References

1. Fortnow, L.: The Golden Ticket: P, NP and the search for the impossible. Princeton University Press, Princeton (2013)
2. Cook, S.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd ACM Symposium on the Theory of Computing, pp. 151–158. ACM, New York (1971)
3. Turing, A.: On computable numbers, with an application to the Etscheidungs problem. Proceedings of the London Mathematical Society 42, 230–265 (1936)
4. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
5. Levin, L.: Universal'nyĭe perebornyĭe zadachi (Universal search problems: in Russian). Problemy Peredachi Informatsii 9(3), 265–266 (1973); Corrected English translation in [6]
6. Trakhtenbrot, R.: A survey of Russian approaches to Perebor (brute-force search) algorithms. Annals of the History of Computing 6(4), 384–400 (1984)

An Investigation on Genomic Repeats

Giuditta Franco and Alessio Milanese

Department of Computer Science, University of Verona,
Strada le grazie 15, Verona, Italy

Abstract. Motivated by a general interest to understand sequence based signaling in the cell, and in particular how (even distal) genomic strings communicate each other in the transcriptional process, we present a bioinformatic investigation on genomic repeats which occur in multiple genes. Unconventional graph based methods to abstractly represent genomes, gene networks, and genomic languages are provided. In particular, the distribution of long repeats along genomic sequences from three specific organisms (genome of *N. equitans*, of *E. coli*, and chromosome IV of *S. cerevisiae*) is computed, and efficiently visualized along the entire sequences, with the unexpected result to have most of them occurring inside genes.

1 Introduction

The human genome is being almost entirely annotated in biochemical terms, within the decade-long federal project ENCODE (Encyclopedia of DNA elements) started in 2003, which has recently given integrated evidence that 80% of the human genome is covered by active regulatory (or functional) elements. New insights into the mechanisms of gene regulation have been given even from an informational viewpoint, along with new data about the correspondence among promoter sequences, specific protein combination bindings, and encoding regions [8]. However, there is a clear lack of a model which explains how major informational processes work to keep the cell alive, whereas the genomic system appears stunningly complex, with lots of redundancies. An interesting point to clarify is *how* regions from the traditional "dark matter" (or junk DNA) communicate with the (close and far) genes they affect, given also the information that promoters and distal elements are engaged in more than 1,000 long-range looping interactions [17].

Several computational (mainly machine learning based) tools have supported the ENCODE project, while numerous alignment-free methods of sequence analysis have been emerging in the literature, among which we would like to point out computational and linguistic approaches based on genomic dictionaries (for example, [3,5,15,9]). In this context, we focus our investigation on genomic repeats, which are recurrent motifs, occurring in several locations along the genome and often inside multiple genes, targeted as possible signals in mechanisms of genomic information exchange.

Every function in the cell is mediated by signals, from circulating hormones, that affect the entire body, to intracellular signaling molecules. There are two

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 149–160, 2013.

types of signaling mechanisms: *i)* protein signals, where an amino acid sequence assumes a certain configuration, which results in a protein activation signal (e.g., by creating a protein phosphorylation site), or in a position signal (where, for example, a specific sequence indicates if the protein has to move in the nucleus or in the mitochondria); and *ii)* nucleotide signals, where the signaling mechanism is mediated by annealing of complementary sequences. The first type is a structural (3D) signal that has been extensively studied, while the second one is related to sequence recognition, and only recently is being object of attention, since the discovery of miRNAs in nematodes in 1993.

The number of repetitions and the localization of specific repeated sequences along the genome has recently attracted attention also because of their correlation with number of mutations in many cancer-related genes (such as PTEN) [20], particularly those with simple repeated sequences in their coding regions [2]. Tumors with microsatellite instability (MSI) are characterized by a massive instability in simple repeated sequences [13], and display a strong microsatellite mutator phenotype (MMP), which (for gastrointestinal cancer in particular) differs in genotype and phenotype from cancers with microsatellite stability (MSS) [2].

There are examples of enormous importance where repeats located in genes turn out to have a relevant role for survival and evolution. For example, the SRP ribonucleotide implicated in the cellular transportation of proteins of any living beings, has an sRNA encoded by the RNA7s gene, which is duplicated many times in sequences called ALU. Some of these occurrences are not funcional, are truncated or mutated, but most of them constitute, only in primates, a major part of the short interspersed repeats (about 300bp, and covering about 10.7% of the human genome). ALU elements (appeared in the evolution of primates between 60 and 70 millions of years ago), once repetitively inserted in existent genes, may alter the rate of protein production, a parameter considered fundamental for the development of various biological characteristics. They may in fact become new exons, meaning new information carried by mRNA outside the nucleus. ALU sequences are more abundant in human organisms than in others, so that often they are helpful to distinguish human DNA from mouse DNA. They are implicated in cancer and in several hereditarian diseases as well.

A final example we would like to report here, where repeat variation inside genes is very important, is related to the fragile X syndrome (a genetic disorder inducing mental retardation), associated to the expansion of the CGG trinucleotide repeat affecting the (FMR1) gene on the X chromosome. Depending on the length of the CGG repeat, an allele may be classified as normal (unaffected by the syndrome), a premutation (at risk of fragile X associated disorders), or a full mutation (usually affected by the syndrome).

Analyzing the presence of repeats in genes may therefore be important for studies of population genetics of human beings and the evolution of primates. Here we present a bioinformatic investigation on genomic repeats, along the line of a recently introduced computational genomic approach based on dictionary [5,4,14,9], where we look at multioccurring genetic motifs as elements

in common among (e.g., paralog) genes, and as an abstract representation of genomes, in order to highlight specific string properties. Both views are based on graph representations. A brief and basic introduction to the genome structure and functions is given in the next section, followed by a computational analysis of genome repeats of three specific organisms, where the longest repeats appear to occur in genes. A recent approach employing repeat based gene networks is recalled in section 4, along with a few interesting preliminary results from work in progress [6], while a discussion about these results and some open problems concludes the paper.

2 Biological Background

The genome contains *i) structural* genes, transcribed into *messenger RNA (mRNA)* and then translated into (possibly more than one) protein, *ii) non-structural* genes, which are transcribed into RNA based regulatory elements, and *iii*) other portions (for example, *centromeres* and *telomeres* located in the middle and at the ends of chromosomes) which are never transcribed. Centromeres, in particular, are devoted to mechanical movements of chromosomes during cell division (it is the location to bind, in order to carry the chromosome to the border of eukaryotic cell): their sequence is not conserved (every organism has a different one) but it is known to be composed by highly repetitive traits.

Structural genes in procaryotes are contiguous segments, whereas eukaryotic genes are segmented in *exons* and *introns* (representing coding and noncoding parts, respectively). The introns are cut off and (some of) the exons are reassembled, in the nucleus, by means of the *(alternative) splicing phase*. In such a way, a specific concatenation of exons (often called *exome*) is then translated into a protein, while mRNA includes *untranslated regions* (UTR) (never translated into protein). One of the main result achieved within the ENCODE project has been the annotation of isoforms for human genes, i.e., the annotation of genes with their (multiple) products due to alternative splicing (a key mechanism in transcription). Main processes including *transcription* and *translation* are depicted in Figure 1, where it is reported how genomic information is copied into different types of coding and non-coding RNAs (by means of the transcription phase). Some RNA molecules are called ncRNAs and have different functions, such as serving aminoacids to the *ribosomes* (machineries which synthesize proteins), by tRNA, or performing the splicing process (in Figure 1, these are called "other RNA"). According to well known data[1], most of the RNA retrieved in the cells is rRNA (approximatively 80% of total RNAs, especially in rapidly growing mammalian cells, e.g., cultured HeLa cells), while the rest of types is present in small proportions (15% tRNA, 3% mRNA, 2% regulatory or other RNAs) which vary from cell to cell. However, the protein-coding mRNA in general constitutes only a small portion of the total RNA.

Single strands of mRNA carry the genomic information outside the cell nucleus, where it is transformed into *pre-mRNAs*, which is then processed in the

[1] See for example `http://www.ncbi.nlm.nih.gov/books/NBK21729/`

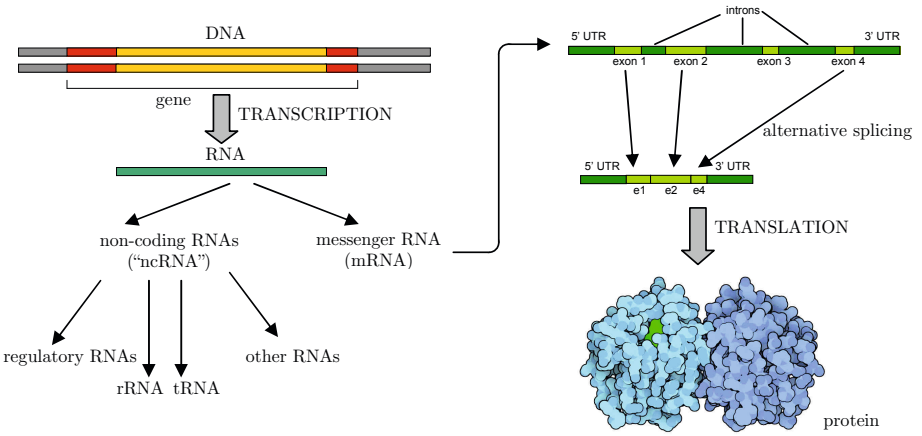


Fig. 1. A sketch to visualize how genomic information guides the assembly of proteins, by means of different types of RNA molecules

cell nucleus to become (mature) mRNAs. This process consists of three phases: a *post-trascriptional capping* where a modified G is attached to the 5′-end, a step-by-step removal of introns present in the pre-mRNA and an (alternative) *splicing* of the exons (performed by *splicesomes*), and a final (post-transcriptional) *polyadenylation* (elongation by a polyA) at the 3′-end. Such a targeted information is moved to the cytoplasm, where the translation into a protein is performed by ribosomes, according to the *genetic code*, which associates three-letter words, called *codons*, to single amino acids (basic protein constituents). Different codons may code for the same amino acid. Transcription is activated by molecular signals (such as hormones), which interact with the regions where promoters, switchers, enhancers, and protein binding sites are located. Enhancers may be located upstream, downstream, or even within the gene they control, and increase the rate of transcription.

Recent discoveries from ENCODE have changed our previous characterization of genic and inter-genic regions. Indeed, 8,801 small RNAs (sRNAs) and 9,640 long non-coding RNAs (lncRNAs) were discovered in relation to human genes, together with about 1,600 (22-nucleotide-long) microRNAs (miRNAs), that function in the post-transcriptional regulation of gene expression by binding to target mRNAs, and regulate the translation via translational repression or target degradation [1]. Despite miRNAs have been found only in eukaryotes, we can observe the mechanism of antisense RNA regulation in phages, plasmids and bacteria [22], confirming that this level of regulation is widespread and as important as protein regulation. In particular, untranslated RNAs consisting of 50 to 400 bases have regulatory function in protein synthesis of bacteria, where they regulate mRNA transcription via base-pairing interactions [11]. In mammalians, miRNAs may target about 60% of genes [10], while all sRNAs (and miRNAs) are known to regulate multiple genes [19]. The above considerations

bring our attention on studies focused on identification and quantification of genomic repeats (longer than 20), along with their relative presence and multi-occurrence in encoding regions, where common repeats seem to represent a form of communication among genes [20].

3 Repeat Analysis

According to a first simplification, genomes may be linearized, and analyzed by synthetic methods just as long sequences. Along with this natural approach, recent alignment-free methods, where a systemic view replaces a sequence local analysis, are based on empirical studies of frequencies of DNA k-mers in whole genomes [7,23,18]. Infogenomics is a methodology [5] based on the analysis of dictionaries of all k-mers occurring in a genome. Main ideas of this approach have been proposed by V. Manca in [14]. Briefly, given an alphabet $\Gamma = \{a, t, c, g\}$ and a genome G (seen as a string over Γ), we call k-genomic dictionary $D_k(G)$ the set of all its k-mers, which corresponds to the bipartition $H_k(G) \cup R_k(G)$, where all k-mers occurring exactly once (called *hapaxes*) are collected in $H_k(G)$ and all the others (occurring at least twice, and called *repeats*) in $R_k(G)$.

In abstract terms, any genomic string corresponds to such a bipartition. An interesting open question is how many other strings have the same bipartition of k-mers, as well as which are the string operations allowing to get them from the original one. If we add some information to the hypothesis, and we start from having the function (also called *k-gram profile*) associating k-mers to their corresponding number of occurrences in the genome, then the question was posed by Ukkonen in his seminal paper [21], and solved by Pevzner in [16]. The question has been formulated by a graph problem, the solutions as Eulerian paths, and the only two types of string operations keeping invariant the k-gram profile have resulted in transpositions and rotations. On the other hand, the question which starts from having the genomic bipartition (or a generalized h-bipartition, for a given threshold h, which separates strings occurring less than h times from strings occurring more than h times) is still open, and may be translated into a graph problem as well. Some preliminary results may be found on the variant, where, given a genome, we look for the characterization of other genomes having the same set of k-hapaxes [2].

In purely biological terms, in the human genome, where coding sequences cover less than 3%, repeats (of "significant length") account for more than 50%. In general, they are known to fall into five different classes [12]: *i)* transposon-derived repeats, often referred to as interspersed repeats, which may be distinguished in SINEs (short interspersed elements, 100-300bp long) and LINEs (6,000-8,000bp long interspersed elements), both of them are non-viral transposons randomly distributed in the genome; *ii)* inactive (partially) retroposed copies of cellular genes (including protein coding genes and small structural RNAs), usually referred to as processed *pseudogenes*; *iii)* simple sequence repeats (SSR), consisting of direct repetitions of relatively short k-mers such

[2] Manuscript in preparation, available on request to the first author.

as poly(A), poly(CA) or poly(CGG); *iv)* segmental duplications, consisting of blocks of around 10,000–300,000bp that have been copied from one region of the genome into another region; and *v)* blocks of tandemly repeated sequences (such as the case of centromeres, and telomeres, the short arms of acrocentric chromosomes). All the repeats described above are present in different proportions in every organism, and are (especially SSR) commonly assumed to be located in non-coding regions, while genetic portions are considered comprised of unique sequences.

The issue investigated in the following is related to the distribution (especially inside genes) of significantly long repeats. The exact number of different repeats of length up to about 20, for 12 organisms, together with a comparison with the randomly permuted genomic sequences, has been carried out in [5], where non-random (i.e., evolutionally selected) repeats turned out to be clearly present in real genomes yet for lengths greater than 12. For the genomic sequences we have investigated, having lengths from 490,885 to 247,000,000 bp, longest genomic repeats reach lengths of thousand bases, while randomly permuted sequences do not contain repeats longer than 22 [5,9]. Hence, we have observed that numerous long repeated motifs occur in genomic sequences, whereas repeat dictionary sizes have been compared among different genomes and repeat lengths [5]. In the

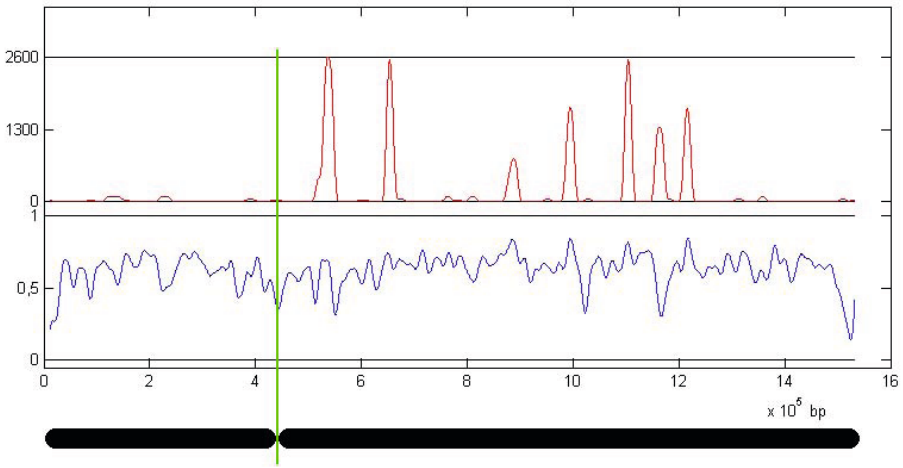


Fig. 2. Longest repeat variation in the chromosome IV of S. cerevisiae. For 12,000bp long windows, shifted along the genome every 300bp, one may see both the length variation of the longest repeat in each window as a red curve (top diagram), and the portion of the window covered by encoding sequences as a blue curve (bottom diagram). Green (vertical) line denotes the presence of a centromere, where we have no genes and repeats long only 100-150 base pairs. The same lack of (genes and) long repeats may be observed inside the telomeres, located at the extremal regions, while picks occur in the genes, covering more than 70% of the chromosome, and having an average length of 1,435bp

following we report the distribution of long repeats along genomic sequences of three specific organisms, and efficiently visualize the results for both the genomes and the entire chromosome.

In Figure 2, the diagram shows longest repeat variation inside chromosome IV of *Saccaromyces cerevisiae* (which is a yeast, an example of unicellular eukaryote, having 17 chromosomes, out of which chromosome IV is the longest one). Windows of length 12,000bp have been shifted of 300bp along the whole genomic sequence, to check the presence of longest repeats (top of Figure 2) in genomic traits, of which it is reported (bottom of Figure 2) the percentage of encoding parts. It has been a surprising result to find out that longest repeats are in most cases located in genes, rather than in inter-genic regions (such as centromeres and telomeres) in biology commonly considered as those with highly repetitive traits. Similar investigations, and similar interesting results, may be observed for

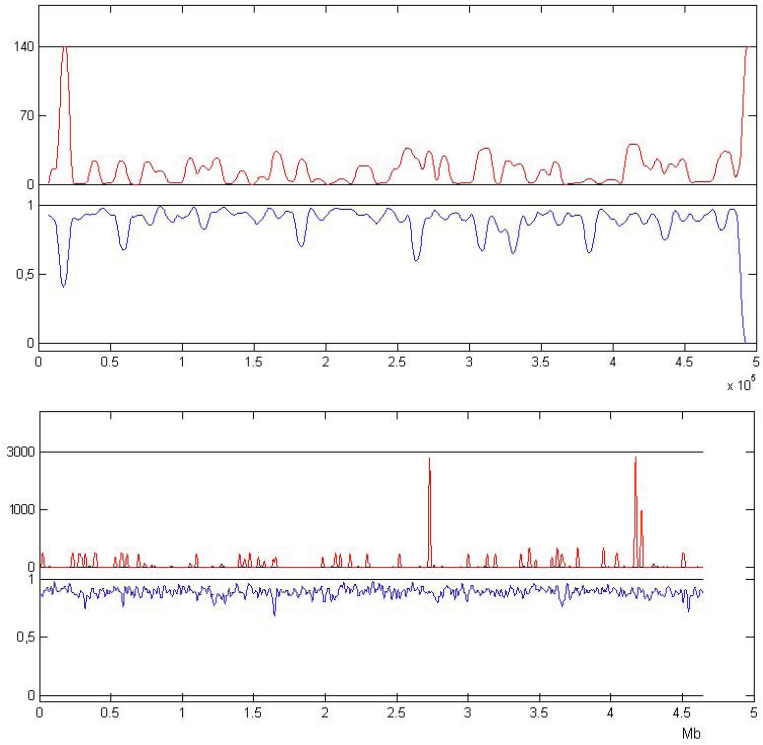


Fig. 3. Longest repeat variation inside the (circular) genomes of *N. equitans* (top) and *E. coli* (bottom). For 12,000bp long windows shifted along the genome each 300bp, one may see both the length variation of the longest repeat in each genomic window as a red curve, and the portion of the window covered by encoding sequences as a blue curve. In both cases picks of longest repeats fall inside genes. Interestingly, for *N. equitans*, the longest repeats occur in the region containing the origin of genome replication

the (circular) genomes of *Nanoarchaeum equitans* (a symbiont, having one of the shortest and more compact genomes, with the highest coding density) and *Escherichia coli* (a bacterium model, a prokaryote having one single chromosome) having no centromeres, and having extremal regions associated to the origin of replication (conventionally considered the starting point, at length zero, of the genome).

In Figure 3, we notice that significantly long repeats may be found mainly inside the encoding regions. In the extremal parts instead, where it is known there are no genes (which cover in both cases about 90% of the genome, and have respectively an average length of 787bp and 1,005bp), we have found a different and interesting behaviour. In *N. equitans*'s genome, the longest repeats are located in the region containing the origin of replication. It is a particular parasitic organism, with a minimal number of genes (and very few paralogs) essential for survival and having very different functions (i.e., sequences): the longest repeat in genes is in fact 53bp long. However, even in this case, most repeats (of any length) have been found principally inside genes, as one may observe in Figure 4.

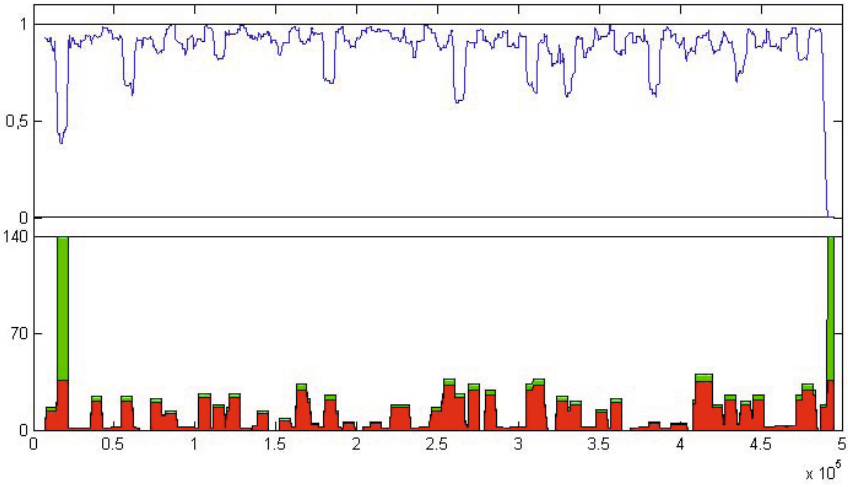


Fig. 4. Repeat variations inside the (circular) genome of *N. equitans*, where 12,000bp long windows have been shifted along the genome each 300bp. In the diagram, one may see the portion of each window covered by encoding sequences as a blue curve (top), and the length of repeats (bottom) respectively, contained inside genes (red rectangles) and outside genes (green rectangles)

4 Gene Networks

Motivated by the results discussed in the previous section, we wonder which genes contain (possibly common) long repeats. A network-based approach to investigate genetic repeats has been given by *repeat sharing gene networks*, defined

in [5] by a collection of genes connected if containing in their own sequence at least one k-repeat in common. More precisely, we define a family of networks N_k (parameterized with a natural number k) which have their gene-nodes connected by edges, labelled by the k-repeats shared between the two corresponding gene sequences. For example, in Figure 5 a gene network is reported for *N. equitans*, for repeat length $k = 30$, where the 30bp long repeats common to connected genes are pointed out for a specific clique. A surprising feature of these networks is that each of their complete clusters (or cliques) has at least one repeat occurring on all its edges, with also the same reading frame in all the involved genes. In the case pointed out in Figure 5, such a repeat is `CCGGAGGTCCCGGGTTCGAATCCCGGCGGG`.

Work in progress [6] is analyzing the structure of these networks (with standard methods of graph theory and gene ontology techniques) for our three specific organisms (*N. equitans*, *E. coli*, and *S. cerevisiae*), and interesting regularity properties have been found, despite the diversity of the three organisms and of their genome structure. For example, the relative (with respect to the number

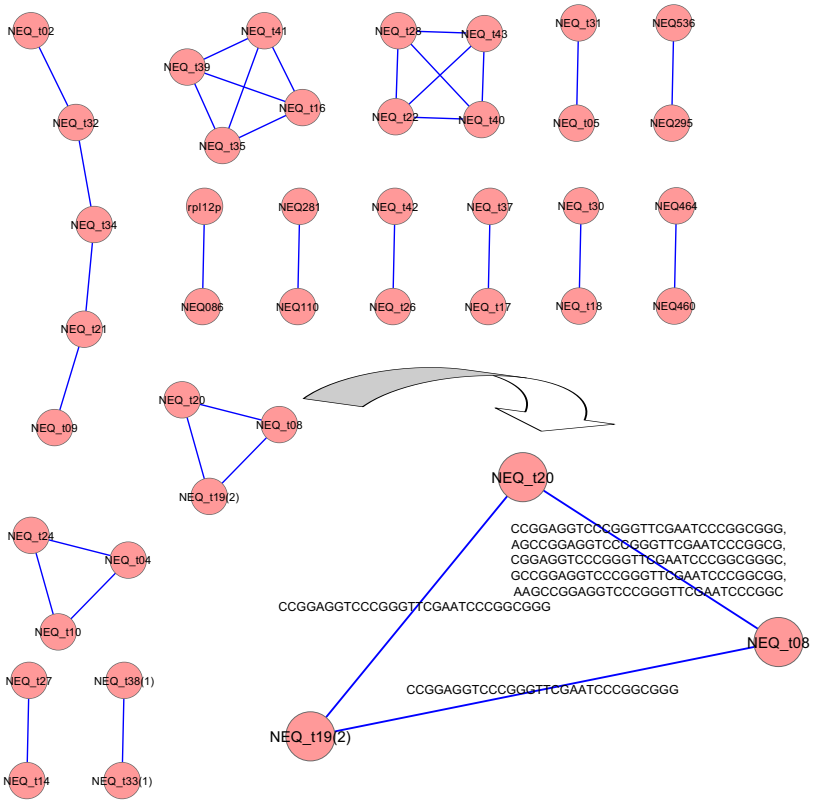


Fig. 5. Repeat sharing gene networks of *N. equitans* for repeat length $k = 30$

of genes involved in the network) variation of the number of genes involved in cliques, with respect to k, appears the same for the three organisms. Moreover, it has been computed that relatively few genes are involved in cliques, and the number of cliques in the network varies (with k) similarly for the three organisms.

As a further empirical result from the study currently developed on these organisms [6], we have discovered that $k > 30$ is the condition to have networks (and cliques) of paralogs and pseudogenes, while we have biologically significant networks (where one may search for size of words of an hypothetic genomic or genetic language) within the k-values range of $20 \leq k \leq 30$. In particular, the study has been carried out by means of an analysis of complete clusters (cliques such as those in Figure 6), which cover all the genes of a network with k high enough (over 30).

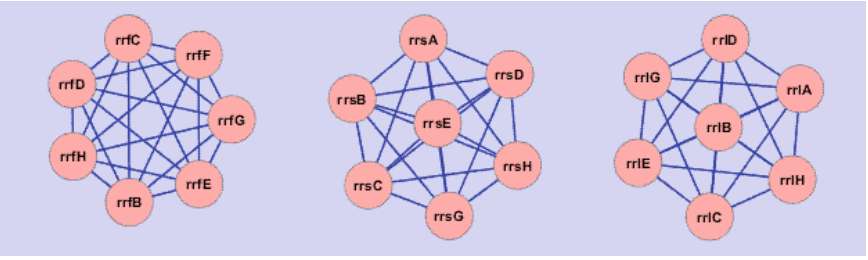


Fig. 6. Different cliques found in the family of repeat sharing gene networks of *E. coli*

5 Conclusions

Repeat motifs covering most part of regulative DNA have been discovered very important to understand both the transcriptional process and the evolution mechanisms. The comprehension of their formation and distribution could justify crucial differences between species. In this paper a systematic analysis of more significant (in terms of length) genomic repeats has been performed and applied to three specific organisms. In all cases very long repeats have been found inside the coding regions, despite the common believe that repetitive regions are mainly inter-genic. Further computational experiments have to be carried out to confirm this result on more complex genomes. Ranges of interest for repeat lengths k have been identified as well, by means of a characterization of repeat sharing gene networks, a method of sequence analysis currently under investigation [6].

As a future work, we would like to apply the infogenomic methodology to better understand the intricate links between promoters and corresponding genes. A first extension of the above repeat analysis into genetic portions could comprehend also the flanking 5′-UTRs, non-coding parts which usually control the stability of mRNA and the efficiency of its translation into a protein. A systematic analysis of transcripts is indeed essential to identify regulatory regions,

where the transcript may be considered as the basic unit of heredity. The bipartition of a genomic dictionary in hapax and repeat words corresponds to a representation of genomic information, and the repeat sharing gene networks (where genes are linked if having common factors) may be of interest to recognize similarities among genes, such as conserved miRNA binding sites.

These networks will be further investigated in their structure, for example by methods from the Random Matrix Theory (which, given the incidence matrices associated to networks, change the eigenvalue gap distribution from Gaussian to a power law), and may be investigated in some of their variants, even having biomedical applications, such as those defined over the exome rather than over the whole genes. Finally, another research line will concern with the intergenomic character of hapaxes and repeats. The question is about which hapaxes (resp. repeats) of a given genome occur in other genomes of a certain class by keeping their status of hapax (resp. repeat) when compared to the new context of words.

Let us finally point out that the ENCODE and the infogenomic projects share a dictionary based view of genome analysis and an attempt to gather global, holistic information contained in genomes, in order to understand how genomes orchestrate coordinated processes to keep alive the cell. Recent advances of computational genomics in these contexts open up new perspectives of genome analysis, focused on the understanding of a genomic code which explains how promoters communicate with corresponding (proximal or distal) genes, with the aim to generate a sequence based dynamical model for mechanisms of gene activation and regulation [9]. We expect that systemic alignment-free methods can be helpful in cases where alignment methods fail. For example, similarity of sequences which code for proteins having very similar functions could be revealed by a dictionary based score, even in those cases where the alignment score is very low. Also, on the other hand, such methods could suggest suitable ranges of parameters to set, for BLAST or other alignment based algorithms, when looking for similar sequences (for example, common words of length 30 seem enough to find paralogs and pseudogenes, which are very similar in their sequence structure). Finally, combinatorial and algorithmical results from stringology field may be exported to investigate, in terms of dictionaries, long (real) strings, in order to identify some biologically significant characterization of genomes.

References

1. Bartel, D.P.: MicroRNAs: Target recognition and regulatory functions. Cell 136(2), 215–233 (2009), doi:10.1016/j.cell.2009.01.002
2. Bilbao, C., et al.: The relationship between microsatellite instability and PTEN gene mutations in endometrial cancer. International Journal Cancer 119(3), 563–570 (2006)
3. Burden, C.J., Jing, J., Wilson, S.R.: Alignment-free sequence comparison for biologically realistic sequences of moderate length. Statistical Applications in Genetics and Molecular Biology 11(1), Article 3 (2012)
4. Castellini, A., et al.: Genome classification by dictionary based indexes. Poster. Presented at the Int. Conf. on Pattern Recognition in Bioinformatics, PRIB (2011)

5. Castellini, A., Franco, G., Manca, V.: A dictionary based informational genome analysis. BMC Genomics 13(1), 485 (2012), doi:10.1186/1471-2164-13-485
6. Castellini, A., Franco, G., Milanese, A.: A genome analysis based on repeat sharing gene networks (in preparation)
7. Chor, B., et al.: Genomic DNA k-mer spectra: models and modalities. Genome Biology 10, R108 (2009)
8. Dunham, I., et al.: (The ENCODE Project Consortium): An integrated encyclopedia of DNA elements in the human genome. Nature 489, 57–74 (2012)
9. Franco, G.: Perspectives in computational genome analysis, book chapter in Discrete and Topological Models in Molecular Biology. Springer (to appear, 2013)
10. Friedman, R.C., Farh, K.K., Burge, C.B., Bartel, D.P.: Most mammalian mRNAs are conserved targets of microRNAs. Genome Res. 19(1), 92–105 (2009)
11. Gottesman, S.: The small RNA regulators of Escherichia coli: roles and mechanisms. Annu. Rev. Microbiol. 58, 303–328 (2004)
12. International Human Genome Sequencing Consortium, Initial sequencing and analysis of the human genome. Nature 409, 860–921 (2001)
13. Liran, I., et al.: Cell lineage analysis of acute leukemia relapse uncovers the role of replication-rate heterogeneity and microsatellite instability. Blood 120, 603–612 (2012)
14. Manca, V.: Infobiotics. Springer (2013)
15. Song, K., Ren, J., Zhai, Z., Liu, X., Deng, M., Sun, F.: Alignment-free sequence comparison based on next-generation sequencing reads. Journal of Computational Biology 20 (2), 64–79 (2013)
16. Pevzner, P.A.: DNA physical mapping and alternating eulerian cycles in colored graphs. Algorithmica, 77–105 (1995)
17. Sanyal, A., et al.: The long-range interaction landscape of gene promoters. Nature 489, 109–113 (2012)
18. Searls, D.B.: The language of genes. Nature 420, 211–217 (2002)
19. Sharma, C.M., Vogel, J.: Experimental approaches for the discovery and characterization of regulatory small RNA. Curr. Opin. Microbiol. 12, 536–546 (2009)
20. Tay, Y., et al.: Coding-independent regulation of the tumor suppressor PTEN by competing endogenous mRNAs. Cell 147(2), 344–357 (2011)
21. Ukkonen, E.: Approximate string matching with q-grams and maximal matches. Theoretical Computer Science 92(1), 191–211 (1992)
22. Wagner, E.G.H., Simon, R.W.: Antisense RNA control in bacteria, phages, and plasmids. Annual Review of Microbiology 48, 713–742 (1994)
23. Zhou, F., Olman, V., Xu, Y.: Barcodes for genomes and applications. BMC Bioinformatics 9, 546 (2008)

Local Computability for Ordinals

Johanna N.Y. Franklin[1], Asher M. Kach[2,⋆], Russell Miller[3,4,⋆⋆], and Reed Solomon[1]

[1] Department of Mathematics, University of Connecticut, 196 Auditorium Road, U-3009, Storrs, CT 06269-3009, U.S.A.
{johanna.franklin,david.solomon}@uconn.edu

[2] Department of Mathematics, University of Chicago, 5734 S. University Avenue, Chicago, IL 60637, U.S.A.
asher.kach@gmail.com

[3] Queens College of the City University of New York, 65-30 Kissena Blvd., Queens, NY 11367-1597, U.S.A.
Russell.Miller@qc.cuny.edu

[4] CUNY Graduate Center, 365 Fifth Avenue, New York, NY 10016, U.S.A.

Abstract. We examine the extent to which well orders satisfy the properties of local computability, which measure how effectively the finite suborders of the ordinal can be presented. Known results prove that all computable ordinals are perfectly locally computable, whereas ω_1^{CK} and larger countable ordinals are not. We show that perfect local computability also fails for uncountable ordinals, and that ordinals $\alpha \geq \omega_1^{\mathrm{CK}}$ are θ-extensionally locally computable for all $\theta < \omega_1^{\mathrm{CK}}$, but not when $\theta > \omega_1^{\mathrm{CK}}$, nor when $\theta = \omega_1^{\mathrm{CK}} \leq \alpha < \omega_1^{\mathrm{CK}} \cdot \omega$.

1 Introduction

Local computability represents an effort to give effective presentations of structures, such as the fields of real and complex numbers, which admit computation on their elements by simple algebraic algorithms and therefore, despite their uncountability, feel as though they ought to have computable presentations. Full definitions and much more analysis are given in [3,4,5], and we offer some basic definitions below. Local computability applies to linear orders as well as to fields and other structures, and the intention of this work is to investigate local computability for ordinals, the most ubiquitous linear orders in mathematical logic. We started with a particular eye on uncountable ordinals, but soon found large (noncomputable) countable ordinals to be of similar interest. For example, can we use the ideas and terminology of local computability to draw distinctions between noncomputable countable ordinals and uncountable ordinals? Or

⋆ The second author was partially supported by a grant from the Packard Foundation through a Post-Doctoral Fellowship.

⋆⋆ The third author was partially supported by grant # DMS–1001306 from the National Science Foundation, by the Centre de Recerca Matemática, the Isaac Newton Institute, and the European Science Foundation, and by several PSC-CUNY grants from the Research Foundation of The City University of New York.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 161–170, 2013.

between limit and successor ordinals? Or between cardinals and noncardinals? In this paper, we give almost complete negative answers to these questions. All noncomputable ordinals are α-extentionally locally computable (defined below) for $\alpha < \omega_1^{\mathrm{CK}}$ and are not β-extensionally locally computable for $\beta > \omega_1^{\mathrm{CK}}$. If $\omega_1^{\mathrm{CK}} \leq \gamma \leq \omega_1^{\mathrm{CK}} \cdot \omega$, then γ is not ω_1^{CK}-extentionally locally computable, but it remains open whether there could be a larger ordinal which is ω_1^{CK}-extentionally locally computable.

We now give the background definitions. In this context, one typically works with a fixed class of structures which is closed under taking finitely generated substructures. For example, one might consider the models of a $\forall$-axiomatizable theory T or (as we do here) a nonaxiomatizable class of models such as the ordinals.

Definition 1. *A* simple cover *of a structure $\mathcal{S}$ is a countable collection $\mathfrak{A}$ of models $\{\mathcal{A}_i\}_{i\in I}$ of T, each generated by a finite tuple $\boldsymbol{a}_i$, such that every finitely generated substructure of $\mathcal{S}$ is isomorphic to some $\mathcal{A}_i$ and every $\mathcal{A}_i$ embeds into $\mathcal{S}$.*

A simple cover is computable *if every $\mathcal{A}_i \in \mathfrak{A}$ is a computable structure with domain an initial segment of ω.*

A simple cover is uniformly computable *if the sequence $\{(\mathcal{A}_i, \boldsymbol{a}_i)\}_{i\in I}$ can be given uniformly computably, including a strong index for each $\boldsymbol{a}_i$.*

Definition 2. *A* cover *of $\mathcal{S}$ consists of a simple cover $\mathfrak{A}$ of $\mathcal{S}$ along with sets $I_{ij}^{\mathfrak{A}}$ (for all $\mathcal{A}_i, \mathcal{A}_j \in \mathfrak{A}$) of injective homomorphisms $f : \mathcal{A}_i \hookrightarrow \mathcal{A}_j$ satisfying:*

- *For all finitely generated substructures $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{S}$, there exists $i, j \in \omega$, $f \in I_{ij}^{\mathfrak{A}}$, and $\beta : \mathcal{A}_i \cong \mathcal{B}$ and $\gamma : \mathcal{A}_j \cong \mathcal{C}$ with $\beta = \gamma \circ f$.*
- *For all k and m and $g \in I_{k,m}^{\mathfrak{A}}$, there exist finitely generated substrutures $\mathcal{D} \subseteq \mathcal{E} \subseteq \mathcal{S}$ and isomorphisms $\delta : A_k \cong \mathcal{D}$ and $\epsilon : A_m \cong \mathcal{E}$ with $\delta = \epsilon \circ g$.*
- *The **Amalgamation Property**: for every i, j, k and all maps $f \in I_{ij}^{\mathfrak{A}}$ and $g \in I_{ik}^{\mathfrak{A}}$, there exists m and maps $h \in I_{jm}^{\mathfrak{A}}$ and $p \in I_{km}^{\mathfrak{A}}$ with $p \circ g = h \circ f$.*

We often refer to the elements of $\mathfrak{A}$ as the objects of the cover and the elements of the $I_{ij}^{\mathfrak{A}}$ as the maps of the cover.

A cover is 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 $i, j \in \omega$,*

$$I_{ij}^{\mathfrak{A}} = \{\varphi_e \restriction \mathcal{A}_i \ : \ \langle i, j, e\rangle \in W\}.$$

The structure $\mathcal{S}$ is locally computable *if it has a uniformly computable cover.*

In past articles on local computability, the Amalgamation Property has not always been included in the definition of a cover. Therefore, in the interest of clarity, we shall occasionally remind the reader that we are working in the context of this property.

Definition 3. *Let $\mathfrak{A}$ be a cover of a structure $\mathcal{S}$. An $\mathcal{A}_i \in \mathfrak{A}$ matches a substructure $\mathcal{B} \subseteq \mathcal{S}$* extensionally *if there is an isomorphism $\beta : \mathcal{A}_i \cong \mathcal{B}$ satisfying:*

- *For every finitely generated* $\mathcal{C}$ *with* $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{S}$, *there exists* $j \in \omega$, $f \in I^{\mathfrak{A}}_{ij}$, *and* $\gamma : \mathcal{A}_j \cong \mathcal{C}$ *with* $\beta = \gamma \circ f$.
- *For every* $m \in \omega$ *and* $g \in I^{\mathfrak{A}}_{i,m}$, *there exists an* $\mathcal{E} \subseteq \mathcal{S}$ *and* $\epsilon : \mathcal{A}_m \cong \mathcal{E}$ *with* $\mathcal{B} \subseteq \mathcal{E}$ *and* $\beta = \epsilon \circ g$.

The map β *is termed an* extensional match *between* $\mathcal{A}_i$ *and* $\mathcal{B}$.

Definition 4. *Let* $\mathfrak{A}$ *be a cover of a structure* $\mathcal{S}$. *Every isomorphism* $\beta : \mathcal{A}_i \cong \mathcal{B}$, *where* $\mathcal{B} \subseteq \mathcal{S}$ *is a finitely generated substructure, is* 0-extensional.

For an ordinal $\theta > 0$, *an isomorphism* $\beta : \mathcal{A}_i \cong \mathcal{B}$ *is* θ-extensional *if:*

- *For every finitely generated* $\mathcal{C}$ *with* $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{S}$ *and every ordinal* $\zeta < \theta$, *there exists* $j \in \omega$, $f \in I^{\mathfrak{A}}_{ij}$, *and a* ζ*-extensional* $\gamma : \mathcal{A}_j \cong \mathcal{C}$ *with* $\beta = \gamma \circ f$.
- *For every* $m \in \omega$, $g \in I^{\mathfrak{A}}_{i,m}$, *and ordinal* $\zeta < \theta$, *there exists* $\mathcal{E} \subseteq \mathcal{S}$ *and* ζ*-extensional* $\epsilon : \mathcal{A}_m \cong \mathcal{E}$ *with* $\mathcal{B} \subseteq \mathcal{E}$ *and* $\beta = \epsilon \circ g$.

A uniformly computable cover $\mathfrak{A}$ *of* $\mathcal{S}$ *is* θ-extensional *if for every* $\mathcal{A}_i \in \mathfrak{A}$ *there is a* θ*-extensional isomorphism* $\beta : \mathcal{A}_i \cong \mathcal{B}$ *to some finitely generated substructure* $\mathcal{B} \subseteq \mathcal{S}$ *and for every finitely generated substructure* $\mathcal{E} \subseteq \mathcal{S}$ *there is a* θ*-extensional isomorphism* $\epsilon : \mathcal{A}_j \cong \mathcal{E}$ *from some* $\mathcal{A}_j \in \mathfrak{A}$.

If such a uniformly computable cover exists, we say that $\mathcal{S}$ *is* θ-extensionally locally computable *or, more simply,* θ-extensional.

Definition 5. *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:*

- *Each element of* M *is an embedding of an* $\mathcal{A}_i$ *into* $\mathcal{S}$.
- *For every* $\mathcal{A}_i \in \mathfrak{A}$, *there exists a* $\beta \in M$ *with domain* $\mathcal{A}_i$.
- *For every finitely generated substructure* $\mathcal{B} \subseteq \mathcal{S}$, *there exists a* $\beta \in M$ *with range* $\mathcal{B}$.
- *For every* $\mathcal{A}_i \in \mathfrak{A}$, *every* $\beta \in M$ *with domain* $\mathcal{A}_i$, *and every finitely generated substructure* $\mathcal{C} \subseteq \mathcal{S}$ *with* $\beta(\mathcal{A}_i) \subseteq \mathcal{C}$, *there exists* $\mathcal{A}_j \in \mathfrak{A}$, $\gamma \in M$ *with domain* $\mathcal{A}_j$ *and image* $\mathcal{C}$, *and* $f \in I^{\mathfrak{A}}_{ij}$ *with* $\beta = \gamma \circ f$.
- *For every* $\mathcal{A}_i \in \mathfrak{A}$, *every* $\beta \in M$ *with domain* $\mathcal{A}_i$, *and every* $\mathcal{A}_m \in \mathfrak{A}$ *and every* $g \in I^{\mathfrak{A}}_{i,m}$, *there exists* $\epsilon \in M$ *with domain* $\mathcal{A}_m$ *with* $\beta = \epsilon \circ g$.

If $\mathcal{S}$ *has a uniformly computable cover* $\mathfrak{A}$ *with a correspondence system* M, *then we say* $\mathcal{S}$ *is* ∞-extensionally locally computable.

However, the definition applies perfectly well to countable structures, and Miller proved the following connection between local computability and computable presentability for countable structures (cf. [5,4]).

Theorem 1. *Let* $\mathcal{S}$ *be a countable structure.* $\mathcal{S}$ *is isomorphic to a computable structure if and only if* $\mathcal{S}$ *has an* ∞*-extensional computable cover (with the AP).*

This equivalence helped to establish ∞-extensionality as the ultimate goal, when one desires to prove that a particular structure of arbitrary cardinality is "nicely presentable." It also justifies our decision to consider only covers with the AP.

2 Failures of Extensionality for Ordinals

We now show that having sufficiently high ordinal levels of extensionality is sufficient for ∞-extensional local computability. We work in $\mathcal{L}_{\omega_1,\omega}$ allowing countable conjunctions and disjunctions, and we use the standard notation Σ_α for $\alpha < \omega_1$ to calibrate the complexity of our formulas. (See [1] for additional background on this logic.)

Lemma 1. *Let $\mathfrak{A}$ be a cover of a structure $\mathcal{S}$. Suppose $\mathcal{A}_i \in \mathfrak{A}$ and $\psi : \mathcal{A}_i \to \mathcal{C}$ is a θ-extensional map onto a substructure $\mathcal{C}$ of $\mathcal{S}$, and let h be an automorphism of $\mathcal{S}$. Then $h \circ \psi$ is also θ-extensional.*

Proof. We induct on θ. For $\theta = 0$, if ψ is 0-extensional, it is an injective homomorphism, and therefore so is $h \circ \psi$. Thus $h \circ \psi$ is 0-extensional.

For $\theta > 0$, if $f \in I_{ij}^{\mathfrak{A}}$ lifts to an inclusion $\mathcal{C} \subseteq \mathcal{D}$ via ψ and a ζ-extensional map φ (for any $\zeta < \theta$), then f also lifts to the inclusion $h(\mathcal{C}) \subseteq h(\mathcal{D})$ via $h \circ \psi$ and the map $h \circ \varphi$. By induction on θ, the map $h \circ \varphi$ is also ζ-extensional. It follows that $h \circ \psi$ is θ-extensional.

Lemma 2. *Suppose that $\mathfrak{A}$ is a computable cover of a structure $\mathcal{S}$, and that $\boldsymbol{a}$ is an n-tuple from an object $\mathcal{A}_i \in \mathfrak{A}$. If φ and ψ are both θ-extensional maps from $\mathcal{A}_i$ into $\mathcal{S}$, then the tuples $\varphi(\boldsymbol{a})$ and $\psi(\boldsymbol{a})$ satisfy exactly the same Σ_θ-formulas in $\mathcal{S}$.*

We shall sometimes refer to the set of these Σ_θ formulas as the *Σ_θ-theory of $\boldsymbol{a}_i$ in* $\mathfrak{A}$, and will speak of $\boldsymbol{a}_i$ *satisfying* various formulas *in* $\mathfrak{A}$. The lemma can be seen as saying that this notion is well-defined: in the theory of the cover $\mathfrak{A}$, $\boldsymbol{a}_i$ satisfies exactly those Σ_θ formulas that its image, under an arbitrary θ-extensional map, satisfies in $\mathcal{S}$. (Alternatively, one can define the Σ_θ-theory of $\boldsymbol{a}_i$ in $\mathfrak{A}$ by using $\exists$-quantifiers to refer to the existence of embeddings $f \in I^{\mathfrak{A}}$ from $\mathcal{A}_i$ into other objects $\mathcal{A}_j$ of $\mathfrak{A}$, such that $(\mathcal{A}_j, f(\boldsymbol{a}_i))$ satisfies the formula inside the $\exists$-quantifier. This is natural, and is equivalent to the above definition.)

Proof. For $\theta = 0$, this follows from φ and ψ both being 0-extensional, i.e., being embeddings of $\mathcal{A}_i$ into $\mathcal{S}$. For $\theta > 0$, suppose

$$\mathcal{S} \models \bigvee_{k \in \omega} (\exists y)\, [P_k(\psi(\boldsymbol{a}), y)]$$

where each P_k is a Π_{ζ_k}-formula with $\zeta_k < \theta$. Fix $k \in \omega$ and $y \in \mathcal{S}$ such that $\mathcal{S} \models P_k(\psi(\boldsymbol{a}), y)$. The inclusion $\mathrm{range}(\psi) \subseteq \mathrm{range}(\psi) \cup \{y\}$ in $\mathcal{S}$ must be the lift of some f in some $I_{ij}^{\mathfrak{A}}$, via ψ and some ζ_k-extensional $\psi' : \mathcal{A}_j \to \mathrm{range}(\psi) \cup \{y\}$. But this f must also lift to an inclusion $\mathrm{range}(\varphi) \subseteq \mathcal{D}$ in $\mathcal{S}$ via φ and some ζ_k-extensional φ'. By induction, the Π_{ζ_k} formula $P_k(\boldsymbol{x}, y)$, being known to hold of $(\psi'(f(\boldsymbol{a}), y) = (\psi(\boldsymbol{a}), y)$, must also hold of $(\varphi'(f(\boldsymbol{a})), z) = (\varphi(\boldsymbol{a}), z)$, where $z := \varphi'(\psi'^{-1}(y))$. Thus $\varphi(\boldsymbol{a})$ also satisfies $\bigvee_k (\exists y)\, [P_k(\boldsymbol{x}, y)]$. Finally, by a symmetric argument, if $\varphi(\boldsymbol{a})$ satisfies this Σ_θ formula, then so does $\psi(\boldsymbol{a})$.

Several definitions of Scott rank exist in the literature (cf. [1]). For our purposes, only the following property of the Scott rank (and not the exact definition) matters: whenever $\mathcal{S}$ is a countable structure of Scott rank ζ and $\boldsymbol{x}$ and $\boldsymbol{y}$ are n-tuples of elements from $\mathcal{S}$ (for any n) which satisfy exactly the same Π_ζ formulas in n variables, there must exist an automorphism of $\mathcal{S}$ mapping each x_i to the corresponding y_i.

Lemma 3. *Fix ordinals θ and ζ with $\theta > \zeta$. Let $\mathfrak{A}$ be a θ-extensional cover of a countable structure $\mathcal{S}$ with Scott rank ζ. Suppose $\mathcal{A}_i \in \mathfrak{A}$ and $\psi : \mathcal{A}_i \to \mathcal{S}$ is a ζ-extensional map. Then ψ is also θ-extensional.*

Proof. Since $\mathfrak{A}$ is a θ-extensional cover, we know that $\mathcal{A}_i$ is the domain of some θ-extensional map $\varphi : \mathcal{A}_i \to \mathcal{S}$. Let $\boldsymbol{a}$ be a finite tuple generating $\mathcal{A}_i$. Then by Lemma 2, the tuples $\varphi(\boldsymbol{a})$ and $\psi(\boldsymbol{a})$ satisfy exactly the same Π_ζ-formulas in $\mathcal{S}$. Since $\mathcal{S}$ has Scott rank ζ, there must be an automorphism h of $\mathcal{S}$ mapping $\varphi(\boldsymbol{a})$ onto $\psi(\boldsymbol{a})$. But then $h \circ \varphi = \psi$ since $\boldsymbol{a}$ generates $\mathcal{A}_i$, and so by Lemma 1, the map ψ is also θ-extensional.

Proposition 1. *For a countable structure $\mathcal{S}$, the following are equivalent.*

1. *The structure $\mathcal{S}$ is computably presentable.*
2. *The structure $\mathcal{S}$ is perfectly locally computable (as defined in [5]).*
3. *The structure $\mathcal{S}$ has an ∞-extensional computable cover with the Amalgamation Property.*
4. *There is an ordinal θ strictly greater than the Scott rank of $\mathcal{S}$, such that $\mathcal{S}$ has a θ-extensional computable cover with the Amalgamation Property.*

Proof. The equivalence of (1), (2), and (3) is shown in [4, Thm 6.3]; some of it was originally proven by Miller and Mulcahey in [5]. Since (3) $\implies$ (4) is trivial, we need only show that (4) $\implies$ (3). Fix a θ-extensional computable cover $\mathfrak{A}$ of $\mathcal{S}$. We claim that the set M of all θ-extensional maps ψ of objects $\mathcal{A}_i$ into $\mathcal{S}$ must be a correspondence system. Clearly every $\mathcal{A}_i$ is the domain of such a map and every finitely generated $\mathcal{D} \subseteq \mathcal{S}$ is the image of such a map. Moreover, for any $\psi \in M$, say with domain $\mathcal{A}_i$, and every $f \in I_{ij}^{\mathfrak{A}}$, we can lift f to an inclusion via ψ and a ζ-extensional φ (since $\zeta < \theta$), and by Lemma 3, φ is also in M. Likewise, every inclusion of range(ψ) is the lift of some $f \in I_{ij}^{\mathfrak{A}}$, for some j, via some ζ-extensional φ, and again, by Lemma 3, this φ actually lies in M.

Corollary 1. *For every $\theta > \omega_1^{\mathrm{CK}}$, the ordinal ω_1^{CK} is not θ-extensionally locally computable.*

Proof. The Scott rank of the ordinal ω_1^{CK} (as a linear order) is exactly ω_1^{CK}. Hence, as a countable structure with no computable presentation, ω_1^{CK} cannot be θ-extensionally locally computable for any $\theta > \omega_1^{\mathrm{CK}}$, by Proposition 1.

In Proposition 2 and Theorem 3, we strengthen this corollary to cover some of the case $\theta = \omega_1^{\mathrm{CK}}$. The situation for $\theta < \omega_1^{\mathrm{CK}}$ will be handled in Theorem 4, and the rest of the case $\theta > \omega_1^{\mathrm{CK}}$ in Theorem 2.

Lemma 4 (Folklore; cf., e.g., [1]). *For each finite sequence of ordinal* $\alpha_0 < \cdots < \alpha_k < \omega_1^{\mathrm{CK}}$*, there is a computable infinitary formula* $\lambda(x_0, \ldots, x_k)$ *(in the language of linear orders) such that for every ordinal* γ*,* $\gamma \models \lambda(\beta_0, \ldots, \beta_k)$ *if and only if* $\beta_i = \alpha_i$ *for each* $i \leq k$*.*

Proposition 2. *There is no* ω_1^{CK}*-extensional computable cover (with AP) of the ordinal* ω_1^{CK} *itself.*

Proof. Suppose $\mathfrak{A}$ were such a cover. By Theorem 1, if there were a correspondence system M for $\mathfrak{A}$ and ω_1^{CK}, then there would be a computable copy of ω_1^{CK}, yielding a contradiction. Our goal is to define such a correspondence system M.

For any $\mathcal{A}_i \in \mathfrak{A}$ with $\mathcal{A}_i = \mathbf{a}$, let ψ and ψ' be ω_1^{CK}-extensional maps from $\mathcal{A}_i$ into ω_1^{CK}. The tuples $\psi(\mathbf{a})$ and $\psi'(\mathbf{a})$ satisfy the same $\Pi_{\omega_1^{\mathrm{CK}}}$ formulas in ω_1^{CK}. By Lemma 4, this forces ψ and ψ' to agree on $\mathbf{a}$, and hence on $\mathcal{A}_i$. Thus, every $\mathcal{A}_i \in \mathfrak{A}$ is the domain of exactly one ω_1^{CK}-extensional map ψ_i into ω_1^{CK}. In fact, for every $\mathcal{A}_i$, there is a $\delta_i < \omega_1^{\mathrm{CK}}$ such that ψ_i is the unique δ_i-extensional map of $\mathcal{A}_i$ into ω_1^{CK}. (We assume δ_i is least with this property.)

Let M be the collection of all ω_1^{CK}-extensional maps ψ_i. We claim M is a correspondence system. The first three properties of a correspondence system follow immediately from the fact that $\mathfrak{A}$ is an ω_1^{CK}-extensional computable cover.

To verify the fourth property, fix $\mathcal{A}_i \in \mathfrak{A}$ and $\beta \in M$ with domain $\mathcal{A}_i$. Since β is ω_1^{CK}-extensional, $\beta = \psi_i$. Fix a finite $\mathcal{C}$ such that $\psi_i(\mathcal{A}_i) \subseteq \mathcal{C} \subseteq \omega_1^{\mathrm{CK}}$. By Lemma 4, let $\theta < \omega_1^{\mathrm{CK}}$ be such that $\mathcal{C}$ is defined by a Σ_θ formula in ω_1^{CK}. Using the fact that $\mathfrak{A}$ is an ω_1^{CK}-extensional cover, fix $\mathcal{A}_j$, $f \in I_{ij}^{\mathfrak{A}}$ and an θ-extensional map γ such that $\beta = \psi_i = \gamma \circ f$. Since $\mathcal{C}$ is defined by a Σ_θ formula, it follows that $\delta_j \leq \theta$ and hence $\gamma = \psi_j$. Therefore, γ is ω_1^{CK}-extensional and hence $\gamma \in M$ as required.

The fifth property follows by an similar argument which we leave to the reader.

Theorem 2. *If* $\alpha > \omega_1^{\mathrm{CK}}$*, then* α *has no* $(\omega_1^{\mathrm{CK}} + 1)$*-extensional computable cover with the Amalgamation Property. Indeed, no computable cover of such an* α *can have any object with an* $(\omega_1^{\mathrm{CK}} + 1)$*-extensional map into* α *whose image contains* ω_1^{CK}*.*

Proof. We show that from such a cover $\mathfrak{A}$, we could construct a computable presentation $\mathcal{S}$ of ω_1^{CK}. Let $\mathcal{A}_{i_0}$ be the object with an $(\omega_1^{\mathrm{CK}}+1)$-extensional map φ_0 such that $\varphi_0(x_0) = \omega_1^{\mathrm{CK}}$ for some $x_0 \in \mathcal{A}_{i_0}$ (hereafter fixed). The argument in a nutshell is that we can watch for embeddings $g \in I^{\mathfrak{A}}$ mapping $\mathcal{A}_{i_0}$ into other objects $\mathcal{A}_j$ of $\mathfrak{A}$. When we find such a g, it must lift to an inclusion in α via φ_0 and some ω_1^{CK}-extensional φ_1, and so every element $y < g(x_0)$ in $\mathcal{A}_j$ is forced to map to some ordinal $< \varphi_1(g(x_0)) = \omega_1^{\mathrm{CK}}$ in α. Since the map is ω_1^{CK}-extensional, Lemma 2 shows that in the theory of $\mathfrak{A}$, y satisfies some identifying formula from Lemma 4. We then use the AP to amalgamate $\mathcal{A}_j$ together with the portion of $\mathcal{S}$ already built, and either we see that y maps to some element already in $\mathcal{S}$, or else we add a new element to $\mathcal{S}$ to correspond to this y. Since every ordinal $< \omega_1^{\mathrm{CK}}$ corresponds to some such y in some such $\mathcal{A}_j$, the $\mathcal{S}$ built this way is actually a copy of ω_1^{CK}.

We start building $\mathcal{S}$ by setting $\mathcal{B}_0 = \{y \in \mathcal{A}_{i_0} : y < x\}$, letting $\mathcal{S}_0$ be the (possibly empty) linear order $\{0, 1, \ldots, |\mathcal{B}_0| - 1\}$ under $<$, and defining $p_0 : \mathcal{S}_0 \to \mathcal{B}_0$ to be an order-isomorphism.

At stage $s + 1$, we begin with a finite order $\mathcal{S}_s$ and with some sequence of objects and embeddings from $\mathfrak{A}$, given effectively:

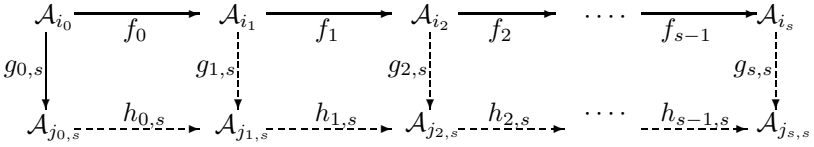
$$\mathcal{A}_{i_0} \hookrightarrow \mathcal{A}_{i_1} \hookrightarrow \cdots \hookrightarrow \mathcal{A}_{i_s},$$

where each embedding $f_t : \mathcal{A}_{i_t} \hookrightarrow \mathcal{A}_{i_{t+1}}$ lies in $I^{\mathfrak{A}}_{i_t i_{t+1}}$. By induction, we know an isomorphism p_s from $\mathcal{S}_s$ onto a suborder $\mathcal{B}_s$ of $\mathcal{A}_{i_s}$, with every element of $\mathcal{B}_s$ below the element $x_s = f_{s-1}(x_{s-1})$ in $\mathcal{A}_{i_s}$, which is the image of x_0 under $(f_{s-1} \circ \cdots \circ f_0)$.

We now search through $I^{\mathfrak{A}}$ for the first map $g_{0,s}$ such that:

- $g_{0,s} \in I^{\mathfrak{A}}_{i_0, j_{0,s}}$ for some $j_{0,s}$; and
- $\mathcal{A}_{j_{0,s}}$ contains exactly one $y < g_{0,s}(x)$ which is not in range$(g_{0,s})$; and
- $g_{0,s}$ has not been considered at any previous stage.

Such a $g_{0,s}$ must exist, since there are infinitely many elements of α lying below ω_1^{CK} satisfying distinct computable infinitary formulas in α. Once we find the least one, we fix it and search for amalgamations: first $j_{1,s} \in \omega$ and $g_{1,s}, h_{0,s} \in I^{\mathfrak{A}}$, then $j_{2,s} \in \omega$ and $g_{2,s}, h_{1,s} \in I^{\mathfrak{A}}$, etc., as shown here:



We define $i_{s+1} = j_{s,s}$ and $f_s = g_{s,s}$, thus adding $\mathcal{A}_{j_{s,s}}$ to the sequence $\mathcal{A}_{i_0}, \mathcal{A}_{i_1}, \ldots$ previously built. If the image of $\mathcal{B}_s$ under f_s already contains the element $y_{s+1} = (h_{s-1,s} \circ h_{s-2,s} \circ \cdots \circ h_{0,s})(g_{0,s}((y))$, then we set $\mathcal{S}_{s+1} = \mathcal{S}_s$ and $p_{s+1} = f_s \circ p_s$. If not, then we extend $\mathcal{S}_s$ to a larger order $\mathcal{S}_{s+1}$ by adding one new element z_{s+1} to $\mathcal{S}_s$, with $p_{s+1}(z_{s+1}) = y_{s+1}$ and $p_{s+1} = f_s \circ p_s$ on the rest of $\mathcal{S}_{s+1}$. The order on $\mathcal{S}_{s+1}$ is defined so that p_{s+1} remains an order isomorphism from $\mathcal{S}_{s+1}$ into the suborder $\mathcal{B}_{s+1} = \mathcal{B}_s \cup \{y_{s+1}\}$ of $\mathcal{A}_{i_{s+1}}$; clearly this is compatible with the order on $\mathcal{S}_s$, and it justifies the inductive hypothesis at the next stage.

This is the entire construction, building the computable linear order $\mathcal{S} = \cup_s \mathcal{S}_s$. We now present the (non-effective) inductive argument that $\mathcal{S} \cong \omega_1^{\mathrm{CK}}$, which proceeds through the same stages just described. At stage 0, of course we have an $(\omega_1^{\mathrm{CK}} + 1)$-extensional map $\varphi_0 : \mathcal{A}_{i_0} \to \alpha$, and we define $\psi_0 = \varphi_0 \circ p_0$, embedding $\mathcal{S}_0$ isomorphically into ω_1^{CK} within α (since $\varphi_0(x_0) = \omega_1^{\mathrm{CK}}$, and p_0 maps all elements of $\mathcal{S}_0$ to elements below x_0 in $\mathcal{A}_{i_0}$).

Now at stage $s + 1$ we chose an embedding $g_{0,s} : \mathcal{A}_{i_0} \hookrightarrow \mathcal{A}_{j_{0,s}}$ from $I^{\mathfrak{A}}$. Since φ_0 is $(\omega_1^{\mathrm{CK}} + 1)$-extensional, this $g_{0,s}$ lifts to an inclusion range$(\varphi_0) \subseteq \mathcal{C}$, for some finite $\mathcal{C} \subseteq \alpha$, via φ_0 and some ω_1^{CK}-extensional map $\varphi_{1,s}$ sending $\mathcal{A}_{j_{0,s}}$ onto $\mathcal{C}$. By Lemma 4, then, the unique $y < g_{0,s}(x)$ in $(\mathcal{A}_{j_{0,s}} - \text{range}(g_{0,s}))$ must

satisfy (in the theory of the cover $\mathfrak{A}$) some computable infinitary formula which uniquely identifies one computable ordinal. Fix some θ_s with $\theta_{s-1} < \theta_s < \omega_1^{\mathrm{CK}}$ large enough that this formula is Σ_{θ_s}.

Now we proceed along the diagram above. Each object $\mathcal{A}_{i_t}$ is the domain of some (θ_s)-extensional map into α, as is each object $\mathcal{A}_{j_{t,s}}$, such that these maps are all compatible with φ_0. To see this, take $(\theta_s + s)$-extensional maps with domains $\mathcal{A}_{i_1}$ and $\mathcal{A}_{j_{0,s}}$, using $(\omega_1^{\mathrm{CK}} + 1)$-extensionality of φ_0; then $(\theta_s + s - 1)$-extensional maps with domains $\mathcal{A}_{i_2}$ and $\mathcal{A}_{j_{1,s}}$, etc. Notice that for an $\mathcal{A}_{j_{t,s}}$ with $t > 0$, we may have several different such maps, depending on the path one takes through the diagram. However, every element y from any $\mathcal{B}_t$ within $\mathcal{A}_{i_t}$ satisfies a Σ_{θ_s} formula from Lemma 4, and therefore has a unique possible image in α under these θ_s-extensional maps: there is only one element in α satisfying that formula. Moreover, for such a $y \in \mathcal{B}_t$, the same holds of the element $g_{t,s}(y)$ of $\mathcal{A}_{j_{t,s}}$ under all θ_s-extensional maps from $\mathcal{A}_{j_{t,s}}$ into α. So all of these maps agree on all elements of $\mathcal{B}_s$ and on their images in $\mathcal{A}_{j_{s,s}}$. Indeed, this remains true even when we allow s to vary: θ_s will be larger for larger s, and $\mathcal{A}_{j_{t,s}}$ may be distinct from $\mathcal{A}_{j_{t,s+1}}$, but each element of any $\mathcal{B}_t$ within each $\mathcal{A}_{i_t}$ in the diagram at stage $s+1$ is mapped to the same element of α by all these maps at this and all subsequent stages. So, to define $\psi_s(z)$ for $z \in \mathcal{S}_s$, we just map z into $\mathcal{B}_s$ using p_s, and then send $p_s(z)$ to its image in α under any one of these θ_s-extensional maps. This defines ψ_s unambiguously on $\mathcal{S}_s$, and each ψ_s is compatible with ψ_{s+1}, because p_{s+1} restricts to p_s and because we noted above that the image of an element of $\mathcal{B}_s$ below the image of x has only one possible image in α under these (sufficiently extensional) maps. So it is clear that this $\psi = \cup_s \psi_s$ is an embedding of $\mathcal{S}$ into ω_1^{CK} within α. Finally, for each element $\gamma \notin \mathrm{range}(\varphi_0)$ of the linear order ω_1^{CK}, there is some j_0 and some map $g_0 : \mathcal{A}_{i_0} \to \mathcal{A}_{j_0}$ which lifts to the inclusion $\mathrm{range}(\varphi_0) \subseteq \mathrm{range}(\varphi_0) \cup \{\gamma\}$, and at some stage s this j_0 and this g_0 will be chosen as $j_{0,s}$ and $g_{0,s}$. At that stage, γ will become the ψ_s-image of some element of $\mathcal{S}_s$, and so the embedding ψ actually maps $\mathcal{S}$ onto ω_1^{CK}. Thus $\mathcal{S}$ is a computable presentation of ω_1^{CK}, which is impossible.

It remains to decide whether an $\alpha \geq \omega_1^{\mathrm{CK}}$ could have an ω_1^{CK}-extensional computable cover. In the initial cases, we can answer this.

Theorem 3. *If $\omega_1^{\mathrm{CK}} \leq \alpha < \omega_1^{\mathrm{CK}} \cdot \omega$, then α has no ω_1^{CK}-extensional computable cover with AP.*

Proof. We sketch the proof, which mixes the techniques used for Proposition 2 and Theorem 2. Now one fixes some i_0 for which $\mathcal{A}_{i_0}$ is the domain of an ω_1^{CK}-extensional map φ_0 onto the finite set $\alpha \cap \{\omega_1^{\mathrm{CK}} \cdot (n+1) : n \in \omega\}$. Consider any j and any $g \in I^{\mathfrak{A}}_{i_0 j}$. Now for every $\theta < \omega_1^{\mathrm{CK}}$, every $g(x)$ maps to $\varphi_0(x)$ by some θ-extensional map, and so each $g(x)$ satisfies a Σ_θ-formula in $\mathfrak{A}$ stating that in the Cantor normal form of $g(x)$, every ω^ζ with $\zeta < \theta$ has coefficient 0. Since this holds for all $\theta < \omega_1^{\mathrm{CK}}$, each ω_1^{CK}-extensional map ψ with domain $\mathcal{A}_j$ must send each of these $g(x)$ to a nonzero multiple of ω_1^{CK} in α. If follows that each element $y \in \mathcal{A}_j$ with $y < \min(\mathrm{range}(g))$ has $\psi(y) < \omega_1^{\mathrm{CK}}$ in α. By Lemma 4, each such y satisfies a Σ_θ formula in $\mathfrak{A}$ which, in all ordinals, can only be satisfied by $\psi(y)$.

This allows us to run the same construction that we did in Theorem 2, going systematically through maps $g \in I^{\mathfrak{A}}$ from $\mathcal{A}_{i_0}$ into any $\mathcal{A}_j$ in such a way that $\min(\text{range}(g)) \neq \min(\mathcal{A}_j)$ and amalgamating those maps into the construction to get a computable presentation of ω_1^{CK}, which is impossible.

3 Extensionality for Ordinals Beyond ω_1^{CK}

Theorem 4. *For each computable ordinal θ, every ordinal α has a θ-extensional computable cover.*

The full proof is too long to present in this context, but we can provide a number of details. We state the key lemmas (in terms of the fixed computable ordinal θ), present the proof of Theorem 4 assuming these lemmas, and end with a sketch of the proofs of the lemmas.

Lemma 5. *If linear orders $\mathcal{S}_0$ and $\mathcal{S}_1$ each have θ-extensional computable covers, then so does their sum $\mathcal{S}_0 + \mathcal{S}_1$.*

Lemma 6. *Each ordinal multiple of ω^θ of the form $\omega^\theta \cdot \beta$ (with $\beta \geq \omega$) has a θ-extensional computable cover.*

To prove Theorem 4, notice that every computable ordinal has a θ-extensional (even ∞-extensional) computable cover. Therefore, fix a noncomputable ordinal α and write $\alpha = \omega^\theta \cdot \beta + \rho$ with $\rho < \omega^\theta$. Since $\rho < \omega^\theta$, ρ is computable and hence has a θ-extensional cover. Since $\beta > \omega$ (because α is not computable), $\omega^\theta \cdot \beta$ has a θ-extensional cover by Lemma 6. Therefore, by Lemma 5, α has a θ-extensional computable cover. So Lemmas 5 and 6 imply Theorem 4.

To prove Lemma 5, fix θ-extensional computable covers $\mathfrak{A}_0$ and $\mathfrak{A}_1$ of $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively. The objects in the θ-extensional computable cover of $\mathcal{S}_0 + \mathcal{S}_1$ have the form $\mathcal{A}_i^0 + \mathcal{A}_j^1$ where $\mathcal{A}_i^0 \in \mathfrak{A}_0$ and $\mathcal{A}_j^1 \in \mathfrak{A}_1$, with the caveat that one of $\mathcal{A}_i^0$ or $\mathcal{A}_j^1$ is allowed to be empty. The injective maps from $\mathcal{A}_i^0 + \mathcal{A}_j^1$ to $\mathcal{A}_k^0 + \mathcal{A}_\ell^1$ are defined in the obvious way, and one checks that this cover is θ-extensional.

The proof of Lemma 6 is notationally cumbersome, but the fundamental idea is that θ-extensionality cannot distinguish between gaps in a linear order of length $\omega^\theta \cdot \gamma$ for varying nonzero values of γ. Each $\mathcal{A}_i$ in our θ-extensional cover $\mathfrak{A}$ of $\omega^\theta \cdot \beta$, is a finite linear order of the form $1 < 2 < \cdots < n$, for some n, together with an n-tuple $\langle \xi_0, \xi_1, \ldots, \xi_{n-1} \rangle$, called its *label*, in which each $\xi_i \in \omega^\theta \cdot 2$. If $\xi_i < \omega^\theta$, then ξ indicates that the gap between i and $i+1$ (or the gap to the left of 1 if $i = 0$) in $\mathcal{A}_i$ should have length ξ_i. If $\xi_i = \omega^\theta + \rho$, then it indicates that this gap has length $\omega^\theta \cdot \gamma + \rho$ for some $\gamma \geq 1$. (Note that the labels are not formally part of $\mathcal{A}_i$. They are merely a denotation to help us keep track of which injective maps to include in $I_{ij}^{\mathfrak{A}}$.)

For each n and each label $\langle \xi_0, \ldots, \xi_{n-1} \rangle$, we include an object $\mathcal{A}_i$ of length n with this label in $\mathfrak{A}$. Let $\mathcal{A}_i$ have domain $\{1, \ldots, n\}$ with label $\langle \xi_0, \ldots, \xi_{n-1} \rangle$ and $\mathcal{A}_j$ have domain $\{1, \ldots, m\}$ with label $\langle \eta_0, \ldots, \eta_{m-1} \rangle$. We include an order preserving map $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$ in $I_{ij}^{\mathfrak{A}}$ if and only if the labels

match in the sense that $\xi_0 = \eta_0 + \cdots + \eta_{f(1)-1}$ and for all $0 < k < n$, $\xi_k = \eta_{f(k)} + \eta_{f(k)+1} + \cdots + \eta_{f(k+1)-1}$. It is straightforward to check that this process defines a computable cover of $\omega^\theta \cdot \beta$. To check that this cover is θ-extensional takes longer and will not be presented here. The key fact is the following lemma, which can be established by induction on ζ. Essentially it says that ζ-extensional maps cannot distinguish one nonzero multiple of ω^ζ from another, so that an object with (say) two elements $1 < 2$ and a gap labeled $\omega^\zeta \cdot \mu + \rho$ between them (where $\rho < \omega^\zeta$) can be used to cover two elements of the ordinal $\omega^\theta \cdot \beta$ by a map ψ, provided that $\psi(1) + \omega^\zeta \cdot \overline{\mu} + \rho = \psi(2)$ for some $\overline{\mu}$ which equals 0 iff $\mu = 0$.

Lemma 7. *Fix $\zeta \leq \theta$ and let $\psi : \mathcal{A}_i \to \omega^\theta \cdot \beta$ be an increasing map. Assume $\mathcal{A}_i$ has domain $\{1, \ldots, n\}$ and label $\langle \xi_0, \ldots, \xi_{n-1} \rangle$. Write each $\xi_k = \omega^\zeta \cdot \mu_k + \rho_k$ with $\rho_k < \omega^\zeta$. If there are ordinals $\overline{\mu}_k$, with $\overline{\mu}_k = 0$ if and only if $\mu_k = 0$, such that*

$$\psi(k) = (\omega^\zeta \cdot \overline{\mu}_0 + \rho_0) + (\omega^\zeta \cdot \overline{\mu}_1 + \rho_1) + \cdots + (\omega^\zeta \cdot \overline{\mu}_{k-1} + \rho_{k-1}),$$

then ψ is a ζ-extensional map from $\mathcal{A}_i$ into $\omega^\theta \cdot \beta$.

This completes the proof sketch for Theorem 4. The full proof is quite technical and is omitted here. We believe that Theorem 4 ought to be a corollary to a more general result connecting effectiveness properties of the θ back-and-forth types of a structure with its being θ-extensionally locally computable. We invite the interested reader to find the deeper connection that has thus far eluded us.

With Theorem 4, the general question of θ-extensionality of ordinals α is now settled in almost all cases. When $\alpha < \omega_1^{\mathrm{CK}}$ or $\theta < \omega_1^{\mathrm{CK}}$, the answer is positive, by Proposition 1 and Theorem 4. When $\omega_1^{\mathrm{CK}} \leq \alpha < \omega_1^{\mathrm{CK}} \cdot \omega$, the answer is negative for every $\theta \geq \omega_1^{\mathrm{CK}}$ by Proposition 2 and Theorem 3. When $\alpha \geq \omega_1^{\mathrm{CK}} \cdot \omega$, the answer is negative for every $\theta > \omega_1^{\mathrm{CK}}$ by Theorem 2. The only case remaining open is that in which $\theta = \omega_1^{\mathrm{CK}}$ and $\alpha \geq \omega_1^{\mathrm{CK}} \cdot \omega$.

References

1. Ash, C.J., Knight, J.F.: Computable Structures and the Hyperarithmetical Hierarchy. Elsevier, Amsterdam (2000)
2. Harizanov, V.S.: Pure computable model theory. In: Handbook of Recursive Mathematics, vol. 1, pp. 3–114. Elsevier, Amsterdam (1998)
3. Miller, R.G.: Locally computable structures. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 575–584. Springer, Heidelberg (2007)
4. Miller, R.G.: Local computability and uncountable structures. Greenberg, N., Hamkins, J.D., Hirschfeldt, D.R., Miller, R.G. (eds.) To appear in the ASL Lecture Notes in Logic volume Effective Mathematics of the Uncountable
5. Miller, R., Mulcahey, D.: Perfect local computability and computable simulations. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 447–456. Springer, Heidelberg (2008)
6. Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, New York (1987)

A Note on the Sequential Version of Π^1_2 Statements

Makoto Fujiwara[1,⋆] and Keita Yokoyama[2,⋆⋆]

[1] Mathematical Institute, Tohoku University, 6-3, Aramaki Aoba, Aoba-ku, Sendai, Miyagi, Japan
sb0m29@math.tohoku.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan
y-keita@jaist.ac.jp

Abstract. In connection with uniform computability and intuitionistic provability, the strength of the sequential version of Π^1_2 theorems has been investigated in reverse mathematics. In some examples, we illustrate that it occasionally depends on the way of formalizing the Π^1_2 statement, so the investigation of sequential strength demands careful attention to the formalization. Moreover our results suggest the optimality of Dorais's uniformization theorems.

1 Introduction

Definition 1 (Sequential version). *The sequential version of a* Π^1_2 *statement having the form:*

$$(\spadesuit) \qquad \forall X\,(\varphi(X) \to \exists Y \psi(X, Y))$$

is the statement

$$\forall \langle X_n \rangle_{n\in\mathbb{N}}\,(\forall n\ \varphi(X_n) \to \exists \langle Y_n \rangle_{n\in\mathbb{N}} \forall n\ \psi(X_n, Y_n))\,,$$

where X *is possibly a tuple of set (or function) variables. Throughout this paper, we denote the sequential version of a statement* T *having a form* $(\spadesuit)$ *as* Seq(T).

Many mathematical statements have the form $(\spadesuit)$, and their sequential forms have been investigated in order to reveal the lack of uniformity of their proof in classical subsystems of second-order arithmetic (e.g., [2,3,6]). For instance, the intermediate value theorem is provable in RCA_0, but the sequential version of it is equivalent to WKL (weak König's lemma), and so, not provable in RCA_0. This is of course caused by the necessity of non-uniformity in the proof in RCA_0.

⋆ Makoto Fujiwara is supported by a grant from Shigakukai.
⋆⋆ Keita Yokoyama is supported by a Japan Society for the Promotion of Science post-doctoral fellowship for young scientists, and by a grant from the John Templeton Foundation.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 171–180, 2013.

However, the strength of the sequential version may increase for another reason. In this paper, we concentrate our attention on Π^1_2 statements having the following syntactical form:

$$(\natural') \qquad \forall X \left(\exists Z \theta(X, Z) \to \exists Y \psi(X, Y)\right), \tag{$\natural'$}$$

where $\theta(X, Z)$ is arithmetical. Despite the fact that $(\natural')$ is, even in intuitionistic predicate logic, equivalent to the following statement:

$$(\natural) \qquad \forall X, Z \left(\theta(X, Z) \to \exists Y \psi(X, Y)\right), \tag{$\natural$}$$

the sequential version of $(\natural')$ is occasionally stronger than that of $(\natural)$ even if $\theta(X, Z)$ has a very weak complexity such as Π^0_1. This is caused by the difficulty of obtaining the sequence of Z in $(\natural')$. Using the finite marriage theorem and the bounded König's lemma, we illustrate this phenomenon. On the other hand, the sequential version of a statement of the form $(\natural')$ is not always stronger than that of $(\natural)$ as we see in the case of the weak weak König's lemma. The important point is that the sequential form of $(\natural')$ captures the difficulty of obtaining a solution Y from X alone while that of $(\natural)$ captures the difficulty of obtaining a solution Y using both X and Z. That is to say, whenever we consider the sequential version of a Π^1_2 statement, we must pay attention to the formalization and what information can be used to obtain a solution.

In addition, it has been recently established in [7] and [1] that the intuitionistic provability of Π^1_2 statements of some syntactical form guarantees its classical sequential provability. Such kind of results are called "uniformization theorems". Our results can be used to show that Dorais's uniformization theorems in [1] are the best possible for the syntactical classes involved.

Throughout this paper, we use the standard notation and terminology in reverse mathematics (cf. [9]). In addition, $\mathbf{x} \subset_{\mathrm{fin}} X$ denotes that $\mathbf{x}$ is a finite subset of X, and $\mathbb{Q}^+$ denotes the set of positive rational numbers. We recall that $\mathsf{WKL}_0 = \mathsf{RCA}_0 + \mathrm{WKL}$ and $\mathsf{ACA}_0 = \mathsf{RCA}_0 + \mathrm{ACA}$ (arithmetical comprehension).

2 The Finite Marriage Theorem

The so-called marriage theorem for finite graphs states that *a finite binary graph* $(B, G; R)$ *satisfying the Hall condition:*

$$\forall \mathbf{x} \subset_{\mathrm{fin}} B \exists \mathbf{y} \subset_{\mathrm{fin}} G \left(|\mathbf{x}| \leq |\mathbf{y}| \wedge \forall g \in \mathbf{y} \exists b \in \mathbf{x} \left((b, g) \in R\right)\right),$$

has an injection $M \subseteq R$ *from* B *to* G. It is well-known that there is a uniform algorithm to construct an injection from a given finite bipartite graph satisfying the Hall condition, which suggests that the sequential version of the finite marriage theorem is provable in RCA_0. However, it depends on the formalization. We provide the following two formalizations of the finite marriage theorem.

FMT :

$$\forall B, G, R, k \left(\left(\begin{array}{c} (B, G; R) \text{ is a bipartite graph} \\ \text{which satisfies the Hall condition} \\ \text{and } k \text{ bounds } B \cup G \end{array} \right) \to \exists M \left(\begin{array}{c} M \subseteq R \\ \text{is injective} \end{array} \right) \right),$$

F′MT :

$$\forall B, G, R \left(\exists k \left(\begin{array}{c} (B, G; R) \text{ is a bipartite graph} \\ \text{which satisfies the Hall condition} \\ \text{and } k \text{ bounds } B \cup G \end{array} \right) \to \exists M \left(\begin{array}{c} M \subseteq R \\ \text{is injective} \end{array} \right) \right),$$

where "k bounds $B \cup G$" denotes that for all $v \in B \cup G$, $v < k$. Note that the premise of $(\ldots \to \ldots)$ in FMT is purely universal. Throughout this paper, we use a little odd notation (e.g., F′MT) to indicate which assumption of uniformity is dropped by sequentializing.

Proposition 2

1. $\mathsf{RCA}_0 \vdash \mathrm{Seq}(\mathrm{FMT})$.
2. $\mathsf{RCA}_0 \vdash \mathrm{Seq}(\mathrm{F'MT}) \leftrightarrow \mathrm{ACA}$.

Proof. (1) A slight recasting of the proof of the finite marriage theorem in RCA_0 ([4, Theorem 2.1]).

(2) $\mathsf{ACA}_0 \vdash \mathrm{Seq}(\mathrm{F'MT})$ follows from the fact that the infinite marriage theorem is provable in ACA_0 ([5, Theorem 2.2]). For the reverse direction, it suffices to find the range of an injection $f : \mathbb{N} \to \mathbb{N}$ ([9, Lemma III.1.3]). The basic idea is to construct, simultaneously in RCA_0, infinite numbers of finite bipartite graphs such that the solution of the i-th graph indicates whether i is in $Rng(f)$ or not. By Σ^0_0 comprehension, take $\langle B_n \rangle_{n \in \mathbb{N}}$ and $\langle G_n \rangle_{n \in \mathbb{N}}$ as

$$b \in B_n \Leftrightarrow b = 0 \vee f\left(\frac{b-2}{2}\right) = n,$$
$$g \in G_n \Leftrightarrow g = 1 \vee f\left(\frac{g-3}{2}\right) = n,$$

which means that in addition to the underlying sequence $\{0, 1\}_{n \in \mathbb{N}}$ of vertices, the odd numbers are divided into $\{B_n\}_{n \in \mathbb{N}}$ and the even numbers are divided into $\{G_n\}_{n \in \mathbb{N}}$ according to f, and take $\langle R_n \rangle_{n \in \mathbb{N}}$ as

$$(b, g) \in R_n \Leftrightarrow (b, g) = (0, 1) \vee \left(b = 0 \wedge f\left(\frac{g-3}{2}\right) = n\right) \vee \left(g = 1 \wedge f\left(\frac{b-2}{2}\right) = n\right).$$

Then it is easy to see that $(B_n, G_n; R_n)$ satisfies the Hall condition for each $n \in \mathbb{N}$. Moreover if n is in the range of f via j, $B_n \cup G_n$ is bounded by $2j + 4$, and otherwise, $B_n \cup G_n$ is bounded by 2. Thus, by Seq(F′MT), there exists a sequence $\langle M_n \rangle_{n \in \mathbb{N}}$ of injections. Put $S := \{n : M_n(0) \neq 1\}$, then S is the range of f by the above construction. □

The previous proposition indicates that ACA is not needed to construct an injection from a finite bipartite graph satisfying the Hall condition, and only used to take a sequence of bounds. In fact, the next proposition follows from the previous proposition immediately. (One can even prove it directly.)

Proposition 3. *The following assertion* SeqB *is equivalent to* ACA *over* RCA_0.

(SeqB) *For any sequence of sets* $\langle X_n \rangle_{n\in\mathbb{N}}$*, if* X_n *is finite for all* n*, then there exists a function* $g : \mathbb{N} \to \mathbb{N}$ *such that* $g(n)$ *bounds* X_n.

Proof. $\mathsf{ACA}_0 \vdash$ SeqB is straightforward. For the reverse direction, it suffices to show Seq(F′MT) from SeqB over RCA_0. Let $\langle (B_n, G_n; R_n) \rangle_{n\in\mathbb{N}}$ be a sequence of finite bipartite graphs satisfying the Hall condition. Using SeqB, we have a function $g : \mathbb{N} \to \mathbb{N}$ such that $g(n)$ bounds $B_n \cup G_n$ for all $n \in \mathbb{N}$. Then the existence of a sequence of injections follows from Seq(FMT). □

3 The Bounded König's Lemma

It is known that the bounded König's lemma, which states that *an infinite tree having a bounding function has an infinite path*, is equivalent to WKL [9, Lemma IV.1.4]. As in the previous section, we provide the two formalizations of it.

$$\mathrm{BKL} : \forall T, g \left(\begin{pmatrix} T \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is an infinite tree} \\ \text{and } g : \mathbb{N} \to \mathbb{N} \text{ bounds } T \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right),$$

$$\mathrm{B'KL} : \forall T \left(\exists g \begin{pmatrix} T \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is an infinite tree} \\ \text{and } g : \mathbb{N} \to \mathbb{N} \text{ bounds } T \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right),$$

where "g bounds T" denotes that for all $\sigma \in T$ and $i < \mathrm{lh}(\sigma)$, $\sigma(i) < g(i)$. In addition, we now treat a weaker version of the bounded König's lemma in which a tree in question is bounded by a constant.

$$\mathrm{B_cKL} : \forall T, k \left(\begin{pmatrix} T \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is an infinite tree} \\ \text{and } k \text{ bounds } T \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right),$$

$$\mathrm{B'_cKL} : \forall T \left(\exists k \begin{pmatrix} T \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is an infinite tree} \\ \text{and } k \text{ bounds } T \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right),$$

where "k bounds T" denotes that for all $\sigma \in T$ and $i < \mathrm{lh}(\sigma)$, $\sigma(i) < k$. Note that the premise of $(\ldots \to \ldots)$ in $\mathrm{B_cKL}$ is purely universal.

Proposition 4

1. $\mathsf{RCA}_0 \vdash \mathrm{Seq(BKL)} \leftrightarrow \mathrm{Seq(B_cKL)} \leftrightarrow \mathrm{WKL}$.
2. $\mathsf{RCA}_0 \vdash \mathrm{Seq(B'KL)} \leftrightarrow \mathrm{Seq(B'_cKL)} \leftrightarrow \mathrm{ACA}$.

Proof. We reason in RCA_0.

(1) WKL implies Seq(WKL) ([6, Lemma 5]), and Seq(WKL) implies Seq(BKL) by imitating the proof of BKL in WKL_0 ([9, Lemma IV.1.4]). The implication from Seq(BKL) to Seq($\mathrm{B_cKL}$) is obvious. That from Seq($\mathrm{B_cKL}$) to WKL follows immediately from the fact that binary trees are bounded by 2.

(2) It is straightforward that ACA implies Seq(B′KL) by imitating the proof of König's lemma in ACA_0 ([9, Lemma III.7.2]). Seq(B′KL) implies Seq($\mathrm{B'_cKL}$). The implication from Seq($\mathrm{B'_cKL}$) to ACA follows from Lemma 11 below. □

In the reverse mathematics of analysis, the bounded König's lemma corresponds to the Heine/Borel compactness of effectively totally bounded complete separable metric spaces. Thus, to consider the strength of a sequential version of a mathematical statement which is related to Heine/Borel compactness, it is important to check which version of bounded König's lemma is needed. Here, we shall consider the maximum principle of continuous functions as an example. The following statement is equivalent to WKL over RCA_0. (See [9, Section IV].)

(MP) For any f, if f is a continuous function from $[-1,1]$ to $\mathbb{R}$, then there exists $a \in [-1,1]$ such that

$$\max\{f(x) : x \in [-1,1]\} = f(a).$$

By an easy consideration, we can see that MP is equivalent to the following.

(MP$^+$) For any f, if f is a continuous function from $(-1,1)$ to $\mathbb{R}$ such that $f(0) > 0$ and $\lim_{x\to\pm 1} f(x) = 0$, then there exists $a \in (-1,1)$ such that

$$\max\{f(x) : x \in (-1,1)\} = f(a).$$

For the sequential version of MP, the following is well-known, actually, it is an easy consequence of $\mathsf{RCA}_0 \vdash \mathrm{WKL} \leftrightarrow \mathrm{Seq(WKL)}$ ([6, Lemma 5]).

Proposition 5. Seq(MP) *is equivalent to* WKL *over* RCA_0.

However, the sequential version of MP$^+$ is strictly stronger than that of MP. (In general, ACA is required to extend a continuous function $f : (-1,1) \to \mathbb{R}$ with $\lim_{x\to\pm 1} f(x) = 0$ into a continuous function from $[-1,1]$ to $\mathbb{R}$.)

Proposition 6. *The following are equivalent over* RCA_0.

1. ACA.
2. *The sequential version of the following statement: for any f, if f is a bounded support continuous function from $\mathbb{R}$ to $\mathbb{R}$, then there exists $a \in \mathbb{R}$ such that* $\max\{f(x) : x \in \mathbb{R}\} = f(a)$. *(Here, f is said to have bounded support if there exists $k \in \mathbb{N}$ such that the closure of $\{x \in \mathbb{R} : f(x) \neq 0\}$ is a subset of $[-k,k]$.)*
3. *The sequential version of the following statement: for any f, if f is a continuous function from $\mathbb{R}$ to $\mathbb{R}$ such that $f(0) > 0$ and $\lim_{x\to\pm\infty} f(x) = 0$, then there exists $a \in \mathbb{R}$ such that* $\max\{f(x) : x \in \mathbb{R}\} = f(a)$.
4. Seq(MP$^+$).

Proof. By modifying the proof of MP $\leftrightarrow$ WKL, we can easily see that 2 is equivalent to the sequential version of the following statement: if $T \subseteq \mathbb{N}^{\mathbb{N}}$ is an infinite tree such that $T \subseteq 2k \times 2^{<\mathbb{N}}$ for some k, then T has an infinite path. Note that this is a weaker version of Seq(B'_{c}KL), and still is equivalent to ACA as in the proof of Lemma 11 below. For a given continuous function f from $\mathbb{R}$ to $\mathbb{R}$ such that $f(0) > 0$ and $\lim_{|x|\to\infty} f(x) = 0$, define a continuous function g as $g(x) = \max\{0, f(x) - f(0)/2\}$. Then, g has bounded support and $\max\{g(x) : x \in \mathbb{R}\} + f(0)/2 = \max\{f(x) : x \in \mathbb{R}\}$, hence we have $2 \leftrightarrow 3$. By an easy rescaling, we have $3 \leftrightarrow 4$. Thus, they are all equivalent to ACA. □

4 The Weak Weak König's Lemma

The weak weak König's lemma, which states that *a binary tree with positive measure has an infinite path*, has an intermediate strength between RCA_0 and WKL_0 ([9, Remark X.1.8]). In this case, both of sequential versions are stronger than the instancewise version and actually equivalent to WKL.

$$\mathrm{WWKL} : \forall T, m \left(\begin{pmatrix} T \subseteq 2^{<\mathbb{N}} \text{ is a tree and} \\ m \in \mathbb{Q}^+ \text{ satisfies } (\mathrm{W}_2) \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right),$$

$$\mathrm{W'WKL} : \forall T \left(\exists m \begin{pmatrix} T \subseteq 2^{<\mathbb{N}} \text{ is a tree and} \\ m \in \mathbb{Q}^+ \text{ satisfies } (\mathrm{W}_2) \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right),$$

where (W_2) denotes

$$\lim_{n\to\infty} \frac{|\{\sigma \in T : \mathrm{lh}(\sigma) = n\}|}{2^n} \geq m.$$

Proposition 7

1. $\mathsf{RCA}_0 \vdash \mathrm{Seq(WWKL)} \leftrightarrow \mathrm{WKL}$. ([2, Theorem 4.1.(2)])
2. $\mathsf{RCA}_0 \vdash \mathrm{Seq(W'WKL)} \leftrightarrow \mathrm{WKL}$.

Proof (of 2). It is easy to show within RCA_0 that for binary tree T, if there exists $m \in \mathbb{Q}^+$ such that $\lim_{n\to\infty} \frac{|\{\sigma \in T : \ \mathrm{lh}(\sigma) = n\}|}{2^n} \geq m$, then T is infinite. Therefore $\mathsf{WKL}_0 \vdash \mathrm{Seq(W'WKL)}$ immediately follows from $\mathsf{WKL}_0 \vdash \mathrm{Seq(WKL)}$ ([6, Lemma 5]). For the reverse direction, Seq(W'WKL) obviously implies Seq(WWKL), which is equivalent to WKL over RCA_0 from (1). □

Remark 8. *Note that the previous proposition does not suggest that the sequential strength of a mathematical statement equivalent to* WWKL *is* WKL *in general. Here, we shall consider Riemann integrability for bounded functions as an example. The following statement is equivalent to* WWKL *over* RCA_0*. (See [8].)*

(Int) *For any f, if f is a continuous function from $[0,1]$ to $[0,1]$, then there exists $r \in \mathbb{R}$ such that*

$$\int_0^1 f(x)\,dx = r.$$

However, Seq(Int) *does not imply* WKL*. This is because* Seq(Int) *follows from the following sequential contrapositive of* W'WKL*:*

$$(\star) \quad \forall T \left(\forall n \begin{pmatrix} T_n \subseteq 2^{<\mathbb{N}} \textit{ is a tree} \\ \textit{which has no path} \end{pmatrix} \to \forall n \lim_{k\to\infty} \frac{|\{\sigma \in T_n : \mathrm{lh}(\sigma) = k\}|}{2^k} = 0 \right).$$

The contrapositive of W'WKL *does not have the form* (♠) *from Definition 1 any more and* $(\star)$ *is trivially equivalent to* WWKL*. Therefore* Seq(Int) *is actually*

equivalent to WWKL. *In fact, for many sequential versions of mathematical statements which are equivalent to* WWKL, *we do not need* Seq(WWKL) *or* Seq(W′WKL) *but* ($\star$).

Next, we shall investigate the effect of uniformity for positive measure more precisely. For this, we shall consider some more variants, namely, bounded König's lemmas with respect to measure.

- $\mathrm{WBKL} : \forall T, m, g \left(\begin{pmatrix} T \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is a tree,} \\ m \in \mathbb{Q}^+ \text{ satisfies } (\mathrm{W_g}), \\ g : \mathbb{N} \to \mathbb{N} \text{ bounds } T \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right)$,

  where $(\mathrm{W_g})$ denotes

$$\lim_{n\to\infty} \frac{|\{\sigma \in T : \mathrm{lh}(\sigma) = n\}|}{\prod_{i<n} g(i)} \geq m.$$

- $\mathrm{WB_cKL} : \forall T, m, k \left(\begin{pmatrix} T \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is a tree,} \\ m \in \mathbb{Q}^+ \text{ satisfies } (\mathrm{W_k}), \\ k \text{ bounds } T \end{pmatrix} \to \exists P \begin{pmatrix} P \text{ is an infinite} \\ \text{path of } T \end{pmatrix} \right)$,

  where $(\mathrm{W_k})$ denotes

$$\lim_{n\to\infty} \frac{|\{\sigma \in T : \mathrm{lh}(\sigma) = n\}|}{k^n} \geq m.$$

Proposition 9. WBKL *and* $\mathrm{WB_cKL}$ *are equivalent to* WWKL *over* $\mathsf{RCA_0}$.

Proof. We reason in $\mathsf{RCA_0}$. WBKL to $\mathrm{WB_cKL}$ to WWKL is trivial. We shall show WBKL from WWKL. Let $T \subseteq \mathbb{N}^{<\mathbb{N}}$ be a tree bounded by $g : \mathbb{N} \to \mathbb{N}$ such that for some $q \in \mathbb{Q}^+$,

$$\lim_{n\to\infty} \frac{|\{\sigma \in T : \mathrm{lh}(\sigma) = n\}|}{\prod_{i<n} g(i)} \geq q.$$

For $\sigma \in \mathbb{N}^{<\mathbb{N}}$, define $l_g(\sigma)$ and $r_g(\sigma)$ as follows:

$$l_g(\sigma) = \sum_{k<\mathrm{lh}(\sigma)} \frac{\sigma(k)}{\prod_{i\leq k} g(i)}, \qquad r_g(\sigma) = l_g(\sigma) + \frac{1}{\prod_{i<\mathrm{lh}(\sigma)} g(i)}.$$

Similarly, for $\sigma \in 2^{<\mathbb{N}}$, define $l_2(\sigma)$ and $r_2(\sigma)$ as follows:

$$l_2(\sigma) = \sum_{k<\mathrm{lh}(\sigma)} \sigma(k) 2^{-k-1}, \qquad r_2(\sigma) = l_2(\sigma) + 2^{-\mathrm{lh}(\sigma)}.$$

Note that $\bigcup_{\sigma\in T, \mathrm{lh}(\sigma)=m} [l_g(\sigma), r_g(\sigma)]$ are disjoint intervals in $[0, 1]$ whose lengths sum to the measure of level m of T and these intervals can be approximated arbitrarily well from within by intervals with dyadic rational endpoints. That is,

for $m \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that

$$\frac{\left|\left\{\sigma \in 2^{<\mathbb{N}} : \begin{array}{l}\mathrm{lh}(\sigma) = N \wedge \\ \exists \tau \in T(\mathrm{lh}(\tau) = m \wedge l_g(\tau) \leq l_2(\sigma) \wedge r_2(\sigma) \leq r_g(\tau))\end{array}\right\}\right|}{2^N}$$
$$> \frac{|\{\sigma \in T : \mathrm{lh}(\sigma) = m\}|}{\prod_{i<m} g(i)} - \frac{q}{2^{m+2}}.$$

We define $h(m)$ as the least such N.

Now we define $T^* \subseteq 2^{\mathbb{N}}$ as

$$\sigma \in T^* \Leftrightarrow$$
$$\forall m < \mathrm{lh}(\sigma) \left(\begin{array}{l} h(m) \leq \mathrm{lh}(\sigma) \rightarrow \exists \tau \in T(\mathrm{lh}(\tau) = m \wedge \\ l_g(\tau) \leq l_2(\sigma \upharpoonright h(m)) \wedge r_2(\sigma \upharpoonright h(m)) \leq r_g(\tau)) \end{array} \right).$$

Then, T^* is a tree such that for all $n \in \mathbb{N}$,

$$\frac{|\{\sigma \in T^* : \mathrm{lh}(\sigma) = n\}|}{2^n} > \frac{|\{\sigma \in T : \mathrm{lh}(\sigma) = n\}|}{\prod_{i<n} g(i)} - \sum_{m<n} \frac{q}{2^{m+2}} \geq \frac{q}{2}.$$

Thus, by WWKL, there exists a path P^* through T^*. For any $m \in \mathbb{N}$, there exists a unique $\tau_m \in T$ such that $\mathrm{lh}(\tau_m) = m$ and $l_g(\tau_m) \leq l_2(P \upharpoonright h(m)) \wedge r_2(P \upharpoonright h(m)) \leq r_g(\tau_m)$. Then, $P = \bigcup_{m \in \mathbb{N}} \tau_m$ is a path through T. □

Next we investigate the sequential strength of the statements in question. The following proposition means that the uniformity for positive-measure does not help to weaken the sequential strength of the bounded König's lemma.

Proposition 10

1. $\mathrm{Seq}(\mathrm{W'BKL})$, $\mathrm{Seq}(\mathrm{WBKL})$, $\mathrm{Seq}(\mathrm{W'B_cKL})$ *and* $\mathrm{Seq}(\mathrm{WB_cKL})$ *are equivalent to* WKL *over* $\mathsf{RCA_0}$.
2. $\mathrm{Seq}(\mathrm{W'B'KL})$, $\mathrm{Seq}(\mathrm{WB'KL})$, $\mathrm{Seq}(\mathrm{W'B'_cKL})$ *and* $\mathrm{Seq}(\mathrm{WB'_cKL})$ *are equivalent to* ACA *over* $\mathsf{RCA_0}$.

Here $\mathrm{WB'KL}$, $\mathrm{W'BKL}$, $\mathrm{W'B'KL}$, $\mathrm{WB'_cKL}$, $\mathrm{W'B_cKL}$, *and* $\mathrm{W'B'_cKL}$ *are defined in the same manner as before, that is,* W′ *(resp.* B′, $\mathrm{B'_c}$*) means that the universal quantifier* $\forall m$ *(resp.* $\forall g, \forall k$*) is moved into* $(\ldots \rightarrow \ldots)$ *as the existential quantifier* $\exists m$ *(resp.* $\exists g, \exists k$*).*

To show the previous proposition, we first show the following lemma.

Lemma 11. $\mathsf{RCA_0} \vdash \mathrm{Seq}(\mathrm{WB'_cKL}) \rightarrow \mathrm{ACA}$*, that is, the following statement implies* ACA *over* $\mathsf{RCA_0}$:

$$\forall \langle T_n \rangle_{n \in \mathbb{N}}, \langle m_n \rangle_{n \in \mathbb{N}} \left(\forall n \exists k \left(\begin{array}{l} T_n \subseteq \mathbb{N}^{<\mathbb{N}} \text{ is a tree,} \\ m_n \in \mathbb{Q}^+ \text{ satisfies } (\mathrm{W_k}) \text{ for } T_n, \\ k \text{ bounds } T_n \end{array} \right) \right.$$
$$\left. \longrightarrow \exists \langle P_n \rangle_{n \in \mathbb{N}} \forall n \, (P_n \text{ is an infinite path of } T_n) \right).$$

Proof. As in the proof of Proposition 2.(2), we shall find the range of an injection $f : \mathbb{N} \to \mathbb{N}$ ([9, Lemma III.1.3]). By Σ^0_0 comprehension, we take a sequence $\langle T_n \rangle_{n\in\mathbb{N}}$ of trees from the given injection f as

$$\sigma \in T_n \;\Leftrightarrow\; \begin{array}{l} \forall i < \mathrm{lh}(\sigma)\Big(\sigma(0) = 0 \wedge \sigma(i+1) \leq 1 \wedge f(i) \neq n\Big) \vee \\ \exists j < \sigma(0)\Big(\forall i < \mathrm{lh}(\sigma)\,(\sigma(i) \leq 2j+1) \wedge f(j) = n\Big). \end{array}$$

Then, each $T_n \subseteq \mathbb{N}^{<\mathbb{N}}$ clearly forms a tree. Put $m_n :\equiv 1/2$. We need to find a required bound k for each n. For given n, if there exists j such that $f(j) = n$, then put $k := 2j + 2$, and otherwise, put $k := 2$. In either case, we can check that k bounds T_n and $m_n (= 1/2)$ satisfies $(\mathrm{W_k})$ for T_n. Thus, by $\mathrm{Seq(WB'_cKL)}$, there exists a sequence $\langle P_n \rangle_{n\in\mathbb{N}}$ of paths. Put $S := \{n : P_n(0) \neq 0\}$. It is easy to see that $P_n(0) \neq 0 \leftrightarrow \exists j\,(f(j) = n)$, namely, S is the range of f. □

Proof (of Proposition 10). We reason in $\mathsf{RCA_0}$.

(1) Each of Seq(W′BKL), Seq(WBKL), Seq(W′B$_c$KL), Seq(WB$_c$KL) follows from Seq(BKL), then also from WKL by Proposition 4.(1). On the other hand, each of them implies Seq(WWKL) which is equivalent to WKL.

(2) Each of Seq(W′B′KL), Seq(WB′KL), Seq(W′B$'_c$KL), Seq(WB$'_c$KL) follows from Seq(B′KL), then also from ACA by Proposition 4.(2). On the other hand, each of Seq(W′B′KL), Seq(WB′KL), Seq(W′B$'_c$KL) implies Seq(WB$'_c$KL) and Seq(WB$'_c$KL) implies ACA by Lemma 11. □

5 The Best Possibility of Dorais's Uniformization Results

The first uniformization theorems are established in [7], which can be applied for Π^1_2 statements of the form (♠) (from Definition 1) with purely universal φ. Dorais has recently shown other uniformization theorems in second-order setting with function-based language, which can be applied for more Π^1_2 statements.

Proposition 12 (Dorais [1])

1. *For any* $\mathrm{T} : \forall f\,(\varphi(f) \to \exists g \psi(f,g))$ *such that* $\varphi(f)$ *is in* N_K *and* $\psi(f,g)$ *is in* Γ_K*, if* $\mathsf{EL} + \mathrm{GC} + \mathrm{CN} \vdash \mathrm{T}$*, then* $\mathsf{RCA} \vdash \mathrm{Seq(T)}$.
2. *For any* $\mathrm{T} : \forall f\,(\varphi(f) \to \exists g \psi(f,g))$ *such that* $\varphi(f)$ *is in* N_L *and* $\psi(f,g)$ *is in* Γ_L*, if* $\mathsf{EL} + \mathrm{WKL} + \mathrm{GC}_L + \mathrm{CN}_L \vdash \mathrm{T}$*, then* $\mathsf{RCA} + \mathrm{WKL} \vdash \mathrm{Seq(T)}$.

We refer the readers to see [1] for precise definitions of each of the symbols in the previous proposition. In fact, the restriction of ψ to Γ_K and Γ_L is not tight and the interest is only in the possibility of extending $\mathrm{N_K}$ and $\mathrm{N_L}$. All purely existential and purely universal formulas are included in N_K. In addition, all formulas of the form $\exists x \leq t \forall z A_{qf}$ are included in N_L. Here we show that $\mathrm{N_K}$ and $\mathrm{N_L}$ cannot be extended to the class including all formulas of the form $\exists x \forall z A_{qf}$ in Proposition 12.

Suppose that in Proposition 12.(1), N_K can be extended to such a class. Since each purely universal formula in set-based language is translated as a

purely universal formula in function-based language by identifying sets with their characteristic functions, the premise of $(\ldots \to \ldots)$ in function-based F′MT (intuitionistically equivalent to function-based FMT) has the form $\exists x \forall z A_{qf}$. Then Proposition 2.(2) derives that function-based F′MT is not provable in $\mathsf{EL} + \mathrm{GC} + \mathrm{CN}$. However, it is provable in EL_0 by transforming the proof of the finite marriage theorem in RCA_0 ([4, Theorem 2.1]).

Next we suppose in Proposition 12.(2), N_L can be extended to such a class. As in the previous paragraph, Proposition 4.(2) derives that function-based $\mathrm{B}'_{\mathrm{c}}\mathrm{KL}$ (intuitionistically equivalent to function-based $\mathrm{B}_{\mathrm{c}}\mathrm{KL}$) is not provable in $\mathsf{EL} + \mathrm{WKL} + \mathrm{GC}_L + \mathrm{CN}_L$. However, it is provable in $\mathsf{EL}_0 + \mathrm{WKL}$ by transforming the proof of the bounded König's lemma in WKL_0 ([9, Lemma IV.1.4]).

Acknowledgment. We are grateful to Ulrich Kohlenbach and anonymous referees for their helpful comments and suggestions on an earlier version of this paper.

References

1. Dorais, F.G.: Classical consequences of continuous choice principles from intuitionistic analysis. Notre Dame Journal to Formal Logic (to appear)
2. Dorais, F.G., Dzhafarov, D.D., Hirst, J.L., Mileti, J.R., Shafer, P.: On uniform relationships between combinatorial problems (to appear)
3. Dorais, F.G., Hirst, J.L., Shafer, P.: Reverse mathematics, trichotomy, and dichotomy. Journal of Logic and Analysis 4(13), 1–14 (2012)
4. Hirst, J.R.: Combinatorics in Subsystems of Second Order Arithmetic. Ph.D. thesis, Pennsylvania State University (1987)
5. Hirst, J.R.: Marriage theorems and reverse mathematics. Contemporary Mathematics 106, 181–196 (1990)
6. Hirst, J.R.: Representations of reals in reverse mathematics. Bull. Pol. Acad. Sci. Math. 55(4), 303–316 (2007)
7. Hirst, J.R., Mummert, C.: Reverse mathematics and uniformity in proofs without excluded middle. Notre Dome Journal to Formal Logic 52(2), 149–162 (2011)
8. Sanders, S., Yokoyama, K.: The Dirac delta function in two settings of Reverse Mathematics. Archive for Mathematical Logic 51, 99–121 (2012)
9. Simpson, S.G.: Association for Symbolic Logic. In: Subsystems of Second Order Arithmetic, 2nd edn. Cambridge University Press (2009)

On Conservative Learning of Recursively Enumerable Languages*

Ziyuan Gao[1], Sanjay Jain[2], and Frank Stephan[2]

[1] Department of Computer Science, University of Regina, Regina, Saskatchewan S4S 0A2, Canada
ziyuan84@yahoo.com
[2] Department of Computer Science, National University of Singapore, Singapore 117417, Republic of Singapore
{sanjay,fstephan}@comp.nus.edu.sg

Abstract. Conservative partial learning is a variant of partial learning whereby the learner, on a text for a target language L, outputs one index e with $L = W_e$ infinitely often and every further hypothesis d is output only finitely often and satisfies $L \not\subseteq W_d$. The present paper studies the learning strength of this notion, comparing it with other learnability criteria such as confident partial learning, explanatory learning, as well as behaviourally correct learning. It is further established that for classes comprising infinite sets, conservative partial learnability is in fact equivalent to explanatory learnability relative to the halting problem.

1 Introduction

The present paper follows Gold's framework of learning in the limit [10] which aimed to understand language acquisition from a mathematical point of view. Gold's model has been studied extensively [1,3,11,15,19] and can be described as follows: The learner processes step by step an infinite presentation of the language to be learnt — such a presentation is called a text — and outputs in parallel to reading the data a sequence of guesses. If the learner's sequence of guesses converges to a single correct description of the underlying concept, then the inference is correct; on the other hand, if it does not converge or converges to an incorrect description, then the inference is incorrect. This learning model is called "explanatory learning" from "positive data" or "text". A class of languages is learnable iff there is a recursive learner which learns every language in the class. Gold [10, Theorem I.8] showed that on one hand the class of all finite languages is learnable while on the other hand this class becomes unlearnable if only one infinite language is added to it.

Various alternative models of learning by inference from positive data have since been proposed [6,15]. In particular, behaviourally correct learning (see [2,3]) in which one requires that almost all the hypotheses of the learner are correct

* Work is supported in part by NUS grant R252-000-420-112 (all three authors) and C252-000-087-001 (S. Jain).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 181–190, 2013.

description of the input language (though these hypotheses may not be syntactically the same). Osherson, Stob and Weinstein [15] generalised the notion of explanatory learning to *partial learning*. Similar to the former setting, a recursive learner receives piecewise information about the elements of an unknown recursively enumerable language. At each stage, the learner is required to output a conjecture based on a pre-assigned hypothesis space — usually taken to be a fixed acceptable numbering of all recursively enumerable sets — and is judged to have successfully inferred a target language if it outputs exactly one correct index of the language infinitely often and outputs any other conjecture only finitely often, though there might be infinitely many false conjectures in total. This is in contrast to explanatory learning where the learner outputs one correct conjecture almost always and only finitely often outputs a false conjecture. As it stands, partial learning is even more general than behaviourally correct learning; in fact, a recursive learner can partially learn the class of all recursively enumerable sets [15, Exercise 7.5A]. Osherson, Stob and Weinstein [15] called an explanatory learner *confident* iff it converges on any text for any language to a hypothesis; this concept was brought over to partial learning [8,9] by defining that the learner must issue exactly one hypothesis infinitely often on any text for any language. This new hybrid of the two learnability notions turns out to be restrictive in the case of language learning [8]: the class of all cofinite sets is not confidently partially learnable. Other learning constraints considered in the inductive inference literature, but studied mainly in the context of explanatory learning, include *consistency* [5,17], *conservativeness* [1] and *prudence* [15].

The present paper continues the study of *conservativeness* and transfers it from explanatory learning to more general criteria like behaviourally correct learning and partial learning. Angluin [1] noted that inference from positive data is often stymied by the problem of overgeneralisation, by which is meant that the learner conjectures a proper superset of the target language, so that it never witnesses a counterexample to its hypothesis from a presentation of positive data. She suggested that learners should be *conservative*, that is, avoid overgeneralisation, and gave sufficient conditions for an indexed family of nonempty recursive languages to be inferable by a conservative learner [1, Theorem 5]. Further research into the properties of conservatively learnable classes [13,18] brought new insights for the case of uniformly r.e. classes and provided a criterion for the case of indexed families. This paper adapts a slight variation of the original conservative learning to fit the partial learning model: a recursive learner is said to *conservatively partially* learn a recursively enumerable language L from text if it partially learns L and, on each text for L, it outputs exactly one correct index for L and all other indices output are not for any proper superset of L.

For ease of notation, the languages in the present paper are taken as subsets of the set of natural numbers, $\mathbb{N}$. In the present work, it is shown that imposing conservativeness yields a close connection between the explanatory, vacillatory and behaviourally correct models of learning and partial learning. To exemplify this point, Theorem 3 states that the set of all conservatively partially learnable classes of r.e. languages strictly subsumes that of all conservatively behaviourally

correctly learnable classes of r.e. languages. On one hand, the class containing all finite sets and the set of natural numbers is not conservatively partially learnable; this shows that this new learning notion constitutes a restriction on ordinary partial learning. On the other hand, conservative partial learning may still be a fairly general concept, as the class of graphs of all recursive functions is learnable according to this criterion (Example 5). Theorem 4 frames a learning criterion that appears to be more general than conservative partial learnability; nevertheless, one can show that this apparent generalisation is equivalent in terms of learning strength to the original definition of conservative partial learning. The paper also revisits vacillatory learning, introduced by Case [4], when it is combined with conservativeness. The principal result obtained here is that conservative vacillatory learning is as powerful as conservative explanatory learning. This result stands in contrast to that for the case when conservativeness is not stipulated: Case [4] provided an example of a vacillatorily learnable class which is not explanatorily learnable.

As further evidence of the tie-in between conservative partial learning and explanatory learning, several results draw comparisons between conservative partial learning and explanatory learning relative to oracles; Theorem 13, for instance, asserts that for any class of infinite r.e. languages, the learning strengths of both notions coincide when the oracle used for the explanatory learner is Turing equivalent to the diagonal halting problem. In addition, this work proposes a characterisation of conservatively partially learnable classes of r.e. languages as an analogue of Angluin's tell-tale criterion [1, Theorem 1]. The original theorem, formulated in the intuitively appealing notion of a family of finite "tell-tale sets", gave necessary and sufficient conditions for an indexed family of recursive languages to be explanatorily learnable from positive data. De Jongh and Kanazawa [13] later generalised Angluin's tell-tale condition to characterise indexed families of r.e. languages. Similarly, Zeugmann, Lange and Kapur [18] proved a characterisation of conservative learnability for indexed families of recursive languages; this was subsequently extended by de Jongh and Kanazawa [13] to the case of indexed families of r.e. languages. In Theorem 16, a characterisation similar to that of Angluin [1] and Zeugmann, Lange and Kapur [18] is given for conservative partial learning.

2 Notation

The notation and terminology from recursion theory adopted in this paper follows in general the book of Rogers [16]. Background on inductive inference can be found in [11]. The symbol $\mathbb{N}$ denotes the set of natural numbers, $\{0, 1, 2, \ldots\}$. Let $\varphi_0, \varphi_1, \varphi_2, \ldots$ denote a fixed acceptable numbering [16] of all partial-recursive functions over $\mathbb{N}$. Given a set S, S^* denotes the set of all finite sequences in S. One defines the e-th r.e. set W_e as $dom(\varphi_e)$ and the e-th canonical finite set by choosing D_e such that $\sum_{x \in D_e} 2^x = e$. Furthermore, $\langle x, y \rangle$ denotes Cantor's pairing function, given by $\langle x, y \rangle = \frac{1}{2}(x + y)(x + y + 1) + y$. The notion $\eta(x)\downarrow$ means that $\eta(x)$ is defined, and $\eta(x)\uparrow$ means that $\eta(x)$ is undefined.

Turing reducibility is denoted by $\leq_{\mathrm{T}}$; $A \leq_{\mathrm{T}} B$ holds if A can be computed via a machine which uses B as an oracle; that is, it can give information on whether or not x belongs to B. $A \equiv_{\mathrm{T}} B$ means that $A \leq_{\mathrm{T}} B$ and $B \leq_{\mathrm{T}} A$ both hold, and $\{A : A \equiv_{\mathrm{T}} B\}$ is called the Turing degree of B. The class of all recursive functions is denoted by REC; the class of all $\{0,1\}$-valued recursive functions is denoted by $REC_{0,1}$. For any partial-recursive function g, we let $\mathrm{graph}(g) = \{\langle x, y\rangle : g(x) \downarrow = y\}$. The symbol $\mathbb{K}$ denotes the diagonal halting problem $\{e : \varphi_e(e)\downarrow\}$.

The jump of a set A is denoted by A' and denotes the relativised halting problem $A' = \{e : \varphi_e^A(e)\downarrow\}$. For any two sets A and B, $A \oplus B = \{2x : x \in A\} \cup \{2y+1 : y \in B\}$. Analogously, $A \oplus B \oplus C = \{3x : x \in A\} \cup \{3y+1 : y \in B\} \cup \{3z+2 : z \in C\}$.

For any $\sigma, \tau \in (\mathbb{N} \cup \{\#\})^*, \sigma \preceq \tau$ if and only if $\sigma = \tau$ or τ is an extension of σ, $\sigma \prec \tau$ if and only if σ is a proper prefix of τ, and $\sigma(n)$ denotes the element in the nth position of σ, starting from $n = 0$. The concatenation of two strings σ and τ shall be denoted by $\sigma \circ \tau$; for convenience, and whenever there is no possibility of confusion, this is occasionally denoted by $\sigma\tau$. Let $\sigma[n]$ denote the sequence $\sigma(0) \circ \sigma(1) \circ \ldots \circ \sigma(n-1)$. The length of σ is denoted by $|\sigma|$.

3 Learnability

Let $\mathcal{C}$ be a class of r.e. languages. Throughout this paper, the mode of data presentation is that of a *text*, by which is meant an infinite sequence of natural numbers and the # symbol. Formally, a *text* T_L for some L in $\mathcal{C}$ is a map $T_L : \mathbb{N} \to \mathbb{N} \cup \{\#\}$ such that $L = \mathrm{content}(T_L)$; here $T_L[n]$ denotes the sequence $T_L(0) \circ T_L(1) \circ \ldots \circ T_L(n-1)$ and the content of a text T, denoted $\mathrm{content}(T)$, is the set of numbers in the range of T. Analogously, for a finite sequence σ, $\mathrm{content}(\sigma)$ is the set of numbers in the range of σ. The main learning criteria studied in this paper are *partial learning*, *explanatory learning* and *behaviourally correct learning*. In the following definitions, M is a recursive function mapping $(\mathbb{N} \cup \{\#\})^*$ into $\mathbb{N} \cup \{?\}$; the ? symbol permits M to abstain from conjecturing at some stage.

Definition 1. (i) [15] M *partially* (Part) learns $\mathcal{C}$ if, for every L in $\mathcal{C}$ and each text T_L for L, there is exactly one index e such that $M(T_L[k]) = e$ for infinitely many k; furthermore, if M outputs e infinitely often on T_L, then $L = W_e$.

(ii) [10] M *explanatorily* (Ex) learns $\mathcal{C}$ if, for every L in $\mathcal{C}$ and each text T_L for L, there is a number n for which $L = W_{M(T_L[n])}$ and, for any $j \geq n$, $M(T_L[j]) = M(T_L[n])$.

(iii) [3] M *behaviourally correctly* (BC) learns $\mathcal{C}$ if, for every L in $\mathcal{C}$ and each text T_L for L, there is a number n for which $L = W_{M(T_L[j])}$ whenever $j \geq n$.

In some cases, learners are equipped with oracles and then $\mathrm{Ex}[A]$ denotes the criterion of explanatory learning with oracle A and so on. For a learning criterion

I, such as Ex and BC above, one also writes I to denote the collection of all I-learnable classes. Furthermore, in some cases, the learner does not use the default hypothesis space $W_0, W_1, W_2, \ldots$ but instead uses a uniformly r.e. hypothesis space $H_0, H_1, H_2, \ldots$ where in the corresponding definitions W_e is replaced by H_e. Here a hypothesis space $H_0, H_1, \ldots$ is called uniformly r.e. hypothesis space, iff $\{\langle i, x\rangle : x \in H_i\}$ is recursively enumerable. The next series of definitions impose additional constraints on the learner.

Definition 2. (i) [8] M is said to *confidently partially* (ConfPart) learn $\mathcal{C}$ if it partially learns $\mathcal{C}$ from text and, on every infinite sequence, outputs exactly one index infinitely often. Note that the requirement to output exactly one index infinitely often applies even for infinite sequences which are not texts for any language in $\mathcal{C}$.

(ii) M is said to *conservatively partially* (ConsvPart) learn $\mathcal{C}$ if it partially learns $\mathcal{C}$ from text and, on each text for every L in $\mathcal{C}$, outputs exactly one correct index for L, and all other indices output are not for any proper superset of L.

(iii) [1] M *conservatively explanatorily* (ConsvEx) learns $\mathcal{C}$ if it Ex learns $\mathcal{C}$ and, for any $\sigma, \tau \in (\mathbb{N} \cup \{\#\})^*$ with $? \neq M(\sigma) \neq M(\sigma \circ \tau) \neq ?$, there is a number x with $x \in \text{range}(\sigma \circ \tau) - W_\sigma$.

(iv) M is said to *conservatively behaviourally correctly* (ConsvBC) learn $\mathcal{C}$ if it BC learns $\mathcal{C}$ and is semantically conservative. Here, a learner M is semantically conservative iff, for any $\sigma, \tau \in (\mathbb{N} \cup \{\#\})^*$ with $W_{M(\sigma)} \neq W_{M(\sigma\circ\tau)}$, there is a number x with $x \in \text{range}(\sigma \circ \tau) - W_{M(\sigma)}$.

(v) [15] M is *prudent* if it learns the class $\{W_{M(\sigma)} : \sigma \in (\mathbb{N} \cup \{\#\})^*, M(\sigma) \neq ?\}$. In other words, M learns every set it conjectures.

(vi) [4] M *vacillatorily* (Vac) learns $\mathcal{C}$ if it BC learns $\mathcal{C}$ from text, and outputs on each text for every L in $\mathcal{C}$ only finitely many different indices.

(vii) M is said to *conservatively vacillatorily* (ConsvVac) learn $\mathcal{C}$ if it ConsvBC learns $\mathcal{C}$ from text, and outputs on each text for every L in $\mathcal{C}$ only finitely many different indices.

(viii) [5] M is *consistent* (Cons) if for all $\sigma \in (\mathbb{N} \cup \{\#\})^*$, $\text{content}(\sigma) \subseteq W_{M(\sigma)}$.

(ix) [14] M is said to *class-comprisingly* (ClsCom) learn a class $\mathcal{C}$ if it learns $\mathcal{C}$ from text with respect to a hypothesis space $\{H_0, H_1, H_2, \ldots\}$. As M learns $\mathcal{C}$, it is immediate that $\mathcal{C} \subseteq \{H_0, H_1, H_2, \ldots\}$.

(x) [14] M is said to *class-preservingly* (ClsPresv) learn $\mathcal{C}$ if it learns $\mathcal{C}$ from text with respect to a hypothesis space $\{H_0, H_1, H_2, \ldots\}$ satisfying $\mathcal{C} = \{H_0, H_1, H_2, \ldots\}$

Note that the requirements (v), (ix), (x) of the above definition can be applied to any learning criterion and are denoted by using Prud, ClsCom or ClsPresv as a prefix of the criterion name. Note that the definition for conservative partial learning given above is the most restrictive of some possible variants. It follows the spirit of conservative explanatory learning where the learner never gives up a correct index for another one. Theorem 4 below shows it to be equivalent to more general possible definitions.

A learner M is said to make a mind change on input $T[n+1]$ (or M makes a mind change from input $T[n]$ to input $T[n+1]$), if $? \neq M(T[n]) \neq M(T[n+1])$. Furthermore, M makes a semantic mind change on input $T[n+1]$, if $? \neq M(T[n]) \neq M(T[n+1])$ and $W_{M(T[n])} \neq W_{M(T[n+1])}$.

4 Conservative Partial Learning

Conservativeness is a learnability constraint that has been studied fairly extensively in the inductive inference literature, especially in the setting of indexed families [1,18]. In this paper, we consider the notion of *conservative partial* language learning; in brief, this is partial learning combined with the constraint that if a learner outputs e infinitely often on a text for some target language L, then none of its other conjectures on this text can contain L as a subset. Gold [10] observed that the class $\{\mathbb{N}\} \cup \{F \subseteq \mathbb{N} : F \text{ is finite}\}$ is not behaviourally correctly learnable, even when granting access to any oracle. Nonetheless, Gold's class is partially learnable from text: it is only necessary to output a canonical index for $\mathbb{N}$ as many times as a new datum (not previously seen) appears, and to conjecture for every initial segment σ of the input text, a canonical index for the finite set content(σ). By contrast, one can show that Gold's class $\{\mathbb{N}\} \cup \{F \subseteq \mathbb{N} : F \text{ is finite}\}$ cannot be conservatively partially learnt.

The main theorem of the present paper establishes a hierarchy of the major learnability notions treated herein when conservativeness constraint is imposed on the learners.

Theorem 3. ConsvEx = ConsvVac $\subset$ ConsvBC $\subset$ ConsvPart $\subset$ ConsvEx[$\mathbb{K}$].

Theorem 3 follows from Theorem 4, Example 5, Theorem 6, Example 7, Theorem 8 and Remark 12 given below. The first of these results shows that the criterion for conservative partial learning may in fact be slightly relaxed. It asserts that for any class $\mathcal{C}$ of r.e. languages to be ConsvPart learnable, it is sufficient that there is a recursive learner which, on every text for a target language L in $\mathcal{C}$, outputs at least one correct index and does not overgeneralise at any stage.

Theorem 4. *Let $\mathcal{C}$ be a class of r.e. languages such that there is a recursive learner which, on any text for some $L \in \mathcal{C}$, outputs at least one correct index for L and does not output an index for a proper superset of L. Then $\mathcal{C}$ is* ConsvPart *learnable.*

Proof. Suppose a recursive learner M satisfying the hypothesis of the theorem is given. First a learner N is defined, which, on any text for some $L \in \mathcal{C}$, outputs exactly one index for L and does not output an index for a proper superset of L. N on input text T (for some set L) is defined as follows. Let $e_n = M(T[n])$. If, content$(T[n]) = \varnothing$, then $N(T[n])$ is a canonical grammar for $\varnothing$. If content$(T) \neq \varnothing$, then let r be minimal such that content$(T[r]) \neq \varnothing$. Then, $N(T[r+i]) = p_i$, where p_{2m} is a canonical index for content$(T[r+m])$ and $p_{2m+1} = q_n^k$, where $m = \langle n, k \rangle$ and $W_{q_n^k}$ is defined as follows:

$W_{q_n^k} = \bigcup\{W_{e_n,s} : W_{e_n,s} \subset W_{e_n,s+1}$ and
$(\forall h < n)[\text{content}(T[r+k]) \not\subseteq W_{e_h,s}]$ and
$(\forall k' < k)(\exists h < n, t < s)[\text{content}(T[r+k']) \subseteq W_{e_h,t}$ and $W_{e_n,t} \subset W_{e_n,s}]\}$.

For $L = \varnothing$, clearly N satisfies the requirements. For $L \in \mathcal{C}$, as M never outputs an index for a proper superset of L on any text for T, N also never outputs an index for a proper superset of L on any text for T.

Suppose $L \in \mathcal{C}$ is a non-empty finite set. Then, N outputs a canonical index for L, as it outputs a canonical index for content$(T[r+k])$, for all k. Furthermore, no index e_n output by M while learning L is a proper superset of L; hence the $W_{e_n,s}$ used in the definitions of $W_{q_n^k}$ satisfy that either $L \not\subseteq W_{e_n}$ or $W_{e_n,s} \subset W_{e_n,s+1} \subseteq L$; hence $L \not\subseteq W_{q_n^k}$. Thus, N outputs one unique index for L and all other indices output are not for a proper superset of L.

Suppose $L \in \mathcal{C}$ is infinite, then there is a least n such that $W_{e_n} = L$. Let k be least such that, for all $m < n$, content$(T[r+k]) \not\subseteq W_{e_m}$. Then there are infinitely many s such that $W_{e_n,s} \subset W_{e_n,s+1}$, content$(T[r+k]) \subseteq W_{e_n,s}$ and for $k' < k$, there exists $m < n$ and $t < s$ such that content$(T[r+k']) \subseteq W_{e_m,t}$ and $W_{e_n,t} \subset W_{e_n,s}$. Thus, $W_{q_n^k} = W_{e_n} = L$. For all $k'' < k$, the second clause in the definition of $W_{q_n^{k''}}$ holds only for finitely many s, and thus, $W_{q_n^{k''}}$ is finite. For all $k'' > k$, the third clause in definition of $W_{q_n^{k''}}$ does not hold, and thus $W_{q_n^{k''}}$ is empty. For all $m < n$, for all k'', clearly, $W_{q_m^{k''}}$ cannot be an index for L nor an index for a superset of L. Furthermore, for all $m > n$, for all k'', as content$(T[r+k'']) \subseteq W_{e_n}$, $W_{q_m^{k''}}$ is finite. Thus, again N outputs one unique index for L and all other indices output are not for a proper superset of L.

Now, one can build a new learner N' that, on any text T, considers the conjectures $c_0, c_1, c_2, c_3, \ldots$ output by N on the given text T. For each conjecture c_i that N outputs, N' outputs c_i at least n times iff there is a stage $s > n$ such that $\forall x < n[x \in W_{c_i,s} \Leftrightarrow x \in \text{content}(T[s])]$ holds. Consequently, N' outputs on T the unique index of L issued by N infinitely often, and it outputs all other conjectures of N only finitely often. N' is therefore a ConsvPart learner of $\mathcal{C}$. □

Whilst conservative partial learnability may appear at first sight to be quite a stringent learning criterion, one can furnish a relatively natural example of a conservatively partially learnable class of r.e. languages which is neither behaviourally correctly nor confidently partially learnable.

Example 5. *Let* $\mathcal{G} = \{\text{graph}(f) : f$ *is recursive*$\}$. *Then* $\mathcal{G}$ *is* ConsvPart *learnable but neither confidently partially learnable nor behaviourally correctly learnable.*

As an immediate consequence of Theorem 4 and Example 5, one has that the notion of conservative partial learning strictly subsumes that of conservative behaviourally correct learning, as claimed in Theorem 3. The subsequent theorem places the last inclusion relation of Theorem 3 in a broader setting, characterising the oracles relative to which a conservatively partially learnable class is also conservatively explanatorily learnable.

Theorem 6. *Given an oracle A,* ConsvPart $\subseteq$ ConsvEx$[A]$ *if and only if* $\mathbb{K} \leq_{\mathrm{T}} A$.

To complement the preceding theorem, one can show that for every nonrecursive oracle A, ConsvEx$[A]$ learning is more powerful than ConsvPart learning.

Example 7. *If A is not recursive, then there is a class which is* ConsvEx$[A]$ *learnable but not conservatively partially learnable.*

Vacillatory learning, when imposed together with conservativeness, is a fairly stringent criterion; the next theorem asserts that it implies conservative explanatory learning in general.

Theorem 8. *If a class of r.e. languages is* ConsvVac *learnable, then it is* ConsvEx *learnable.*

In the next example, it is shown that if one imposes the condition that the learner must use a class-preserving hypothesis space, then semantic conservativeness does not in general imply its syntactic analogue for explanatory learning.

Example 9. *If A is a nonrecursive r.e. set, then the class $\mathcal{C} = \{A \cup \{x\} : x \notin A\}$ is semantically conservatively and explanatorily learnable using a class-preserving hypothesis space, as well as syntactically conservatively and prudently explanatorily learnable using a class-comprising hypothesis space, but the class $\mathcal{C}$ is not* ConsvEx *learnable using a class-preserving hypothesis space.*

Fulk [7] proved that every explanatorily learnable class is also prudently explanatorily learnable; Jain, Stephan and Ye [12] established the corresponding result for behaviourally correct learning. In connection to these results, one may ask whether the relation in Theorem 8 still holds when the learner is required to be prudent. The following theorem answers this question affirmatively, showing that every conservatively explanatorily learnable class of r.e. languages is also conservatively explanatorily learnable by a prudent learner.

Theorem 10. *If a class of r.e. languages is* ConsvEx *learnable, then it is* PrudConsvEx *learnable.*

The following example emphasises the distinction between prudence and the use of a class-preserving hypothesis space; even when combined with conservativeness, prudence is not sufficient to guarantee that a uniformly r.e. class of languages is conservatively partially learnable with respect to a class-preserving hypothesis space.

Example 11. *Let* $\mathcal{L} = \{L_{\langle d,2s\rangle} : d, s \in \mathbb{N}\} \cup \{L_{\langle d,2s+1\rangle} : d, s \in \mathbb{N}\}$, *where*

$$L_{\langle d,2s\rangle} = \begin{cases} \{d\} & \text{if } W_{d,s} = W_d; \\ \{d, t+1\} & \text{if } t \text{ is the first stage with } W_{d,s} \subset W_{d,t}, \end{cases}$$
$$L_{\langle d,2s+1\rangle} = \{d, d+s+1\}.$$

$\mathcal{L}$ *is* PrudConsvEx *learnable but not* ClsPresvConsvPart *learnable.*

Remark 12. One also has that the conservatively vacillatorily learnable classes of r.e. languages constitute a strict subset of the prudently conservatively behaviourally correctly learnable classes when a class-preserving hypothesis space is enforced. For example, the class $\{\mathbb{K} \cup D : D \text{ is finite}\}$ is prudently conservatively behaviourally correctly learnable using a class-preserving hypothesis space, but it is not vacillatorily learnable (using any hypothesis space).

Coming to a more particular setting, the following result demonstrates the equivalence of conservative partial learnability and Ex[$\mathbb{K}$] learnability for classes comprising infinite sets. The hypothesis that all the languages in the class be infinite cannot, however, be dropped, as may be seen from Example 7.

Theorem 13. *Let $\mathcal{C}$ be a class of infinite r.e. sets. Then $\mathcal{C}$ is* ConsvPart *learnable if and only if it is* Ex[$\mathbb{K}$] *learnable.*

The subsequent example gives an instance of a set A such that $A \not\leq_{\mathrm{T}} \mathbb{K}$ and the relation Ex[A] $\subset$ ConsvPart no longer holds, even when confined to classes of infinite sets.

Example 14. *The class of infinite sets*

$$\mathcal{C} = \{\{e\} \oplus (W_e \cup D) : D \text{ is finite and } W_e \text{ is cofinite}\} \cup \{\{e\} \oplus \mathbb{N} : e \in \mathbb{N}\}$$

is Ex[$\mathbb{K}'$] *learnable but not conservatively partially learnable.*

The following theorem sharpens Theorem 3, demonstrating that by imposing the further learning constraint of *prudence*, prudent ConsvBC learnability does not in general guarantee prudent ConsvPart learnability.

Theorem 15. *There is a class of r.e. languages which is* ConsvPart *learnable and prudently* ConsvBC *learnable, but not prudently* ConsvPart *learnable.*

Angluin [1] introduced the concept of a *tell-tale set* of a language L to be learnt: it is a finite subset E_L of L such that there is no language L' in the class of languages to be learnt which satisfies $E_L \subseteq L' \subset L$. She characterised indexed families of recursive languages which are explanatorily learnable from positive data using this notion. The next two theorems give characterisation for conservative partial learnability as well as conservative Ex[$\mathbb{K}$] learnability in a similar fashion.

Theorem 16. *A class $\mathcal{C}$ is* ConsvPart *learnable iff there is a recursive sequence of pairs (i_e, j_e) such that*

1. *$D_{i_e} \subseteq W_{j_e}$ for all e;*
2. *For all $L \in \mathcal{C}$ there is an e with $L = W_{j_e}$;*
3. *For all d, e, if $D_{i_e} \subseteq W_{j_d} \subset W_{j_e}$ then $W_{j_d} \notin \mathcal{C}$.*

Theorem 17. *A class $\mathcal{C}$ is* ConsvEx[$\mathbb{K}$] *learnable iff there is a recursive sequence of pairs (i_e, j_e) such that*

1. *$W_{i_e} \subseteq W_{j_e}$ for all e;*
2. *For all $L \in \mathcal{C}$ there is an e with W_{i_e} being finite and $L = W_{j_e}$;*
3. *For all d, e, if W_{i_e} is finite and $W_{i_e} \subseteq W_{j_d} \subset W_{j_e}$ then $W_{j_d} \notin \mathcal{C}$.*

References

1. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45(2), 117–135 (1980)
2. Bārzdiņš, J.: Two theorems on the limiting synthesis of functions. In: Theory of Algorithms and Programs, vol. 1, pp. 82–88. Latvian State University (1974) (in Russian)
3. Case, J., Lynes, C.: Machine inductive inference and language identification. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP 1982. LNCS, vol. 140, pp. 107–115. Springer, Heidelberg (1982)
4. Case, J.: The power of vacillation in language learning. SIAM Journal on Computing 28(6), 1941–1969 (1999)
5. Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Information and Control 28, 125–155 (1975)
6. Feldman, J.: Some decidability results on grammatical inference and complexity. Information and Control 20, 244–262 (1972)
7. Fulk, M.: Prudence and other conditions on formal language learning. Information and Computation 85, 1–11 (1990)
8. Gao, Z., Stephan, F., Wu, G., Yamamoto, A.: Learning families of closed sets in matroids. In: Dinneen, M.J., Khoussainov, B., Nies, A. (eds.) WTCS 2012 (Calude Festschrift). LNCS, vol. 7160, pp. 120–139. Springer, Heidelberg (2012)
9. Gao, Z., Stephan, F.: Confident and consistent partial learning of recursive functions. In: Bshouty, N.H., Stoltz, G., Vayatis, N., Zeugmann, T. (eds.) ALT 2012. LNCS (LNAI), vol. 7568, pp. 51–65. Springer, Heidelberg (2012)
10. Gold, M.E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
11. Jain, S., Osherson, D., Royer, J.S., Sharma, A.: Systems That Learn: An Introduction to Learning Theory. MIT Press, Cambridge (1999)
12. Jain, S., Stephan, F., Ye, N.: Prescribed learning of r.e. classes. Theoretical Computer Science 410(19), 1796–1806 (2009)
13. de Jongh, D., Kanazawa, M.: Angluin's theorem for indexed families of r.e. sets and applications. In: Proceedings of the Ninth Annual Conference on Computational Learning Theory, pp. 193–204. ACM Press (1996)
14. Lange, S., Zeugmann, T.: Language learning in dependence on the space of hypotheses. In: Proceedings of the Sixth Annual Conference on Computational Learning Theory, pp. 127–136. ACM Press (1993)
15. Osherson, D.N., Stob, M., Weinstein, S.: Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists. MIT Press, Cambridge (1986)
16. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. MIT Press, Cambridge (1987)
17. Wiehagen, R., Zeugmann, T.: Learning and consistency. In: Lange, S., Jantke, K.P. (eds.) GOSLER 1994. LNCS, vol. 961, pp. 1–24. Springer, Heidelberg (1995)
18. Zeugmann, T., Lange, S., Kapur, S.: Characterizations of monotonic and dual monotonic language learning. Information and Computation 120(2), 155–173 (1995)
19. Zeugmann, T., Zilles, S.: Learning recursive functions: a survey. Theoretical Computer Science 397(1-3), 4–56 (2008)

Topology of Asymptotic Cones and Non-deterministic Polynomial Time Computations

Anthony Gasperin

Theoretical Computer Science Department, University of Geneva, Switzerland
anthony.gasperin@unige.ch

Abstract. In this paper we study the topology of asymptotic cones of groups constructed from $\mathcal{S}$-machines running in polynomial time. In particular we directly construct an $\mathcal{S}$-machine for an **NP**-complete problem. Using a part of the machinery shaped by Sapir,Birget and Rips we construct its associated group and we show that every asymptotic cone of this group is not simply connected. The proof is rather geometric and use an argument similar to the one developed by Sapir and Olshanskii. This work aims to give a topological characterization of non-deterministic time complexity class.

1 Introduction

Let (X, d_X) a metric space $s = (s_n)$ a sequence of points in X, $d = (d_n)$ an increasing sequences of numbers with $\lim d_n = \infty$ and let $\omega : P(\mathbb{N}) \to \{0, 1\}$ be a non-principal ultrafilter. An *asymptotic cone* of $\mathrm{Con}_\omega(X, s, d)$ of (X, d_X) is the subset of the cartesian product $X^{\mathbb{N}}$ consisting of sequences $(x_i)_{i\in\mathbb{N}}$ with $\lim_\omega \frac{d_X(s_i,x_i)}{d_i} < \infty$ where two sequences (x_i) and (y_i) are equivalent if and only if $\lim_\omega \frac{d_x(x_i,y_i)}{d_i} = 0$. The distance between two elements $(x_i), (y_i)$ in the asymptotic cone $\mathrm{Con}_\omega(X, s, d)$ is defined as $\lim_\omega \frac{d_X(x_i,y_i)}{d_i}$. Here $\lim_\omega$ is defined as follows. If a_n is a bounded sequence of real numbers then $\lim_\omega(a_n)$ is the unique number a such that for every $\epsilon > 0$, $\omega(\ \{n \mid |a_n - a| < \epsilon\}\ ) = 1$. The *asymptotic cones* of a finitely generated group G are asymptotic cones of the Cayley graph of G and it well known that they do not depend on the choice of the sequence s. It is then assumed that $s = (1)$ where 1 is the identity. Given an ultrafilter ω and an increasing sequence of numbers d the asymptotic cone of a finitely generated group G is then noted $\mathrm{Con}_\omega(X, d)$.

A function $f : \mathbb{N} \to \mathbb{N}$ is an isoperimetric function of a finite presentation $\langle X, R\rangle$ of a group G if every word w in X, that is equal to 1 in G, is freely equal to a product of conjugates $\prod_{i=1}^{m} x_i^{-1} r_i x_i$ where r_i or r_i^{-1} is in R, x_i is in $(X \cup X^{-1})^*$ and $m \leq f(|w|)$. *Dehn's function* of a finite presentation $\langle X, R\rangle$ is defined as the smallest *isoperimetric* function of the presentation.

In [1–3] the connections between Dehn's functions, asymptotic geometry of groups and computational complexity of the word problem are discussed. In [2]

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 191–199, 2013.

Gromov showed that if all *asymptotic cones* of a group G are simply connected then G is finitely presented, has polynomial isoperimetric function and linear isodiametric function. Papasoglu [4] proved that if a finitely presented group has quadratic isoperimetric function then all its asymptotic cones are simply connected. Sapir, Birget and Rips in [5] introduced the concept of $\mathcal{S}$-machines to show that the word problem of a finitely generated group is decidable in polynomial time if and only if this group can be embedded into a group with polynomial isoperimetric function. Olshanskii and Sapir in [6] constructed a group with polynomial isoperimetric function, linear isodiametric function and non-simply connected asymptotic cones, the group is roughly an $\mathcal{S}$-machine introduced in [5]. In [7] they also constructed a group with two non-homeomorphic asymptotic cones using the concept of $\mathcal{S}$-machine.

In this paper we construct a group from the well known **NP**-complete problem 3SAT. The construction does not use the full machinery of Sapir, Birget and Rips. Indeed we construct directly an $\mathcal{S}$-machine for 3SAT. Then we use a tiny part of the construction of Sapir, Birget and Rips to construct the corresponding group. We finally show, in the same fashion as the proof in [6] that every asymptotic cones of this group is not simply connected. The topological characterization of the asymptotic cones corresponding to non-deterministic time computation is interesting in the following sense; one can take an **NEXP**-complete problem and construct its associated group G using the machinery of Sapir, Birget and Rips. Since $\mathbf{NEXP} \neq \mathbf{NP}$ the isoperimetric functions of G are not polynomial and thus applying the result of Gromov [2] one can conclude that G has at least one asymptotic cone which is not simply connected. Our work is then a first step in the construction of an answer to the question: "Is the difference between **NEXP** and **NP** reflected in term of topology?". The partial answer we get is rather negative. Indeed, considering a group directly constructed from an $\mathcal{S}$-machine the type of diagram that turns every asymptotic cones not simply connected are not diagram representing accepting computations.

2 Preliminaries

This section introduces briefly the machinery introduced by Sapir, Birget and Rips in [5]. We need to explain, at least superficially, what is an S-machine, how it works and especially how it leads to the construction of group.

2.1 *S*-Machines

This section is closely modeled on [5], we recall the notion of S-machine defined in the work of Sapir, Birget and Rips in [5]. As we already mentioned, our construction will not use the whole machinery, in particular our S-machine will be simpler than the one describe in the following. Indeed the construction in [5] is far more general and then one has to be careful when constructing an S-machine simulating some Turing machine. Our work does not need such a care since it

focuses on the construction of a group from an $\mathcal{S}$-machine and does not need the characterization of Dehn's function.

[5] defined S-machines as rewriting systems. An S-machine then comes with a *hardware*, a *language of admissible words*, and a set of *rewriting rules*. A *hardware* of an S-machine is a pair (Y, Q) where Y is an n-vector of (not necessarily disjoint) sets Y_i, Q is an $(n+1)$-vector of disjoints sets Q_i with $(\bigcup Y_i) \cap (\bigcup Q_i) = \emptyset$. The elements of $\bigcup Y_i$ are called *tape letters*; the elements of $\bigcup Q_i$ are called *state letters*. With every hardware $\mathcal{S} = (Y, Q)$ one can associate the *language of admissible words* $L(\mathcal{S}) = Q_1 F(Y_1) Q_2 \cdots F(Y_n) Q_{n+1}$ where $F(Y_i)$ is the language of all reduced group words in the alphabet $Y_j \cup Y_j^{-1}$. This language completely determines the hardware. One can then describe the language of admissible words instead of describing the hardware $\mathcal{S}$. If $1 \leq i < j \leq n$ and $W = q_1 u_1 q_2 \cdots u_n q_{n+1}$ is an admissible word, $q_i \in Q_i, u_i \in (Y_i \cup Y_i^{-1})^*$ then the subword $q_i u_i \cdots q_j$ of W is called the (Q_i, Q_j)-subword of W $(i < j)$. The rewriting rules (*S-rules*) have the following form:

$$[U_1 \to V_1, \ldots, U_m \to V_m]$$

where the following conditions hold: Each U_i is a subword of an admissible word starting with a Q_l-letter and ending with Q_r-letter. If $i < j$ then $r(i) < l(j)$, where $r(i)$ is the end of U_i and $l(j)$ the start of U_j. Each V_i is a subword of and admissible word whose Q-letters belong to $Q_{l(i)} \cup \cdots \cup Q_{r(i)}$. The machine applies an S-rule to a word W replacing simultaneously subwords U_i by subword $V_i, i = 1, \ldots, m$.

As mentioned in [5] there exists a natural way to convert a Turing machine M into an S-machine $\mathcal{S}$; one can concatenate all tapes of the given machine M together and replace every command $aq\omega \to q'\omega$ by $a^{-1}q'\omega$. Unfortunately the S-machine constructed following this natural way will not inherit most of the properties of the original machine M. According to [5] the main problem is that it is nontrivial to construct an S-machine which recognizes only positive powers of a letter. Thus in order to construct an S-machine $\mathcal{S}(M)$ that will inherit the desired properties of a Turing machine M, Sapir, Birget and Rips in [5] constructed eleven S-machines and then used them to construct the final S-machine $\mathcal{S}(M)$ simulating M. The construction is quite involved and nontrivial, cf. [5] for details.

Taking any Turing machine $M = \langle X, Y, Q, \Theta, \boldsymbol{s}_1, \boldsymbol{s}_0 \rangle$ and modifying it in a specific way, [5] constructs an S-machine $\mathcal{S}(M)$ simulating M. The S-machine constructed in [5] is quite long to define, next we explain briefly the main part of the construction, for proofs and deeper understanding of the whole machinery please refer to [5]. The main idea of the construction is to simulate the initial machine M using eleven S-machines $S_1, S_2, \ldots, S_9, S_\alpha, S_\omega$. We will explain how the machines $S_4, S_9, S_\alpha, S_\omega$ are used in the construction of $\mathcal{S}(M)$. The others S-machines are used to construct S_4 and S_9 and are rather of technical importance. First we need by describing what is an admissible word of the S-machine $\mathcal{S}(M)$. For every $q \in Q$ the word $q\omega$ is denoted by F_q, in every command of M the word $q\omega$ is replaced by F_q. Left marker on tape i is denoted by E_i. This gives a Turing Machine M' such that the configurations of each tape have the form

$E_i u F_q$ where u is a word in the alphabet of tape i and every command or its inverse has one of the forms:

$$\{F_{q_1} \to F_{q_1'}, \ldots, aF_{q_i} \to F_{q_i'}, \ldots, F_{q_k} \to F_{q_k'}\} \tag{1}$$

where $a \in Y$ or

$$\{F_{q_1} \to F_{q_1'}, \ldots, E_i F_{q_i} \to E_i F_{q_i'}, \ldots, F_{q_k} \to F_{q_k'}\}. \tag{2}$$

An admissible word of the considered $\mathcal{S}(M)$ machine is a product of three parts. The first part has the form

$$E(0)\alpha^{n_1} x(0) \alpha^{n_2} F(0).$$

The second part is a product of k words of the form

$$E(i)v_i x(i) w_i F(i) E'(i) p(i) \Delta_{i,1} q(i) \Delta_{i,2} r(i) \Delta_{i,3} s(i) \Delta_{i,4} t(i) \Delta_{i,5} u(i) \Delta_{i,6} \overline{p}(i)$$
$$\Delta_{i,7} \overline{q}(i) \Delta_{i,8} \overline{r}(i) \Delta_{i,9} \overline{s}(i) \Delta_{i,10} \overline{t}(i) \Delta_{i,11} \overline{u}(i) \Delta_{i,12} F'(i)$$

for $i = 1, \ldots, k$. The third part has the form

$$E'(k+1)\omega^{n_1'} x'(k+1) \omega^{n_2'} F'(k+1).$$

Here v_i, w_i are group words in the alphabet Y_i of tape i, and $\Delta_{i,j}$ is a power of δ. The letters $E(i), x(i), F(i), E'(i), p(i), q(i), r(i), s(i), t(i), u(i), \overline{p}(i), \overline{q}(i), \overline{r}(i), \overline{s}(i), \overline{t}(i), \overline{u}(i), F'(i)$ belong to disjoint sets of state letters.

The letters $x(i), p(i), q(i), r(i), s(i), t(i), u(i), \overline{p}(i), \overline{q}(i), \overline{r}(i), \overline{s}(i), \overline{t}(i), \overline{u}(i)$ are called standard and are included into the corresponding sets $\mathbf{X}(i), \mathbf{P}(i), \mathbf{R}(i), \mathbf{S}(i), \mathbf{T}(i), \mathbf{U}(i), \overline{\mathbf{P}}(i), \overline{\mathbf{Q}}(i), \overline{\mathbf{R}}(i), \overline{\mathbf{S}}(i), \overline{\mathbf{T}}(i), \overline{\mathbf{U}}(i), (i = 1, \ldots, k)$. Let τ be a command in Θ of the form (1) (command of the form (1) are are called *positive* and their inverse *negative*). For every $\gamma \in \{4, 9, \alpha, \omega\}$ and for each component $V(i)$ of the vector of sets of state letters, the letter $V(i, \tau, \gamma)$ are included into $V(i)$ where $V \in \{P, Q, R, S, T, U, \overline{P}, \overline{Q}, \overline{R}, \overline{S}, \overline{T}, \overline{U}\}$. For each S-machine $S_\gamma, \gamma \in \{4, 9, \alpha, \omega\}$ a copy of S_γ is considered where every state letter z is replaced by $z(j, \tau, \gamma)$ where $j = i$ if $\gamma = 4, 9, j = 0$ if $\gamma = \alpha$ and $j = k + 1$ if $\gamma = \omega$. These state letters are included into the corresponding sets. The state letters we just described are all the state letters of $\mathcal{S}(M)$. The rules of $\mathcal{S}(M)$ are the rules of $\mathcal{S}_4(\tau), \mathcal{S}_9(\tau), \mathcal{S}_\gamma(\tau), \mathcal{S}_\omega(\tau)$ for all $\tau \in \Theta$ of the form (1) plus the connecting rules. Basically the connecting rules allows to go from a machine to another one, there are five such rules: $R_4(\tau), R_{4,\alpha}(\tau), R_{\alpha,\omega}(\tau), R_{\omega,9}(\tau), R_9(\tau)$. They can be described informally as follows. $R_4(\tau)$ turns on the machine $\mathcal{S}_4(\tau)$. $R_{4,\alpha}(\tau)$ turns on the machine $\mathcal{S}_\alpha(\tau)$ when $\mathcal{S}_4(\tau)$ finishes its work, $R_{\alpha,\omega}(\tau), R_{\omega,9}(\tau)$ do the same with the corresponding S-machines. $R_9(\tau)$ turns off $\mathcal{S}_9(\tau)$ and gets the machine ready to simulate the next transition from Θ. This machinery contains all the necessary steps to simulate a rule of the machine M.

Formally speaking, to every configuration $c = (E_1 v_1 F_{q_1}, \ldots, E_k v_k F_{q_k})$ of the machine M is associated the following admissible word $\sigma(c)$ of $\mathcal{S}(M)$:

$$E(0)\alpha^n x(0)F(0)E(1)v_1x(1)F_{q_1}(1)E'(1)p(1)\delta^{||v_1||}q(1)r(1)s(1)t(1)u(1)\overline{p}(1)$$
$$\overline{q}(1)\overline{r}(1)\overline{s}(1)\overline{t}(1)\overline{u}(1)F'_{q_1}(1)\dots E(k)v_kx(k)F_{q_k}(k)$$
$$E'(k)p(k)\delta^{||v_k||}q(k)r(k)s(k)t(k)u(k)\overline{p}(k)\overline{q}(k)$$
$$\overline{r}(k)\overline{s}(k)\overline{t}(k)\overline{u}(k)F'_{q_k}(k)E'(k+1)x'(k+1)\omega^nF'(k+1),$$

where $||v||$ is the algebraic sum of the degree of the letters in v.

Now we present how [5] converts an S-machine $\mathcal{S}(M)$ into a group presentation, once again this part is strongly modeled on [5]. Let $\mathcal{S}(M)$ be the S-machine as constructed. Let Y be the vector of sets of tape letters, and let Q be the vector of state letters of S. One can remark that Q has $17k+6$ components which [5] denotes by $Q_1,\dots,Q_{17k+6}$. [5] notices that $Q_1=\mathbf{E}(0), Q_2=\mathbf{X}(0), Q_3=\mathbf{F}(0), Q_{17k+4}=\mathbf{E}'(k+1), Q_{17k+5}=\mathbf{X}(k+1), Q_{17k+6}=\mathbf{F}'(k+1)$. Let $\mathbf{\Theta}_+$ the set of positive rules of $\mathcal{S}(M)$ and N a positive integer. To construct their group $G_N(\mathcal{S})$, Sapir, Birget and Rips take the following generating sets :

$$A=\bigcup_{i=1}^{17k+6}Q_i\cup\{\alpha,\omega,\delta\}\cup\bigcup_{i=1}^{k}Y_i\cup\{\kappa_j|j=1,\dots,2N\}\cup\mathbf{\Theta}_+. \tag{3}$$

and the following set $P_N(\mathcal{S})$ of relations:

1. *Transitions relations.* These relations correspond to elements of $\mathbf{\Theta}_+$. Let $\tau\in\mathbf{\Theta}_+, \tau=[U_1\to V_1,\dots,U_p\to V_p]$. Then relations $\tau^{-1}U_1\tau=V_1,\dots,\tau^{-1}U_p\tau=V_p$ are included into $P_N(\mathcal{S})$. If for some j from 1 to $17k+6$ the letters from Q_j do not appear in any of the U_i then the relations $\tau^{-1}q_j\tau=q_j$ for every $q_j\in Q_j$ are also included.
2. *Auxiliary relations.* These are all possible relations of the form $\tau x=x\tau$ where $x\in\{\alpha,\omega,\delta\}\cup\bigcup_{i=1}^{k}Y_i, \tau\in\mathbf{\Theta}_+$.
3. *The hub relation.* For every word u let $K(u)$ denote the following word:

$$K(u)\equiv(u^{-1}\kappa_1u\kappa_2u^{-1}\kappa_3u\kappa_4\dots u^{-1}\kappa_{2N-1}u\kappa_{2N})\times$$
$$(\kappa_{2N}u^{-1}\kappa_{2N-1}u\dots\kappa_2u^{-1}\kappa_1u)^{-1}.$$

Then the relation hub is $K(W_0)=1$, where W_0 is the accepting configuration of the $\mathcal{S}$-machine.

The objective of Sapir, Birget and Rips [5] in constructing such groups is to prove the following theorem :

Theorem 1. *[5] Let $L\subseteq X^+$ be a language accepted by a Turing machine M with a time function $T(n)$ for which $T(n)^4$ is superadditive. Then there exists a finitely presented group $G(M)=\langle A\rangle$ with Dehn's function equivalent to $T(n)^4$, the smallest, isodiametric function equivalent to $T^3(n)$, and there exists an injective map $K:X^+\to(A\cup A^{-1})^+$ such that*

1. *$u\in L$ if and only if $K(u)=1$ in G;*
2. *$K(u)$ has length $O(|u|)^2$ and is computable in time $O(|u|)$.*

3 Construction of $G_{\mathbf{SAT}}$

In this section we shall detail the construction of an $\mathcal{S}$-machine for an **NP**-complete problem, then following a part of [5] we construct a group from the $\mathcal{S}$-machine. As we shall see, when one consider only positive admissible word the $\mathcal{S}$-machine recognizes satisfiable 3SAT formula. The fact is that it is difficult to design an $\mathcal{S}$-machine that exactly simulate a Turing machine, indeed the addition of inverse rules extends the accepted language of the machine. Sapir, Birget and Rips in [5] shows how to overcome this difficulty in a general setting. The drawback of the construction is that the time function of the original machine is increased polynomially. Moreover the upper bound gave in their work for Dehn's function of the associated group take into account this polynomially jump. Thus if one wishes to study the topology of asymptotic cones of group constructed from $\mathcal{S}$-machine one has to find a way to get round the simulation since it can changes drastically the topology of the asymptotic cones.

Let us first define the set of tape and state letters of the first machine. The state sets are $Q_0 = \{\alpha\}, Q_1 = \{q_0, q_b, q_{b_1}, q_c, q_d, q_f, q_r\}, Q_2 = \{p_0, p_1, p_e, p_r, p_f\}$, $Q_3 = \{u_0, u_1, u_e, u_r, u_f\}, Q_4 = \{v_0, v_1, v_e, v_r, v_f\}, Q_5 = \{z_0, z_1, z_e, z_r, z_f\}, Q_6 = \{t_0, t_1, t_2, t_3, t_f\}$. The letter sets are $Y_0 = \{0, 1, \xi, \gamma, \sharp\}, Y_1 = \{0, 1, \xi, \gamma, \sharp\}, Y_2 = \{0, 1, \xi, \gamma, \eta, F, T, \zeta\}, Y_3 = \{0, 1, \xi, \gamma, \eta, F, T, \zeta\}, Y_3 = \{0, 1, \xi, \gamma, \eta, F, T, \zeta\}, Y_4 = \{0, 1, \xi, \gamma, \eta, F, T, \zeta\}, Y_5 = \{T, F\}$. We need to encode every input $(x_1 \vee x_2 \vee x_3) \wedge \cdots \wedge (x_k \vee x_{k+1} \vee x_{k+2})$ of the problem 3SAT. We choose "ξ" and "$\sharp$" to represents respectively "$\vee$" and "$\wedge$". Every x_i appearing in a clause is encoded by the binary representation $\mathrm{bin}(i)$ of i. The symbol $\overline{x}_i$ is represented by the sequence $\gamma\mathrm{bin}(i)$. Intuitively our machine works as follows; it reads an x_i, checks if a value has been already assigned to x_i or $\overline{x}_i$, if not it assigns a value to it. Next we detail the rules of the machine, let us call it $\mathcal{S}_0$.

We first define a set of rules that allows the copy of the symbol reads and that assigns to it a value. The admissible word corresponding to the start configuration of a computation on ω is $w_s = \alpha q_0 \omega p_0 u_0 v_0 z_0 t_0$.

$$
\begin{aligned}
I_0 &: [q_0 \to q_c, p_0u_0v_0z_0t_0 \to p_0u_0v_0z_0t_0],\\
I_1 &: [q_0 \to q_d\gamma^{-1}, p_0u_0v_0z_0t_0 \to p_0u_0v_0z_0t_0],\\
R_0 &: [q_c \to q_cx^{-1}, v_0yz_0 \to yv_0xz_0],\\
R_1 &: [q_c \to q_c\xi^{-1}, v_0yz_0 \to yv_0\eta Tz_0, Xt_i \to Tt_{i+1}],\\
R_2 &: [q_c \to q_c\xi^{-1}, v_0yz_0 \to yv_0\eta Fz_0, Xt_i \to Xt_{i+1}],\\
R_3 &: [q_c \to q_b\sharp^{-1}, v_0yz_0 \to yv_0\zeta z_0, Tt_3 \to Tt_1],\\
R_4 &: [q_c \to q_p\sharp^{-1}, v_0yz_0 \to yv_p\zeta z_p, Ft_3 \to t_p],\\
T_0 &: [q_d \to q_dx^{-1}, v_0yz_0 \to yv_0xz_0],\\
T_1 &: [q_d \to q_d\xi^{-1}, v_0yz_0 \to yv_0\eta Tz_0, Xt_i \to Xt_{i+1}],\\
T_2 &: [q_d \to q_d\xi^{-1}, v_0yz_0 \to yv_0\eta Fz_0, Xt_i \to Tt_{i+1}],\\
T_3 &: [q_d \to q_b\sharp^{-1}, v_0yz_0 \to yv_0\zeta z_0, Tt_3 \to Tt_1], \text{ and}
\end{aligned}
$$

$$T_4 : [q_d \to q_p \sharp^{-1}, v_0 y z_0 \to y v_p \zeta z_p, F t_3 \to t_p].$$

Here $x \in \{0,1\}$, $i \in \{1,2,3\}$. Assuming the input is a positive admissible word corresponding to a 3SAT instance, the rules just defined allow to copy the code of a literal in a clause, moreover it assigns to it a value. One has to be careful reading the rule T_i, R_i, even if they look the same they differ slightly. Indeed the rule T_i are applied in the case where a γ has been detected, then when the value of the binary representation is saved and set to value (T, F) the current value of the formula is update with the inverse value (rules T_1, T_2). Now let us define the rules that check if a literal has been recorded and update the computation with its value.

$$\begin{aligned}
A_1 &: [q_b \to x q_b x^{-1}, u_1 \to x u_1 x^{-1}],\\
A_2 &: [q_b \to \xi q_{b_1} \xi^{-1}, u_1 \to \eta u_1 \eta^{-1}],\\
A_3 &: [q_{b_1} \to q_e, u_1 \to F u_e F^{-1}, X t_i \to X t_{i+1}],\\
A_4 &: [q_{b_1} \to q_e, u_1 \to T u_e T^{-1}, X t_i \to T t_{i+1}],\\
B_1 &: [\alpha \to \alpha x^{-1}, q_e \to q_e, u_e \to y^{-1} u_e y],\\
B_2 &: [\alpha q_e \to \alpha q_e, u_e \to y^{-1} u_e y],\\
B_3 &: [\alpha \to \alpha q_b q_e^{-1}, p_0 u_e \to p_0 u_0],\\
C_1 &: [q_b \to x q_r x^{-1}, u_1 \to y u_r y^{-1}], y \neq x, x \neq \xi, y \neq \eta,\\
C_2 &: [q_r \to x^{-1} q_r x, u_r \to y^{-1} u_r y],\\
C_3 &: [\alpha q_r \to q_c, u_r \to u_0], \text{ and}\\
F &: [q_b \to q_b p_f p_0^{-1}, u_0 \to u_f, v_0 \to v_f, z_0 \to z_f, T t_1 \to t_f].
\end{aligned}$$

Rules C_1, C_2, C_3 are applied when the string is not found, they bring the machine in the state that will record the literal and set its value. Again, as we did for the first set of rule we defined, the rule regarding the symbol $\overline{x_i}$ must be defined. Since they are essentially the same we omitted them. It is not difficult to see that considering only application of positive rule and given a positive admissible word corresponding to a start configuration of a word w representing a 3SAT formula the machine goes in the final configuration if and only if the formula is satisfiable. But since an $\mathcal{S}$-machine has also the inverse rules, other words can be accepted.

Now we are going to see how the group is constructed from the $\mathcal{S}$-machine, basically it follows the construction of Sapir, Birget and Rips, but it is far less complicated since we do not need the machinery that allows the simulation. We define the group $G_N(\mathcal{S}_0)$ constructed from the following generating set

$$A = \bigcup_{i=1}^{i=6} Q_i \cup \bigcup_{i=1}^{i=5} Y_i \cup \{\kappa_j | j = 1, \ldots, 2N\} \cup \theta_+. \tag{4}$$

where θ_+ is the set of positive rules and κ_j the symbols added to define the hub relation. N is a fixed integer such that $N \geq 6$ for some small cancellation reason (cf. [5] for more detailed explanations).

The presentation is defined in the same way that the presentation introduces in last section, but considering the generating set just defined. The group has some properties similar to the one defined by Sapir, Birget and Rips. In particular it is immediate to conclude that the same arguments, involving computational sectors and computational discs, allow to conclude the following lemma of [5], but restricted to the positive admissible word.

Lemma 1. *For every positive admissible word W of the machine $\mathcal{S}$, there exists a computation of $\mathcal{S}$ connecting W and W_0 if and only if $K(W) = 1$ in the group G_{SAT}.*

In our case W_0 is the accepting configuration $q_f p_f v_f z_f u_f t_f$. For the rest of the paper the group will be denoted G_{SAT}.

3.1 Topology of Asymptotic Cones of G_{SAT}

In this section we show that every asymptotic cones of G_{SAT} is not simply connected. For that we need to see exactly what happen when a rule is applied in the formal construction. Remember that the symbol are encoded in binary. Let us consider the rule R_0. The rule R_0 writes a 0 or a 1 between $v_0 z_0$. We shall focus on the case where a 1 is written between $v_0 z_0$. One can remark that the rule T_0 works exactly as R_0 for the word $v_0 z_0$.

Thus in the group G_{SAT} there exists τ_1, τ_2, such that τ_1 is of the form $\tau_1^{-1} v_0 y z_0 \tau_1 = y v_0 1 z_0$, τ_2 is of the form $\tau_2^{-1} y v_0 1 z_0 \tau_2 = v_0 y z_0$ where $y = \varepsilon$ or $y = 1$. Moreover $\tau_1 \tau_2 \neq 1$. Now we can consider for each $n \in \mathbb{N}$ the van Kampen diagram Δ_n defined as follows :

- the top side and bottom side of Δ_n are labeled by $(v_0 z_0)^n$
- the right and left side are labeled by $\tau_1^n \tau_2^n$

We shall show that this kind of diagram contradicts the following statement noticed by Gromov [2]:

Suppose that an asymptotic cone $\mathrm{Con}_\omega(G, d)$ is simply connected then for every $M > 1$ there exists a number k such that for every constant $C \geq 1$, every loop l in the Cayley graph of G satisfying $\frac{1}{C} d_m \leq |l| \leq C d_m$ for any sufficient large m, bounds a disc that can be subdivided into k subdisc with perimeter at most $\frac{|l|}{M}$. Now we shall show the result, the proof use the same argument as the one in [6]. That is we show that no loop corresponding to Δ_n can bound a disc decomposed into at most $k \leq \sqrt{n}$. First let us recall what is an x-band, for every letter x. An x-edge in a van Kampen diagram is an edge labeled by $x^{\pm 1}$. An x-cell is a cell whose boundary contains an x-edge. An x-band in a diagram is a sequence of cells containing x-edges, such that every two consecutive cells share an x-edge. The boundary of the union of cells from an x-band $\mathcal{B}$ has the form $s^{-1} p e q^{-1}$ where s, e are the only x-edges on the boundary representing respectively the start and the end of the band. The paths p, q are called the *sides* of $\mathcal{B}$.

Theorem 2. *Every asymptotic cone of G_{SAT} is not simply connected.*

Proof. Let $\mathrm{Con}_\omega(G_{\mathrm{SAT}}, (d_i))$ be an asymptotic cone of G_{SAT}. Fix $n = d_m$ for a large m. Let $M = 6 = C$ and k of the Gromov statement, assume that each loop $\frac{1}{C} d_m \leq |l| \leq C d_m$ can be divided into $\sqrt{n}$ subdiagrams of perimeter n. Let $\mathcal{B}$ be maximal Θ-band in Δ_n. Since Θ-bands in Δ_n do not intersect, $\mathcal{B}$ has a side label by $(1^{i-1} v_0 1 z_0)^i$ which is equivalent in G to $\tau_1^{-i}(v_0 z_0)^i \tau_1^i$. Since the number of diagram $k \leq \sqrt{n}$ there exists a diagram Δ_i with a perimeter of length n containing the word $w = (z_0)1^{n-1}(v_0 1 z_0)$. One can choose Δ_i such that the boundary is equal to $b_1 b_2$, b_1 contains all Θ-edges, $|b_1| \leq \frac{n}{2}$ and a subword b_1' of b_1 reduced to w. Focusing on the diagram Γ with boundary $b_1' w$ we can remark that each v_0-band and z_0-band beginning on w end on b_1'. But each letter "1" is obtained by the application of a rule τ_1. There are at most $\frac{n}{2}$ application of the rule τ_1 in the diagram Γ. Moreover Γ has neither v_0-cells nor z_0-cells. Thus every 1-band beginning on b_1' must ends on w. But that is impossible since $|b_1'| < n$.

The theorem shows that to consider the topology of asymptotic cones for groups construct from $\mathcal{S}$-machine as an interesting tool to study computation, one has to be careful in the construction. The topology of the asymptotic cones appears to be highly dependent of the interactions between the rules of the machine. In future work we shall consider $\mathcal{S}$-machines for **NP**-complete problem that avoid the construction of diagrams of type Δ_n. As a main motivation we will try to find a group construct from an **NP**-complete problem having a simply connected asymptotic cone.

References

1. Rips, E., Birget, J.C., Olshanskii, A.Y., Sapir, M.V.: Isoperimetric function of groups and computational complexity of the word problem. Annals of Mathematics 156(2), 476–518 (2002)
2. Gromov, M.: Asymptotics invariants of infinite groups. Geometric Group Theory 2, 1–295 (1993)
3. Olshanskii, A.Y., Sapir, M.V.: Length and area functions on groups and quasi-isometric higman embeddings. Intern. J. Algebra and Comput. 1, 137–170 (1991)
4. Papsoglu, P.: On the asymptotic cones of groups satisfying a quadratic isoperimetric inequality. J. Differential Geometry 44, 789–806 (1996)
5. Birget, J.C., Sapir, M.V., Rips, E.: Isoperimetric and isodiametric functions of groups. Annals of Mathematics 156, 345–466 (2002)
6. Olshanskii, A.Y., Sapir, M.V.: Groups with non-simply connected asymptotic cones. In: Topology and Asymptotic Aspects of Group Theory. Contemp. Math., vol. 394, pp. 203–208 (2006)
7. Olshanskii, A.Y., Sapir, M.V.: A finitely presented group with two non-homeomorphic asymptotic cones. Internat. J. Algebra. Comput. 17(2), 421–426 (2007)

On Decidable and Computable Models of Theories

Alexander Gavruskin* and Bakhadyr Khoussainov**

Department of Computer Science, University of Auckland, New Zealand
a.gavruskin@auckland.ac.nz, bmk@cs.auckland.ac.nz

Abstract. In this paper we obtain two results using amalgamation classes and Fraïssé limits. First, we construct a decidable theory T whose types are all decidable yet whose prime model is not decidable. Millar [15] constructed such example but his example uses an infinite language in an essential way. Our example uses one binary predicate symbol, that is, the models we construct are graphs. Second, for any finite lattice $\mathcal{F}$ we construct a theory T with countably many models such that the fundamental order determined by T is isomorphic to $\mathcal{F}$. As a by-product of this example, we propose the investigation of computable and decidable models of T by connecting them to the fundamental order of T.

1. Introduction

Let T be a complete consistent first order theory of a countable language. Let $\mathrm{Mod}(T)$ be the class of all countable models of T. We are interested in those models of $\mathrm{Mod}(T)$ that can be described effectively. Effectiveness can be introduced through considering diagrams of the models of T as follows.

For a model $\mathcal{M} \in \mathrm{Mod}(T)$, expand the language of $\mathcal{M}$ by adding constant symbols c_m for all elements m of $\mathcal{M}$. Denote the expanded language by $L_{\mathcal{M}}$ and the expanded structure by $(\mathcal{M}, c_m)_{m\in M}$. The *atomic diagram* of $\mathcal{M}$ is the set:

$$\mathcal{DA}(\mathcal{M}) \leftrightharpoons \{\varphi \mid (\mathcal{M}, c_m)_{m\in M} \models \varphi \text{ and } \varphi \text{ is a quantifier free sentence of } L_{\mathcal{M}}\}$$

We say that $\mathcal{M}$ is a *computable model* of T if $\mathcal{DA}(\mathcal{M})$ is a decidable set. A stronger notion of effectivity can be introduced by considering the *elementary diagram*:

$$\mathcal{D}(\mathcal{M}) \leftrightharpoons \{\varphi \mid (\mathcal{M}, c_m)_{m\in M} \models \varphi \text{ and } \varphi \text{ is a sentence of } L_{\mathcal{M}}\}$$

We say that $\mathcal{M}$ is a *decidable model* of T if $\mathcal{D}(\mathcal{M})$ is a decidable set. Clearly, decidable models are computable. We slightly abuse our definition and refer to

* A. Gavruskin is supported by the FRDF grant of the University of Auckland and by the Federal Target Program "Research and Training Specialists in Innovative Russia, 2009-2013", Contracts No Π1227 and No 16.740.11.0567.

** B. Khoussainov is partially supported by Marsden Fund grant of the Royal Society of New Zealand.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 200–209, 2013.

a model $\mathcal{M}$ as computable (decidable) if the isomorphism type of $\mathcal{M}$ contains a computable (decidable) model. Thus, being computable or decidable is a property of the isomorphism types of the models.

There are theories with computable but without decidable models. For instance, the arithmetic $(\omega; +, \times, \leqslant, 0, 1)$ is a computable model of the Peano arithmetic PA, and PA has no decidable models [5].

The standard Henkin construction produces a model of T whose elementary diagram is decidable with an oracle for T. Hence, the theory T is decidable if and only if it has a decidable model [5]. Given a decidable theory T, one can ask many natural questions about decidability of models of T: Is the prime model of T decidable? Is the saturated model of T decidable? Is there a homogeneous decidable model of T? Is there a decidable model of T that omits a given (non-principal) type of T? By the early 80s all these questions have been answered in a series of papers. Goncharov and Nurtazin [6] and independently Harrington [8] proved that the existence of a uniform procedure that lists all principal types of T is a necessary and sufficient condition for T to possess a decidable prime model. Similarly, Morley [17] proved that T has a saturated decidable model if and only if the class of all types of T is uniformly decidable. Peretyatkin [19] and independently Goncharov [4] gave criteria for a homogeneous model of T to be decidable. Millar [15,16] built decidable models of T that omit certain collections of non-principal types of a given theory. Recently, a series of papers have been published that investigate degree-theoretic aspects of the results mentioned [2,7,13]. For instance, Hirschfeldt [9] proves that if all types of T are decidable then the prime model of T has a copy decidable in any given non-recursive degree.

In this paper we obtain two results using amalgamation classes and Fraïssé limits. In the first part of this paper we construct a decidable theory T all of whose types are decidable yet whose prime model is not decidable. We note that Millar [15] constructed such an example but his example uses an infinite language in an essential way. Our example uses just one binary predicate symbol, that is, the models we construct will be graphs. Our construction uses Fraïssé limits that code a family of sets of natural numbers. We note that such a theory T must have the following properties: (1) T has a saturated model since T has countably many types [21], (2) T has a prime model since T has a saturated model [21], (3) T does not have a decidable saturated model as otherwise the prime model would be decidable [5], (4) The family of all principal types of T and the family of all types of T both are not uniformly computable as mentioned above, (5) The prime model is decidable in any non-recursive degree.

In the second part of the paper, for any finite lattice $\mathcal{F}$ we construct a complete theory T with countably many models such that the fundamental order determined by T is isomorphic to $\mathcal{F}$. In this case we say that T *realises* $\mathcal{F}$. Now we explain the fundamental order associated with T. By a *type* p of the theory T we mean a maximal consistent with T set of formulas in at most n variables, where n is fixed. We use the notation $S(T)$ for the collection of all types of T. For a model $\mathcal{A}$ of T, the set of all types realised in $\mathcal{A}$ is called the *type spectrum of* T

and is denoted by $TySp(\mathcal{A})$ as in [16]. We say that type q *is weaker than* type p, written $q \preceq_T p$, if any model of T realising p must also realise q. In this case we also say that p *is stronger than* q. If $p \preceq_T q$ and $q \preceq_T p$ then we say that p and q *are equivalent* and denote this by $p \sim q$. We denote the equivalence class of type p by $[p]$. It is easy to see that $\preceq_T$ defines an order on the quotient set $S(T)/\sim$. We call this order, following Lascar–Poizat [14] and Baldwin–Berman [1], *the fundamental order* of the theory. Clearly, the equivalence class of a principal type is the least element in the fundamental order. Sudoplatov in [22] studies fundamental orders of theories through syntactic conditions. In this sense, his approach is somewhat orthogonal to ours as we employ algebraic tools such as amalgamation classes and Fraïssé limits.

Baldwin and Berman [1] investigate the fundamental order on 1-types over models rather then "pure" types (types over the empty set) as considered by us in this paper. They show that lattices can be realised as fundamental orders. In [20] Poizat proves that the fundamental order need not be a lattice and that the fundamental order over models must be lower semimodular. It is not clear, however, whether our results can be derived from the work of Baldwin, Berman, and Poizat. In addition, our results are independent, self-contained and use quite different machinery as opposed to the proofs used in [1] and [20].

If the theory T has countably many types, then we can recast the fundamental order $\preceq_T$ as follows. On the class of all countable models $\mathrm{Mod}(T)$ of T consider the elementary embedding relation $\preceq$. This is a pre-partial order on $\mathrm{Mod}(T)$. Say that two models $\mathcal{A}$ and $\mathcal{B}$ of T are $\approx$-equivalent if $\mathcal{A} \preceq \mathcal{B}$ and $\mathcal{B} \preceq \mathcal{A}$. The $\approx$-equivalence class of model $\mathcal{A}$ is denoted by $[\mathcal{A}]$. Thus, $\preceq$ determines a partial order on the quotient set $\mathrm{Mod}(T)/\approx$. It turns out that the partial order $(\mathrm{Mod}(T)/\approx;\ \preceq)$ is isomorphic to the fundamental order $(S(T)/\sim;\ \preceq_T)$. The isomorphism is explicitly defined as follows. Let p be a type of T. There exists a model $\mathcal{B}$ such that $TySp(\mathcal{B}) = \{q \mid q \preceq_T p\}$. In this case we say that $\mathcal{B}$ *strictly realises* p. Set $\mathcal{K}(p) = \{\mathcal{B} \mid \mathcal{B}$ strictly realises $p\}$. The mapping $[p] \to \mathcal{K}(p)$ establishes the desired isomorphism from $(S(T)/\sim;\ \preceq_T)$ to $(\mathrm{Mod}(T)/\approx;\ \preceq)$.

Having the fundamental order $(S(T)/\sim;\ \preceq_T)$, one can now refine many questions about computable and decidable models of the theory T. In particular, one can ask if there is decidable or computable model in the class $\mathcal{K}(p)$ for a given type p. As corollaries of the second result of this paper we obtained the following: (1) For any finite lattice $\mathcal{F}$ and any $p \in \mathcal{F}$, there exists a decidable theory T such that T realises $\mathcal{F}$, T has countably many models, and all decidable models of T belong to the classes $\mathcal{K}(q)$, where $q \preceq_T p$. (2) For any finite lattice $\mathcal{F}$, there exists a theory T such that T realises $\mathcal{F}$, T has countably many models, and for all $p \in \mathcal{F}$ each class $\mathcal{K}(p)$ contains exactly one computable model.

2. A Theory Whose All Types Are Decidable but Which Has No Decidable Prime Model

We construct a theory T whose all types are decidable which has no decidable prime model. The language of the theory consists of one binary predicate symbol. Our construction uses Fraïssé limits that code a family of sets of natural numbers.

The preorder $\leqslant_{\mathcal{S}}$. Let $\mathcal{S} = \{A_0, A_1, \ldots\}$ be a family of finite sets of natural numbers. On the set $2^{<\omega}$ of finite sets, using $\mathcal{S}$, we introduce the following preorder $\leqslant_{\mathcal{S}}$. Say that a set $X \in 2^{<\omega}$ is $\mathcal{S}$-less than or equal to $Y \in 2^{<\omega}$, written $X \leqslant_{\mathcal{S}} Y$, if there exists an n such that for all $i \geqslant n$ we have $Y \subseteq A_i$ implies $X \subseteq A_i$. The set $(\mathcal{D}, \leqslant_{\mathcal{S}})$ is a preordered set for all $\mathcal{D} \subseteq 2^{<\omega}$. Note that $\leqslant_{\mathcal{S}}$ does not depend on the enumeration of $\mathcal{S}$. There is an equivalence relation $\sim_{\mathcal{S}}$ associated with $\leqslant_{\mathcal{S}}$: $X \sim_{\mathcal{S}} Y$ if and only if $X \leqslant_{\mathcal{S}} Y$ and $Y \leqslant_{\mathcal{S}} X$. Denote the equivalence class of X by $[X]$. Note that for all $X, Y \in 2^{<\omega}$ for which there are at most finitely many $A \in \mathcal{S}$ containing X or Y, we have $X \sim_{\mathcal{S}} Y$. By $\mathbf{1}$ we denote the $\sim_{\mathcal{S}}$-equivalence class of $X \in 2^{<\omega}$ for which there are at most finitely many $A \in \mathcal{S}$ containing X. Clearly, $\mathbf{1}$ is the greatest element in $\leqslant_{\mathcal{S}}$ preorder if it exists. Thus, interesting cases arise when we restrict ourselves to those $X \in 2^{\omega}$ that appear in infinitely many $A \in \mathcal{S}$.

As an example, consider the infinite binary tree T. With this tree associate a family $\mathcal{S} = \{B_x \mid x \in T,\ B_x \subset 2^{<\omega}\}$ with the following properties:

1. For every node $x \in T$, the set B_x is of the form $A_x \cup \{n_x\} \in \mathcal{S}$.
2. The mapping $x \to A_x$ is an injection.
3. The mapping $x \to n_x$ is an injection.
4. $(\cup_x A_x) \cap \{n_x \mid x \in T\} = \emptyset$.
5. For all $x, y \in T$, x is a prefix of y if and only if $A_x \subseteq A_y$.

For all $x, y \in T$ we have $A_x \cup \{n_x\} \sim_{\mathcal{S}} A_y \cup \{n_y\}$. Also, we have $A_x \leqslant_{\mathcal{S}} A_y$ if and only if $x \leqslant y$ in the tree T. Moreover, if $x \neq y$ then $A_x \not\sim_{\mathcal{S}} A_y$.

The following proposition shows that $\leqslant_{\mathcal{S}}$ is quite a general construction in the sense that all countable preorders can be represented through appropriate $\leqslant_{\mathcal{S}}$ and $\mathcal{D} \subseteq 2^{<\omega}$. We leave the proof of the proposition out as well as most of other proofs due to the space limit.

Proposition 1. *Let $(A, \leqslant)$ be an at most countable preordered set. There exist a family of finite sets $\mathcal{S} = \{A_0, A_1, \ldots\}$ and $\mathcal{D} \subseteq 2^{<\omega}$ such that the preordered set $(A, \leqslant)$ is isomorphic to $(\mathcal{D}, \leqslant_{\mathcal{S}})$.*

A chain is an infinite sequence $A_0 \leqslant_{\mathcal{S}} A_1 \leqslant_{\mathcal{S}} \ldots$ of finite subsets of ω.

Definition 1. We say that a subset $L \subseteq \omega$ is an *$\mathcal{S}$-limit point* if there exists a chain $A_0 \leqslant_{\mathcal{S}} A_1 \leqslant_{\mathcal{S}} \ldots$ such that each $[A_i] <_{\mathcal{S}} [\mathbf{1}]$ and L is the union $\cup_i L_i$, where each L_i is the union of all sets in the $\sim_{\mathcal{S}}$-equivalence class $[A_i]$.

In the example above each path η through the tree T defines the following limit point $L_\eta = \cup_{x \in \eta} A_x$. Note that if L is a limit point and X is a finite subset of L then X appears in infinitely many $A \in \mathcal{S}$. If $\mathcal{S}$ is clear from the context we use the term *limit point* instead of $\mathcal{S}$-limit point.

Let $\mathcal{S}$ be a family of finite sets. We shall be coding families $\mathcal{S}$ into prime models of theories. In order to capture non-prime models of those theories, we single out certain families extending $\mathcal{S}$ in the next definition.

Definition 2. A family $\mathcal{S}'$ is called a *non-principal extension* of $\mathcal{S}$ if $\mathcal{S} \subset \mathcal{S}'$ and every $B \in \mathcal{S}' \setminus \mathcal{S}$ is an infinite $\mathcal{S}$-limit point. If all infinite limit points of $\mathcal{S}$ are in $\mathcal{S}'$ then we call $\mathcal{S}'$ a *saturated non-principal extension* of $\mathcal{S}$.

Proposition 2. *There exists a family $\mathcal{S}$ of finite sets and its non-principal extension $\mathcal{S}'$ such that the following properties hold:*

1. *All limit points of $\mathcal{S}$ are infinite.*
2. *$\mathcal{S}'$ is a saturated non-principal extension of $\mathcal{S}$.*
3. *$\mathcal{S}'$ is uniformly computable.*
4. *$\mathcal{S}$ is not uniformly computably enumerable.*

The theory $T_{\mathcal{S}}$. Let $\mathcal{S}$ be a family of subsets of ω. Let $B_0, B_1, \ldots$ be a computably enumerable enumeration of $\mathcal{S}$. We define a complete theory $T_{\mathcal{S}}$ based on the family $\mathcal{S}$. The theory will be defined through a Fraïssé limit type of construction. The idea of the construction follows [11].

We briefly recall the construction of Fraïssé limits. Let $\mathcal{K}$ be an infinite class of finite structures (over a finite relational language) closed under isomorphisms. Assume that the class $\mathcal{K}$ has the following properties:

1. Hereditary property (*HP*): for all $A \in \mathcal{K}$, if B is a substructure of A then $B \in \mathcal{K}$.
2. Joint embedding property (*JEP*): for all $A, B \in \mathcal{K}$ there exists a $C \in \mathcal{K}$, such that A and B can be embedded into C.
3. Amalgamation property (*AP*): for all $A, B, C \in \mathcal{K}$, if $f\colon A \to C$ and $g\colon A \to B$ are embeddings then there exists a structure $D \in \mathcal{K}$ and embeddings $h\colon B \to D$ and $k\colon C \to D$ such that $kf = hg$ on A.

Recall that a structure is *ultrahomogeneous* if any finite partial isomorphism of the structure into itself can be extended to an automorphism of the structure. The *age* of a structure D is the class of all finite structures embeddable in D. The following is a well known result in model theory connecting ultrahomogeneous structures with classes that possess *HP*, *JEP* and *AP* (see, for instance, [10]):

Theorem. *For any infinite class $\mathcal{K}$ that has HP, JEP and AP there exists a unique at most countable ultrahomogeneous structure* $\lim(\mathcal{K})$ *whose age coincides with $\mathcal{K}$. Moreover, the structure* $\lim(\mathcal{K})$ *is $\aleph_0$-categorical.*

The structure $\lim(\mathcal{K})$ is called the *Fraïssé limit* of the class $\mathcal{K}$. We shall use this theorem in our construction below.

For our purpose we now define special classes of finite structures that have properties *HP*, *JEP*, and *AP*. A *cycle* of length $n \geqslant 3$ is the graph $C_n = (\{1, \ldots, n\}, E)$ with $E = \{(1,2), (2,1), (2,3), (3,2), \ldots, (n-1,n), (n,n-1), (n,1), (1,n)\}$. A graph G *contains* a cycle of length n if there is an embedding from C_n into G.

For $Y \subset \omega$, $0, 1, 2 \notin Y$, $Y \neq \emptyset$, consider the following class of finite directed graphs: $\mathcal{K}(Y)$=$\{(V, E) \mid$ If (V, E) contains a cycle of length n then $n \in Y\}$. The following is an easy lemma:

Lemma 1. *The class $\mathcal{K}(Y)$ possesses properties HP, JEP and AP.*

We construct the following a structure $\mathcal{A}_{\mathcal{S}}$ based on the uniformly computable sequence $B_0, B_1, \ldots$ of the family $\mathcal{S}$ given above. For each $B_x \in \mathcal{S}$ consider the limit structure $\lim \mathcal{K}(B_x)$. One can uniformly construct a sequence

$$\lim \mathcal{K}(B_0),\ \lim \mathcal{K}(B_1),\ \lim \mathcal{K}(B_2), \ldots$$

of these structures such that (1) the graphs in this sequence are all pairwise disjoint; (2) the union of domains of these graphs is ω; and (3) the sequence is uniformly computable meaning that the set $\{(n, m) \mid m \in \lim \mathcal{K}(B_n)\}$ is computable.

Here is now a description of the structure $\mathcal{A}_{\mathcal{S}}$. The signature of $\mathcal{A}_{\mathcal{S}}$ has one binary relation E. The domain of the structure is ω. The relation E is the union of all edges of graphs that appear in the sequence above. Clearly, E is a computable edge relation. Thus, the structure $\mathcal{A}_{\mathcal{S}}$ is a computable graph.

Define $T_{\mathcal{S}}$ to be the first order theory of the structure $\mathcal{A}_{\mathcal{S}}$ built above. The next lemma connects decidability of the theory $T_{\mathcal{S}}$ with a uniformity condition put on the family $\mathcal{S}$:

Lemma 2. *The structure $\mathcal{A}_{\mathcal{S}}$ is decidable if and only if the $\mathcal{S} \cup \{\cup \mathcal{S}\}$ is uniformly computable.*

Using the lemmas above, we can prove the following theorem.

Theorem 1. *There exists a decidable theory T in the language of one binary predicate such that all types of T are all decidable but both prime and saturated models of T are not.*

3. Finite Lattices and Fundamental Orders

The main theorem of this section is to show that every finite lattice can be realised as the fundamental order of a theory.

An amalgamation class. We consider a class $\mathcal{K}$ of finite structures of the signature σ that has one binary predicate $\leqslant$, a ternary predicate R, and a finite set of constants $\{f_0, f_1, \ldots, f_n\}$. The axioms for the class $\mathcal{K}$ are given below. For the axioms we fix a finite lattice $\mathcal{F}$.

1) The relation $\leqslant$ orders the set of constants $\{f_0, \ldots, f_n\}$ such that the resulting partially ordered set $(\{f_0, \ldots, f_n\}, \leqslant)$ is isomorphic to $\mathcal{F}$. We denote this set of constants by F. We always assume that f_0 is the least element and f_n is the greatest element with respect to $\leqslant$.
2) The relation $\leqslant$ is false in any pair of elements x, y such that $x, y \notin F$. By Tr, we denote the set $\{x \mid x \not\leqslant x\}$. We also declare that $Tr \neq \emptyset$. It is clear that Tr and F have no elements in common.
   For the next set of axioms, we write $x \leqslant_f y$ instead of $R(f, x, y)$.
3) For each $f \in F$ we have the axiom: $x \leqslant_f y \to (f \in F \ \&\ x, y \in Tr)$.
4) For each $f \in F$, the relation $\leqslant_f$ orders Tr as a tree.
5) (**Downwards monotonicity axiom**): $(x \leqslant_f y \ \&\ g \leqslant f) \to x \leqslant_g y$.

6) (**Uplifting axiom**): For all $f, g, h \in F$ such that h is the least upper bound of f and g we have the axiom: $(x \leqslant_f y \ \& \ x \leqslant_g y) \to x \leqslant_h y$. This axiom implies that the intersection of $\leqslant_f$ and $\leqslant_g$ equals $\leqslant_h$.

There are trivial finite models that satisfy the axioms. For instance, on the set Tr we can always let the tree order $\leqslant_f$ be the same for all $f \in F$. The next proposition shows that there are non-trivial finite models satisfying the axioms.

Proposition 3. *For any finite lattice $\mathcal{F}$ there exists a finite structure $\mathcal{A}$ of signature σ such that (1) $\mathcal{A}$ is a model of the axioms, and (2) the set $\{\leqslant_f | \ f \in F\}$ under $\supseteq$ is isomorphic to F.*

For a structure $\mathcal{A}$ from $\mathcal{K}$, let $Tr(A)$ denote the tree part of its domain and $F(A)$ its constants part. So we have $Tr(A) = A \setminus F(A)$ and $A = Tr(A) \cup F(A)$. Note that if $\mathcal{A}$ is a model of the axioms, then for all $x, y \in Tr$ either x and y are not $\leqslant_g$-comparable for each $g \in F$, or there exists a unique $f \in F$ such that for all $g \in F$ we have x and y are $\leqslant_g$-comparable if and only if $g \leqslant f$.

Lemma 3. *The class $\mathcal{K}$ possesses properties HP, JEP and AP.*

Let $\mathcal{A}'$ be the Fraïssé limit of the class $\mathcal{K}$. Since $\mathcal{K}$ is the class of relational structures of finite signature, we have the following:

Proposition 4. *The model $\mathcal{A}'$ is ultrahomogeneous and $\aleph_0$-categorical.*

The model $\mathcal{A}$. Consider the model $\mathcal{A}'$ built above. Extend the structure $\mathcal{A}'$ by adding a new countable set S of elements to the domain of $\mathcal{A}'$; so, $A' \cap S = \emptyset$. We also add a new relation symbol p to the signature of $\mathcal{A}'$. The interpretation of p will be a partial function $F \times S \to Tr(\mathcal{A}')$. We shall write $p_f(s)$ instead of $p(f, s)$. After introducing some notations, we shall give a precise definition of $p_f(s)$ for $f \in F$ and $s \in S$. This extended structure is denoted by $\mathcal{A}$.

We now define a sequence $\Phi_1, \Phi_2, \ldots$ of non-principal filters in $Tr(\mathcal{A}')$. Initially, we set $\Phi_0 = Tr(\mathcal{A}')$. Each Φ_{i+1} is chosen to be a non-principal $\leqslant_{f_n}$-filter of Φ_i for all $i \in \omega$. Recall that f_n is the greatest element in the lattice $\mathcal{F}$. Note that all filters are pairwise isomorphic since A' is ultrahomogeneous.

For all $i \in \omega$, set $\Delta_i = \Phi_i \setminus \Phi_{i+1}$. For each $f \in F$, we also define

$$\Delta_{0,f} = \{x \mid (\exists y, z \in \Delta_0) \ y <_f x <_f z\},$$
$$\Delta_{i,f} = \{x \mid (\exists y, z \in \Delta_i) \ y <_f x <_f z\} \setminus \bigcup_{j<i} \Delta_{j,f} \ \text{(for all } i > 0).$$

It is clear that $\Delta_i = \Delta_{i,f_n}$ for each i, and for all $f \in F$ the sequence $\Delta_{0,f}, \Delta_{1,f}, \ldots$ consists of pairwise disjoint subsets. One can choose the sequence $\Phi_1, \Phi_2, \ldots$ of the filters so that for each f from F we have $Tr(\mathcal{A}') = \cup_{i \in \omega} \Delta_{i,f}$.

Let $\{S_f\}_{f \in F}$ be a partition of S into infinite subsets. We now define the function $p_f : S_f \to Tr(\mathcal{A}')$ such that (1) the function $p_f : S_f \to Tr(\mathcal{A}')$ is onto, and (2) for each $x \in \Delta_{i,f}$ the set $p_f^{-1}(x)$ has cardinality $i + 1$. Denote the new structure thus obtained by $\mathcal{A}$. This is our desired structure.

Let T be the elementary theory $Th(\mathcal{A})$ of $\mathcal{A}$. Our goal is to show that the theory T is the desired one. We introduce a notation in order to simplify our exposition. The intuition comes from the known example $(Q, \leqslant, c_0, c_1, \ldots)$ of the order of rationals where the constants form a strictly increasing chain. The theory of this structure has 3 models (See, for instance, [10]). For elements $f \in F$, write $c_i \leqslant_f x$ to indicate that $|p_f^{-1}(x)| \geqslant i+1$. Similarly, write $c_i \not\leqslant_f x$ for $|p_f^{-1}(x)| < i+1$; and write $x \leqslant_f c_i$ for $(\exists y \geqslant_f x)\ |p_f^{-1}(y)| = i+1$. Note each c_i neither is a constant nor an element of our structure.

Characterisations of models of T. Let $\mathcal{B}$ be a countable model of T. For $f \in F$, set $B_f = \{x \mid c_i \leqslant_f x \text{ for all } i\}$. Note that if $B_f \neq \emptyset$, then B_f is a $\leqslant_f$-filter but not necessarily a non-principal filter. If B_f is a principal filter and z is its least element, then we call such elements *B_f-least.*

Lemma 4. *If a is a B_f-least and b is a B_g-least element, then $a = b$.*

For the model $\mathcal{B}$, and $f \in F$, define the ordinal α_f as follows:

- If B_f is principal then $\alpha_f = \omega + \omega$.
- If B_f is non-principal, and there are a and g such that a is the B_g-least element, $c_k \leqslant_f a$, and $c_{k+1} \not\leqslant_f a$, then $\alpha_f = \omega + k$.
- If B_f is non-principal and there is no g such that B_g is principal, then $\alpha_f = \omega$.
- If $B_f = \emptyset$ and there are a and g such that a is the B_g-least element, $c_k \leqslant_f a$, and $c_{k+1} \not\leqslant_f a$, then $\alpha_f = k$.
- If $B_f = \emptyset$ and there is no g such that B_g is principal, then $\alpha_f = 0$.

Definition 3. We call the function $\{(f, \alpha_f) \mid f \in F\}$ the *characteristic* of $\mathcal{B}$ and denote it by $ch(\mathcal{B})$.

The next lemma tells us what functions from F into $\omega + \omega + 1$ can be realised as characteristics of models of T.

Lemma 5. *A function $\alpha : F \to \omega + \omega + 1$ is the characteristic of a model of T if and only if it satisfies the following conditions:*

1. *The function α is decreasing, that is, for all $f, g \in F$, $f \leqslant g$ implies $\alpha_g \leqslant \alpha_f$.*
2. *For all $f, g \in F$, if $f \leqslant g$, $\alpha_g = k$, and $\alpha_f = \omega + l$, then $k \leqslant l$.*
3. *If $h = lub\{f, g\}$ then*
$$\alpha_h = \begin{cases} \min\{k, l\}, & \text{if } \{\alpha_f, \alpha_g\} = \{k, \omega + l\}, \\ \min\{\alpha_f, \alpha_g\}, & \text{otherwise.} \end{cases}$$
4. *If α_f is a successor ordinal then there is g such that $\alpha_g = \omega + \omega$.*

Lemma 6. *Any two countable models of T are isomorphic if and only if the models possess the same characteristic.*

As an immediate consequence of Lemmas 5 and 6 we have the following:

Corollary 1. *The theory T has countably many countable models.*

Let $\mathcal{A}$ and $\mathcal{B}$ be models of T. Define $g_1^{\mathcal{A}}$ to be the least upper bound of the set $\{f \in A \mid \alpha_f \geqslant \omega\}$ as in the proof of Lemma 5. Define similarly $g_1^{\mathcal{B}}$ in the model $\mathcal{B}$. The following lemma is the description of the type spectra of models of T.

Lemma 7. *Two models $\mathcal{A}$ and $\mathcal{B}$ of T, $TySp(\mathcal{A}) = TySp(\mathcal{B})$ iff $g_1^{\mathcal{A}} = g_1^{\mathcal{B}}$.*

The next lemma is a direct corollary of Lemma 7.

Lemma 8. *The lattice $\mathcal{F}$ is isomorphic to the fundamental order of T without the least element.*

The proof of the theorem below follows from the lemmas above.

Theorem 2. *For each finite lattice $\mathcal{F}$, there exists a complete theory T whose fundamental order is isomorphic to $\mathcal{F}$. Moreover, T has countably many countable models and the language of T is finite.*

We note that the theorem above can be generalised to finite upper semi-lattices with the least element. However, the proofs are more involved and technically cumbersome.

4. Applications

We present two computability theoretic applications of Theorem 2. For these applications, recall the mapping $[p] \to \mathcal{K}(p)$ from the fundamental order into the class of subclasses of all models of T explained in the introduction.

Theorem 3. *For every finite lattice $\mathcal{F}$ and every element $p \in \mathcal{F}$, there exists a decidable theory T of finite signature with countably many models such that:*

1. *The fundamental order $(S(T)/\sim;\ \preceq_T)$ of T is isomorphic to $\mathcal{F}$.*
2. *The class $\mathcal{K}(q)$ has a decidable model if and only if $q \preceq_T p$.*

We note that this theorem provides a full description of all possible sub-lattices of the given lattice $\mathcal{F}$ that can be realised by decidable models. Indeed, those sub-lattices must be ideals in $\mathcal{F}$.

Theorem 4. *For every finite lattice $\mathcal{F}$, there exists a theory T of finite signature with countably many models such that:*

1. *The fundamental order $(S(T)/\sim;\ \preceq_T)$ of T without the least element is isomorphic to $\mathcal{F}$.*
2. *For all $p \in \mathcal{F}$ each class $\mathcal{K}(p)$ contains infinitely many models of which exactly one is computable.*

The proof of this theorem uses methods developed in [3] and [12].

References

1. Baldwin, J., Berman, J.: Concrete representations of lattices and the fundamental order. Classification Theory, 24–31 (1987)
2. Csima, B., Kalimullin, I.: Degree spectra and immunity properties. Mathematical Logic Quarterly 56(1), 67–77 (2010)
3. Gavruskin, A.: Computable limit models. In: Programs, Proofs, Processes—CiE, pp. 188–193 (2010)
4. Goncharov, S.: Strong constructivizability of homogeneous models. Algebra and Logic 17(4), 247–263 (1978)
5. Goncharov, S., Ershov, Y.: Constructive Models. Consultants Bureau, New York (2000)
6. Goncharov, S., Nurtazin, A.: Constructive models of complete solvable theories. Algebra and Logic 12(2), 67–77 (1973)
7. Greenberg, N., Montalbán, A., Slaman, T.: Relative to any non-hyperarithmetic set. preprint arXiv:1110.1907 (2011)
8. Harrington, L.: Recursively presentable prime models. The Journal of Symbolic Logic 39(2), 305–309 (1974)
9. Hirschfeldt, D.: Computable trees, prime models, and relative decidability. Proceedings of the American Mathematical Society 134(5), 1495–1498 (2006)
10. Hodges, W.: Model Theory. In: Encyclopaedia of Mathematics and Its Applications, vol. 42, Cambridge University Press (1993)
11. Khoussainov, B., Semukhin, P., Stephan, F.: Applications of Kolmogorov complexity to computable model theory. The Journal of Symbolic Logic 72(3), 1041–1054 (2007)
12. Khoussainov, B., Nies, A., Shore, R.: Computable models of theories with few models. Notre Dame Journal of Formal Logic 38(2), 165–178 (1997)
13. Lange, K.: The degree spectra of homogeneous models. Journal of Symbolic Logic 73(3), 1009–1028 (2008)
14. Lascar, D., Poizat, B.: An introduction to forking. The Journal of Symbolic Logic 44(3), 330–350 (1979)
15. Millar, T.: Foundations of recursive model theory. Annals of Mathematical Logic 13, 45–72 (1978)
16. Millar, T.: Omitting types, type spectrums, and decidability. Journal of Symbolic Logic 48(1), 171–181 (1983)
17. Morley, M.: Decidable Models. Israel Journal of Mathematics 25, 233–240 (1976)
18. Peretyatkin, M.: On complete theories with a finite number of denumerable models. Algebra and Logic 12(5), 310–326 (1973)
19. Peretyatkin, M.: Criterion for strong constructivizability of a homogeneous model. Algebra and Logic 17(4), 290–301 (1978)
20. Poizat, B.: Attention a la Marche! Journal of Symbolic Logic 51(3), 570–585 (1986)
21. Sacks, G.: Saturated model theory, 2nd edn. World Scientific Publishing Company Incorporated (2010)
22. Sudoplatov, S.: Complete theories with finitely many countable models I, II. Algebra and Logic 43(1), 62–69 (2004)

Discovering Hidden Repetitions in Words*

Paweł Gawrychowski[1], Florin Manea[2], and Dirk Nowotka[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
gawry@cs.uni.wroc.pl
[2] Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Kiel, Germany
{flm,dn}@informatik.uni-kiel.de

Abstract. Pseudo-repetitions are a natural generalization of the classical notion of repetitions in sequences: they are the repeated concatenation of a word and its encoding under a certain morphism or antimorphism. We approach the problem of deciding whether there exists an anti-/morphism for which a word is a pseudo-repetition. In other words, we try to discover whether a word has a hidden repetitive structure. We show that some variants of this problem are efficiently solvable, while some others are **NP**-complete.

1 Introduction

A word is a *repetition* if it equals a repeated concatenation of one of its prefixes. A word w is a *pseudo-repetition* if it equals a repeated concatenation of one of its prefixes t and its image $f(t)$ under some morphism or antimorphism (for short "anti-/morphism") f, thus $w \in t\{t, f(t)\}^+$.

These generalised repetitions (introduced in [1]) draw their motivations from computational biology and natural computing, namely the facts that the Watson-Crick complement can be formalised as an antimorphic involution and both a single-stranded DNA and its complement (that is, its image under such an involution) basically encode the same information. However, pseudo-repetitions occur in some other real life situations. For instance, in music: repetitions of some fragment, in its initial form but also slightly modified, are used to provide unity to a musical piece. Moreover, the concept of *ternary (song) form* is also used: three consecutive musical fragments such that the first and third ones are identical, while the second one is constructed in order to provide a contrast to the other two (which can be formalised sometimes by seeing it as the image of the other parts under some anti-/morphism); this musical concept appears in even more general forms, involving pseudo-repetitions of more than three factors. Besides the motivation coming from natural computing or musical theory, pseudo-repetitions seem to be of intrinsic theoretical interest, as they generalise a combinatorics on words concept that is central both in theory and applications.

The results obtained so far on pseudo-repetitions were both of combinatorial [1–3] and of algorithmic [4] nature. We continue here the study of algorithmic

* P. Gawrychowski is supported by the *NCN* grant 2011/01/D/ST6/07164, F. Manea by the *DFG* grant 596676, D. Nowotka by the *DFG Heisenberg* grant 590179.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 210–219, 2013.

problems related to pseudo-repetitions. More precisely, we study how efficiently we can discover whether an input word has a hidden repetitive structure, i.e, find an anti-/morphism f for which that word becomes an f-repetition. The problem seems natural to us: basically, we look at a text and want to find an encoded repetitive structures in it, without actually knowing the encoding scheme.

Based on both combinatorial and data-structure results, our approach is aimed to provide a better understanding of the concept of pseudo-repetition itself as well as to enrich a set of algorithmic tools that can be actually used in its originating fields, computational biology and natural computing.

Some Basic Concepts. For more detailed definitions we refer to [5].

Let V be a finite alphabet. We denote by V^* the set of all words over V and by V^k the set of all words of length k. The *length* of a word $w \in V^*$ is denoted by $|w|$. The *empty word* is denoted by λ. Moreover, we denote by $\mathrm{alph}(w)$ the alphabet of all letters that occur in w. In the problems discussed in this paper we are given as input a word w of length n and we assume that the letters of w are in fact integers from $\{1, \ldots, n\}$ and w is seen as a sequence of integers. This is a common assumption in algorithmic on words (cf., e.g., [6]).

A word u is a *factor* of a word v if $v = xuy$, for some x, y; also, u is a *prefix* of v if $x = \lambda$ and a *suffix* of v if $y = \lambda$. We denote by $w[i]$ the symbol at position i in w and by $w[i..j]$ the factor $w[i]w[i+1]\ldots w[j]$ of w starting at position i and ending at position j. For simplicity, we assume that $w[i..j] = \lambda$ if $i > j$. A word u occurs in w at position i if u is a prefix of $w[i..|w|]$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. If w cannot be expressed as a power of another word, then w is *primitive*. If $w = u^n$ with $n \geq 2$ and u primitive, then u is called the primitive root of w. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$. By $\mathrm{per}(w)$ we denote the smallest period of w.

The following well-known result is useful in our investigation:

Theorem 1 (Fine and Wilf [7]). *Let u and v be in V^*, and $d = \gcd(|u|, |v|)$. If two words $\alpha \in u\{u, v\}^+$ and $\beta \in v\{u, v\}^+$ have a common prefix of length at least $|u| + |v| - d$, then u and v are powers of a common word of length d.*

A function $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for all $x, y \in V^*$; f is an antimorphism if $f(xy) = f(y)f(x)$ for all $x, y \in V^*$. Note that to define an anti-/morphism it is enough to give the definitions of $f(a)$, for all $a \in V$. We say that f is *uniform* if there exists a number k with $f(a) \in V^k$, for all $a \in V$; if $k = 1$ then f is called *literal*. If $f(a) = \lambda$ for some $a \in V$, then f is called *erasing*, otherwise *non-erasing*. The vector T_f of $|V|$ natural numbers with $T_f[a] = |f(a)|$ is called the length-type of the anti-/morphism f in the following. If $V = \{a_1, \ldots, a_n\}$, T is a vector of n natural numbers $T[a_1], \ldots, T[a_n]$, and $x = b_1 \cdots b_k$ with $b_i \in V$ for all i, we denote by $T(x) = \sum_{i \leq k} T[b_i]$, the length of the image of x under any anti-/morphism of length type T defined on V.

We say that a word w is an *f-repetition*, or, alternatively, an f-power, if w is in $t\{t, f(t)\}^+$, for some prefix t of w; for simplicity, if $w \in t\{t, f(t)\}^+$ then w is called an f-power of root t. If w is not an f-power, then w is *f-primitive*.

For example, the word *abcaab* is primitive from the classical point of view (i.e., **1**-primitive, where **1** is the identical morphism) as well as f-primitive, for the morphism f defined by $f(a) = b$, $f(b) = a$ and $f(c) = c$. However, when considering the morphism $f(a) = c$, $f(b) = a$ and $f(c) = b$, we get that *abcaab* is the concatenation of ab, $ca = f(ab)$, and ab, thus, being an f-repetition.

Finally, the computational model we use is the standard unit-cost RAM with logarithmic word size. Also, all logarithms appearing here are in base 2.

2 The Problem

In [4], an efficient solution for the problem of deciding, given a word w and an anti-/morphism f, whether w is an f-repetition was given. Here we approach a more challenging problem. Namely, we are interested in deciding whether there exists an anti-/morphism f for which a given word w is an f-repetition. Basically, we check whether a given word has an intrinsic (yet hidden) repetitive structure. Note that in the case approached in [4] the main difficulty was to find a prefix x of w such that $w \in x\{x, f(x)\}^*$. The case we discuss here seems more involved: not only we need to find two factors x and y such that $w \in x\{x, y\}^*$, i.e., a suitable decompositions of w, but we also have to decide the existence of an anti-/morphism f with $f(x) = y$. The problem is defined in the following.

Problem 1. Given $w \in V^+$, decide whether there exists an anti-/morphism $f : V^* \to V^*$ and a prefix t of w such that $w \in t\{t, f(t)\}^+$.

The unrestricted version of the problem is, however, trivial. We can always give a positive answer for input words of length greater than 2. It is enough to take the (non-erasing) anti-/morphism f that maps the first letter of w, namely $w[1]$, to $w[2..n]$, where $n = |w|$. Clearly, $w = w[1]f(w[1])$, so w is indeed an f-repetition. When the input word has length 1 or 0, the answer is negative.

On the other hand, when we add a series of simple restrictions to the initial statement, the problem becomes more interesting. The restrictions we define are of two types: either we restrict the desired form of f, and try to find anti-/morphisms of given length type, or we restrict the repetitive structure of w by requiring that it consists in at least three repeating factors or that the root of the pseudo-repetition has length at least 2.

In the first case, when the input consists both in the word w and the length type of the anti-/morphism we are trying to find, we obtain a series of polynomial time solutions for Problem 1. More precisely, in the most general case we can decide whether there exists an anti-/morphism f such that w is an f-repetition in $\mathcal{O}(n(\log n)^2)$ time. Note that deciding whether a word is an f-repetition when f is known took only $\mathcal{O}(n \log n)$ time [4]. When we search for an uniform morphism we solve the problem in optimal linear time time. This matches the complexity of deciding, for a given uniform anti-/morphism f, whether a given word is an f-repetition, obtained in [4]. This result covers also the case of literal anti-/morphism, extensively approached in the literature (cf., e.g., [1, 3]). Our solutions are based both on combinatorial results regarding the structure of pseudo-repetitions and on the usage of efficient data-structures.

For the second kind of restrictions, the length type of f is no longer given. In this case, we want to check, for instance, whether there exist a prefix t and an anti-/morphism f such that w is an f-repetition that consists in the concatenation of at least 3 factors t or $f(t)$. The most general case as well as the case when we add the supplementary restriction that f is non-erasing are **NP**-complete; the case when f is uniform (but of unknown length type) is tractable. The problem of checking whether there exists a prefix t, with $|t| \geq 2$, and a non-erasing anti-/morphism f such that $w \in t\{t, f(t)\}^+$ is also **NP**-complete; this problem becomes tractable for erasing or uniform anti-/morphisms.

3 Basic Tools

The following classical combinatorial results are used in this paper; for proofs and details, cf. the handbook [5, Chapter 9] and the references therein.

Lemma 1. *Let $w \in V^n$ be a word, $\mathrm{PS}_w = \{u | u$ primitive $, u^2$ prefix of $w\}$, and $\mathrm{PR}_w = \{u \mid u$ is primitive, u^2 is a factor of $w\}$.*
(1) Let $u_1, u_2, u_3 \in \mathrm{PS}_w$ be words such that $|u_1| < |u_2| < |u_3|$. Then $2|u_1| < |u_3|$. As a consequence, $|\mathrm{PS}_w| \leq 2 \log n$.
(2) We compute all the pairs (i, j) such that $w[i..j] = u^2$ for some $u \in \mathrm{PR}_w$ in $\mathcal{O}(n \log n)$ time. Moreover, $|\mathrm{PR}_w| \in \mathcal{O}(n)$.
(3) We compute the values $\mathrm{per}(w[1..i])$ for all $i \in \{1, \ldots, n\}$ in linear time $\mathcal{O}(n)$.

The next lemma was given in [4].

Lemma 2. *Let $u, v \in V^+$ and $w \in \{u, v\}^* \setminus \{u\}^*$ be words such that $|u| \leq |v|$ and u and v are not powers of the same word. Let $M = \max\{p \mid u^p$ is a prefix of $w\}$ and $N = \max\{p \mid u^p$ is a prefix of $v\}$. Then $M \geq N$. Also, if $M = N$ then $w \in v\{u, v\}^*$, while if $M > N$ then either it is the case that $w \in u^{M-N} v\{u, v\}^* \setminus u^{M-N-1} vuV^*$, or we have that $w \in u^{M-N-1} v\{u, v\}^+ \setminus u^{M-N} vV^*$ and $N > 0$.*

We also recall basic facts about the data structures we use. For a word u, with $|u| = n$, over $V \subseteq \{1, \ldots, n\}$ we can build in linear time a suffix array structure as well as data structures allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $u[i..n]$ and $u[j..n]$ of u, denoted $\mathrm{LCP}(i, j)$. These structures are called LCP data structures in the following. For details, cf., e.g., [6, 8]. The following results can be easily shown.

Remark 1. (1) A number p is a period of w if and only if $\mathrm{LCP}(1, p+1) = |w| - p$.
(2) Given two words $x, y \in V^*$, for which we have LCP-data structures, and an array T of $|V|$ integers, we can decide in $\mathcal{O}(\mathrm{per}(x))$ time the existence of an anti-/morphism f of length type T such that $f(x) = y$.
(3) Let w be a word. First, for $1 \leq i \leq n$, we construct the list of pairs (i, j) such that $w[i..j] = u^2$ with u primitive. By Lemma 1 this takes $\mathcal{O}(n \log n)$ time. Further, we put together all these lists and sort the resulting list according to the lexicographical order of the words encoded by the pairs ((i, j) encodes $w[i..j]$ and we just choose some order on the alphabet of w). This can be done in $\mathcal{O}(n(\log n)^2)$ time, using LCP queries to compare the words encoded by two

pairs. Next, we construct in $\mathcal{O}(n \log n)$ time the set PR_w, which has $\mathcal{O}(n)$ elements by Lemma 1. Each element u of PR_w is encoded by the pair (i, j) such that $w[i..j]$ is the first occurrence of u^2 in w. Moreover, while computing PR_w, we store for each pair (i', j') such that $w[i'..j'] = u^2$ for some $u \in \mathrm{PR}_w$ the actual pair (i, j) used to encode u in PR_w. ◁

4 Efficient Solutions: Known Length Type

Recall that we are given a word w, and want to check whether there exists an anti-/morphism f such that w is an f-repetition; assume that the length type T of f is also given as input. We only present the case of morphisms, as the case of antimorphisms is similar. The only difference is, in fact, the way the check in Remark 1.(2) is implemented.

Finding suitable decompositions. We begin by noting that if $w \in t\{t, f(t)\}^+$ for a prefix t and a morphism f, then there exists a primitive prefix t' of w such that $w \in t'\{t', f(t')\}$. Therefore, we algorithmically construct the set $S = \{(x, y) \mid w \in x\{x, y\}^+ \setminus \{x\}^+, x \text{ primitive}, |y| = T(x)\} \cup \{(x, \perp) \mid x \text{ primitive}, w \in \{x\}^+\}$. If this set contains the pair $(x, \perp)$ then the input word w is a repetition, and we can already give a positive answer to Problem 1, while if the pair (x, y) belongs to S then $w \in x\{x, y\}^*$ and there may be a morphism f of length type T such that $f(x) = y$; this morphism, however, remains to be found.

Basically, our algorithm works as follows. For each proper and primitive prefix $x = w[1..i]$ of the input word w we check whether $w \in \{x\}^+$. If this is not the case, we must find a factor y, whose length m equals $T(x)$, such that $w \in x\{x, y\}^+$; clearly such a factor y occurs at position j' in w such that $w[1..j'-1] \in \{x\}^+$. To find a factor y as above, we first compute the value j such that $w[1..j-1] \in \{x\}^+$ and x does not occur at position j in w. If $m < i = |x|$ then y may occur at any of the positions $j+1$ or $j-i+1$ (otherwise, we get from Theorem 1, that x is not primitive). Further, if $m \geq i$ and $i \mid m$ (which is, in fact, equivalent to $i \mid n$) then y may occur at any of the positions $j - it + 1$ for $0 \leq t \leq \frac{m}{i} - 1$. Finally, if $m > i$ and $i \nmid m$ then y may occur at any of the positions $j - it + 1$ for $0 \leq t \leq \lceil \frac{m}{i} \rceil$ (again, otherwise, we get that x is a proper factor of xx, so it is not primitive, a contradiction). In all cases, for the prefix x we identify at most $\lceil \frac{m}{i} \rceil + 1$ possibilities to choose the factor y. For each of these possibilities we obtain a pair (x, y) and we check whether $w \in x\{x, y\}^+$. This is done using an approach defined in [4] and essentially based on Lemma 2. More precisely, with every choice of y we also identify a prefix $w[1..s-1]$ that is in $\{x\}^+y$; we only have to check whether the suffix $w[s..n] \in \{x, y\}^*$. This can be done using Lemma 2. More precisely, for u being the shorter word of x and y and v being the longer one, we compute $M = \max\{p \mid u^p \text{ prefix of } w[s..n]\}$ and $N = \max\{p \mid u^p \text{ prefix of } v\}$. Further, $w[s..n]$ is either in $\{u\}^*$ and we set $s = n + 1$, or $u^{M-N}v$ occurs at position s, case in which we set $s = s + (M - N)|u| + |v|$ and repeat the analysis for the new suffix $w[s..n] \in \{x, y\}^*$, or, finally, $M > N$ and $u^{M-N-1}vu$ occurs at position s, case in which we set $s = s + (M - N - 1)|u| + |v|$ and, again,

repeat the argument for the new suffix $w[s+1..n]$. The algorithm adds (x, y) to S whenever $s = n+1$, and it needs $\mathcal{O}(\frac{n}{\max\{|x|,|y|\}})$ steps to check whether $w \in \{x, y\}^*$. The soundness of this approach follows from the explanations given above, and one can show that its time complexity is $\mathcal{O}(n \log n)$, provided that we construct and use LCP-data structures for w.

Known length type: finding the function. We present a solution for Problem 1 in the general case. Assume that the input of our problem consists in a word w and a list T of at most n numbers, giving the length of $f(a)$ for all the letters $a \in \text{alph}(w)$, in the order of their appearance in w. A naive solution of the problem runs, clearly, in $\mathcal{O}(n^2 \log n)$.

Intuitively, our more efficient approach is the following. We try to find in the set S a pair (x, y) such that x can be mapped to y by a morphism of length type T. However, trying each pair individually takes too much time. Therefore, S is split into several sets whose elements share common combinatorial properties and can be processed simultaneously. Each such set is then analysed separately in an efficient manner. The technical details are described in the following.

Initially, the word w is processed as in Remark 1.(3). Then, for each pair $v \in \text{PR}_w$ we construct an empty set $B(v)$; we keep track of the minimal element contained by each such set: when an element is inserted in it, the minimum is updated. We also compute inductively (and store) the values $T(w[1..i])$ for $1 \leq i \leq |w|$ in linear time. Finally, for each $j \leq n$ there exists at most one number i_j such that $w[1..j] = w[1..i_j]y$, with $|y| = T(w[1..i_j])$. As $i = i_j$ if and only if $j = i + T(w[1..i])$, computing these numbers takes linear time.

Next, we compute the set S, as described above. While computing this set we can already split it in two subsets $S_1 = \{(x, y) \mid w \in x^2\{x, y\}^*\}$ and $S_2 = \{(x, y) \mid w \in xy\{x, y\}^*\}$; clearly, $S_1 \cup S_2 = S$, $|S_1| \in \mathcal{O}(n \log n)$, and $|S_2| \in \mathcal{O}(n)$. Moreover, for each $(x, y) \in S_2$ we put i in the set $B(v)$ if $x = w[1..i]$ and v equals the primitive root of y. Using the preprocessing phase described above, this last step takes $\mathcal{O}(n)$ time in total. The rest of our solution consists in a separate analysis of the sets S_1 and S_2, checking whether one of them contains a pair (x, y) for which there exists a morphism f of length type T with $f(x) = y$.

We start with S_1. By Remark 1.(2), we check each pair (x, y) from this set in time $\mathcal{O}(\text{per}(x))$. Thus, the time needed to verify all the pairs in S_1 is upper bounded by $\mathcal{O}\left(\sum_{x \in \text{PS}_w} |x| \left(|f(x)|/|x| + 2\right)\right) \in \mathcal{O}(n \log n)$. Indeed, $|\text{PS}_w| \leq 2 \log n$ by Lemma 1 and for each x we have at most $\lceil |f(x)|/|x| \rceil + 1$ pairs $(x, y) \in S$ to check, and the previously announced upper bound follows.

We continue with the analysis of the set S_2. This case is more involved. We first split S_2 in two subsets S_3 and S_4. In S_3 we put all the pairs (x, y) with $\text{per}(x) > \frac{|x|}{2}$, while $S_4 = S_2 \setminus S_3$. Then we analyse S_3 and S_4 separately.

The analysis of S_3 can be implemented faster than checking its elements one by one. We partition S_3 into the sets $S_3^k = \{(x, y) \in S_3 \mid 2^k \leq |x| < 2^{k+1}\}$, for $0 \leq k \leq \lceil \log n \rceil$. As for each prefix x of w there is at most one pair (x, y) in S_3^k (and in S_3), we can store these sets so that checking whether a pair (x, y) is indeed in S_3^k is done in $\mathcal{O}(1)$ time for every k.

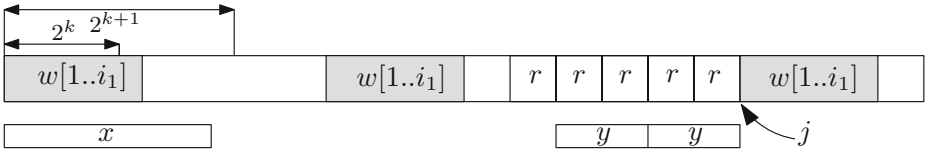


Fig. 1. The analysis of the set S_3^k

Let us now fix one k, and let $S_3^k = \{(w[1..i_1], y_1), \ldots, (w[1..i_s], y_s)\}$, where $i_\ell < i_{\ell+1}$ for $1 \leq \ell \leq s-1$. Clearly, x starts with $w[1..i_1]$, for all $(x, y) \in S_3^k$. Thus, in a decomposition of w in factors x and y, such that xy occurs as prefix of w, all the factors x appear on positions where $w[1..i_1]$ occurs in w. Accordingly, we identify the positions where $w[1..i_1]$ occurs in w using a linear time string matching algorithm. There are at most $\frac{n}{2^{k-1}}$ such positions, as $w[1..i_1] \geq 2^k$ and $\mathrm{per}(w[1..i_1]) > 2^{k-1}$; let $j_1, \ldots, j_p$ be these positions.

Since for each $1 \leq \ell \leq s$ the word w has a decomposition in factors $w[1..i_\ell]$ and y_ℓ, there exists $j'_\ell \in \{j_1, \ldots, j_p, n+1\}$ such that $w[1..j'_\ell - 1] \in w[1..i_\ell]\{y_\ell\}^+$ and $w[j_\ell..n] \in \{w[1..i_\ell], y_\ell\}^*$. Hence, there exist $(x, y) \in S_3^k$ and a morphism f of length type T such that $f(x) = y$ if and only if there exist $j \in \{j_1, \ldots, j_p, n+1\}$, $(x, y) \in S_3^k$, and a morphism f of length type T such that $w[1..j-1] \in x\{y\}^+$ and $f(x) = y$. We check whether there exists j fulfilling these conditions.

For each $j \in \{j_1, \ldots, j_p, n+1\}$ we run the following processing. We first decide in $\mathcal{O}(2^{k+1})$ time whether there exist two words x and y such that $2^k \leq |x| < 2^{k+1}$, $w[1..j-1] = xy$, and $|y| = T(x)$. If yes, by Remark 1.(2), we check in $\mathcal{O}(2^{k+1})$ time whether there exists f of length type T with $f(x) = y$. Moreover, if $(x, y) \in S$ then we found a solution of Problem 1. If no solution is found in this way, we further check whether there exist two words x and y such that $2^k \leq |x| < 2^{k+1}$, $w[1..j-1] \in xy\{y\}^+$, and $|y| = T(x)$. Let us assume that such a pair (x, y) exists. According to Lemma 1 there exists a set $S_j = \{r_1, \ldots, r_t\}$ of primitive words, with $t \leq 2 \log n$, such that r^2 is a suffix of $w[1..j]$ for all $r \in S_j$. As $w[1..j]$ ends with y^2, it follows that $y = r^\ell$ for some $r \in S_j$ and $\ell \geq 1$. Hence, for each $r \in S_j$, we proceed as follows. We go through the prefixes $w[1..i]$ of $w[1..2^{k+1}]$ and try to construct a morphism f of length type T that maps $w[1..i]$ into a prefix of a power of r. The image of these prefixes can be computed inductively: $f(w[1]) = r^{t_1} r'_1$, where r'_1 is a prefix of r and $t_1|r| + |r'| = T[w[1]]$, and, further, $f(w[i+1]) = r''_i r^{t_{i+1}} r'_{i+1}$, where $r'_i r''_i = r$, r'_{i+1} is a prefix of r, and $t_i|r| + |r'_{i+1}| + |r''_i| = T[w[i+1]]$. Clearly, the image of each letter $w[i]$ can be uniquely associated to the triple $(|r''_{i-1}|, t_i, |r'_i|)$. It is not hard to see that the process of computing these images fails whenever we associate to a letter two different images. On the other hand, each time we find a prefix $w[1..i]$ that can be mapped to r^q, for some $q > 0$, we check whether $|r|$ is a period of $w[i+1..j-1]$ and whether $q|r|$ divides $|j-i-1|$. If all these hold, and, also, $(w[1..i], r^q) \in S_3^k$, then Problem 1 can be answered positively. This concludes the analysis of S_3.

The time needed to analyse S_3 as above is $\mathcal{O}(n(\log n)^2)$.

If we did not find any solution in the set S_3, we continue with the analysis of S_4. Again, since for each prefix x of w there is at most one pair $(x, y) \in S_4$, we can store S_4 so that checking whether it contains such a pair takes $\mathcal{O}(1)$ time.

Note that if $(x, y) \in S_4$ then $\mathrm{per}(x) = d$ with $w[1..d]$ primitive and $(w[1..d])^2$ prefix of x. Thus, one can split S_4 into the sets $S_4^d = \{(x, y) \in S_4 \mid \mathrm{per}(x) = d\}$, for all d such that $w[1..d]$ is primitive and $(w[1..d])^2$ is a prefix of w. There are at most $2 \log n$ such sets and they partition S_4. We analyse each one separately.

We fix a value d such that $w[1..d]$ is primitive and $(w[1..d])^2$ is a prefix of w. It is not hard to see that there exist two numbers e_d and f_d such that $\mathrm{per}(w[1..i]) = d$ if and only if $e_d \leq i \leq f_d$. Moreover, if $d' < d$, then $f_{d'} < e_d$. As in the analysis of S_3^k, we run a linear time pattern matching algorithm to locate the positions where $w[1..d]$ occurs in w. Let $j_1, \ldots, j_s$ be these positions; note that s depends on d, but we omit writing this to keep the notation simpler. Since $w[1..d]$ is primitive, it occurs at most $\frac{n}{d}$ times in w, so $s \leq \frac{n}{d}$. Recall that for each $j \in \{j_1, \ldots, j_s\}$ there exists at most one value i_j such that $w[1..j-1] = w[1..i_j]y$ with $y = T(w[1..i])$; we already computed and stored these values and we can retrieve each of them in $\mathcal{O}(1)$ time. Now, for each j, if the value i_j is defined and $e_d \leq i_j \leq f_d$, we check in $\mathcal{O}(d)$ time whether there exists a morphism f of length type T such that $f(w[1..i_j]) = w[i_j + 1..j - 1]$. If yes and $(w[1..i_j], w[i_j + 1..j - 1]) \in S_4$ then the instance of Problem 1 defined by w and T has a positive answer.

Now, for each $j \in \{j_1, \ldots, j_s\}$ we check whether $w[1..j-1] \in xy\{y\}^+$ for some $(x, y) \in S_4^d$ such that there is a morphism f of length type T with $f(x) = y$. Let $S_j = \{r_1, \ldots, r_t\}$ be the set of primitive words whose squares are suffixes of $w[1..j-1]$. As y^2 is a suffix of $w[1..j-1]$ we get that $y \in \{r\}^+$ for some $r \in S_j$.

We first discuss the case when $y = r^p$ with $p > 1$. Clearly, y has a prefix v^2, where $|v| = T(w[1..d])$; $|v|$ is also a period of y. By Theorem 1, since $y = r^p$ with r primitive, it follows that $v = r^s$ for $s = \frac{T(w[1..d])}{|r|}$. Hence, we check whether r^{4s} occurs as a suffix of $w[1..j-1]$ (i.e., $w[1..j-1]$ ends with a long enough power of r, allowing us to find a suitable y) and whether there is a morphism f of length type T with $f(w[1..d]) = r^s$. This takes $\mathcal{O}(d)$ time, by Remark 1.(2). If there exists such an f, as well as an $i \in B(r)$ with $e_d \leq i \leq f_d$, Problem 1 can be answered positively. Indeed, we have a pair $(w[1..i], z) \in S$ where $\mathrm{per}(w[1..i]) = d$ and z a power of r. Thus, $w \in w[1..i]\{w[1..i], z\}^+$, $T(w[1..i]) = |z|$, and, as f exists, $w[1..d]$ can be mapped to the $z[1..m]$ for $m = T(w[1..d])$. It follows that $w[1..i]$ can be mapped to z, and this proves our point. If no i as above exists, then we must consider the case $y = r$ as well as another choices for d, j, and r. However, checking for each d, j, and r as above whether $B(r)$ contains an element i in the range $[e_d, f_d]$ is not efficient. It is better to do all these checks after we finished considering all cases and identified all the ranges we have to verify. Basically, for each primitively rooted square r^2 occurring in w we shall have $\mathcal{O}(\log n)$ range queries (at most one for each d), whose ranges do not overlap; checking all of them at once takes $\mathcal{O}(|B(r)|)$ time, by considering the elements of $B(r)$ one by one. So the time needed for all r's is $\mathcal{O}(n \log n)$, as the total number of elements of the sets $B(r)$ is less than $|S|$.

The case when $y = r$ is, however, simpler. We just have to see if the minimum i of $B(r)$ fulfils $e_d \leq i \leq f_d$, and check in $\mathcal{O}(d)$ time whether $w[1..i]$ can be mapped to r. If yes, we answer Problem 1 positively.

This analysis of the set S_4 takes, again, $\mathcal{O}(n(\log n)^2)$ time.

Now we can say whether there exists a pair $(x, y) \in S$ and a morphism f of length type T such that $f(x) = y$. Hence, we gave a solution for Problem 1, when the length type of f is known. This solution's time complexity is $\mathcal{O}(n(\log n)^2)$.

Final remarks. Problem 1 can be solved optimally, in $\mathcal{O}(n)$ time, when f is uniform. The result is based on several data-structures developed in [4] and a thorough analysis of the combinatorial properties of the elements of S. A summary of the results obtained in this section is the following.

Theorem 2. *Given a word w and a vector T of $|V|$ numbers, we decide whether there exists an anti-/morphism f of length type T such that $w \in t\{t, f(t)\}^+$ in $\mathcal{O}(n(\log n)^2)$ time. If T defines uniform anti-/morphisms we need $\mathcal{O}(n)$ time.*

5 Unknown Length Type

As already mentioned, the most general form of Problem 1 is trivial. Besides considering the cases when the length type of the function we search is given, there are two other natural ways to restrict Problem 1 in order to make it non-trivial for functions f of unknown length type. One variant is obtained by asking that the root t has at least two letters, and another one by asking that w is an f-repetition consisting of at least three factors. Some cases remain tractable. First, we can decide in linear time the existence of a general anti-/morphism f and of a prefix t of w such that $w \in t\{t, f(t)\}^+$ and $|t| \geq 2$. Second, both restricted variants Problem 1 are solvable in $\mathcal{O}(n^2)$ when f should be uniform (of unknown length type). Other cases are computationally hard, as shown next.

Recall first the pattern-description problem:

Problem 2. Given $x, y \in V^*$ decide the existence of a morphism g with $g(x) = y$.

This problem is **NP**-complete and it remains as hard for g non-erasing (cf. [9]).

We begin by considering the restriction requiring that the root of the pseudo-repetition has at least 2 letters. The only case left open is when the function we look for is non-erasing; otherwise the problem can be solved efficiently.

Problem 3. Given $w \in V^+$ decide the existence of a non-erasing anti-/morphism $f : V^* \to V^*$ and a prefix t of w with $|t| \geq 2$ such that $w \in t\{t, f(t)\}^+$

This problem is clearly in **NP**, either if the searched function f is a morphism or an anti-morphism. We show it is **NP**-complete by giving a polynomial-time reduction from Problem 2, the case when the morphism g from that problem is non-erasing. We only show here the **NP**-completeness of the variant where we search for a morphism f, as the case of searching antimorphisms is similar.

Assume that we have an input instance of the pattern-description problem, namely two words x and y, over an alphabet V, and want to decide whether

there exists a non-erasing morphism g such that $g(x) = y$. Let $w = a^n x b^n y$, where $n = 2\max\{|x|, |y|\}$ and $a, b \notin V$. We show there exists a non-erasing morphism g such that $g(x) = y$ if and only if there exist a non-erasing morphism $f : (V \cup \{a, b\})^* \to (V \cup \{a, b\})^*$ and a prefix t of w with $|t| \geq 2$ such that $w \in t\{t, f(t)\}^+$. The left to right implication is immediate. For the other implication, assuming first that $t = a^k$ we obtain that b must appear in $f(t)$ as, otherwise, w would not be in $t\{t, f(t)\}^*$. Further, since $k \geq 2$ we obtain a contradiction, as w should be a k-repetition, and it is not such a repetition. Therefore, $t = a^n x'$. We obtain that $f(t) = (f(a))^n f(x')$. But the only two factors of the form u^n of w are a^n and b^n, so $(f(a))^n = b^n$ and $f(a) = b$. Now, it follows that $x' = x$ and $f(x) = y$, so we can take $g = f$. This concludes the proof of this implication.

Hence, we exhibited a polynomial time reduction from Problem 2 to Problem 3. Thus, this problem is **NP**-complete.

Let us now overview the case when w is an f-repetition of at least three factors. In the most general variant, the problem asks to decide, for a given word $w \in V$, whether there exist an anti-/morphism $f : V^* \to V^*$ and a prefix t of w such that $w \in t\{t, f(t)\}\{t, f(t)\}^+$. This variant of the problem is **NP**-complete, just as its restriction to the case when the function f is non-erasing.

We summarise the main results of this section in the following theorem:

Theorem 3. *For a word $w \in V^+$, deciding the existence of an anti-/morphism $f : V^* \to V^*$ and a prefix t of w such that $w \in t\{t, f(t)\}^+$ with $|t| \geq 2$ (respectively, $w \in t\{t, f(t)\}\{t, f(t)\}^+$) is solvable in linear time (respectively,* **NP***-complete) in the general case, is* **NP***-complete for f non-erasing, and is solvable in $\mathcal{O}(n^2)$ time for f uniform.*

References

1. Czeizler, E., Kari, L., Seki, S.: On a special class of primitive words. Theoretical Computer Science 411, 617–630 (2010)
2. Manea, F., Mercaş, R., Nowotka, D.: Fine and Wilf's theorem and pseudo-repetitions. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 668–680. Springer, Heidelberg (2012)
3. Manea, F., Müller, M., Nowotka, D.: The avoidability of cubes under permutations. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 416–427. Springer, Heidelberg (2012)
4. Gawrychowski, P., Manea, F., Mercaş, R., Nowotka, D., Tiseanu, C.: Finding pseudo-repetitions. In: Proc. STACS (to appear, 2013)
5. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press (2007)
6. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM 53, 918–936 (2006)
7. Fine, N.J., Wilf, H.S.: Uniqueness theorem for periodic functions. Proceedings of the American Mathematical Society 16, 109–114 (1965)
8. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York (1997)
9. Ehrenfeucht, A., Rozenberg, G.: Finding a Homomorphism Between Two Words is NP-Complete. Inf. Process. Lett. 9(2), 86–88 (1979)

Language Forbidding-Enforcing Systems Defining DNA Codewords

Daniela Genova

Department of Mathematics and Statistics, University of North Florida,
Jacksonville, FL 32224, U.S.A.
d.genova@unf.edu

Abstract. DNA code word design is an interesting and important area of research in DNA computing and generating a large set of DNA strands that satisfy a given set of constraints is a difficult and important problem. On the other hand, forbidding and enforcing systems (fe-systems) are a molecularly inspired model of computation that defines structures based on constraints. This paper reinforces the connection between fe-systems and DNA codes by using the single language model of fe-systems to characterize a variety of DNA codes that avoid certain types of cross hybridizations. Some known methods of generating good DNA code words which have been tested experimentally are generalized by fe-systems. Finally, it is shown how the theoretical definitions by fe-systems can be used as a computational tool and also to model laboratory experiments.

1 Introduction

The study of DNA codes emerged from the attempt to design DNA strands for the purpose of using them to perform computation in such a way that mismatched pairings due to Watson-Crick complementarity are minimized. When a set of DNA molecules is used to perform computation, a variety of undesirable total or partial cross-hybridization between the molecules may occur during a polymerase chain reaction, self-assembly, or in the extraction step. The problem of unwanted hybridization has been studied extensively theoretically, algorithmically, and experimentally and numerous solutions have been proposed, e.g., [2,5,19]. Recently, the authors in [7] used DNA metric spaces to design a large set of DNA code words and showed that this problem is NP-complete.

The theoretical study of DNA codeword design begun by Kari et al. [13] spans a vast body of literature. In [15] the notion of θ-k-codes, capturing a variety of unwanted partial bindings (Fig. 1), was introduced. For a suitable k, a θ-k-code also avoids cross bindings disallowed by θ-comma-free, θ-strict, θ-intercode, θ-infix, θ-prefix, and θ-suffix codes. Maximal bond-free languages were studied in [17] and, most recently, a structural characterization of bond-free languages that leads to a polynomial time algorithm was provided [18].

Inspired by the non-deterministic nature of molecular reactions, forbidding-enforcing systems (fe-systems) were introduced by Rozenberg et al. in [3,4] as a model of computation that defines classes of languages. This variant of fe-systems

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 220–229, 2013.

Fig. 1. Various cross hybridizations of molecules avoided by a θ-k-code: one molecule contains subword of length k and the other its complement. In this figure $k = 6$.

uses one fe-system (two sets of constraints) to define a family of languages (fe-family). Properties of fe-systems defining fe-families and their information processing capabilities have been studied extensively, e.g., [3,4,10]. Using a topological approach, it was shown in [10] that these fe-families are different than the Chomsky's classes and thus, the tools to study them from formal language theory do not readily apply. In [12], fe-families fe-systems were used to model both theoretical results and laboratory experiments related to DNA codeword design. It was shown that fe-family fe-systems can define entire classes of certain DNA codes and, in general, the theory of fe-systems is very applicable to the study of DNA codes.

Different variants of fe-systems models have been proposed, e.g., in membrane computing [1], to model self-assembly of graphs [6], and to define classes of graphs [11]. The single language model of fe-systems, which uses one fe-system to define a single language (fe-language) as opposed to a class of languages (fe-family) was introduced in [8] and some normal forms for this model were proved in [9]. Two advantages of the fe-language model compared to the fe-family model are that first, it can be studied with tools from formal language theory, because, as it was shown in [8], fe-language fe-systems can define languages from the entire spectrum of Chomsky's hierarchy, alternatively to grammars and automata (thus, we can study for example, the computational complexity of fe-language fe-systems, which is an unsurmountable task for fe-family fe-systems) and second, fe-language fe-systems are more applicable to laboratory experiments (since, for example, they can better model the result of a wet computation, which is a set of molecules, i.e., a set of words, as opposed to a set of languages).

The purpose of this paper is to show how the single language model of fe-systems can be used to define large sets of DNA codewords and more importantly, to discuss the potential of these fe-systems for algorithmic and laboratory implementation of theoretical results.

Definitions of DNA involution codes, fe-systems, fe-languages, and some of their properties are recalled in Section 2. Section 3 presents DNA codes characterizations by fe-language fe-systems. Section 4 discusses how fe-systems can be applied to (theoretically) define "good" codewords that have also been tested experimentally and Section 5 shows how the theoretical definitions from Section 4 can be used to design step-by-step computational procedures that can be applied to experimental work.

2 Basic Concepts and Definitions

An alphabet is denoted by Σ, the length of a word w over Σ by $|w|$ and the empty word by λ. The free monoid Σ^* contains all words over Σ and the free semigroup $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. For $k \geq 1$, $\Sigma^k = \{w \in \Sigma^* \mid |w| = k\}$ and $\Sigma^{\leqslant k} = \{w \in \Sigma^* \mid |w| \leq k\}$. For $w \in \Sigma^*$, the set of subwords of w is $\mathrm{Sub}\,(w) = \{u \mid \exists v_1, v_2 \in \Sigma^*, v_1 u v_2 = w\}$ and $\mathrm{Sub}_k(w) = \mathrm{Sub}\,(w) \cap \Sigma^k$. Define $\mathrm{Sub}\,(L) = \cup_{w \in L} \mathrm{Sub}\,(w)$.

2.1 DNA Involution Codes

Using the definitions from [13,14,15,16], an involution $\theta : \Sigma \to \Sigma$ of a set Σ is a mapping such that θ^2 equals the identity mapping, i.e., $\theta(\theta(x)) = x$, for all $x \in \Sigma$. For words $u, v \in \Sigma^*$ and morphic θ we have that $\theta(uv) = \theta(u)\theta(v)$ and for an antimorphic θ, $\theta(uv) = \theta(v)\theta(u)$.

Definition 1. Given an alphabet Σ, let $\theta : \Sigma^* \to \Sigma^*$ be a morphic or an antimorphic involution and $X \subseteq \Sigma^+$. Then the set (language) X is called a:

1. *θ-subword-k-m-code* for some positive integers k and m if for all $u \in \Sigma^k$ we have $\Sigma^* u \Sigma^i \theta(u) \Sigma^* \cap X = \varnothing$ for all $1 \leq i \leq m$.
2. *θ-subword-k-code* for some positive integer k if for all $u \in \Sigma^k$ we have $\Sigma^* u \Sigma^i \theta(u) \Sigma^* \cap X = \varnothing$ for all $i \geq 1$.
3. *θ-strict-code* if $X \cap \theta(X) = \varnothing$.
4. *θ-prefix-code* if $X \cap \theta(X)\Sigma^+ = \varnothing$.
5. *θ-suffix-code* if $X \cap \Sigma^+\theta(X) = \varnothing$.
6. *θ-bifix-code* if X is both a θ-prefix-code and a θ-suffix-code.
7. *θ-intercode of index m* for some integer $m \geq 1$ if $X^{m+1} \cap \Sigma^+\theta(X^m)\Sigma^+ = \varnothing$.
8. *θ-infix-code* if $\Sigma^*\theta(X)\Sigma^+ \cap X = \varnothing$ and $\Sigma^+\theta(X)\Sigma^* \cap X = \varnothing$.
9. *θ-comma-free-code* if $X^2 \cap \Sigma^+\theta(X)\Sigma^+ = \varnothing$.
10. *θ-k-code* for some integer $k > 0$ if $\mathrm{Sub}_k(X) \cap \mathrm{Sub}_k(\theta(X)) = \varnothing$.

A set $X \subseteq \Sigma^+$ is said to be a *θ-strict-P-code* if X is both a θ-P-code and a θ-strict-code, where $P \in \{$prefix, suffix, infix, bifix, comma-free, intercode, k-(code)$\}$.

Example 2. (From [12]) Let $X = \{aa, baa\}$ be a language over the alphabet $\Sigma = \{a, b\}$ and θ be a morphic involution on Σ with $\theta(a) = b$ and $\theta(b) = a$. Then $\theta(X) = \{bb, abb\}$. Note that X is a θ-infix-code since none of the subwords of X

are in $\theta(X)$ and X is a θ-comma-free-code since $X^2 = \{a^4, a^2ba^2, ba^4, ba^2ba^2\}$ and none of the words in $\theta(X)$ appears as a subword of any word in X^2. Also, note that $\mathrm{Sub}_3(X) \cap \mathrm{Sub}_3(\theta(X)) = \varnothing$. Hence, X is a θ-3-code. The word $baa = ba\theta(b)$ and hence X is not a θ-subword-1-code but there is no word of the type $uxv\theta(x)w$ in X with $u, w \in \Sigma^*$, $v \in \Sigma^+$ and $|x| = 2$. Thus, X is a θ-subword-2-code.

2.2 Forbidding-Enforcing Systems Defining fe-Languages

The definitions and notation for fe-family fe-systems are from [4]. The single language (fe-language) fe-systems model used to characterize DNA codes in this paper was introduced in [8] and related definitions and examples are recalled.

Definition 3. A *forbidding set* $\mathcal{F}$ over Σ is a family of non-empty finite subsets F of Σ^+ called *forbidders*. A word w is *consistent with a forbidder* F, denoted by $w\,con\,F$, iff $F \not\subseteq \mathrm{Sub}(w)$. A word w is *consistent with a forbidding set* $\mathcal{F}$ denoted by $w\,con\,\mathcal{F}$, iff $w\,con\,F$ for all $F \in \mathcal{F}$. If w is not consistent with $\mathcal{F}$, the notation is $w\,ncon\,\mathcal{F}$. The language $L(\mathcal{F}) = \{w \mid w\,con\,\mathcal{F}\}$ (called a *forbidding* or *f-language* for short) is said to be *defined* by the forbidding set $\mathcal{F}$.

Example 4. For $\Sigma = \{a, b\}$ and for the forbidding set $\mathcal{F} = \{\{ab, ba\}, \{aa, bb\}\}$ from [4] and others, $L(\mathcal{F}) = \{\, a^n, b^n, ab^n, a^nb, ba^n, b^na \mid n \geq 0\}$.

Remark 5. Observe that $L(\mathcal{F})$ in the above example is precisely the union of the (maximal and thus all) languages in $\mathcal{L}(\mathcal{F})$ (defined in [4]). It was proved in Theorem 1 in [8] and Theorem 3 in [9] that this is always the case, i.e given a forbidding set $\mathcal{F}$, $L(\mathcal{F}) = \cup_{L \in \mathcal{L}(\mathcal{F})} L$.

Note that if nothing is forbidden, then everything is allowed, i.e., $L(\mathcal{F}) = \Sigma^*$ if and only if $\mathcal{F} = \varnothing$.

The second constraint in any fe-systems model is an enforcing set. Note that the definition for enforcing set for fe-families (see [4]) is different than that for an enforcing set for fe-languages (see [8]). It should be clear from the context which definition is intended.

Definition 6. An *enforcing set* $\mathcal{E}$ over Σ is a family of ordered pairs (x, Y) called *enforcers*, such that $x \in \Sigma^*$ and $Y = \{y_1, \ldots, y_n\}$ where $y_i \in \Sigma^+$ for $i = 1, \ldots, n$, $x \in \mathrm{Sub}(y_i)$ and $x \neq y_i$ for every $y_i \in Y$. A word w *satisfies an enforcer* (x, Y) denoted $(w\,sat\,(x, Y))$ iff $w = uxv$ for some $u, v \in \Sigma^*$ implies that there exists $y_i \in Y$ and $u_1, u_2, v_1, v_2 \in \Sigma^*$ such that $y_i = u_2xv_2$ and $w = u_1u_2xv_2v_1$. A word w *satisfies an enforcing set* $\mathcal{E}$, denoted $w\,sat\,\mathcal{E}$, iff w satisfies every enforcer in that set. Otherwise, w does not satisfy $\mathcal{E}$, written $w\,nsat\,\mathcal{E}$. The enforcing set $\mathcal{E}$ is said to *define* the language $L(\mathcal{E}) = \{w \mid w\,sat\,\mathcal{E}\}$ (called an *e-language*).

If $x \notin \mathrm{Sub}(w)$, w satisfies the enforcer trivially. If $x = \lambda$, a word from Y has to be a subword of w in order for w to satisfy that enforcer.

Observe that if nothing is enforced everything is allowed, i.e., $L(\mathcal{E}) = \Sigma^*$ if and only if $\mathcal{E} = \varnothing$.

Example 7. Let $\Sigma = \{a\}$ and $L = \{a^{2^n} \mid n \geq 0\}$. Then, the enforcing set $\mathcal{E} = \{(\lambda, \{a, aa\})\} \cup \{(a^{2^i+1}, \{a^{2^{i+1}}\}) \mid i \geq 1\}$ defines L, i.e., $L = L(\mathcal{E})$.

Preserving the idea of a forbidding-enforcing system from [4], [8] defines a forbidding-enforcing language (fe-language) analogously.

Definition 8. A *forbidding-enforcing system* over an alphabet Σ is an ordered pair $(\mathcal{F}, \mathcal{E})$, such that $\mathcal{F}$ is a forbidding set and $\mathcal{E}$ is an enforcing set. The language $L(\mathcal{F}, \mathcal{E})$ defined by this system (called an *fe-language*) consists of all words in Σ^* that are consistent with $\mathcal{F}$ and satisfy $\mathcal{E}$, i.e., $L(\mathcal{F}, \mathcal{E}) = L(\mathcal{F}) \cap L(\mathcal{E})$.

Example 9.
1. Let $\mathcal{F} = \{\{ba\}\}$ and $\mathcal{E}_1 = \{(\lambda, \{a\})\} \cup \{(a^i, \{a^{i+1}, a^i b^i\}) \mid i \geq 1\}$. Then, $L_1 = L(\mathcal{F}, \mathcal{E}_1) = \{a^n b^m \mid n \leq m \text{ and } n, m \geq 1\}$.
2. Let $\mathcal{F} = \{\{ba\}\}$ and $\mathcal{E}_2 = \{(\lambda, \{b\})\} \cup \{(b^i, \{b^{i+1}, a^i b^i\}) \mid i \geq 1\}$. Then, $L_2 = L(\mathcal{F}, \mathcal{E}_2) = \{a^n b^m \mid n \geq m \text{ and } n, m \geq 1\}$.

It follows from Example 9 and from the property for fe-systems $L(\mathcal{F} \cup \mathcal{F}', \mathcal{E} \cup \mathcal{E}') = L(\mathcal{F}, \mathcal{E}) \cap L(\mathcal{F}', \mathcal{E}')$ that $L = L_1 \cap L_2 = \{a^n b^n \mid n \geq 1\} = L(\mathcal{F}, \mathcal{E}_1 \cup \mathcal{E}_2)$.

3 Characterizing Involution Codes by fe-Language fe-Systems

In this section, the single language model of fe-systems is used to characterize some involution codes. The notation for fe-systems from Subsection 2.2 is observed, i.e., an fe-system $(\mathcal{F}, \mathcal{E})$ is understood to define the fe-language $L(\mathcal{F}, \mathcal{E})$. Observe that $L(\mathcal{F}, \varnothing) = L(\mathcal{F})$ for every forbidding set $\mathcal{F}$ and $L(\varnothing, \mathcal{E}) = L(\mathcal{E})$ for every enforcing set $\mathcal{E}$. In this sense, all f-languages and all e-languages are fe-languages.

3.1 Characterizations by f-Languages

Since forbidding sets are defined in the same way for both fe-systems models, some results for f-languages can be proved using similar techniques as the ones used for f-families. However, they do not follow readily from Remark 5, since there are languages $L \subseteq L(F)$ such that $L \notin \mathcal{L}(\mathcal{F})$. Some results from Section 3.1 in [12], which also hold for f-languages are presented here. Others can be adjusted for f-languages with similar proofs as in [12], as well, to obtain characterizations for θ-subword-k-m, θ-strict-infix, θ-intercode of index m, θ-comma-free-code, etc. Assume $\lambda \notin L$ for $L \subseteq L(\mathcal{F})$, Σ is given, and θ is a morphic or antimorphic involution on Σ^*.

Proposition 10. *Let $k \geq 1$ be an integer. Consider $\mathcal{F} = \{\{u, \theta(u)\} \mid u \in \Sigma^k\}$. Then for every $L \subseteq L(\mathcal{F})$, L is a θ-subword-k-code.*

Proposition 11. *Let $X \subset \Sigma^+$ be given and let $\mathcal{F}_u = \{\{ua\} \mid a \in \Sigma\} \cup \{\{au\} \mid a \in \Sigma\}$ and let $\mathcal{F} = \cup_{u \in \theta(X)} \mathcal{F}_u$. Then X is θ-infix if and only if $X \subseteq L(\mathcal{F})$.*

Proposition 12. *Given a finite alphabet Σ and a fixed $k \geq 1$ let $\Sigma^k = P \cup Q \cup R$ such that $\theta(P) = Q$, $\theta(x) = x$ for all $x \in R$ where P, Q, and R are pairwise disjoint. Let $H = Q \cup R$ and let $\mathcal{F} = \{\{u\} \mid u \in H\}$. Then, for all $L \subseteq L(\mathcal{F})$, L is a θ-k-code.*

3.2 Characterizations by e-Languages

Let Σ be an alphabet, $X \subset \Sigma^+$ be given and θ be a morphic or antimorphic involution on Σ^*. The first result characterizes θ-prefix-codes by enforcing sets.

Proposition 13. *For every $u \in \theta(X)$ construct $\mathcal{E}_u = \{(u, \{au, ub \mid a, b \in \Sigma\})\} \cup \{(ub, \{aub, ub^2 \mid a \in \Sigma\}) \mid b \in \Sigma\} \cup \{(ub^2, \{aub^2, ub^3 \mid a \in \Sigma\}) \mid b \in \Sigma\} \cup \ldots \cup \{(ub^n, \{aub^n, ub^{n+1} \mid a \in \Sigma\}) \mid b \in \Sigma\} \cup \ldots$ and let $\mathcal{E} = \cup_{u \in \theta(X)} \mathcal{E}_u$. Then, X is a θ-prefix-code if and only if $X \subseteq L(\mathcal{E})$.*

Proof. Assume X is a θ-prefix-code. Then, $X \cap \theta(X)\Sigma^+ = \varnothing$. If for every $u \in \theta(X), u \notin sub\,(X)$, the enforcers are satisfied trivially. Otherwise, for some $u \in \theta(X)$, $u \in \mathrm{Sub}\,(X)$. Then there is $x \in X$ such that $u \in \mathrm{Sub}\,(x)$ and u is not a prefix of x. Thus, u must be enclosed in au in x for some $a \in \Sigma$, i.e., $x = saut$ for some $s, t \in \Sigma^*$. Hence, x satisfies the first enforcer of $\mathcal{E}_u$. If $ub^i \in \mathrm{Sub}\,(x)$ for some $b \in \Sigma$ and some $i \geq 1$, then ub^i is enclosed in aub^i in x and x satisfies this enforcer as well. Hence, for all $x \in X$ such that $u \in \mathrm{Sub}\,(x)$, x satisfies every enforcer in $\mathcal{E}_u$, i.e., $X \subseteq L(\mathcal{E}_u)$. Similarly, $X \subseteq L(\mathcal{E}_v)$ for any $v \in \theta(X), v \neq u$. Therefore, $X \subseteq L(\mathcal{E})$.

Conversely, assume that X is not a θ-prefix-code. Then, $X \cap \theta(X)\Sigma^+ \neq \varnothing$. This implies that there exists $x \in X$ such that $x = ut$ for some $u \in \theta(X)$ and some $t \in \Sigma^+$. Then, there exists $b \in \Sigma$ such that $ub \in \mathrm{Pref}\,(x)$ and there exists $i \geq 1$ such that $ub^i \in \mathrm{Pref}\,(x)$ and $ub^{i+1} \notin \mathrm{Pref}\,(x)$. Since ub^i cannot be enclosed in either aub^i or ub^{i+1}, it follows that $x\, nsat\, (ub^i, \{aub^i, ub^{i+1} \mid a \in \Sigma\})$. Since x does not satisfy this enforcer, $x\, nsat\, \mathcal{E}$. Thus, $X \not\subseteq L(\mathcal{E})$. □

By symmetry of the above argument, one can obtain the following characterization of θ-sufix-codes.

Proposition 14. *For every $u \in \theta(X)$ construct $\mathcal{E}_u = \{(u, \{au, ub \mid a, b \in \Sigma\})\} \cup \{(au, \{aub, a^2u \mid b \in \Sigma\}) \mid a \in \Sigma\} \cup \{(a^2u, \{a^2ub, a^3u \mid b \in \Sigma\}) \mid a \in \Sigma\} \cup \ldots \cup \{(a^nu, \{a^nub, a^{n+1}u \mid b \in \Sigma\}) \mid a \in \Sigma\} \cup \ldots$ and let $\mathcal{E} = \cup_{u \in \theta(X)} \mathcal{E}_u$. Then, X is a θ-suffix-code if and only if $X \subseteq L(\mathcal{E})$.*

Let $\mathcal{E}_1$ be the enforcing set from Proposition 13 and $\mathcal{E}_2$ be the enforcing set from Proposition 14. Then X is a θ-bifix-code if and only if X is both θ-prefix and θ-suffix (a fact confirmed here by properties of fe-systems, as well), i.e., if and only if $X \subseteq L(\mathcal{E}_1)$ and $X \subseteq L(\mathcal{E}_2)$ if and only if $X \subseteq L(\mathcal{E}_1 \cup \mathcal{E}_2)$, where the last equivalence follows from properties of enforcing sets. Thus, from the enforcing property $L(\mathcal{E}_1 \cup \mathcal{E}_2) = L(\mathcal{E}_1) \cap L(\mathcal{E}_2)$ (which holds for any two enforcing sets), from $\mathcal{E}_1$ and $\mathcal{E}_2$, and Propositions 13 and 14, we obtain the following characterization of θ-bifix-codes.

Proposition 15. *For every* $u \in \theta(X)$ *construct* $\mathcal{E}'_u = \{(u, \{au, ub \mid a, b \in \Sigma\})\} \cup \{(ub, \{aub, ub^2 \mid a \in \Sigma\}) \mid b \in \Sigma\} \cup \{(ub^2, \{aub^2, ub^3 \mid a \in \Sigma\}) \mid b \in \Sigma\} \cup \ldots \cup \{(ub^n, \{aub^n, ub^{n+1} \mid a \in \Sigma\}) \mid b \in \Sigma\} \cup \ldots$ *and let* $\mathcal{E}_1 = \cup_{u \in \theta(X)} \mathcal{E}'_u$. *Also, for every* $u \in \theta(X)$ *construct* $\mathcal{E}''_u = \{(u, \{au, ub \mid a, b \in \Sigma\})\} \cup \{(au, \{aub, a^2u \mid b \in \Sigma\}) \mid a \in \Sigma\} \cup \{(a^2u, \{a^2ub, a^3u \mid b \in \Sigma\}) \mid a \in \Sigma\} \cup \ldots \cup \{(a^nu, \{a^nub, a^{n+1}u \mid b \in \Sigma\}) \mid a \in \Sigma\} \cup \ldots$ *and let* $\mathcal{E}_2 = \cup_{u \in \theta(X)} \mathcal{E}''_u$. *Finally, let* $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$. *Then,* X *is a* θ*-bifix-code if and only if* $X \subseteq L(\mathcal{E})$.

4 Defining Good DNA Codes by Language fe-Systems

Unlike the previous section, where the goal was to show how each of the constraints can work independently to characterize DNA codes, the fe-systems discussed in this and the following section show how fe-systems with both nonempty forbidding sets and non-empty enforcing sets can be used and how the two constraints can "act together" to define or generate structures. Section 4 in [12] provides an illustration of how fe-family fe-systems can be used to generalize theoretical results and describe applications, as in Proposition 4.9 and the related experiment from [15]. Proposition 31, Example 32, and Corollary 33 from [12] can be adjusted for f-languages and proved in a similar manner as in [12]. However, the enforcing sets are differently defined for the fe-language and the fe-family models, and hence, the second part of results from Section 4 in [12] differ for each model. This difference accounts for the better applicability of the fe-language model to experimental work. The next result is analogous to Proposition 34 in [12] for fe-languages.

Proposition 16. *Let* Σ *be an alphabet and* θ *a morphic or an antimorphic involution such that* $\theta(a) \neq a$ *for each symbol* $a \in \Sigma$. *Let* $b, c \in \Sigma$ *such that* $\theta(b) = c$ *and* $k \geq 2$. *Consider the forbidding set* $\mathcal{F} = \{\{c\}\} \cup \{\{u\} \mid u \in (\Sigma \setminus \{b, c\})^k\}$ *and the enforcing set* $\mathcal{E} = \{(\lambda, \{ub \mid u \in (\Sigma \setminus \{c\})^{k-1}\})\} \cup \{(awb, \{ubvb \mid u, v \in (\Sigma \setminus \{c\})^{k-1}\}) \mid w \in (\Sigma \setminus \{c\})^{k-1}$ *and* $a \in (\Sigma \setminus \{c\})\} \cup \{(wba, \{ubvb \mid u, v \in (\Sigma \setminus \{c\})^{k-1}\}) \mid w \in (\Sigma \setminus \{c\})^{k-1}$ *and* $a \in (\Sigma \setminus \{c\})\}$. *If* $L \subseteq L(\mathcal{F}, \mathcal{E})$ *then* L *is a* θ*-*k*-code. Furthermore, if* $L \subseteq L(\mathcal{F}, \mathcal{E})$ *then* $L^+ \subseteq L(\mathcal{F}, \mathcal{E})$.

Proof. If L is not a θ-k-code, then by Proposition 31 in [12] adjusted for fe-languages, $L \not\subseteq L(\mathcal{F})$ and hence $L \not\subseteq L(\mathcal{F}, \mathcal{E})$. Assume that $L \subseteq L(\mathcal{F}, \mathcal{E})$ and let u and v be any two words in L. Their concatenation uv is also in $L(\mathcal{F}, \mathcal{E})$, and hence, $L^2 \subseteq L(\mathcal{F}, \mathcal{E})$. By induction, $L^n \subseteq L(\mathcal{F}, \mathcal{E})$ for all $n \geq 1$ and thus $L^+ \subseteq L(\mathcal{F}, \mathcal{E})$. □

The above proof shows that L does not need to equal L^+. This shows that one can take any subset of $L(\mathcal{F}, \mathcal{E})$ for a starting point, which is more general than [15, Proposition 4.9] where $X = X^+$. While Proposition 16 uses fe-systems as a defining tool or alternatively as a one-step-computation, i.e., accepting the desired structures out of all possible structures in one step, the next corollary uses fe-systems as a generative tool, i.e., to generate larger words starting from smaller ones, as explained in the next section.

Let $\Delta = \{A, C, T, G\}$ and θ be a morphic or an antimorphic involution. To model closer the experiment in [15], let θ be the Watson-Crick complementarity such that $\theta(A) = T$, $\theta(T) = A$, $\theta(C) = G$, and $\theta(G) = C$, where θ is antimorphic and consider the following consequence of Proposition 16.

Corollary 17. *Let the alphabet be Δ and integers $k, n \geq 2$ be given. Let $\mathcal{F} = \{\{G\}\} \cup \{\{u\} \mid u \in \{A, T\}^k\} \cup \{\{z\} \mid |z| = nk+1$ and $z \in \Delta^*\}$, and $\mathcal{E} = \mathcal{E}_0 \cup_{i=1}^{n-1} \mathcal{E}_i$ where $\mathcal{E}_0 = \{(\lambda, \{uC \mid u \in \{A, T, C\}^{k-1}\})\}$ and $\mathcal{E}_i = \{(w_1 C w_2 C \dots w_i C, \{u_1 C u_2 C \dots u_{i+1} C \mid u_j \in \{A, T, C\}^{k-1}\}) \mid w_l \in \{A, T, C\}^{k-1}\}$. Then, $L \subseteq L(\mathcal{F}, \mathcal{E})$ implies L is a θ-k-code. Furthermore, if $L \subseteq L(\mathcal{F}, \mathcal{E})$ then $L^+ \subseteq L(\mathcal{F}, \mathcal{E})$.*

5 Generating Good DNA Codes by Language fe-Systems

This section provides intuition on how language fe-systems can be used to perform computation on one hand and to bridge theoretical results about DNA involution codes with their application on another. The concept of computing with fe-systems was introduced in [4] for the families of languages model of fe-systems. The Γ-tree presented in [4] captures the idea of such computation: one can start with smaller sets of strands and "build" larger sets by applying enforcers to produce larger sets that are also consistent with the forbidding conditions. This general concept for the family of languages model of fe-systems also applies to the single language model of fe-systems, i.e., one can start with small words and apply enforcers in such a way, step-by-step, such that larger words are generated which comply with the forbidding conditions. The general notion of computing by fe-systems, using their different variants is a vast topic by itself.

The following example illustrates how a fe-language fe-system may be applied to *translate* a theoretical result into a lab experiment. Consider Proposition 4.9 and the related lab experiment in [15] where 10 θ-5-codes were constructed using the methods in Proposition 4.9 and tested experimentally. Their DNA sequences are presented as $K_1 - K_{10}$ in Table 1 in [15]. We use the fe-system from Corollary 17 in the previous section to describe the steps in the following algorithm.

Algorithm: Fix an integer $k \geq 1$ (for θ-k-codes). Fix an integer $n \geq 1$ for the number of desired annealing steps.

1. Input: Construct strands of type $(\{A, C, T\}^{k-1}C)$ and put them in the test tube S (applies the enforcer in $\mathcal{E}_0$ and all forbidders except type $\{z\}$)
2. For $1 \leq i \leq n-1$ let strands anneal i times (apply enforcers $\mathcal{E}_i$)
3. Discard strands longer than $m = nk$ (apply forbidders $\{\{z\} || z| = m + 1\}$)
4. Output: Detect (S) (the remaining strands $w \in S$ are such that $w \in L(\mathcal{F}, \mathcal{E})$)

In our example, the initial test tube S contains words (strands) like $AATAC$, $ATCAC, ATTTC, TACCC$ used to construct K_1. Then, applying the enforcers $\mathcal{E}_1$ will produce $2k$ or 10-letter words such as $AATACATCAC, ATCACATTTC$, $ATTTCTACCC$ which are 10-letter subwords of K_1, as well as, all possible concatenations of words in S. The enforcers $\mathcal{E}_2$ will produce all $3k$ or 15-letter words from S^3 and so on. The sequences $K_1 - K_{10}$ will result after applying enforcers

$\mathcal{E}_3$ to produce all possible 20-mers in S^4 including the sequences $K_1 - K_{10}$, e.g., $K_1 = AATACATCACATTTCTACCC$. If desired, the process may start with $K_1 - K_{10}$, as in the experiment, and continue by annealing the strands from there. The enforcers applied at each step will only be the ones applicable to the system and the forbidders $\{z\}$ model discarding sequences of higher than desired length and terminating the process when desired length m is reached.

6 Concluding Remarks

Since forbidding-enforcing systems impose restrictions on the subwords of a word or a language, they can be used to model the restrictions imposed by unwanted hybridizations and thus, provide a natural framework to study DNA codes. This paper shows how fe-language fe-systems can be used as a defining tool by proving a series of characterizations of DNA involution codes by fe-language fe-systems. It also presents motivation for applying fe-systems to the study of DNA codes by discussing the computational nature of fe-systems, their applicability to experimental work and their generative capacity to model the evolution of a molecular system.

Future work in fe-systems points to many directions: investigating the computational complexity of fe-language fe-systems and developing their applications as a bridge between theoretical results and practical implementation, such as applying the results in [17,18] to experimental work; investigating the relationship between different variants of fe-systems, such as graph fe-systems (introduced in [11]) and single language fe-systems (introduced in [8]) to study their similarities as computational tools and to contribute to characterizing structures of automata recognizing given languages, for example; investigating each of the models theoretically in the search of new properties of the underlying structures and similarities of different types of structures via fe-systems, e.g., between graphs and languages.

Acknowledgement. This work has been partially supported by a UNF Faculty Development Scholarship Grant.

References

1. Cavaliere, M., Jonoska, N.: Forbidding and enforcing in membrane computing. Natural Computing 2, 215–228 (2003), doi:10.1023/A:1025492906773
2. Deaton, R., Murphy, R., Rose, J., Garzon, M., Franceschetti, D., Stevens Jr., S.: A DNA based implementation of an evolutionary search for good encodings for DNA computation. In: Proc. IEEE Int. Conf. on Evolutionary Computation, ICEC 1997, pp. 267–271 (1997)
3. Ehrenfeucht, A., Hoogeboom, H.J., Rozenberg, G., van Vugt, N.: Forbidding and enforcing. In: Winfree, E., Gifford, D.K. (eds.) DNA Based Computers V. AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 54, pp. 195–206 (2000)

4. Ehrenfeucht, A., Rozenberg, G.: Forbidding-enforcing systems. Theoretical Computer Science 292, 611–638 (2003), doi:10.1016/S0304-3975(01)00088-3
5. Faulhammer, D., Cukras, A.R., Lipton, R.J., Landweber, L.F.: Molecular Computation: RNA Solutions to Chess Problems. Proceedings of the National Academy of Sciences 97(4), 1385–1389 (2000)
6. Franco, G., Jonoska, N.: Forbidding and Enforcing Conditions in DNA Self-assembly of Graphs. Nanotechnology: Science and Computation, Natural Computing Series, Part I, pp. 105–118 (2006), doi:10.1007/3-540-30296-4-6
7. Garzon, M., Phan, V.: On codeword design in metric spaces. Natural Computing 8(3), 571–588 (2009), doi:10.1007/s11047-008-9088-6
8. Genova, D.: Defining Languages by Forbidding-Enforcing Systems. In: Löwe, B., Normann, D., Soskov, I., Soskova, A. (eds.) CiE 2011. LNCS, vol. 6735, pp. 92–101. Springer, Heidelberg (2011)
9. Genova, D.: Forbidding Sets and Normal Forms for Language Forbidding-Enforcing Systems. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 289–300. Springer, Heidelberg (2012)
10. Genova, D., Jonoska, N.: Topological Properties of Forbidding-Enforcing Systems. Journal of Automata, Languages and Combinatorics 11(4), 375–397 (2006)
11. Genova, D., Jonoska, N.: Forbidding and enforcing on graphs. Theoretical Computer Science 429, 108–117 (2012), doi:10.1016/j.tcs.2011.12.029
12. Genova, D., Mahalingam, K.: Generating DNA Code Words Using Forbidding and Enforcing Systems. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) TPNC 2012. LNCS, vol. 7505, pp. 147–160. Springer, Heidelberg (2012)
13. Hussini, S., Kari, L., Konstantinidis, S.: Coding properties of DNA languages. Theoretical Computer Science 290, 1557–1579 (2003); PII: S0304-3975(02)00069-5
14. Jonoska, N., Mahalingam, K.: Languages of DNA Based Code Words. In: Chen, J., Reif, J.H. (eds.) DNA 2003. LNCS, vol. 2943, pp. 61–73. Springer, Heidelberg (2004)
15. Jonoska, N., Mahalingam, K., Chen, J.: Involution codes: with application to DNA coded languages. Natural Computing 4, 141–162 (2005), doi:10.1007/s11047-004-4009-9
16. Kari, L., Mahalingam, K.: DNA Codes and Their Properties. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 127–142. Springer, Heidelberg (2006)
17. Kari, L., Konstantinidis, S., Sosík, P.: Bond-Free Languages: Formalizations, Maximality and Construction Methods. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 169–181. Springer, Heidelberg (2005)
18. Konstantinidis, S., Santean, N.: Computing Maximal Kleene Closures That Are Embeddable in a Given Constrained DNA Language. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 115–129. Springer, Heidelberg (2011)
19. Marathe, A., Condon, A.E., Corn, R.M.: On combinatorial word design. In: Preliminary Preproceedings of the 5th International Meeting on DNA Based Computers, Boston, pp. 75–88 (1999)

Computing K-Trivial Sets by Incomplete Random Sets

Noam Greenberg

School of Mathematics, Statistics and Operations Research,
Victoria University of Wellington, New Zealand
greenberg@msor.vuw.ac.nz

I discuss part of the solution for the ML-covering problem [1]. This passes through analytic notions such as martingale convergence and Lebesgue density; an understanding of the class of cost functions which characterizes K-triviality; and identifying the correct notion of randomness which corresponds to computing K-trivial sets, together with the construction of a smart K-trivial set. This is joint work with Bienvenu, Kučera, Nies, and Turetsky.

Reference

1. Miller, J.S., Nies, A.: Randomness and computability: open questions. Bull. Symb. Logic 12, 390–410 (2006)

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, p. 230, 2013.

Cardinal-Recognizing Infinite Time Turing Machines

Miha E. Habič

The Graduate Center of the City University of New York, Mathematics Program,
365 Fifth Avenue, New York, NY 10016, USA
mhabic@gc.cuny.edu

Abstract. We introduce a model of infinitary computation which enhances the infinite time Turing machine model slightly but in a natural way by giving the machines the capability of detecting cardinal stages of computation. The computational strength with respect to ITTMs is determined to be precisely that of the strong halting problem and the nature of the new characteristic ordinals (clockable, writable, etc.) is explored.

Various notions of infinitary computability have now been studied for several decades. The concept that we can somehow utilize infinity to accommodate our computations is at the same time both appealing and dangerous; appealing, since we are often in a position where we could answer some question if only we could look at the output of some algorithm after an infinite amount of steps, and dangerous, since this sort of greediness must inevitably lead to disappointment when we suddenly reach the limits of our model. At that point we must decide whether to push on and strengthen our model in some way or to abandon it in favour of some (apparently) alternative model. But if we do not wish to abandon our original idea, how to strengthen it in such a way that it remains both interesting and intuitive?

The behaviour of infinite time Turing machines (ITTMs), first introduced in [1], has by now been extensively explored and various characteristics have been determined. While we are far from reaching full understanding of the model, we nevertheless already feel the urge to generalize further. Perhaps the most direct generalization are the ordinal Turing machines of [2], where both the machine tape and running time are allowed to range into the transfinite. But perhaps this modification seems too strong; with it all constructible sets of ordinals are computable. We would be satisfied with the minimal nontrivial expansion of the ITTM model, i.e., something which computes the appropriate halting problem but no more. Of course, we can easily achieve this goal within the ITTM framework by considering oracle computations, but this somehow doesn't seem satisfactory. We intend to give what we feel is a more natural solution to this problem but which falls short of the 'omnipotence' of ordinal Turing machines.

We assume some familiarity with the concepts and notation related to ITTMs and computability theory in general. In particular, we fix at the outset some

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 231–240, 2013.

uniform way of coding programs, whole machine configurations and countable ordinals as reals (i.e., infinite binary sequences). Given a code p for a program, $\varphi_p(x)$ denotes the computation of p with x as input, while $\varphi_p(x)\downarrow$ means that this computation halts. By the output of a computation stabilizing, we mean that from some time onward the contents of the output tape do not change during that computation (but the computation itself need not halt or even be aware of the stabilization). An ordinal is *clockable* if it is the halting time of some ITTM computation with empty input; the supremum of the clockable ordinals is denoted γ. A real is *writable* if it is the output of some halting ITTM computation with empty input, *eventually writable* if it is the output of a stabilized ITTM computation with empty input, and *accidentally writable* if it appears on any tape at any time during an ITTM computation with empty input. An ordinal is (eventually/accidentally) writable if the real coding it is such. The suprema of the writable/eventually writable/accidentally writable ordinals are denoted λ, ζ and Σ, respectively.

1 The Model and Its Computational Power

We build on the standard ITTM framework in which the machines have three tapes and cell values at limit stages are calculated according to the lim sup rule. Our proposed model has the same hardware and behaviour, with one exception: there is a special state, called the *cardinal* state, which is used instead of the *limit* state at cardinal stages of the computation. To be precise, if κ is an uncountable[1] cardinal, the configuration of our machine at stage κ is as follows: the head is on the first cell of the tape, the machine is in the *cardinal* state and the cell values are the lim sup of the previous values. The machine handles non-cardinal limit ordinal stages like an ordinary ITTM. We call these machines *cardinal-recognizing infinite time Turing machines* (CRITTMs). We also define CRITTM-computable functions on Cantor space 2^ω and CRITTM-(semi)decidable subsets of 2^ω analogously to the ITTM case.

Our first task should be to verify that our proposed machines actually possess the computing power we desired of them. Recall the characteristic undecidable problems of ITTM computation:

- the weak halting problem $0^\triangledown = \{p; \varphi_p(0)\downarrow\}$,
- the strong halting problem $0^\blacktriangledown = \{(p,x); \varphi_p(x)\downarrow\}$,
- the stabilization problem $S = \{(p,x);$ the output of $\varphi_p(x)$ stabilizes$\}$.

Proposition 1. *The sets $0^\triangledown, 0^\blacktriangledown$ and S are CRITTM-decidable.*

Proof. The decidability of $0^\triangledown$ is clearly reducible to the decidability of $0^\blacktriangledown$. Similarly, the decidability of $0^\blacktriangledown$ is reducible to the decidability of S: given a program p and an input x, construct a new program p' which acts like p but also flashes a

[1] We restrict to uncountable cardinals mainly to ensure that verbatim copies of ITTM programs work as expected. There is no difference in power between this convention and using the *cardinal* state at all cardinal stages of computation.

designated cell (called a *flag*) after completing each instruction of p; the program p halts on x iff p' stabilizes on x.

It therefore remains to show that S is CRITTM-decidable. Consider the following algorithm: given a pair (p, x), simulate $\varphi_p(x)$ and flash a flag each time the simulated output changes. When a *cardinal* state is attained, output 'no' if the flag is showing 1 and 'yes' if it is showing 0. This algorithm decides S. Indeed, recall that every ITTM computation either halts or begins repeating at some countable time and so the question of stabilization is completely settled by (or before) time ω_1 when the *cardinal* state is first reached. □

In the above proof we made use of the fact that every ITTM computation halts or begins repeating before time ω_1. An analogous property also holds for CRITTM computations. To state this correctly we must, as in the ITTM case, clarify what we mean by an infinite computation repeating. In the ITTM model this is the case when the machine configuration is the same at two limit ordinal stages $\alpha < \beta$ and no cell that is showing 0 at stage α ever shows a 1 between these two stages. This configuration will then repeat (at least) ω many times and the last clause ensures that the configuration at the limit of these repeats will again be the same. The machine then has no alternative but to continue its sisyphean task.

With the appropriate modifications the same description of repeating also works in the case of CRITTMs. Specifically, we say that a CRITTM computation repeats if the machine configuration is the same at two cardinal stages $\kappa < \lambda$ and no cell that is showing 0 at stage κ ever shows a 1 between these two stages. Note that we do not require the stages κ and λ to be limit cardinals. This is not needed since the same argument as before shows that the machine cannot escape this strong repeating pattern: the once repeated configuration will repeat again ω many times[2] and the limit configuration will be the same.

Proposition 2. *Every CRITTM computation halts or begins repeating before time* $\aleph_{\omega_1}$.

Proof. The proof is very similar to the one in the ITTM case. Assume that the machine hasn't halted by time $\aleph_{\omega_1}$. By a cofinality argument there is a countable ordinal α_0 such that all of the cells that will have stabilized before time $\aleph_{\omega_1}$ have already done so by time $\aleph_{\alpha_0}$. Similarly, there is a countable ordinal $\alpha_1 > \alpha_0$ such that each of the nonstabilized cells will have flipped its value at least once between $\aleph_{\alpha_0}$ and $\aleph_{\alpha_1}$. Construct the increasing sequence of ordinals $(\alpha_n)_{n<\omega}$ in this way and let $\alpha_\omega = \sup_n \alpha_n < \omega_1$. Consider the configuration of the machine at time $\aleph_{\alpha_\omega}$. The stabilized cells are showing their values and the nonstabilizing cells have flipped unboundedly many times below $\aleph_{\alpha_\omega}$ and are thus showing 1. This is the same configuration as at time $\aleph_{\omega_1}$. If we now repeat the construction starting with some α'_0 above α_ω, we find the same configuration at some time $\aleph_{\alpha'_\omega}$. Then, by construction, the computation between $\aleph_{\alpha_\omega}$ and $\aleph_{\alpha'_\omega}$ repeats. □

[2] Note that the length of time between repeats gets progressively longer. It will follow from later results that this additional time cannot be used in any meaningful way.

We conclude this section by determining the precise computational power of CRITTMs. Recall that we introduced them as an attempt to define a minimal natural strengthening of the ITTM model which could decide the strong ITTM halting problem without explicitly adding a halting oracle. It turns out that we hit our mark perfectly as the following theorem shows.

Theorem 3. *CRITTMs compute the same functions as ITTMs with a* $0^{\blacktriangledown}$ *oracle. In this sense the two models are computationally equivalent.*[3]

Proof. We have already seen in Proposition 1 that $0^{\blacktriangledown}$ is decidable by a CRITTM, so any computation of an ITTM with a $0^{\blacktriangledown}$ oracle can be simulated on a CRITTM. We must now show that the converse holds.

Consider a CRITTM computation $\varphi_p(x)$. By Proposition 2 this computation halts or begins repeating in the CRITTM sense by time $\aleph_{\omega_1}$. However, in each time interval $[\aleph_\alpha, \aleph_{\alpha+1})$ the computation is in effect an ITTM computation and thus begins repeating in the ITTM sense by time $\aleph_\alpha + \beta_\alpha$ for some countable β_α. We see that the machine performs no useful computations for longer and longer periods of time. It is this behaviour that allows us to compress the CRITTM computation. Given a CRITTM program p, we let $\tilde{p}$ be the program p with all mention of the *cardinal* state omitted (transforming it into an ITTM program). We then define p_* as the following ITTM program: given an input z, decode it into the contents of the three machine tapes, the machine state and the head position and run $\tilde{p}$ on this decoded configuration. By setting aside space to store information on which cells have changed value since the start of the simulation and at each step comparing the current simulated configuration to the one coded by z, the program p_* can halt if the simulation of $\tilde{p}$ repeats the configuration coded by z in the ITTM sense.

Consider the following program for an ITTM with a $0^{\blacktriangledown}$ oracle which simulates the CRITTM computation $\varphi_p(x)$. First, construct the program p_*. At each subsequent step, code the simulated configuration into a real z and query the oracle whether $\varphi_{p_*}(z)$ halts. If not, simulate a step of p. Otherwise, $\varphi_p(x)$ has reached its ITTM repeating configuration between cardinal stages, which must also be the configuration at the next cardinal time. We can therefore jump ahead in our simulation, setting its state to *cardinal* and moving the head back to the beginning of the tape, after which we continue with our procedure. Clearly, if $\varphi_p(x)$ halts, this simulation will halt with the same output. □

2 Clockable and Writable Ordinals

In this section we wish to examine the various classes of ordinals connected with CRITTM computation. In particular, we are interested in CRITTM-clockable and (eventually/accidentally) CRITTM-writable ordinals, where these concepts

[3] This theorem gives an alternative proof (and was originally conceived by observing) that the sets $0^{\blacktriangledown}$ and S are infinite time Turing equivalent; the strong halting oracle allows us to foresee repeating patterns.

are defined analogously to the ITTM case. In the sequel, when we write clockable/writable, we always mean ITTM-clockable/writable.

Let us first present some examples regarding CRITTM-clockable ordinals.

- The countable CRITTM-clockable ordinals are precisely the clockable ordinals. Clearly all clockable ordinals are CRITTM-clockable. In fact, since we stipulated that the *cardinal* state is only used at uncountable stages, the very same program that clocks α on an ITTM can be used to clock it on a CRITTM. Furthermore, no new clockable ordinals appear since at countable stages CRITTMs behave no differently to ITTMs.
- If $\aleph_\alpha$ and $\aleph_\beta$ are CRITTM-clockable then so are $\aleph_{\alpha+\beta}$ and $\aleph_{\alpha\cdot\beta}$. We only give a sketch of the algorithm for the first part, leaving the second to the reader. If $\aleph_\alpha$ and $\aleph_\beta$ are CRITTM-clockable, we can modify the algorithms clocking them to only use the even- and odd-numbered cells on the tape, respectively, without changing the running time. We then design an algorithm which uses these modified programs to first clock $\aleph_\alpha$ and then run the program for clocking $\aleph_\beta$, keeping track of which part of the computation we are executing by means of a master flag. This clocks $\aleph_{\alpha+\beta}$.

There is a very important observation to be made regarding the last bullet point above. The observant reader will have noticed the awkward wording in the algorithm given; we are referring to the phrase "run the program for clocking $\aleph_\beta$". Why not simply say "clock $\aleph_\beta$"? We feel that the issue is most easily seen through an example: knowing the algorithm for clocking ω_1, can we devise an algorithm for clocking $\omega_1+\omega_1$? Surely we can. Just run two copies of the program clocking ω_1 one after another. But does this actually clock $\omega_1+\omega_1$? Let us take a closer look and fix the program clocking ω_1 to be the program which waits for the first *cardinal* state that appears and then immediately halts. Consider what happens when we run two copies of this program in succession. The first copy waits for the *cardinal* state, which appears at time ω_1, and passes control to the second copy. The second copy then waits for the *cardinal* state. However, this state doesn't occur at time $\omega_1+\omega_1$, but only at ω_2. We remark that this phenomenon doesn't occur in ITTM computation due to a certain homogeneity the class of cardinals lacks; the next limit ordinal above an ordinal α is always $\alpha+\omega$, but the next cardinal above α has no such uniform description. It could, perhaps, be argued that this issue makes the notion of the length of a CRITTM computation meaningless, as this quantity changes based on the time at which we run the computation.

The discussion in the previous paragraph leads us to formulate the following theorem, which places strong restrictions on decompositions of CRITTM-clockable ordinals.

Theorem 4. *Let α and β be ordinals with $|\alpha| \geq |\beta|$. If $\alpha+\beta$ is CRITTM-clockable then β is countable.*

Proof. The assumption $|\alpha| \geq |\beta|$ may be restated as $|\alpha+\beta| = |\alpha|$. Therefore the last cardinal stage passed in a CRITTM computation of length $\alpha+\beta$ is $|\alpha|$.

In particular, the *cardinal* state doesn't appear during the last β steps of computation.

Let p be the program clocking $\alpha + \beta$ and let x_α be (a real coding) the content of the machine tapes after α many steps of computation. Since no *cardinal* states appear during the last β many steps of the computation, we are, in effect, performing an ITTM computation with input x_α which halts after β many steps. As we know the halting times of ITTM computations to be countable, β must be countable. □

Corollary 5. *CRITTM-clockable ordinals are not closed under ordinal arithmetic.*

Proof. Consider, as before, $\omega_1 + \omega_1$. Theorem 4 implies that this ordinal is not CRITTM-clockable. □

Theorem 4 raises some interesting questions. For example, given α, which countable 'tails' β give CRITTM-clockable ordinals? If α is CRITTM-clockable we might be inclined to say that precisely the clockable β arise, but this is incorrect. Keep in mind that the first part of the computation might have produced some useful output for us to utilize during the last β steps. In particular, assuming α is uncountable, any writable real may be produced to serve as input to the subsequent computation. The following proposition illustrates this fact.

Proposition 6. *The ordinal $\omega_1 + \Sigma + \omega$ is CRITTM-clockable.*[4]

Proof. In [3] it is shown that the configuration of the universal ITTM[5] at time ζ is its repeating configuration and that it repeats for the first time at time Σ. This configuration also appears at time ω_1. Let our CRITTM simulate the universal ITTM until time ω_1 at which point it stores the simulated configuration and starts a new simulation of the universal ITTM. The machine also checks at limit steps whether the current simulated configuration is the same as the stored configuration and whether this has happened for the second time. This will occur at time $\omega_1 + \Sigma$, our machine will detect this and halt at time $\omega_1 + \Sigma + \omega$. □

Another question we might ask is how CRITTM-clockability propagates. It is easily shown that α^+, the cardinal successor of α, is CRITTM-clockable if α is. Does it also inherit to cardinals in reverse? That is to say, if α is CRITTM-clockable, must $|\alpha|$ be as well? The following theorem shows that this fails in the most spectacular way possible.

Theorem 7. *There is a CRITTM-nonclockable cardinal κ such that the ordinal $\kappa + \omega$ is CRITTM-clockable.*

[4] This example is due to Joel David Hamkins.

[5] The universal ITTM is the machine that dovetails the simulation of all ITTM programs on empty input. It can be used to produce a stream consisting of all accidentally writable reals.

Proof. Let κ be the first CRITTM-nonclockable cardinal. To show that $\kappa + \omega$ is CRITTM-clockable, dovetail the simulations of all CRITTM programs on input 0. We can perform this in such a way that after λ many steps, for λ a cardinal, we have simulated exactly λ many steps of each computation. When a *cardinal* state is reached we use the next ω many steps to check whether any of the simulated computations halt at the next step. If a computation halts we continue with the simulation. Otherwise, we've happened upon the first CRITTM-nonclockable cardinal κ and we halt, having clocked $\kappa + \omega$. □

The theorem shows that the fact that κ^+ is CRITTM-clockable doesn't imply that κ itself is CRITTM-clockable. The proof is merely the trivial observation that we have shown that $\kappa + \omega$ is CRITTM-clockable, where κ is the least CRITTM-nonclockable cardinal, hence κ^+ is CRITTM-clockable. This is in stark contrast to the speed-up theorem of [1], which states that if $\alpha + n$ is clockable for some finite n then α is clockable as well.

We now consider CRITTM-writable reals and ordinals and their more transient relatives. The same proofs as in the ITTM case show that the class of accidentally CRITTM-writable reals properly contains the class of eventually CRITTM-writable reals which properly contains the class of CRITTM-writable reals and that the corresponding classes of ordinals are downward closed. Also, no additional effort need go into proving that the classes of eventually CRITTM-writable and CRITTM-writable ordinals are closed under ordinal arithmetic.

In keeping with the notation from the ITTM model, we introduce the following ordinals

$$\lambda_{\mathrm{C}} = \sup\{\alpha; \alpha \text{ is CRITTM-writable}\}$$
$$\zeta_{\mathrm{C}} = \sup\{\alpha; \alpha \text{ is eventually CRITTM-writable}\}$$
$$\Sigma_{\mathrm{C}} = \sup\{\alpha; \alpha \text{ is accidentally CRITTM-writable}\}$$

The same proof that shows $\lambda < \zeta < \Sigma$ in the ITTM model can be used here to show that $\lambda_{\mathrm{C}} < \zeta_{\mathrm{C}} < \Sigma_{\mathrm{C}}$. We also have the following proposition.

Proposition 8. *Σ_{C} is a countable ordinal.*

Proof. Accidentally CRITTM-writable ordinals are countable by definition. We now prove that there are only countably many accidentally CRITTM-writable reals, whence the proposition follows immediately.

Consider a CRITTM program p. We have shown that p either halts or has repeated by time $\aleph_\alpha$ for some countable α. Recall that the computation of p on input x either halts or repeats (in the ITTM sense) at least once by some countable time δ_x. Let $\beta < \alpha$. Within the time interval $[\aleph_\beta, \aleph_{\beta+1})$ the program p behaves just like an ITTM program and must therefore halt or repeat by time $\aleph_\beta + \delta_{x_\beta}$, where x_β codes the contents of the tapes at time $\aleph_\beta$. This means that the program p in fact produces only countably many reals in the time interval $[\aleph_\beta, \aleph_{\beta+1})$. Repeating this argument for all $\beta < \alpha$, we see that only countably many reals appear during the computation of p and since there are only countably many programs, there can only be countably many accidentally CRITTM-writable reals. □

But what is the relation between the ordinals $\lambda_C, \zeta_C, \Sigma_C$ and their ITTM counterparts? Clearly the supremum of a given CRITTM class of ordinals is at least as big as the supremum of the corresponding ITTM class, i.e., every (eventually/accidentally) writable ordinal is also (eventually/accidentally) CRITTM-writable, but can we say more? It is in fact easy to see that every eventually writable real is CRITTM-writable: if a real x is eventually written by a program p, we may use a CRITTM to simulate p for ω_1 many steps, at which point we halt, having written x. Therefore $\lambda_C \geq \zeta$ and we can go even further.

Proposition 9. *ζ is CRITTM-writable.*

Proof. We begin by dividing the output tape into ω many ω-blocks. We now proceed to enumerate all ITTM programs. When a program p is enumerated, first determine whether $\varphi_p(0)$ stabilizes and, if it does and if its stabilized output codes an ordinal[6], copy its output to the next empty ω block on the output tape. Having done this, resume enumerating programs. To deal with each particular program we need to pass only a single cardinal stage. Therefore we shall have copied all eventually writable ordinals onto the output tape by time $\aleph_\omega$, which we can recognize. At this point we use ω many steps to combine the codes on the output tape into a code for the sum of all eventually writable ordinals and then halt.

This CRITTM algorithm writes an ordinal which is at least as big as ζ, therefore ζ is CRITTM-writable. □

We have now seen that many new ordinals become writable and eventually writable when passing from the ITTM model to the CRITTM model. But what about accidentally writable ordinals?

Question 10. Is $\Sigma = \Sigma_C$?

While we currently do not have an answer to this question, let us present a brief heuristic justification for why we believe the proposed equality to be true. We have seen that $\lambda < \lambda_C$ and $\zeta < \zeta_C$, but these inequalities are not particularly surprising given that the concepts of (eventual) writability are intimately connected with the computational power of the particular model under observation. Accidental writability, however, seems to be a much more robust notion. It is not at all clear how any real could appear during a CRITTM computation and not during an ITTM computation since, as we have seen, CRITTM computations are more or less just ITTM computations with some irrelevant padding inserted.

We now turn to the CRITTM counterpart of γ, the supremum of the clockable ordinals:

$$\gamma_C = \sup\{\alpha; \alpha \text{ is CRITTM-clockable}\}$$

Welch shows in [3] that $\gamma = \lambda$. We intend to prove a somewhat similar statement in the context of CRITTMs.

[6] Checking whether a real codes an ordinal can be done using the count-through algorithm of [1].

Lemma 11. *If α is clockable then $\aleph_\alpha$ is CRITTM-clockable.*

In the proof we shall introduce an algorithm, based on an argument of [1], which we call the *cardinal step count-through algorithm* and which will be very useful in what follows.

Proof. Since α is clockable, we can write a real coding it in a countable number of steps. We now perform the count-through algorithm[7] of [1] on this code, but with a small modification: after each step of the algorithm our machine records the current head position and state in some way and waits for the next *cardinal* state, at which point it decodes the previous configuration and continues the algorithm. Of course, this modification requires keeping track of limit cardinal stages, when all previous information should be discarded, but this can be dealt with using the same bookkeeping devices that are used to keep track of limit-of-limits stages in ITTM computations. This CRITTM algorithm clocks $\aleph_\alpha$. □

Based on this lemma we can state that $\gamma_{\mathrm{C}} \geq \aleph_\gamma$. While the possibility of having $\gamma_{\mathrm{C}} = \aleph_\gamma$ is certainly alluring, this equality does not hold. We can easily use the cardinal step count-through algorithm to show that for each CRITTM-writable ordinal α there exists a CRITTM-clockable ordinal at least as large as $\aleph_\alpha$. Therefore we must have $\gamma_{\mathrm{C}} \geq \aleph_{\lambda_{\mathrm{C}}} > \aleph_\gamma$.

Since there is a CRITTM-clockable cardinal above any CRITTM-clockable ordinal (the cardinal successor works), γ_{C} must be a cardinal. We now proceed to determine exactly which cardinal it is.

Lemma 12. *Every CRITTM computation with input 0 repeats its $\aleph_{\zeta_{\mathrm{C}}}$ configuration at $\aleph_{\Sigma_{\mathrm{C}}}$.*

In the interest of brevity we omit the slightly technical proof. Let us just remark that the proof is a straightforward generalization of the arguments in [3] to the present context.

Theorem 13. $\gamma_{\mathrm{C}} = \aleph_{\lambda_{\mathrm{C}}}$

Proof. We have already seen that $\gamma_{\mathrm{C}} \geq \aleph_{\lambda_{\mathrm{C}}}$. Lemma 12 implies that every halting CRITTM computation with input 0 must halt before time $\aleph_{\zeta_{\mathrm{C}}}$. We therefore have $\gamma_{\mathrm{C}} \leq \aleph_{\zeta_{\mathrm{C}}}$. Consider a halting CRITTM program p. Begin enumerating accidentally CRITTM-writable ordinals α. For each α use the cardinal step count-through algorithm to simulate the computation of $\varphi_p(0)$ up to $\aleph_\alpha$, while simultaneously keeping track of the deleted initial segment β of α. Eventually an α will be enumerated which is large enough that the simulation halts before time $\aleph_\alpha$. When this happens, we halt with output $\beta + 1$. Therefore the program p halts by time $\aleph_{\beta'}$ for some writable β', which means that $\gamma_{\mathrm{C}} \leq \aleph_{\lambda_{\mathrm{C}}}$. Putting this together, we have shown that $\gamma_{\mathrm{C}} = \aleph_{\lambda_{\mathrm{C}}}$. □

[7] That is, we use the improved version of the algorithm, which counts through a code for α in α many steps.

Upon reflection, the properties of CRITTMs explored in this paper require very little of the structure of the class of cardinals. One could equally well have considered *A-recognizing* ITTMs for some club class of limit ordinals A. For example, we could have considered ITTMs with a state recognizing ordinal multiples of Σ. Some of our results would generalize, but note that we have often used the result that any ITTM computation halts or repeats before $\aleph_1$ and this property holds with Σ only for ITTM computations with empty input, so we cannot expect to get an equivalent model.

The generalization to arbitrary A leads to some familiar results (e.g., any computation in this model either halts or begins repeating before the ω_1-st element of A), but it also has serious issues. Since the class A may lack the uniformity of the classes of limit ordinals or cardinals, even the existence of a universal machine is not clear (if there is a change in the frequency of the A-stages and the machine cannot anticipate this, a running simulation may not produce correct results).

Question 14. What properties should the club class A have to get a meaningful notion of A-recognizing ITTMs?

Instead of studying them in isolation, we can also compare the A-recognizing ITTM models for various A. This leads to the concept of reductions between them. For example, any A-recognizing ITTM can simulate a limit ordinal recognizing ITTM (which is just an ordinary ITTM). This endows the collection of club classes of ordinals with a reducibility degree structure, similar to but distinct from the usual Turing degrees.

Question 15. What are the degrees thus obtained?

Acknowledgements. The author wishes to thank Joel David Hamkins for the many fruitful discussions and his suggestions of improvements to the paper.

The author's work was supported in part by the Slovene Human Resources and Scholarship Fund.

References

1. Hamkins, J.D., Lewis, A.: Infinite time Turing machines. Journal of Symbolic Logic 65(2), 567–604 (2000)
2. Koepke, P.: Turing computations on ordinals. Bulletin of Symbolic Logic 11(3), 377–397 (2005)
3. Welch, P.: The length of infinite time Turing computations. Bulletin of the London Mathematical Society 32(2), 129–136 (2000)

'Stored Program Concept' Considered Harmful: History and Historiography*

Thomas Haigh

School of Information Studies, University of Wisconsin–Milwaukee, Milwaukee, U.S.A.
thaigh@computer.org

Abstract. Historians agree that the stored program concept was formulated in 1945 and that its adoption was the most important single step in the development of modern computing. But the "concept" has never been properly defined, and its complex history has left it overloaded with different meanings. The paper surveys its use and development and attempts to separate it into three distinct aspects, each with its own history and each amenable to more precise definition.

1 Introduction

It is a truth universally agreed that implementation of the "stored program concept" in the late-1940s was the most important dividing line in computer history, separating modern computers from their less evolved predecessors. Historians also agree that the concept was first stated in the "First Draft of a Report on the EDVAC," (hereafter "First Draft") circulated under the name of John von Neumann in 1945 [14]. While the true balance of credit for the ideas contained in this document is widely and heatedly debated, its fundamental influence on the development of modern computing is not.

Yet when historian Doron Swade delivered an address [13] to celebrate the sixtieth anniversary of the 1949 EDSAC computer, generally considered the first useful stored program machine, he began with the startling observation we do not really agree on why the concept is so important. For years Swade had "assumed that the significance of the stored program must be self-evident" and attributed his own confusion to personal inadequacy. Eventually he "became bold and began asking" computer historians and pioneers to explain it. Their answers were "all different," with the question of whether "the primary benefit was one of principle or practice frustratingly blurred." Swade concluded that

> There was one feature of all the responses about which there was complete agreement: no one challenged the status of the stored program as

* This paper draws extensively on ideas and analysis developed during my ongoing collaboration with Mark Priestley and Crispin Rope on a project exploring the ENIAC's 1948 conversion to a new programming method and its use for the first computerized Monte Carlo calculations. In particular the definitions given of the "modern programming paradigm" were formulated during discussion with Priestley.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 241–251, 2013.

> the defining feature of the modern digital electronic computer.... While the reasons given for this were different, none discounted its seminal significance. But it seems that we struggle when required to articulate its significance in simple terms and the apparent mix of principle and practice frustrates clarity. The problem, I think, is that we have never actually agreed that the "stored program concept" is. The concept is sometimes treated as an approach to programming, sometimes as a new kind of architecture. Some authors conflate it with the idea of a universal machine and associate it with Turing's ideas on computability. Sometimes the idea is defined very narrowly, with attention to the interchangeable storage of programs and data, and sometimes as a grand cluster of ideas accumulated over time. Self-modifying code may or may not loom large in the discussion. Some authors use "stored program concept" and "von Neumann architecture" interchangeably, while others attempt to separate them.

As a historian, my instinct is to explain this proliferation of meanings and associations historically, going back to the conventional origin point of the stored program concept in 1945 and sketching its subsequent development in the hands of different groups of people with different intellectual agendas.

My own attention was drawn to the concept of "stored program" as we investigated modifications made to ENIAC in early 1948 by a team working closely with John von Neumann. These changes incorporated key elements of the stored-program approach several months before Manchester University's "Baby" computer executed what is usually called the world's first stored program in the summer of the same year. Confusion of the kind noted by Swade makes it impossible to clarify the status of the converted ENIAC, and several other important early machines, as "stored program" or "not stored program."

The final part of the paper attempts to begin the work of separating the cluster of ideas treated as part of the stored program concept by identifying three distinct paradigms stated in von Neumann's seminal 1945 report that were incorporated as standard features by most machines of the 1950s.

2 History of the Stored Program Concept

2.1 The 1945 EDVAC Report

The "First Draft of a Report on the EDVAC" does not, despite the role it has been assigned in later historical work, function very well as a standards document to rigorously define the concept of "stored program." Most notably the word "program" never appears. Von Neumann consistently preferred "code" to "program" and wrote of "memory" rather than "storage."

Our current attachment to the term "stored program" as a description for computers built along lines proposed for EDVAC thus needs some historical explanation. Read literally the term conveys very little. Any program that can be executed by a computer must be stored in some medium. The First Draft

itself observed that "instructions must be given in some form which the device can sense: Punched into a system of punchcards or on teletype tape, magnetically impressed on steel tape or wire, photographically impressed on motion picture film, wired into one or more fixed or exchangeable plugboards—this list being by no means necessarily complete." [14].

The First Draft argued for the collocation of code and data, though only tentatively: "While it appeared that various parts of this memory have to perform functions which differ somewhat in their purpose, it is nevertheless tempting to treat the entire memory as one organ, and to have its parts as interchangeable as possible..." [14]. Von Neumann believed that the data requirements of the problems he was interested in were large enough to require a memory of unprecedented size, and that the program code would, in comparison, be quite small. Using one set of mechanisms to manipulate both would simplify EDVAC.

2.2 Initial Reception

The First Draft circulated quickly between those interested in building computers, and almost immediately established the model for the next generation of computer project. However, early discussion of the EDVAC approach addressed a range of innovative features, with no consensus that the particular method of program storage was the most important one. Several early introductory computing books focused primarily on the method of programming, documenting the instruction set of von Neumann's planned computer at the Institute for Advanced Studies.

To many of the computer builders of the 1940s, including ENIAC creators J. Persper Eckert and John Mauchley whose contribution to the new design was profound, the advantage of what is sometimes called the "EDVAC-type" approach was seen primarily as a way of building a powerful and flexible computer with a relatively small number of expensive and unreliable vacuum tubes. Creation of a large high-speed memory was the key engineering challenge posed by this approach. Ease of programming and speed of changeover from one problem to another were acknowledged as benefits of this approach, and were related to the storage of code and data in the same electronic memory.

2.3 The Phrase "Stored Program"

We could not find the phrase "stored program in any of the early computing conference proceedings and primers published in the 1940s. Its earliest known usage occurred in 1949 by a small team at IBM's Poughkeepsie facility producing the "Test Assembly," IBM's first EDVAC-type computer. This experimental system was built around firm's first electronic calculator, its 604 Electronic Calculating Punch, which became the arithmetic unit of the lashed-up computer. They added a new control unit, cathode ray tube memory, and magnetic drum.

An internal proposal written by Nathaniel Rochester in 1949 [11] noted that the plug-board approach was not viable with large programs, which could be

solved by reading "the calculating program into the machine on a deck of tabulating cards and to retain it, along with the numerical data, in the storage section of the calculator." To distinguish between the program held on the 604's standard plug board and the new-style programs stored either in the 250 word electronic memory or on the drum the team began to call the latter the "stored program." Rochester's document was titled "A Calculator Using Electrostatic Storage and a Stored Program." (Thanks go to Peggy Kidwell of the Smithsonian for alerting me to this document).

Within IBM the meaning of "stored program" quickly evolved from a literal description of a particular kind of programming mechanism into a general description of EDVAC-type machines. During the 1950s the phrase pops up occasionally in conference papers, particularly those delivered by IBM employees, but as all powerful digital computers by then followed this model people generally just referred to "large-scale digital computer."

2.4 Historians Adopt "Stored Program"

In the late 1970s and 1980s the history of computing emerges as an academic subfield. Early work focused on the machines of the 1940s. Following the lead of pioneer turned historian Herman Goldstine the idea of the "stored program computer" was borrowed from technical discourse, developing from a fairly obscure term into a central concept in heated debates over what should be considered the first true computer and why.

Put simply, the community resolved this intractable and distracting wrangle by agreeing on a string of words that clarified what each of the key early machines had accomplished. ENIAC, for example, became the first "large-scale general-purpose digital electronic computer to be fully operational." The Cambridge EDSAC and Manchester Baby were recognized as the first operational stored program computers, patterned after the EDVAC design, but there was little need or interest in defining the "concept" more rigorously to identify its necessary and sufficient characteristics.

More recently, historians of computing largely turned their attention away from the 1940s, having reached a consensus on the honors to be granted to each early machine and formulated a consensus narrative on the events of the decade. As Swade noted, the stored program concept is still enshrined in popular and scholarly histories as a key transition but receives little new analysis.

2.5 The Stored Program Concept Meets Universality

From the late 1950s onward, with the rise of theoretical computer science and a new enthusiasm for work on abstract models of computation, the digital computer was increasingly reinterpreted as an embodiment of the universal Turing machine. Its key benefit was therefore the ability of the computer to treat instructions as data and modify them programmatically. The confusion encountered by Swade seems to reflect differences in opinion held by those influenced by the pragmatism of the 1940s and those favoring the theoretical concerns developed

later. The resurgence of the stored program concept, now as a concept for historical discussion, went along with its increasing identification with foundational ideas from the new discipline of computer science.

In recent decades the manipulation of programs and data interchangeably in the same memory units has increasingly been taken as the key defining characteristic of the stored program computer, and thus of modern computers. In turn, the concepts of stored program and general purpose computer have sometimes conflated with the more formal concept a computer being Turing complete or "universal" if equipped with a memory of infinite size. To cite just three of many recent examples of this conventional wisdom: the Wikipedia page on "stored program computer" currently defines it as "one which stores program instructions in electronic memory. Often the definition is extended with the requirement that the treatment of programs and data in memory be interchangeable... the stored program computer idea can be traced back to the 1936 theoretical concept of a universal Turing machine." In his recent *Computing: A Concise History* [5], Paul Ceruzzi defined stored program computers as storing "both their instructions—the programs—and the data on which those instructions operate in the same physical memory device..." and suggested that this "extended Turing's ideas into the design of practical machinery." Even Swade himself retreated from the endearingly bold confession of confusion quoted earlier to the rather conventional conclusion that "the internal stored program... is the practical realization of Turing universality" and thus conferred "plasticity of function, which in large part accounts for the remarkable proliferation of computers and computer-like artifacts." [13]

Arguing about the influence Turing might or might not have exerted over von Neumann has become an enjoyable parlor game for historians of computing. That question aside, one will find very few references to Turing's theoretical work among the discussions of those building computers in the 1940s [10,8]. Atsushi Akera [1] has suggested that the retroactive embrace of Turing as a foundation for this practical work is tied to the emphasis within computer science, as it emerged as a distinct discipline during the late-1950s and 1960s, on abstract models of computation. In later discussion the advantages of stored program machines were often justified according to the theoretical concerns of later years rather than the pragmatic issues of primary importance to their designers.

In this sense, the search for the logical foundations of computing and the search for is historical foundations may pull us in opposing directions. It was its very purity and abstraction from the messy details of actual hardware that earned the Turing Machine its iconic place within computer science. The ideas of Turing completeness and of the Universal Turing Machine served to decouple theoretical computer science from the material world of computing platforms and architectures. For example, once a virtual computer built within Conway's Game of Life was shown to be computationally equivalent to a Universal Turing Machine that single fact told us that, with sufficient time and a large enough cellular matrix, this computer could execute the same algorithms as any machine build from conventional components. The intellectual utility of this approach is

clear, as is its strategic benefit to the early computer science community at a time in which it was struggling to separate itself intellectually from mathematics, electronic engineering, and scientific service work to other disciplines.

The late Michael Mahoney struggled for many years to encapsulate the history of theoretical computer science [9]. His great theme was the need of scientific communities to construct their own historical narratives. Mahoney saw theoretical computer science as an assemblage of mathematical tools originally developed in quite separate contexts, from group theory and Lambda calculus through to Chomsky's hierarchy of grammars. On a still larger scale, work on mathematical logic and on the engineering of calculating machines both had long but largely distinct histories. Yet from within the discipline and from the present-day viewpoint the connections between these things came to seem obvious and history is often written as if work over the centuries had been directed towards the development of the computer or as if the computing pioneers of the 1940s were inspired primarily by the work of Turing.

As Mahoney wrote, the interest of practitioners in "finding a history... has its real dangers" because while scholarly historians and practitioners "both seek a history" it is "not for the same purpose and not from the same standpoint." [9] Abstraction is the soul of computer science, but as historians we lose something vital if we abstract away from the historical grubbiness of early computer projects, their focus on engineering challenges, their specific goals and roots in the thinking of the 1940s. The abstraction from real computers and real computing practice provide by a focus on Turing completeness is good for theory but bad for history. For example, the effort by Raul Rojas to claim Konrad Zuse's 1943 Z3 computer as universal [12] is an impressive party trick, but diverges entirely from the way in which the machine was designed, how it was actually used, or indeed from anything that would have made sense in the 1940s. The programming method described construction of an impossibly long paper tape, and a massive loss of computational performance. Calculation would have been quicker by hand. To me the real lesson is that the Z3 could have been Turing complete with only minor design changes, but wasn't because the concept and its benefits were not yet widely understood. Indeed, Zuse later claimed to have considered and rejected treating program instructions as data while working on its design. The past really is a foreign country. Yet Rojas was able to raise the status of the machine, and German pride, with this appeal to the world of theory.

In this context, it is worth noting that von Neumann's 1945 report specifically forbade unrestricted code modification, a notable conceptual divergence from what is now understood as the Turing machine model of the universal computer. The EDVAC described therein did rely on code modification for many common operations, including loop termination, other kinds of conditional branching, and altering the address data is fetched from (for example to obtain a value from a different cell within an array each time a block of code is looped through). This reflected a broader design philosophy of radically simplifying computer architecture by replacing the special purpose mechanisms common in earlier designs with a small number of general purpose mechanisms.

However, von Neumann's instruction set for EDVAC explicitly prevented instructions from being fully overwritten [10,6]. Only address fields could be changed. One of the 32 bits in each word of memory flagged it as holding either program or data. A transfer operation applied to an instruction word would overwrite only the address field.

3 Beyond "Stored Program"

Like the Goto statement, discussion of the "stored program concept" has outlived the purpose for which it was created and provides a shortcut to confusion. The time has come to replace it, as an analytical category, with a set of more specific alternatives amenable to clear and precise definition. Below I discuss one proposed partial replacement in some detail and sketch two more.

3.1 The Modern Code Paradigm

The first of these is the "modern code paradigm." This new term describes the program-related elements of the 1945 "First Draft..." design that become standard features of 1950s computer design. Some items specified in the report were ignored or changed by actual computer designers (such as the lack of a dedicated conditional branch instruction), while some common code capabilities of 1950s computers (such as index registers) came from other sources.

Looking for novel code-related features from the 1945 First Draft that had become taken-for-granted features of computers a decade later illuminates the process by which a sprawling, idiosyncratic and brilliant document became a dominant paradigm for the builders of computers.

As one reviewer of this paper noted, this analysis parallels the work of computing theorists to build abstract models of computation based around the stored program approach rather than Turing machines. These include the Random Access Machine (RAM) and Random Access Stored Program machine (RASM). Space does not permit further comment, except to point out that the objectives of the historian and the theorist remain distinct. Whereas the theorist looks for the minimal necessary capabilities for universality, my objective here is to define the maximal set of features present in the 1945 draft that actually made it into the standard designs of the 1950s.

1. **The program is executed completely automatically.** To quote the First Draft, "Once these instructions are given to the device, it must be able to carry them out completely and without any need for further intelligent human intervention." This was essential for electronic machines, whereas manual intervention at branch points had been workable with slower devices such as the Harvard Mark I.
2. **The program is written as a single sequence of instructions, known as "orders" in the First Draft, which are stored in numbered memory locations along with data.** These instructions control all aspects of

the machine's operations. The same mechanisms are used to read code and data. As discussed earlier, the First Draft did specify the explicit demarcation of memory locations holding code from those holding data. It also pointed toward the idea of a program as a readable text: "it is usually convenient that the minor cycles expressing the successive steps in a sequence of logical instructions should follow each other automatically."

3. **Each instruction within the program specifies one of a set of atomic operations made available to the programmer.** This was usually done by beginning the instruction with one of a small number of operation codes. Some operation codes are followed by argument fields specifying a memory location with which to work or other parameters. EDVAC orders would have required between 9 and 22 bits to express. Actual machines usually followed this pattern. The main exception comes with Alan Turing's Ace design and its derivatives, which stuck close to the underlying hardware by coding all instructions as data transfers between sources and destinations.
4. **The program's instructions are usually executed in a predetermined sequence.** According to the First Draft, the machine "should be instructed, after each order, where to find the next order that it is to carry out." In the EDVAC this was to be represented implicitly by the sequence in which they were stored, as in "normal routine" it "should obey the orders in the temporal sequence in which they naturally appear."
5. **However, a program can instruct the computer to depart from this ordinary sequence and jump to a different point in the program.** "There must, however, be orders available which may be used at the exceptional occasions referred to, to instruct CC to transfer its connection [i.e., fetch the next instruction from] any other desired point" in memory." This provided capabilities such as jumps and subroutine returns.
6. **The address on which an instruction acts can change during the course of the program's execution.** That applies to the source or destination of data for calculations or the destination of a jump. This address modification capability was expressed rather cryptically in the First Draft, the final sentence of which noted that when a number was transferred to a memory location holding an instruction only the final thirteen digits, representing the address $\mu\varrho$, should be overwritten. Actual computers achieved functionally equivalent capability through some combination of unrestricted code modification, indirect addressing mechanisms, and conditional branch instructions.

A consequence of the above was that the logical complexity of the program was limited only by memory space available to hold instructions and working data. This contrasted with the dependence of machines such as the original ENIAC or SSEC on a variety of resources such as program lines, plug board capacity, or tape readers as potential limitations on logical program complexity.

3.2 The von Neumann Architecture

The modern code paradigm is not intended a new name for the "stored program concept" or as an idea encompassing the full scope of meanings associated with the latter. Indeed, the more specific scope of the former is a large part of its appeal. There were clearly several other aspects of the First Draft and subsequent publications by members of von Neumann's group in Princeton that had a major influence on later computer builders. To adapt an existing term, one of these facets might be called the "von Neumann architectural paradigm." This includes the basic structure of "organs" found in the report, including the separation of memory from control and arithmetic. Associated with this are the serialization of computation, so that only one operation takes place at a time, and the routing of all memory transfers through the central arithmetic unit. Also the system of special purpose registers to serve as source and destination for arithmetic and logic instructions, and to provide a program counter and instruction register for control purposes. The von Neumann architecture has general been more clearly defined within the technical literature than has the stored program concept. One might, as several have, dispute the extent to which it is fair to attach only von Neumann's name to these concepts. "EDVAC architecture paradigm" could serve as an alternative.

3.3 The EDVAC Hardware Paradigm

The third major facet might be termed the "EDVAC hardware paradigm." The EDVAC approach appealed to early computer builders in large part as a way of building powerful, flexible machines using a relatively small number of components. Influential hardware ideas in the "First Draft" report include use of delay line or storage tube memory, building logic entirely from electronic components, representing all quantities in binary, and keeping special purpose or duplicate hardware mechanisms to a minimum (von Neumann considered that a multiplier would justify itself, but that duplicating adders or providing hardware for more specialized functions would provide little benefit). With the possible exception of the memory technologies discussed these hardware features were not unique, but collectively represented a bold commitment to new technologies at a time when computing group within Harvard, Bell Labs, and IBM were still drawing up plans for new high-end machines based on relay storage and paper tape control. Thus we believe that the hardware choices specified for EDVAC in the First Draft function as a paradigm, in Thomas Kuhn's core sense of a powerful and tangible exemplar [7].

3.4 Separate Trajectories

These three paradigms have intertwined early histories, but were always at least partially separable and ultimately diverged. Many machines of the 1940s implemented some aspects of the EDVAC paradigms but not others. Alan Booth's

ARC followed both the modern code paradigm and the von Neumann architecture but implemented them using relay hardware. Martin Campbell-Kelly observed that Booth's claimed operation date of 12th May 1948 would make this "the first operational EDVAC-type stored program computer (although it was not of course electronic)." [3] Alan Turing's design for the ACE adopted von Neumann's architecture and followed EDVAC's hardware paradigm but relied on a different kind of instruction format with no conventional operation codes. As Campbell-Kelly noted, "Most computers are sufficiently alike that a knowledgeable programmer can get a fairly good appreciation of a machine from its instruction format and a table of operation codes. The Pilot ACE is an exception because its architecture was quite unlike that of any modern computer...." [4] ENIAC after its 1948 conversion followed the modern code paradigm with surprising faithfulness. The feel and structure of its program code bears an unmistakable kinship with those produced for other early machines.

The machines of the mid-1950s tended to implement all three paradigmatic aspects of the First Draft's design for EDVAC. The paradigmatic influences of these three were diverging again by the end of the decade. Its relevance as a hardware paradigm faded first, as transistors and core memories made vacuum tubes and delay lines obsolete. The von Neumann architectural paradigm enjoyed a longer life, though its primacy was gradually chipped away as innovations such as parallel processing, message passing interfaces, instruction pipelining, direct memory access by peripherals, stacks, and addressable registers gradually erased its radical minimalism. In contrast the modern code paradigm has remained largely intact as a description of the machine language executed by processors (though not of the languages used by humans to write programs). It was extended and made more specific in many ways, not least by von Neumann's own 1946 description of the planned structure of his Institute for Advanced Studies machine [2]. It was not, however, overturned.

References

1. Akera, A.: Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research. MIT Press, Cambridge (2006)
2. Burks, A.W., Goldstine, H.H., von Neumann, J.: Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. Institute for Advanced Studies, Princeton (1946)
3. Campbell-Kelly, M.: Foundations of Computer Programming in Britain (1945-1955). Ph.D. thesis, Mathematics and Computer Studies, Sunderland Polytechnic (1980)
4. Campbell-Kelly, M.: Programming the Pilot Ace: Early Programming Activity at the National Physical Laboratory. Annals of the History of Computing 3, 133–162 (1981)
5. Ceruzzi, P.: Computing: A Concise History. MIT Press, MA (2012)
6. Godfrey, M.D., Hendry, D.F.: The Computer as von Neumann Planned It. IEEE Annals of the History of Computing 15, 11–21 (1993)
7. Kuhn, T.S.: Second Thoughts on Paradigms. The Essential Tension: Selected Studies in Scientific Tradition and Change, pp. 293–319. University of Chicago Press, Chicago (1979)

8. Lavington, S. (ed.): Alan Turing and his Contemporaries. British Informatics Society Ltd., Swindon (2012)
9. Mahoney, M.S., Haigh, T. (eds.): Histories of Computing. Harvard University Press, Cambridge (2011)
10. Priestley, M.: A Science of Operations: Machines, Logic, and the Invention of Programming. Springer, New York (2011)
11. Rochester, N.: A Calculator Using Electrostatic Storage and a Stored Program. IBM's Early Computers Sources collection. IBM Archives (1949)
12. Rojas, R.: How to Make Zuse's Z3 a Universal Computer. IEEE Annals of the History of Computing 20, 51–54 (1998)
13. Swade, D.: Inventing the User: EDSAC in Context. The Computer Journal 54, 143–147 (2011)
14. von Neumann, J.: First Draft of a Report on the EDVAC. IEEE Annals of the History of Computing 15, 27–75 (1993)

The Complexity of Interior Point Methods for Solving Discounted Turn-Based Stochastic Games*

Thomas Dueholm Hansen[1,2] and Rasmus Ibsen-Jensen[1]

[1] Department of Computer Science, Aarhus University, Aarhus, Denmark
{tdh,rij}@cs.au.dk
[2] Department of Computer Science, Tel Aviv University, Tel Aviv, Israel

Abstract. We study the problem of solving discounted, two player, turn based, stochastic games (2TBSGs). Jurdziński and Savani showed that in the case of deterministic games the problem can be reduced to solving P-matrix linear complementarity problems (LCPs). We show that the same reduction also works for general 2TBSGs. This implies that a number of interior point methods can be used to solve 2TBSGs. We consider two such algorithms: the unified interior point method of Kojima, Megiddo, Noma, and Yoshise, and the interior point potential reduction algorithm of Kojima, Megiddo, and Ye. The algorithms run in time $O((1+\kappa)n^{3.5}L)$ and $O(\frac{-\delta}{\theta}n^4 \log \epsilon^{-1})$, respectively, when applied to an LCP defined by an $n \times n$ matrix M that can be described with L bits, and where the potential reduction algorithm returns an ϵ-optimal solution. The parameters κ, δ, and θ depend on the matrix M. We show that for 2TBSGs with n states and discount factor γ we get $\kappa = \Theta(\frac{n}{(1-\gamma)^2})$, $-\delta = \Theta(\frac{\sqrt{n}}{1-\gamma})$, and $1/\theta = \Theta(\frac{n}{(1-\gamma)^2})$ in the worst case. The lower bounds for κ, $-\delta$, and $1/\theta$ are all obtained using the same family of deterministic games.

1 Introduction

Two-Player Turn-Based Stochastic Games (2TBSGs). A *two-player turn-based stochastic game* (2TBSG) is a game played by two players (Player 1 and Player 2) on a finite state graph for an infinite number of rounds. The graph is partitioned into two sets of states S^1 (belonging to Player 1) and S^2 (belonging to Player 2). Whenever the current state i is from S^k, Player k chooses an action a emanating from state i, and the next state is then given by a probability

* Supported by the Sino-Danish Center for the Theory of Interactive Computation, funded by the Danish National Research Foundation and the National Science Foundation of China (under the grant 61061130540); and by the Center for research in the Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council. Thomas Dueholm Hansen is a recipient of the Google Europe Fellowship in Game Theory, and this research was supported in part by this Google Fellowship; he was also supported in part by The Danish Council for Independent Research | Natural Sciences (grant no. 12-126512).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 252–262, 2013.

distribution, depending on a. In each round there is a probability of $1-\gamma > 0$ of ending the game, where γ is the *discount factor* of the game. Every action has an associated cost. The objective of Player 1 is to *minimize* the expected sum of costs, and the objective of Player 2 is to *maximize* the expected sum of costs, i.e., the game is a zero-sum game. Our results will be for the case when all states have 2 actions.

The class of (turn-based) stochastic games was introduced by Shapley [20] in 1953, and it has received much attention over the following decades. For books on the subject, cf., e.g., Neyman and Sorin [17] and Filar and Vrieze [5]. Shapley showed that states in such games have a *value* that can be enforced by both players (*determinacy*). We shall in this paper consider the problem of *solving* such games, that is, for each state i finding the value of that state.

Classical Algorithms for Solving 2TBSGs. 2TBSGs form an intriguing class of games whose status in many ways resembles that of linear programming 40 years ago. They can be solved efficiently with *strategy iteration* algorithms, resembling the simplex method for linear programming, but no polynomial time algorithm is known. Strategy iteration algorithms were first described by Rao *et al.* [18]. Hansen, Miltersen, and Zwick [9] recently showed that the standard strategy iteration algorithm solves 2TBSGs with a fixed discount, γ, in *strongly* polynomial time. Prior to this result a polynomial bound by Littman [16] was known for the case when γ is fixed. Littman showed that Shapley's [20] *value iteration* algorithm can be used to solve discounted 2TBSGs in time $O(\frac{nmL}{1-\gamma} \log \frac{1}{1-\gamma})$, where n is the number of states, m is the number of actions, and L is the number of bits needed to represent the game. For a more thorough introduction to the background of the problem we refer to Hansen *et al.* [9] and the references therein.

Interior Point Methods. One may hope that a polynomial time algorithm for solving 2TBSGs in the general case, when the discount factor γ is not fixed (i.e., when it is given as part of the input), can be obtained through the use of *interior point methods*. This was also suggested by Jurdziński and Savani [11] and Hansen *et al.* [9]. The first interior point method was introduced by Karmarkar [12] in 1984 to solve linear programs in polynomial time. Since then the technique has been studied extensively and applied in other contexts. Cf., e.g., Ye [21]. In particular, interior point methods can be used to solve *P-matrix linear complementarity problems*, which, in turn, can be used to solve 2TBSGs. This will be the focus of the present paper.

***P*-Matrix Linear Complementarity Problems (LCPs).** A linear complementarity problem (LCP) is defined as follows: Given an $(n \times n)$-matrix M and a vector $\mathbf{q} \in \mathbb{R}^n$, find two vectors $\mathbf{w}, \mathbf{z} \in \mathbb{R}^n$, such that $\mathbf{w} = \mathbf{q} + M\mathbf{z}$ and $\mathbf{w}^\mathsf{T}\mathbf{z} = 0$ and $\mathbf{w}, \mathbf{z} \geq \mathbf{0}$. LCPs have also received much attention. For books on the subject cf., e.g., Cottle *et al.* [4] and Ye [21].

Jurdziński and Savani [11] showed that solving a *deterministic* 2TBSG G, i.e., every action leads to a single state with probability 1, can be reduced to solving an LCP $(M, \mathbf{q})$. Gärtner and Rüst [6] gave a similar reduction from simple

stochastic games; a class of games that is polynomially equivalent to 2TBSGs (cf. [1]). Moreover, Jurdziński and Savani [11], and Gärtner and Rüst [6], showed that the resulting matrix M is a *P-matrix* (i.e., all principal sub-matrices have a positive determinant). We show that the reduction of Jurdziński and Savani also works for general 2TBSGs, and that the resulting matrix M is again a P-matrix.

Krishnamurthy *et al.* [15] recently gave a survey on various stochastic games and LCP formulations of those.

The Unified Interior Point Method. There exist various interior point methods for solving P-matrix LCPs. The algorithm that we consider in this extended abstract is the unified interior point method of Kojima, Megiddo, Noma, and Yoshise [13]. The unified interior point method solves an LCP whose matrix $M \in \mathbb{R}^{n\times n}$ is a $P_*(\kappa)$-*matrix* in time $O((1+\kappa)n^{3.5}L)$, where L is the number of bits needed to describe M. A matrix M is a $P_*(\kappa)$-matrix, for $\kappa \geq 0$, if and only if for all vectors $\mathbf{x} \in \mathbb{R}^n$, we have that $\mathbf{x}^\mathsf{T}(M\mathbf{x}) + 4\kappa \sum_{i\in\delta_+(M)} \mathbf{x}_i(M\mathbf{x})_i \geq 0$, where $\delta_+(M) = \{i \in [n] \mid \mathbf{x}_i(M\mathbf{x})_i > 0\}$. If M is a P-matrix then it is also a $P_*(\kappa)$-matrix for some $\kappa \geq 0$. Hence, the algorithm can be used to solve 2TBSGs.

Following the work of Kojima *et al.* [13], many algorithms with complexity polynomial in κ, L, and n have been introduced. For recent examples, cf., e.g., [3,2,10].

An Interior Point Potential Reduction Algorithm. In the full version of the paper [8] we also consider a second interior point method: the potential reduction algorithm of Kojima, Megiddo, and Ye [14]. See also Ye [21]. The potential reduction algorithm is an interior point method that takes as input a P-matrix LCP and a parameter $\epsilon > 0$, and produces an approximate solution $\mathbf{w}, \mathbf{z}$, such that $\mathbf{w}^\mathsf{T}\mathbf{z} < \epsilon$, $\mathbf{w} = \mathbf{q} + M\mathbf{z}$, and $\mathbf{w}, \mathbf{z} \geq \mathbf{0}$. The running time of the algorithm is $O(\frac{-\delta}{\theta} n^4 \log \epsilon^{-1})$, where δ is the least eigenvalue of $\frac{M+M^\mathsf{T}}{2}$, and θ is the positive P-matrix number of M, that is, $\theta = \min_{\|\mathbf{x}\|_2=1} \max_{i\in\{1,\ldots,n\}} \mathbf{x}_i(M\mathbf{x})_i$. We refer to $\frac{-\delta}{\theta}$ as the condition number of M. The analysis involving the condition number appears in Ye [21].

In his ph.d. thesis, Rüst [19] shows that there exists a simple stochastic game for which the P-matrix LCPs resulting from the reduction of Gärtner and Rüst [6] has a large condition number. The example of Rüst contains a parameter that can essentially be viewed as the discount factor γ for 2TBSGs, and he shows that the condition number can depend linearly on $\frac{1}{1-\gamma}$. To be more precise, Rüst [19] shows that the matrix M resulting from the reduction of Gärtner and Rüst [6] has positive P-matrix number smaller than 1, and that the smallest eigenvalue of the matrix $\frac{M+M^\mathsf{T}}{2}$ is $-\Omega(\frac{1}{1-\gamma})$. This bound can be viewed as a precursor for some of our results.

1.1 Our Contributions

Our contributions are as follows. We show that the reduction by Jurdziński and Savani [11] from deterministic 2TBSGs to P-matrix LCPs generalizes to 2TBSGs without modification. Although the reduction is the same we provide

an alternative proof that the resulting matrix is a P-matrix. Furthermore, let G be any 2TBSG with n states and let M_G be the matrix obtained from the reduction of Jurdziński and Savani [11].

(*i*) We show that M_G is a $P_*(\kappa)$-matrix for $\kappa = \frac{n}{(1-\gamma)^2}$. This implies that the running time of the unified interior point method of Kojima *et al.* [13] for 2TBSGs is at most $O(\frac{n^{4.5}L}{(1-\gamma)^2})$. We also show that there exists a family of 2TBSGs, G_n, such that the corresponding matrices, M_{G_n}, are not $P_*(\kappa)$-matrices for $\kappa = \Omega\big(\frac{n}{(1-\gamma)^2}\big)$.

(*ii*) We show that the matrix $\frac{M_G + M_G^\intercal}{2}$ has smallest eigenvalue at least $-O\big(\frac{\sqrt{n}}{1-\gamma}\big)$. We also show that there exists a family of 2TBSGs, G_n, such that the corresponding matrices $\frac{M_{G_n} + M_{G_n}^\intercal}{2}$ have smallest eigenvalue less than $-\Omega\big(\frac{\sqrt{n}}{1-\gamma}\big)$.

(*iii*) We show that the positive P-matrix number $\theta(M_G)$ is at least $\Omega\big(\frac{(1-\gamma)^2}{n}\big)$. We also show that there exists a familiy of 2TBSGs, G_n, such that the corresponding matrices M_{G_n} have positive P-matrix number, $\theta(M_{G_n})$, at most $O\big(\frac{(1-\gamma)^2}{n}\big)$.

The results in (*ii*) and (*iii*) have been deferred to the full version of the paper [8]. They together imply that the running time of the potential reduction algorithm of Kojima *et al.* [14] for 2TBSGs is at most $O(\frac{n^{5.5}\log\epsilon^{-1}}{(1-\gamma)^3})$. The familiy of 2TBSGs G_n mentioned in (*i*), (*ii*), and (*iii*) is, in fact, the same. Hence, we get matching upper and lower bounds for the parameters of both the unified interior point method and the potential reduction algorithm. Also, the games G_n are deterministic, so the same lower bounds hold for deterministic 2TBSGs.

It should be noted that although our results for existing interior point methods for solving 2TBSGs are negative, it is still possible that other (possibly new) interior point methods can solve 2TBSGs efficiently. In fact, we believe that this remains an important question for future research.

1.2 Overview

In Section 2 we formally introduce the various classes of problems under consideration. More precisely, in Subsection 2.1 we define LCPs, and in Subsection 2.2 we define 2TBSG and give the reduction from 2TBSGs to P-matrix LCPs. In Section 3 we estimate the κ for which the matrices of 2TBSGs are $P_*(\kappa)$-matrices, thus, giving a bound on the running time of Kojima *et al.*'s unified interior point method [13]. The studies of smallest eigenvalues and positive P-matrix numbers have been deferred to the full version of the paper [8]. Thus, we do not present the bound on the running time of the potential reduction algorithm of Kojima *et al.* [14] in this extended abstract.

2 Preliminaries

2.1 Linear Complementarity Problems

Definition 1 (Linear complementarity problems). *A* linear complementarity problem *(LCP) is a pair* $(M, \mathbf{q})$*, where* M *is an* $(n \times n)$*-matrix and* $\mathbf{q}$ *is an* n*-vector. A* solution *to the LCP* $(M, \mathbf{q})$ *is a pair of vectors* $(\mathbf{w}, \mathbf{z}) \in \mathbb{R}^n$ *such that* $\mathbf{w} = \mathbf{q} + M\mathbf{z}$ *and* $\mathbf{w}^\intercal \mathbf{z} = 0$ *and* $\mathbf{w}, \mathbf{z} \geq \mathbf{0}$*.*

We shall now define various types of matrices for which interior point methods are known to solve the corresponding LCPs.

Definition 2 (P-matrix). *A matrix* $M \in \mathbb{R}^{n \times n}$ *is a* P-matrix *if and only if all principal sub-matrices have a positive determinant.*

The following lemma gives an alternative definition of P-matrices (cf., e.g., [4, Theorem 3.3.4]).

Lemma 1. *A matrix* $M \in \mathbb{R}^{n \times n}$ *is a* P-matrix *if and only if for all* n*-vectors* $\mathbf{x} \neq \mathbf{0}$ *there is an* $i \in [n]$ *such that* $\mathbf{x}_i (M\mathbf{x})_i > 0$*.*

Definition 3 (Positive P-matrix number). *For a matrix* $M \in \mathbb{R}^{n \times n}$*, the positive* P*-matrix number is* $\theta(M) = \min_{\|\mathbf{x}\|_2 = 1} \max_{i \in [n]} \mathbf{x}_i (M\mathbf{x})_i$*.*

According to Lemma 1, $\theta(M) > 0$ if and only if M is a P-matrix.

Definition 4 ($P_*(\kappa)$-matrix). *A matrix* $M \in \mathbb{R}^{n \times n}$ *is a* $P_*(\kappa)$-matrix*, for* $\kappa \geq 0$*, if and only if for all vectors* $\mathbf{x} \in \mathbb{R}^n$*, we have that*

$$\sum_{i \in \delta_-} \mathbf{x}_i (M\mathbf{x})_i + (1 + 4\kappa) \sum_{i \in \delta_+} \mathbf{x}_i (M\mathbf{x})_i \geq 0 \ ,$$

where $\delta_- = \{i \in [n] \mid \mathbf{x}_i (M\mathbf{x})_i < 0\}$ *and* $\delta_+ = \{i \in [n] \mid \mathbf{x}_i (M\mathbf{x})_i > 0\}$*. We say that* M *is a* P_*-matrix *if and only if it is a* $P_*(\kappa)$*-matrix for some* $\kappa \geq 0$*.*

Kojima *et al.* [13] showed that every P-matrix is also a P_*-matrix. By definition, a matrix M is a $P_*(0)$-matrix if and only if it is *positive semi-definite*. The set of symmetric P-matrices is exactly the set of positive semi-definite matrices.

2.2 Two-Player Turn-Based Stochastic Games

Definition 5 (Two-player turn-based stochastic games). *A* two-player turn-based stochastic game *(2TBSG) is a tuple,* $G = (S^1, S^2, (A_i)_{i \in S^1 \cup S^2}, p, c, \gamma)$*, where*

- S^k*, for* $k \in \{1, 2\}$*, is the set of* states *belonging to Player* k*. We let* $S = S^1 \cup S^2$ *be the set of all states, and we assume that* S^1 *and* S^2 *are disjoint.*
- A_i*, for* $i \in S$*, is the set of* actions *applicable from state* i*. We let* $A = \bigcup_{i \in S} A_i$ *be the set of all actions. We assume that* A_i *and* A_j *are disjoint for* $i \neq j$*, and that* $A_i \neq \emptyset$ *for all* $i \in S$*.*

- $p : A \to \Delta(S)$ *is a map from actions to probability distributions over states.*
- $c : A \to \mathbb{R}$ *is a function that assigns a* cost *to every action.*
- $\gamma < 1$ *is a (positive)* discount factor*.*

We let $n = |S|$ and $m = |A|$. Furthermore, we let $A^k = \bigcup_{i \in S^k} A_i$, for $k \in \{1, 2\}$.

We say that an action a is *deterministic* if it moves to a single state with probability 1, i.e., if $p(a)_j = 1$ for some $j \in S$. If all the actions of a 2TBSG G are deterministic we say that G is deterministic.

Plays and Outcomes. A 2TBSG is played as follows. At the beginning of a *play* a pebble is placed on some state $i_0 \in S$. Whenever the pebble is moved to a state $i \in S^k$, Player k chooses an action $a \in A_i$ and the pebble is moved at random according to the probability distribution $p(a)$ to a new state j. Let a^t be the t'th chosen action for every $t \geq 0$. Then the *outcome* of the play, paid by Player 1 to Player 2 is $\sum_{t \geq 0} \gamma^t \cdot c(a^t)$.

We shall now give a way to explicitly represent a 2TBSG using vectors and matrices. It will later simplify the notation in our constructions and proofs.

Definition 6 (Matrix representation). *Let $G = (S^1, S^2, (A_i)_{i \in S^1 \cup S^2}, p, c, \gamma)$ be a 2TBSG. Assume WLOG that $S = [n] = \{1, \ldots, n\}$ and $A = [m] = \{1, \ldots, m\}$.*

- *We define the* probability matrix $P \in \mathbb{R}^{m \times n}$ *by $P_{a,i} = (p(a))_i$, for all $a \in A$ and $i \in S$.*
- *We define the* cost vector $\mathbf{c} \in \mathbb{R}^m$ *by $\mathbf{c}_a = c(a)$, for all $a \in A$.*
- *We define the* source matrix $J \in \{0, 1\}^{m \times n}$ *by $J_{a,i} = 1$ if and only if $a \in A_i$, for all $a \in A$ and $i \in S$.*
- *We define the* ownership matrix $\mathcal{I} \in \{-1, 0, 1\}^{n \times n}$ *by $\mathcal{I}_{i,j} = 0$ if $i \neq j$, $\mathcal{I}_{i,i} = -1$ if $i \in S^1$, and $\mathcal{I}_{i,i} = 1$ if $i \in S^2$.*

The entry $P_{a,i}$ is the probability of moving to state i when using action a. For a matrix $P \in \mathbb{R}^{m \times n}$ and a subset of indices $B \subseteq [m]$, we let P_B be the submatrix of P consisting of rows with indices in B. Also, for any $i \in [m]$, we let $P_i \in \mathbb{R}^{1 \times n}$ be the i-th row of P. We use similar notation for vectors.

Definition 7 (Strategies and strategy profiles). *A* strategy $\sigma^k : S^k \to A^k$ *for Player $k \in \{1, 2\}$ maps every state $i \in S^k$ to an action $\sigma^k(i) \in A_i$ applicable from state i. A* strategy profile $\sigma = (\sigma^1, \sigma^2)$ *is a pair of strategies, one for each player. We let Σ^k be the set of strategies for Player k, and $\Sigma = \Sigma^1 \times \Sigma^2$ be the set of strategy profiles.*

We view a strategy profile $\sigma = (\sigma^1, \sigma^2)$ as a map $\sigma : S \to A$ from states to actions, such that $\sigma(i) = \sigma^k(i)$ for all $i \in S^k$ and $k \in \{1, 2\}$.

A strategy $\sigma^k \in \Sigma^k$ can be viewed as a subset $\sigma^k \subseteq A^k$ of actions such that $\sigma^k \cap A_i = \{\sigma^k(i)\}$ for all $i \in S^k$. A strategy profile $\sigma = (\sigma^1, \sigma^2) \in \Sigma$ can be viewed similarly as a subset of actions $\sigma = \sigma^1 \cup \sigma^2 \subseteq A$. Note that P_σ is an $n \times n$ matrix for every $\sigma \in \Sigma$. We assume WLOG that actions are ordered such that $J_\sigma = I$, where I is the identity matrix, for all $\sigma \in \Sigma$.

The matrix P_σ defines a Markov chain. In particular, the probability of being in the j-th state after t steps when starting in state i is $(P_\sigma^t)_{i,j}$. We say that the players *play according to* σ if whenever the pebble is on state $i \in S^k$, Player k uses action $\sigma(i)$. Let $i \in S$ be some state and t some number. The expected cost of the t-th action used is $(P_\sigma^t)_i \mathbf{c}_\sigma$. In particular, the expected outcome is $\sum_{t=0}^{\infty} \gamma^t (P_\sigma^t)_i \mathbf{c}_\sigma$. The following lemma shows that this infinite series always converges.

Lemma 2. *For every strategy profile $\sigma \in \Sigma$ the matrix $(I - \gamma P_\sigma)$ is non-singular, and $(I - \gamma P_\sigma)^{-1} = \sum_{t=0}^{\infty} \gamma^t P_\sigma^t$.*

The simple proof of Lemma 2 has been omitted. For details, cf., e.g., [7].

Definition 8 (Value vectors). *For every strategy profile $\sigma \in \Sigma$ we define the* value vector $\mathbf{v}^\sigma \in \mathbb{R}^n$ *by* $\mathbf{v}^\sigma = (I - \gamma P_\sigma)^{-1} \mathbf{c}_\sigma$.

The i-th component of the value vector $\mathbf{v}^\sigma$, for a given strategy profile σ, is the expected outcome over plays starting in $i \in S$, when the players play according to σ.

It follows from Lemma 2 and Definition 8 that $\mathbf{v}^\sigma$ is the unique solution to:

$$\mathbf{v}^\sigma = \mathbf{c}_\sigma + \gamma P_\sigma \mathbf{v}^\sigma \ . \tag{1}$$

We say that a vector $\mathbf{v}^* \in \mathbb{R}^n$ is an *optimal value vector* if and only if:

$$\forall i \in S : \ \mathbf{v}_i^* = \min_{\sigma^1 \in \Sigma^1} \max_{\sigma^2 \in \Sigma^2} \mathbf{v}_i^{(\sigma^1, \sigma^2)} = \max_{\sigma^2 \in \Sigma^2} \min_{\sigma^1 \in \Sigma^1} \mathbf{v}_i^{(\sigma^1, \sigma^2)} \ .$$

Shapley [20] showed that there always exists a unique optimal value vector. By solving a 2TBSG G we mean finding the optimal value vector. The following theorem was proved by Jurdziński and Savani [11] for deterministic 2TBSGs. In the full version of the paper [8] we show that the reduction of Jurdziński and Savani [11] generalizes to general 2TBSGs without modification, giving us:

Theorem 1. *Let $G = (S^1, S^2, (A_i)_{i \in S}, p, c, \gamma)$ be a 2TBSG with matrix representation $(P, \mathbf{c}, J, \mathcal{I}, \gamma)$. Assume that the set of actions A can be partitioned into two disjoint strategy profiles σ and τ. Define:*

$$\begin{aligned} M_{G,\sigma,\tau} &= \mathcal{I}(I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}\mathcal{I} \\ \mathbf{q}_{G,\sigma,\tau} &= \mathcal{I}(I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}\mathbf{c}_\tau - \mathcal{I}\mathbf{c}_\sigma \ . \end{aligned}$$

Then from a solution $(\mathbf{w}, \mathbf{z})$ to the LCP $(M_{G,\sigma,\tau}, \mathbf{q}_{G,\sigma,\tau})$ we can derive a solution to the 2TBSG G as the optimal value vector is $\mathbf{v}^* = (I - \gamma P_\tau)^{-1}(\mathbf{c}_\tau + \mathcal{I}\mathbf{z})$. Also, $M_{G,\sigma,\tau}$ is a P-matrix.

To prove that the matrix $M_{G,\sigma,\tau}$ is a P-matrix, we use the following lemma. The lemma is also used in the later parts of the paper. To understand the use of $\mathbf{v}$ in the lemma observe that $\mathbf{x}^\mathsf{T}(I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}\mathbf{x} = \mathbf{x}^\mathsf{T}(I - \gamma P_\sigma)\mathbf{v}$.

Lemma 3. *Let* $\mathbf{x}$ *be a non-zero vector,* $\mathbf{v} = (I - \gamma P_\tau)^{-1}\mathbf{x}$*, and* $j \in \operatorname{argmax}_i |\mathbf{v}_i|$*. Then we have that:*

$$|\mathbf{x}_j| \;\geq\; (1-\gamma)\,|\mathbf{v}_j| \quad . \tag{2}$$

$$\forall i : |\mathbf{x}_i| \;\leq\; (1+\gamma)\,|\mathbf{v}_j| \quad . \tag{3}$$

$$\mathbf{x}_j((I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}\mathbf{x})_j \;\geq\; (1-\gamma)\,|\mathbf{x}_j\mathbf{v}_j| \;>\; 0 \ . \tag{4}$$

Proof. Observe first that $\mathbf{v}$ is the unique solution to $\mathbf{v} = \mathbf{x} + \gamma P_\tau \mathbf{v}$. In fact, we can interpret $\mathbf{v}$ as the value vector for τ when the costs $\mathbf{c}_\tau$ have been replaced by $\mathbf{x}$. If $v = \mathbf{0}$ then this implies that $\mathbf{0} = \mathbf{x} + \mathbf{0} \neq \mathbf{0}$ which is a contradiction. Thus, $\mathbf{v} \neq \mathbf{0}$ and in particular $\mathbf{v}_j \neq 0$. Since, for every i, the entries of $(P_\tau)_i$ are non-negative and sum to one we have that $|\gamma(P_\tau)_i\mathbf{v}| \leq \gamma\,|\mathbf{v}_j|$. The equations $\mathbf{v}_i = \mathbf{x}_i + \gamma(P_\tau)_i\mathbf{v}$, for all i, then imply that:

$$\begin{aligned} |\mathbf{x}_j| \;&=\; |\mathbf{v}_j - \gamma(P_\tau)_j\mathbf{v}| \;\geq\; |\mathbf{v}_j| - |\gamma(P_\tau)_j\mathbf{v}| \;\geq\; |\mathbf{v}_j| - |\gamma\mathbf{v}_j| \;=\; (1-\gamma)\,|\mathbf{v}_j| \\ \forall i:\ |\mathbf{x}_i| \;&=\; |\mathbf{v}_i - \gamma(P_\tau)_i\mathbf{v}| \;\leq\; |\mathbf{v}_i| + |\gamma(P_\tau)_i\mathbf{v}| \;\leq\; |\mathbf{v}_j| + |\gamma\mathbf{v}_j| \;=\; (1+\gamma)\,|\mathbf{v}_j| \end{aligned}$$

This proves (2) and (3).

We next observe that $\mathbf{v}_j$ and $\mathbf{x}_j$ have the same sign. This again follows from $|\gamma(P_\tau)_j\mathbf{v}| \leq \gamma\,|\mathbf{v}_j|$ and $\mathbf{v}_j = \mathbf{x}_j + \gamma(P_\tau)_j\mathbf{v}$. Using that $\operatorname{sgn}(\mathbf{v}_j) = \operatorname{sgn}(\mathbf{x}_j)$ we can now see that:

$$\begin{aligned} \mathbf{x}_j((I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}\mathbf{x})_j \;&=\; \mathbf{x}_j((I - \gamma P_\sigma)\mathbf{v})_j \;=\; \mathbf{x}_j\mathbf{v}_j - \gamma\mathbf{x}_j(P_\sigma)_j\mathbf{v} \\ &\geq\; \mathbf{x}_j\mathbf{v}_j - \gamma\mathbf{x}_j\mathbf{v}_j \;=\; (1-\gamma)\mathbf{x}_j\mathbf{v}_j \;>\; 0 \ . \end{aligned}$$

This proves (4). □

We know from Lemma 1 that the matrix $M_{G,\sigma,\tau}$ is a P-matrix if and only if for every $\mathbf{x} \neq \mathbf{0}$ there exists a $j \in [n]$ such that $\mathbf{x}_j(M_{G,\sigma,\tau}\mathbf{x})_j > 0$. Since $\mathcal{I}\mathbf{x} \neq \mathbf{0}$, inequality (4) in Lemma 3 shows that $\mathbf{x}_j(M_{G,\sigma,\tau}\mathbf{x})_j > 0$ for $j \in \operatorname{argmax}_i \left|((I - \gamma P_\tau)^{-1}\mathcal{I}\mathbf{x})_i\right|$. Hence, $M_{G,\sigma,\tau}$ is a P-matrix.

Recall that Kojima *et al.* [13] showed that every P-matrix is a P_*-matrix. Hence, we have shown that $M_{G,\sigma,\tau}$ is a P_*-matrix.

3 The $P_*(\kappa)$ Property for 2TBSGs

Let G be a 2TBSG with matrix representation $(P, \mathbf{c}, J, \mathcal{I}, \gamma)$, and let σ and τ be two disjoint strategy profiles that form a partition of the set of actions of G. Recall that G can be solved by solving the LCP $(M_{G,\sigma,\tau}, \mathbf{q}_{G,\sigma,\tau})$. In this section we provide essentially tight upper and lower bounds on the smallest number κ for which the matrix $M_{G,\sigma,\tau}$ is guaranteed to be a $P_*(\kappa)$-matrix. More precisely, we first show that for $\kappa = \frac{n}{(1-\gamma)^2}$, the matrix $M_{G,\sigma,\tau}$ is always a $P_*(\kappa)$-matrix. We then also show that for every $n > 2$ and $\gamma < 1$ there exists a game G_n, and two strategy profiles σ_n and τ_n, such that M_{G_n,σ_n,τ_n} is not a $P_*(\kappa)$-matrix for any $\kappa < \frac{\gamma^2(n-2)}{8(1-\gamma)^2} - \frac{1}{4}$. It follows that the unified interior point method of Kojima

et al. [13] solves the 2TBSG G in time $O(\frac{n^{4.5}L}{(1-\gamma)^2})$, where L is the number of bits required to describe G, and that this bound can not be improved further only by bounding κ.

Recall that $M_{G,\sigma,\tau} = \mathcal{I}(I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}\mathcal{I}$, and define $M := \mathcal{I}M_{G,\sigma,\tau}\mathcal{I} = (I - \gamma P_\sigma)(I - \gamma P_\tau)^{-1}$. It is easy to see that $M_{G,\sigma,\tau}$ is a $P_*(\kappa)$-matrix for some $\kappa \geq 0$ if and only if M is. Indeed, the inequality of Definition 4 must hold for all $\mathbf{x} \in \mathbb{R}^n$, and we can therefore substitute $\mathbf{x}$ by $\mathcal{I}\mathbf{x}$. Hence, it suffices to bound the κ for which M is a $P_*(\kappa)$-matrix.

The proof of the following theorem can be found in the full version of the paper [8].

Theorem 2. *Let n and $0 < \gamma < 1$ be given. For any γ-discounted 2TBSG G with n states, the matrix $M_{G,\sigma,\tau}$, where σ and τ partition the actions of G, is a $P_*(\kappa)$-matrix for $\kappa = \frac{n}{(1-\gamma)^2}$.*

We next present a lower bound that essentially matches the upper bound given in Theorem 2. The gap between the upper and lower bounds is close to a factor of 8 for γ going to 1. We are mostly interested in the case when γ is very close to 1, since it is known that the problem can be solved in strongly polynomial time when γ is a fixed constant [9]. We establish the lower bound using the family of games $\{G_n \mid n > 2\}$ shown in Figure 1. Figure 1 also shows two strategies σ_n and τ_n shown as solid and dashed arrows, respectively. Formally, the games are defined as follows.

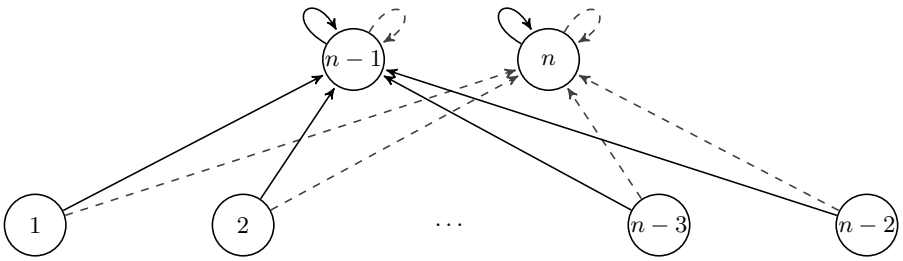


Fig. 1. An example game G_n and two strategy profiles σ_n (solid) and τ_n (dashed), where M_{G_n,σ_n,τ_n} essentially matches the upper bound given in Theorem 2.

The Game G_n. For a given n, let G_n be the following game containing n states, all belonging to Player 2. For $i \leq n-2$, state i has two actions: one leading to state $n-1$ and one leading to state n. State $n-1$ and n have two self-loops each. I.e., the game is deterministic. The cost vector $\mathbf{c}$ can be arbitrary, and the discount factor γ will be specified by the analysis. We also define two disjoint strategy profiles σ_n and τ_n that partition the set of actions. The strategy profile σ_n contains for all states $i \leq n-2$ the action leading to state $n-1$, and the strategy profile τ_n contains for all states $i \leq n-2$ the action leading to state n. Furthermore, at states $n-1$ and n, each strategy profile contains a self-loop.

Straightforward calculations give the following theorem (cf. [8]).

Theorem 3. *Let $n > 2$ and $0 < \gamma < 1$ be given. For the 2TBSG G_n, the matrix M_{G_n,σ_n,τ_n} is not a $P_*(\kappa)$-matrix, for $\kappa < \frac{\gamma^2(n-2)}{8(1-\gamma)^2} - \frac{1}{4} = \Omega\big(\frac{\gamma^2 n}{(1-\gamma)^2}\big)$.*

References

1. Andersson, D., Miltersen, P.B.: The complexity of solving stochastic games on graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 112–121. Springer, Heidelberg (2009)
2. Chen, H., Zhang, M., Zhao, Y.: A class of new large-update primal-dual interior-point algorithms for $P_*(\kappa)$ linear complementarity problems. In: Yu, W., He, H., Zhang, N. (eds.) ISNN 2009, Part III. LNCS, vol. 5553, pp. 77–87. Springer, Heidelberg (2009)
3. Cho, G.-M.: A new large-update interior point algorithm for $P_*(\kappa)$ linear complementarity problems. Journal of Computational and Applied Mathematics 216, 265–278 (2008)
4. Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. In: Computer Science and Scientific Computing. Academic Press, Boston (1992)
5. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer (1997)
6. Gärtner, B., Rüst, L.: Simple stochastic games and P-matrix generalized linear complementarity problems. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 209–220. Springer, Heidelberg (2005)
7. Hansen, T.D.: Worst-case Analysis of Strategy Iteration and the Simplex Method. PhD thesis, Aarhus University (2012)
8. Hansen, T.D., Ibsen-Jensen, R.: The complexity of interior point methods for solving discounted turn-based stochastic games (2013), `http://www.cs.au.dk/~tdh/papers/bounding_kappa.pdf`
9. Hansen, T.D., Miltersen, P.B., Zwick, U.: Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. In: Proc. of 2nd ICS, pp. 253–263 (2011)
10. Illés, T., Nagy, M., Terlaky, T.: A polynomial path-following interior point algorithm for general linear complementarity problems. Journal of Global Optimization 47, 329–342 (2010)
11. Jurdziński, M., Savani, R.: A simple P-matrix linear complementarity problem for discounted games. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 283–293. Springer, Heidelberg (2008)
12. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica 4(4), 373–395 (1984)
13. Kojima, M., Megiddo, N., Noma, T., Yoshise, A.: A unified approach to interior point algorithms for linear complementarity problems: A summary. Oper. Res. Lett. 10(5), 247–254 (1991)
14. Kojima, M., Megiddo, N., Ye, Y.: An interior point potential reduction algorithm for the linear complementarity problem. Mathematical Programming 54, 54–267 (1992)
15. Krishnamurthy, N., Parthasarathy, T., Ravindran, G.: Solving subclasses of multi-player stochastic games via linear complementarity problem formulations – a survey and some new results. Optimization and Engineering 13, 435–457 (2012)
16. Littman, M.L.: Algorithms for sequential decision making. PhD thesis, Brown University (1996)

17. Neyman, A., Sorin, S.: Stochastic games and applications. Kluwer Academic Publishers, Dordrecht (2003)
18. Rao, S.S., Chandrasekaran, R., Nair, K.P.K.: Algorithms for discounted stochastic games. Journal of Optimization Theory and Applications 11, 627–637 (1973)
19. Rüst, L.Y.: The P-matrix linear complementarity problem. PhD thesis, ETH Zürich (2007)
20. Shapley, L.S.: Stochastic games. Proc. Nat. Acad. Sci. U.S.A. 39(10), 1095–1100 (1953)
21. Ye, Y.: Interior Point Algorithms: Theory and Analysis. Wiley-Interscience series in discrete mathematics and optimization. Wiley (1998)

The Computation of Nature, Or: Does the Computer Drive Science and Technology?

Ulf Hashagen

Deutsches Museum, The Research Institute for the History of Science and Technology, 80306 München, Germany

It has often been claimed that the computer has not only revolutionized everyday life but has also affected the sciences in a fundamental manner. Even in national systems of innovation which had initially reacted with a fair amount of reserve to the computer as a new scientific instrument (such as Germany and France; cf., e.g., [33,18]), it is today a commonplace to speak about the "computer revolution" in the sciences [27]. In his path breaking book *Revolution in Science*, Cohen diagnoses that a general revolutionary change in the sciences had followed from the invention of the computer. While he asserts that the scientific revolution in astronomy in the 17th century was not based on the newly invented telescope but on the intellect of Galileo Galilei, he maintains in contrast that the "case is different for the computer, which [...] has affected the thinking of scientists and the formulation of theories in a fundamental way, as in the case of the new computer models for world meteorology" [10, pp. 9-10 & 20-22].

The modern electronic digital computer had originally been invented as an extremely fast and programmable calculator to solve mathematical problems in the sciences and in technology in the 1940s. Although the use of the computer as a new form of a scientific instrument became more and more widespread in the sciences from the 1960s onwards, historians of science and technology did not pay much attention to this important development. Although history of computing has been established as a sub-discipline of the history of technology during the last decades and contributed to a better understanding of the development of hardware and software as well as of the advent of the information age,[1] there are still large gaps in our knowledge on the history of "scientific computing". There are only a few studies that have contributed to our understanding of the use of computers in the many fields of science and/or research institutions. Research to date has largely focused on the development of supercomputing at the large national research laboratories in the United States [30,38], the use of the computer in high-energy physics [39] and in Roentgen crystallography [12], as well as to the efforts to computerize bio-medical research in the 1950s and 1960s [34].

A second related problem is that we know only a little about the different new computer based methods which became established in the various national innovation systems in the second half of the 20th century. The, admittedly, very

[1] For an overview on the history of computing, cf. [8,9,17].

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 263–270, 2013.

small number of historians and philosophers researching on these subjects have for the most part concentrated on just one phenomenon: computer simulation (cf., e.g., [37,24]). Computer simulation, such is the basic assumption, has opened a third way of doing science besides experiment and theory. And this is said to have caused the revolutionary change by computers (cf., e.g., [21,28,44]) There are a number of issues in this field on which a detailed historical analysis would be useful, and it is doubtful whether "simulation" was really invented with the digital computer—instead it should be accepted that there are several classes of "simulations" with different epistemological qualities.

In general, the question of whether the computer changed scientific practice and the setting of the agenda in various scientific disciplines has not found much attention in the history of science. In his pioneering study on *John von Neumann and the Origins of Modern Computing*, William Aspray has shown how a noted mathematician "invented" the computer as a mathematical machine and how the computer decisively transformed numerical mathematics as a field of study [3,2]. Secondly, the introduction of numerical methods to weather forecasting and the change of meteorological practice by using computers has been analyzed in a few studies on the development of "scientific computing" in meteorology (cf., e.g., [11,29,16]). Finally, the British historian of computing, Jon Agar, has made an attempt to analyze the changes in various disciplines triggered by the advent of the computer. According to Agar, "computerization" of a discipline in the 1950s only occurred if "material practices of computation" had already existed beforehand [1]. In support of this assessment, historical studies show that (social) networks between the objects of research (or artifacts), the computing machines, the computer-staff, and the scientists and engineers, as well as the formation of formalized working structures and working routines, did have great importance for the "computerization" of a field or discipline.

In view of contemporary science journals and books, it becomes clear that the "picture" of a uniform, omnipotent and omnipresent computer revolution in all disciplines and in every nation does not correspond to the facts. There was rather a plurality of processes of computerization with different repercussions on the different disciplines. Examples are: Firstly, it is quite clear that the development of high-performance computation was of enormous importance in many fields of science and technology. This holds for all fields in which ordinary and/or partial differential equations and integral equations are used in mathematical modeling of natural phenomena, as for instance in fluid dynamics, reactor design, chemical reactions, astrophysics, crystallography, DNA-sequencing, and geophysics (cf., e.g., [13,36]). These fields are, secondly, closely connected to the phenomenon of computer simulation being used for pure research in the sciences as well as for the design of technical artifacts. Thirdly, the practice of computer based instrumentation and computer based experiments became all-important in many fields of the sciences and medicine, as the computer was more and more needed to control the ever more complicated experiments. Fourthly, in some fields methods of information retrieval were of utmost significance (cf., e.g., [6]). The upshot of all this is that in almost all cases it was one thing that was fundamentally changed

by the computer: the time-economy of science. (Today, scientists and engineers take it for granted that the electronic computer and appropriate software systems that deliver rapid and reliable data for most of their scientific problems, but this has been only the case since the late 20th century. The case of the British *Nautical Almanac and Astronomical Ephemeris*—an annual publication describing the positions of the moon and other celestial bodies for the purpose navigation at sea which has been published by *HM Nautical Almanac Office* since 1767—serves to illustrate how time-consuming, laborious and tedious the work of the human computers for rather easy scientific problems has been until the first half of the 20th century [43].

The above-cited metaphor of the computer as a revolutionary artifact implies—as the recently deceased American historian of science Michael S. Mahoney aptly remarked some years ago—the image of a revolutionary technology changing all parts overnight and splitting all societal groups in two separate parties: either one jumps on the continuously accelerating bandwagon or one comes to a standstill as a "dinosaur" at the platform [32]. It seems quite easy to identify candidates for the "jumping-on-the-bandwagon". Dorothy Crowfoot Hodgkin's discovery of the structure of the vitamin B_{12}, for which she was awarded the Nobel Prize, was essentially based on the use of one of the first available electronic computers in Britain [1]. Or: the design of the hydrogen bomb was essentially based on the use of novel mathematical models and methods computed with the then new electronic computers [14]. Or, to put it another way, one could examine an important question of the history of technology in an new way: Does technology drive science? (Cf. [40].)

At the same time the computer-based "knowledge-revolution" in the sciences seems strongly connected with a "qualitative decline" in the falsification of scientific theories or models. It would appear, therefore, that a digital "Pandora's Box" had been opened and that the scientists are no longer masters of the situation, since the verification or validation of computer-based models does not seem to be justified from a philosophical point of view. Naomi Oreskes, in her article about Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences (published together with philosopher Kristin Shrader-Frechette and geologist Kenneth Belitz) in *Science*, raised fundamental doubts about the common practice of numerical modeling in the sciences. The authors stated that "verification and validation of numerical models of natural systems is impossible", since "natural systems are never closed and because model results are always non-unique". They eventually came to the conclusion that the "primary value of models is heuristic" [2].

To better understand the development that led to this situation, as described in [35], we shall briefly analyze the development of computational fluid dynamics as it was one of the most important origins for the development of computational

[2] Cf. [35]. This article met with very positive responses in the sciences as well as in the history of science and in science studies. It became the starting point for a series of articles on the use of computer simulations in geoscience and in climate science. Cf., e.g., [20].

physics in general. Today, this field is "by far the largest user of high performance computing in engineering" [7] and at the same time is of enormous significance in pure research in physics (cf., e.g., [5]).

In the 17th century, scientist had begun to describe natural phenomena in the exact sciences (astronomy, physics, ...) by physical models which in turn were described in mathematical structures. As a result—to portray it in simplified terms—the "Galilei Principle" (physical phenomena can be described by mathematical models) and the "Newton Principle" (physical phenomena can be described mathematically by differential equations) became crucial for the further development of the exact sciences. This made the theoretical physicist and Nobel laureate, Eugene Wigner, wonder about the miracle of the *Unreasonable Effectiveness of Mathematics in the Natural Sciences* [42]. Since the existence and uniqueness of a solution of an ordinary differential equation (initial value problem) can be proven under certain (quite general) conditions, the Newton principle served as a basis for a deterministic conception of the world of the exact sciences until the end of the 19th century. On the other hand it became apparent to scientists during the 19th century that it was by no means possible to predict and/or to compute all physical phenomena. The history of the three-body problem in celestial mechanics is the most striking example yet of a quite simple physical problem which proved to be extremely difficult to handle. After more than two hundred years of research by outstanding mathematicians and astronomers concentrating all energies on finding an analytical solution of the nonlinear differential equation problem in the 18th and 19th century, it was proven that the three-body problem has no analytical solution in terms of algebraic integrals (cf., e.g., [4]). Even if it is known since the early 20th century that the three-body problem could be solved in quite general cases through a convergent power series which unfortunately converges extremely slowly and is therefore futile for all practical purposes. As the solution of the three body problem was not only of theoretical interest but had also important applications in practical astronomy such as navigation, since the 18th century astronomers "simulated" the solution of the three body problem by developing complicated numerical approximation methods and by using teams of human computers for the tedious work of calculation (cf., e.g., [19]).

Another case of a non-linear problem, Navier-Stokes equations (non-linear partial differential equations) in hydrodynamics, became the starting point of modern computer-based methods of "scientific computing" and sounded the bell for a new round in the relation between theory, experiment and computation in the exact sciences. At the outset of this "science as software"[3] were the ideas of the Hungarian mathematician John von Neumann, who was confronted with the problem to solve hydrodynamic problems for the Manhattan Project in the second world war. Von Neumann failed miserably in his attempts to solve non-linear partial differential equations of hydrodynamics with the traditional methods of analysis. This led him to the conclusion that these difficult problems could only

[3] Cf. [31]. My argumentation in this paragraph partly follows the same line as in Mahoney's article.

be addressed if one would divorce the close marriage of physics and the classical methods of analysis: "Our present analytical methods seem unsuitable for the solution of the important problems arising in connection with non-linear partial differential equations and, in fact, with virtually all types of non-linear problems in pure mathematics [15]". Since, in the case of the non-linear partial differential equations of hydrodynamics, the strategy of using teams of human computers for the tedious work of calculation was also a hopeless undertaking, von Neumann in 1946 suggested to use numerical methods and new nonexistent "Large Scale Computing Machines" instead. From there, von Neumann ultimately arrived at a fundamentally new philosophical position on the relationship between natural phenomena, physical models, and mathematical models; and he proposed to discard the "Galilei Principle". Instead of describing natural phenomena in the exact sciences by physical models which in turn were described in mathematical structures, von Neumann suggested to describe natural phenomena only with a mathematical model, which was characterized by him as a "mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena [41]".

This method had instantaneous consequences for the scientific understanding of physical processes, as the solution was no longer a scientific theory "explaining" nature, but only a numerical result. Numerical solutions did obviously not provide for similar insights into physical processes as analytical solutions had done. Moreover, it became much more difficult to "explain" the discrepancies ("errors") between the numerical solutions and the results of experiments. Also, from a philosophical point of view, it became difficult to compare *Theory-1* with *Theory-2* with regard to experimental results when the model of constructive falsifiability (Lakatos) was applied [25]. This, of course, leads to the logical conclusion that the door was opened to a variety of potential new mistakes, ignorance and contingency. Furthermore, it became soon apparent that the "generation" of numerical solutions in computational fluid dynamics led mathematicians, physicists, and engineers to fundamentally new mathematical problems. Apart from the general problem of rounding errors in the field of numerical mathematics being newly defined by digital computers, it turned out that the numerical solution of partial differential equations was a tricky mathematical problem, since the vividness (Anschaulichkeit) of mathematical approaches to numerical solutions could easily result in false mathematical models. Moreover, over the last few decades the rise of computational science and the widespread use of large software tools in various fields of science and engineering has made the reproducibility of results principally more and more uncertain, if the source code of the used software is inaccessible [22].

The development set in motion by von Neumann had far-reaching consequences for the evolution of science and technology in the late 20th century. On the one hand the new field of computational fluid dynamics developed into a design tool for engineers, and on the other hand computational fluid dynamics made novel "mathematical experiments" in hydrodynamics possible. This, in turn, made it something like the model for the development of the methods of

computational science in other fields [26]. At the end of the 20th century, computational fluid dynamics seems to have developed into what Terry Shinn has called research technology being characterized by its "pragmatic universality" and by its "robustness" in dealing with mistakes, ignorance, contingency and errors [23]. This process can be further analyzed by asking: How did physicists and engineers succeed in handling problems of errors, ignorance and contingency in computational fluid dynamics despite the fundamental philosophical problems of computer-based models in the sciences and in engineering? In the end a careful historical analysis of this questions will hopefully help historians of science and technology as well as scientists and engineers in their understanding of Cohen's assertion: By what means and to what extent does the computer change science and technology?

References

1. Agar, J.: What Difference Did Computers Make? Social Studies of Science 36, 869–907 (2006)
2. Aspray, W.: The Transformation of Numerical Analysis by the Computer: An Example From the Work of John von Neumann. In: Rowe, D.E., et al. (eds.) History of Modern Mathematics, vol. 2, pp. 307–322. Academic Press (1989)
3. Aspray, W.: John von Neumann and the Origins of Modern Computing. MIT Press (1990)
4. Barrow-Green, J.: Poincaré and the Three Body Problem. American Mathematical Society (1997)
5. Birkhoff, G.: Numerical Fluid Dynamics. SIAM Review 25, 1–34 (1983)
6. Bowker, G.C.: Archival Technology in the Historical Sciences 1800-1997. History and Technology 15, 69–87 (1998)
7. Cant, S.: High-Performance Computing in Computational Fluid Dynamics: Progress and Challenges. Philosophical Transactions: Mathematical, Physical and Engineering Sciences 360, 1211–1225 (2002)
8. Campbell-Kelly, M., Aspray, W.: Computer: A History of the Information Machine. Westview Press (2004)
9. Ceruzzi, P.E.: A History of Modern Computing. MIT Press (2003)
10. Cohen, I.B.: Revolution in Science. Belknap Press (1985)
11. Dahan Dalmédico, A.: History and Epistemology of Models: Meteorology (1946–1963) as a Case Study. Archive for History of Exact Sciences 55, 395–422 (2001)
12. de Chadarevian, S.: Designs for Life: Molecular Biology After World War II. Cambridge University Press (2002)
13. Fernbach, S., Taub, A.H. (eds.): Computers and Their Role in the Physical Sciences. Gordon & Breach (1970)
14. Galison, P.: Computer Simulations and the Trading Zone. In: Galison, P., Stump, D.J. (eds.) The Disunity of Science: Boundaries, Contexts, and Power, pp. 118–157. Stanford University Press (1996)
15. Goldstine, H.H., von Neumann, J.: On the Principles of Large-Scale Computing Machines. In: von Neumann, J. (ed.) Collected Works, vol. 5, pp. 1–33. Pergamon Press, Oxford (1963)
16. Harper, K.C.: Weather by the Numbers: The Genesis of Modern Meteorology. MIT Press (2008)

17. Hashagen, U., et al. (eds.): History of Computing: Software Issues. Springer (2002)
18. Hashagen, U.: Rechner für die Wissenschaft: "Scientific Computing" und Informatik im deutschen Wissenschaftssystem 1870-1970. In: Hashagen, U., Hellige, H.D. (eds.) Rechnende Maschinen im Wandel: Mathematik, Technik, Gesellschaft. Deutsches Museum, pp. 111–152 (2011)
19. Hashagen, U.: Rechner für die Wissenschaft. Die Entwicklung des "Wissenschaftlichen Rechnens" in der Astronomie in Deutschland 1870-1945. Habilitation Thesis, LMU Munich (2011)
20. Heymann, M.: Modeling Reality. Practice, Knowledge, and Uncertainty in Atmospheric Transport Simulation. Historical Studies in the Physical and Biological Sciences 37, 49–85 (2006)
21. Humphreys, P.: Extending Ourselves. Computational Science, Empiricism, and Scientific Method. Oxford University Press (2004)
22. Ince, D.C., Hatton, L., Graham-Cumming, J.: The Case for Open Computer Programs. Nature 482, 485–488 (2012)
23. Joerges, B., Shinn, T.: Research-Technology. Instrumentation Between Science, State and Industry. Wissenschaftszentrum Berlin for Sozialforschung (2000)
24. Keller, E.F.: Models, Simulation, and Computer Experiments. In: Radder, H. (ed.) The Philosophy of Scientific Experimentation, pp. 198–215. Univ. of Pittsburgh Press (2003)
25. Lakatos, I.: The Methodology of Scientific Research Programmes. Cambridge University Press (1978)
26. Lax, P.: The Flowering of Applied Mathematics in America. In: Duren, P.L. (ed.) A Century of Mathematics in America, vol. 2, pp. 455–466. American Mathematical Society (1989)
27. Lengauer, T. (ed.): Computermodelle in der Wissenschaft–zwischen Analyse, Vorhersage und Suggestion. Deutsche Akademie der Naturforscher Leopoldina (2011)
28. Lenhard, J., Kueppers, G., Shinn, T. (eds.): Simulation: Pragmatic Construction of Reality. Springer (2006)
29. Lynch, P.: The Emergence of Numerical Weather Prediction: Richardson's Dream. Cambridge University Press (2006)
30. MacKenzie, D.: The Influence of the Los Alamos and Livermore National Laboratories on the Development of Supercomputing. Annals of the History of Computing 13, 179–201 (1991)
31. Mahoney, M.S.: Software as Science—Science as Software. In: [17], pp. 25–48.
32. Mahoney, M.S.: The Histories of Computing(s). Interdisciplinary Science Reviews 30, 119–135 (2005)
33. Mounier-Kuhn, P.: Computer Science in French Universities: Early Entrants and Latecomers. Information and Culture 47, 414–456 (2012)
34. November, J.: Biomedical Computing: Digitizing Life in the United States. Johns Hopkins Univ. Press (2012)
35. Oreskes, N., Shrader-Frechette, K., Belitz, K.: Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. Science 263, 641–646 (1994)
36. Rohrlich, F.: Computer Simulation in the Physical Sciences. In: PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association, pp. 507–518 (1990)
37. Schweber, S.S.: Complex Systems, Modelling and Simulation Studies. History and Philosophy of Science Part B 31, 583–609 (2000)
38. Seidel, R.W.: "Crunching Numbers": Computers and Physical Research in the AEC Laboratories. History and Technology 15, 31–68 (1998)

39. Seidel, R.W.: From Factory to Farm: Dissemination of Computing in High-Energy Physics. Historical Studies in the Natural Sciences 38, 479–507 (2008)
40. Smith, M.R., Marx, L. (eds.): Does Technology Drive History? The Dilemma of Technological Determinism. MIT Press (1994)
41. von Neumann, J.: Method in the Physical Sciences. In: Leary, L. (ed.) The Unity of Knowledge, pp. 157–164. Doubleday (1955)
42. Wigner, E.P.: The Unreasonable Effectiveness of Mathematics in the Natural Sciences. Communications on Pure and Applied Mathematics 13, 1–14 (1960)
43. Wilkins, G.A.: The Making of Astronomical Tables in HM Nautical Almanac Office. In: Campbell-Kelly, M., Croarken, M., Flood, R., Robson, E. (eds.) The History of Mathematical Tables. From Sumer to Spreadsheets, pp. 295–320. Oxford University Press (2003)
44. Winsberg, E.: Science in the Age of Computer Simulation. University of Chicago Press (2010)

Negative Glues and Non-determinism in Nanocomputations by Self-assembly

Lila Kari

Department of Computer Science, University of Western Ontario,
London, Ontario, N6A 5B7, Canada
lila@csd.uwo.ca

Tile-based self-assembly is a model of "algorithmic crystal growth" in which square "tiles" represent molecules that bind to each other via highly-specific bonds on their four sides, driven by random mixing in solution but constrained by the local binding rules of the tile bonds. Winfree defined a model of tile-based self-assembly known as the abstract Tile Assembly Model (aTAM), [4]. The fundamental components of this model are un-rotatable, but translatable square "tile types" whose sides are labeled with "glues" representing binding sites. Two tiles that are placed next to each other are attracted with strength determined by the glues where they abut, and, in the aTAM, a tile binds to an assembly if it is attracted on all matching sides with total strength at least a certain threshold value τ. Assembly begins from a "seed" tile and progresses in a stepwise fashion until no more tiles may attach. In his aTAM model Winfree postulated negative (i.e., repulsive) interactions between tiles to be physically plausible and, subsequently, Reif, Sahu, and Yin, [3], studied negative interactions in the context of reversible attachment operations.

Herein I explore the power of negative interactions with irreversible attachments, and describe two main results, [2]. The first one is an impossibility theorem: After t steps of assembly, $\Omega(t)$ tiles will be forever bound to an assembly, unable to detach. Thus negative glue strengths do not afford unlimited power to reuse tiles. The second result is a positive one: We construct a set of tiles that can simulate an s-space-bounded, t-time-bounded Turing machine, while ensuring that no intermediate assembly grows larger than $O(s)$, rather than $O(s \cdot t)$ as required by the standard Turing machine simulation with tiles. In addition to the space-bounded Turing machine simulation, we show another example application of negative glues: reducing the number of tile types required to assemble "thin" $o(n \times (\log n/\log\log n))$ rectangles.

I also address the role of nondeterminism in the Tile Assembly Model, in particular how its use in tile systems affects the resource requirements, [1]. We show that for infinitely many $c \in N$, there is a finite shape S that is self-assembled by a tile system (meaning that all of the various terminal assemblies produced by the tile system have shape S) with c tile types, but every deterministic tile system that self-assembles S (i.e., has only one terminal assembly, whose shape is S) needs more than c tile types. We extend the technique to prove that the problem of finding the minimum number of tile types that self-assemble a given

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 271–272, 2013.

finite shape is Σ_2^{P}-complete. We then show an analogous "computability theoretic" result: There exists an infinite shape S that is self-assembled by a tile system but not by any deterministic tile system.

References

1. Bryans, N., Chiniforooshan, E., Doty, D., Kari, L., Seki, S.: The power of nondeterminism in self-assembly. Theory of Computing 9, 1–29 (2013)
2. Doty, D., Kari, L., Masson, B.: Negative interactions in irreversible self-assembly. Algorithmica 66(1), 153–172 (2013)
3. Reif, J.H., Sahu, S., Yin, P.: Complexity of graph self-assembly in accretive systems and self-destructible systems. In: Carbone, A., Pierce, N.A. (eds.) DNA 2005. LNCS, vol. 3892, pp. 257–274. Springer, Heidelberg (2006)
4. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology (1998)

Structures without Scattered-Automatic Presentation*

Alexander Kartzow[1] and Philipp Schlicht[2,**]

[1] Institut für Informatik, Universität Leipzig, Germany
kartzow@informatik.uni-leipzig.de
[2] Mathematisches Institut, Rheinische Friedrich-Wilhelms-Universität Bonn, Germany
schlicht@math.uni-bonn.de

Abstract. Bruyère and Carton lifted the notion of finite automata reading infinite words to finite automata reading words with shape an arbitrary linear order $\mathfrak{L}$. Automata on finite words can be used to represent infinite structures, the so-called word-automatic structures. Analogously, for a linear order $\mathfrak{L}$ there is the class of $\mathfrak{L}$-automatic structures. In this paper we prove the following limitations on the class of $\mathfrak{L}$-automatic structures for a fixed $\mathfrak{L}$ of finite condensation rank $1+\alpha$.

Firstly, no scattered linear order with finite condensation rank above $\omega^{\alpha+1}$ is $\mathfrak{L}$-automatic. In particular, every $\mathfrak{L}$-automatic ordinal is below $\omega^{\omega^{\alpha+1}}$. Secondly, we provide bounds on the (ordinal) height of well-founded order trees that are $\mathfrak{L}$-automatic. If α is finite or $\mathfrak{L}$ is an ordinal, the height of such a tree is bounded by $\omega^{\alpha+1}$. Finally, we separate the class of tree-automatic structures from that of $\mathfrak{L}$-automatic structures for any ordinal $\mathfrak{L}$: the countable atomless boolean algebra is known to be tree-automatic, but we show that it is not $\mathfrak{L}$-automatic.

1 Introduction

Finite automata play a crucial role in many areas of computer science. In particular, finite automata have been used to represent certain classes of possibly infinite structures. The basic notion of this branch of research is the class of automatic structures (cf. [10]): a structure is automatic if its domain as well as its relations are recognised by (synchronous multi-tape) finite automata processing finite words. This class has the remarkable property that the first-order theory of any automatic structure is decidable. One goal in the theory of automatic structures is a classification of those structures that are automatic (cf. [4,12,11,9,13]). Besides finite automata reading *finite* or *infinite words* there are also finite automata reading finite or infinite *trees*. Using such automata as representation of structures leads to the notion of tree-automatic structures [2]. The classification of tree-automatic structures is less advanced but some results have been

* Omitted proofs can be found in the arXiv-Version [8]

** The first author is supported by the DFG research project GELO.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 273–283, 2013.

obtained in the last years (cf. [4,6,7]). Bruyère and Carton [3] adapted the notion of finite automata such that they can process words that have the shape of some fixed linear order. If the linear order is countable and scattered, the corresponding class of languages possesses the good closure properties of the class of languages of finite automata for finite words (i.e., closure under intersection, union, complement, and projection) and emptiness of a given language is decidable. Thus, these automata are also well-suited for representing structures. Given a fixed scattered linear order $\mathfrak{L}$ this leads to the notion of $\mathfrak{L}$-automatic structures. In case that $\mathfrak{L}$ is an ordinal Schlicht and Stephan [16] as well as Finkel and Todorcevic [5] studied the classes of $\mathfrak{L}$-automatic ordinals and $\mathfrak{L}$-automatic linear orders. Here we study $\mathfrak{L}$-automatic linear orders for any scattered linear order $\mathfrak{L}$ and we study $\mathfrak{L}$-automatic well-founded order forests, i.e., forests (seen as partial orders) without infinite branches.

1. If a linear order is $\mathfrak{L}$-automatic and $\mathfrak{L}$ has finite condensation rank at most $1 + \alpha$, then it is a finite sum of linear orders of condensation rank below $\omega^{\alpha+1}$. As already shown in [16], this bound is optimal.
2. If a well-founded order forest is $\mathfrak{L}$-automatic for some ordinal $\mathfrak{L}$, then its ordinal height is bounded by $\mathfrak{L} \cdot \omega$.
   If a well-founded order forest is $\mathfrak{L}$-automatic for $\mathfrak{L}$ some linear order of condensation rank $n \in \mathbb{N}$, then its ordinal height is bounded by ω^{n+1}.
   These two bounds are optimal.
3. A well-founded $\mathfrak{L}$-automatic order forest has ordinal height bounded by $\omega^{\omega\cdot(\alpha+1)}$ where α is the finite condensation rank of $\mathfrak{L}$.

In order to prove Claims 1 and 3 we observe that the notion of *finite-type products* from [16] and the notion of *sum-augmentations of tamely colourable box-augmentations* from [7,6], even though defined in completely different terms, have a common underlying idea. We introduce a new notion of tamely colourable sum-of-box augmentations that refines both notions and allows to prove a variant of Delhommé's decomposition method (cf. [4]) for the case of $\mathfrak{L}$-automatic structures. The main results then follow as corollaries using results from [6] and [7]. For the other two results, we provide an $\mathfrak{L}$-automatic scattered linear ordering of all $\mathfrak{L}$-shaped words if $\mathfrak{L}$ has finite condensation rank $n \in \mathbb{N}$ or if $\mathfrak{L}$ is an ordinal. Extending work from [13], we provide a connection between the height of a tree and the finite condensation rank of its Kleene-Brouwer ordering (with respect to this $\mathfrak{L}$-automatic ordering) that allows to derive the better bounds stated in Claim 2.

As a very sketchy summary of these results, one could say that we adapt techniques previously used on trees to use them on linear orders. This raises the question whether there is a deeper connection between $\mathfrak{L}$-automatic structures and tree-automatic structures. It is known that all ω^n-automatic structures are tree-automatic (cf. [5]). Moreover, from [16] and [4] it follows that ω^{ω^ω} is ω^ω-automatic but not tree-automatic. It is open so far whether every tree-automatic structure is $\mathfrak{L}$-automatic for some linear order $\mathfrak{L}$. We make a first step towards a negative answer by showing that the countable atomless boolean algebra is not $\mathfrak{L}$-automatic for any ordinal $\mathfrak{L}$ (while it is tree-automatic [1]).

2 Preliminaries

2.1 Scattered Linear Orders

In this section, we recall basic notions concerning scattered linear orders. For a detailed introduction, we refer the reader to [15]. A linear order $(L, \leq)$ is *scattered* if there is no embedding of the rational numbers into $(L, \leq)$.

Given a scattered linear order $\mathfrak{L} = (L, \leq)$, an equivalence relation $\sim$ is called a *condensation* if each $\sim$ class is an interval of $\mathfrak{L}$. We then write $\mathfrak{L}/\sim := (L/\sim, \leq')$ for the linear order of the $\sim$ classes induced by $\leq$ (i.e., for $\sim$-classes x, y, $x \leq' y$ iff there are $k \in x, l \in y$ such that $k \leq l$). As usual, for $\mathfrak{L}$ a scattered linear order and l, l' elements of $\mathfrak{L}$, we write $[l, l']$ for the closed interval between l and l'. For each ordinal α we define the *α-th condensation* $\sim_\alpha$ by $x \sim_0 y$ iff $x = y$, $x \sim_{\alpha+1} y$ if the closed interval $[x, y]$ in $\mathfrak{L}/\sim_\alpha$ is finite and for a limit ordinal β, $x \sim_\beta y$ if there is an $\alpha < \beta$ such that $x \sim_\alpha y$. The *finite condensation rank* $\mathsf{FC}(\mathfrak{L})$ is the minimal ordinal α such that $\mathfrak{L}/\sim_\alpha$ is a one-element order. We also let $\mathsf{FC}_*(\mathfrak{L})$ be the minimal ordinal α such that $\mathfrak{L}/\sim_\alpha$ is a finite order. There is such an ordinal α if and only if $\mathfrak{L}$ is scattered. It is obvious from these definitions that $\mathsf{FC}_*(\mathfrak{L}) \leq \mathsf{FC}(\mathfrak{L}) \leq \mathsf{FC}_*(\mathfrak{L}) + 1$.

As usual, for a linear order $\mathfrak{L} = (L, \leq)$ and a sequence of linear orders $(\mathfrak{L}_i)_{i \in \mathfrak{L}}$ we denote by $\sum_{i \in L} \mathfrak{L}_i$ the $\mathfrak{L}$-sum of the $(\mathfrak{L}_i)_{i \in \mathfrak{L}}$.

We conclude this section by recalling the notion of Dedekind cuts of a linear order. Let $\mathfrak{L} = (L, \leq)$ be a linear order. A *cut* of $\mathfrak{L}$ is a pair $c = (C, D)$ where C is a downward closed subset $C \subseteq L$ and $D = L \setminus C$. We write $\mathsf{Cuts}(\mathfrak{L})$ for the *set of all cuts of* $\mathfrak{L}$. For cuts c, d, we say that c and d are the consecutive cuts around some $l \in L$ if $c = (C, D)$ and $d = (C', D')$ such that $C = \{x \in L \mid x < l\}$ and $C' = \{x \in L \mid x \leq l\}$. $\mathsf{Cuts}(\mathfrak{L})$ can be naturally equipped with an order (also denoted by $\leq$) via $c = (C, D) \leq d = (C', D')$ if $C \subseteq C'$. We say a cut $c = (C, D)$ has no direct predecessor (or direct successor), if it has no direct predecessor (or direct successor, respectively) with respect to $\leq$. Let us finally introduce a notation for values appearing arbitrarily close to some cut (from below or from above, respectively).

Definition 1. *Let $\mathfrak{L} = (L, \leq)$ be a linear order, and $w : \mathsf{Cuts}(\mathfrak{L}) \to A$. For $c = (C, D) \in \mathsf{Cuts}(\mathfrak{L})$, set $\lim_{c^-} w := \{a \in A \mid \forall l \in C \exists l' \in C \quad l \leq l'$ and $w(l') = a\}$ and $\lim_{c^+} w := \{a \in A \mid \forall l \in D \exists l' \in D \quad l' \leq l$ and $w(l') = a\}$.*

2.2 Automata for Scattered Words and Scattered-Automatic Structures

For this section, we fix an arbitrary linear order $\mathfrak{L} = (L, \leq)$.

Definition 2. *Let $\Sigma_\diamond$ be some finite alphabet with $\diamond \in \Sigma_\diamond$. An $\mathfrak{L}$-word (over Σ) is a map $L \to \Sigma_\diamond$. An $\mathfrak{L}$-word w is* finite *if the support $\mathsf{supp}(w) := \{l \in L \mid w(l) \neq \diamond\}$ of w is finite. $W(\mathfrak{L})$ denotes the set of $\mathfrak{L}$-words.*

The usual notion of a convolution of finite words used in automata theory can be easily lifted to the case of $\mathfrak{L}$-words.

Definition 3. *Let w_1, w_2 be $\mathfrak{L}$-words over alphabets Σ_1 and Σ_2, respectively. The* convolution *$w_1 \otimes w_2$ is the $\mathfrak{L}$-word over $\Sigma_1 \times \Sigma_2$ given by $[w \otimes v](l) := (w(l), v(l))$.*

We recall Bruyère and Carton's definition of automata for $\mathfrak{L}$-words [3]. Then we introduce the notion of (finite word) $\mathfrak{L}$-automatic structures generalising the notion of ordinal-automatic structures from [16].

Definition 4. *An* $\mathfrak{L}$-automaton *is a tuple $\mathcal{A} = (Q, \Sigma, I, F, \Delta)$ where Q is a finite set of states, Σ a finite alphabet, $I \subseteq Q$ the initial and $F \subseteq Q$ the final states and Δ is a subset of $(Q \times \Sigma \times Q) \cup \left(2^Q \times Q\right) \cup \left(Q \times 2^Q\right)$ called the transition relation.*

Transitions in $Q \times \Sigma \times Q$ are called *successor transitions*, transitions in $2^Q \times Q$ are called *right limit transitions*, and transitions in $Q \times 2^Q$ are called *left limit transitions*.

Definition 5. *A* run *of $\mathcal{A}$ on the $\mathfrak{L}$-word w is a map $r : \mathsf{Cuts}(\mathfrak{L}) \to Q$ such that*

- *$(r(c), w(l), r(d)) \in \Delta$ for all $l \in L$ and all consecutive cuts c, d around l,*
- *$(\lim_{c^-} r, r(c)) \in \Delta$ for all cuts $c \in \mathsf{Cuts}(\mathfrak{L}) \setminus \{(\emptyset, L)\}$ without direct predecessor,*
- *$(r(c), \lim_{c^+} r) \in \Delta$ for all cuts $c \in \mathsf{Cuts}(\mathfrak{L}) \setminus \{(L, \emptyset)\}$ without direct successor.*

The run r is accepting *if $r((\emptyset, L)) \in I$ and $r((L, \emptyset)) \in F$. The* language *of $\mathcal{A}$ consists of all $\mathfrak{L}$-words w such that there is an accepting run of $\mathcal{A}$ on w. For some $\mathfrak{L}$-word w and states q, q' of $\mathcal{A}$ we write $q \xrightarrow[\mathcal{A}]{w} q'$ if there is a run r of $\mathcal{A}$ on w such that $r((\emptyset, L)) = q$ and $r((L, \emptyset)) = q'$.*

Example 6. The following $\mathfrak{L}$-automaton accepts the set of finite $\mathfrak{L}$-words over the alphabet Σ. Let $\mathcal{A} = (Q, \Sigma, I, F, \Delta)$ with $Q = \{e_l, e_r, n, p\}$, $I = \{n\}, F = \{n, p\}$, and

$$\Delta = \{(n, \diamond, n), (p, \diamond, n)\} \cup \{(n, \sigma, p), (p, \sigma, p) \mid \sigma \in \Sigma \setminus \{\diamond\}\}$$
$$\cup \{(\{n\}, n), (n, \{n\}), (p, \{n\}), (\{p\}, e_l), (e_r, \{p\}), (\{n, p\}, e_l), (e_r, \{n, p\})\}.$$

For each $w \in W(\mathfrak{L})$, $r((C, D)) = \begin{cases} p & \text{if } \max(C) \text{ exists and } \max(C) \in \mathsf{supp}(w) \\ n & \text{otherwise,} \end{cases}$
defines an accepting run if w is a finite $\mathfrak{L}$-word. On an $\mathfrak{L}$-word w with infinite support, the successor transitions require infinitely many occurrences of state p. But then some limit position is marked with an error state e_l or e_r (where l means 'from left' and r 'from right') and the run cannot be continued.

Automata on words (or infinite words or trees or infinite trees) have been applied fruitfully for representing structures. This can be lifted to the setting of $\mathfrak{L}$-words and leads to the notion of (oracle)-$\mathfrak{L}$-automatic structures.

Definition 7. *Fix an $\mathfrak{L}$-word o (called an oracle). A structure $\mathfrak{A} = (A, R_1, R_2, \ldots, R_m)$ is* $\mathfrak{L}$-o-automatic *if there are $\mathfrak{L}$-automata $\mathcal{A}, \mathcal{A}_1, \ldots, \mathcal{A}_m$ such that*

- *$\mathcal{A}$ represents the domain of $\mathfrak{A}$ in the sense that $A = \{w \mid w \otimes o \in L(\mathcal{A})\}$, and*
- *for each $i \leq m$, $\mathcal{A}_i$ represents R_i in the sense that $R_i = \{(w_1, w_2 \ldots, w_{r_i}) \mid w_1 \otimes w_2 \otimes \cdots \otimes w_{r_i} \otimes o \in L(\mathcal{A}_i)\}$, where r_i is the arity of relation R_i.*

We say that an $\mathfrak{L}$-o-automatic structure is finite word $\mathfrak{L}$-o-automatic *if its domain consists only of finite $\mathfrak{L}$-words. Let $\mathcal{F}_{\mathfrak{L}}$ denote the class of all finite word $\mathfrak{L}$-oracle-automatic graphs.*

For the constantly $\diamond$-valued oracle o ($\forall x \in \mathfrak{L}\ o(x) = \diamond$), we call an $\mathfrak{L}$-o-automatic structure $\mathfrak{L}$-automatic. We call some structure $\mathfrak{A}$ scattered-automatic (scattered-oracle-automatic, respectively) if there is some scattered linear order $\mathfrak{L}'$ (and some oracle o) such that $\mathfrak{A}$ is finite word $\mathfrak{L}'$-automatic ($\mathfrak{L}'$-o-automatic, respectively).

Rispal and Carton [14] showed that $\mathfrak{L}$-oracle-automata are closed under complementation if $\mathfrak{L}$ is countable and scattered which implies the following Proposition.

Proposition 8. *If $\mathfrak{L}$ is a countable scattered linear order, the set of finite word $\mathfrak{L}$-o-automatic structures is closed under first-order definable relations.*

2.3 Order Forests

Definition 9. *An (order)* forest *is a partial order $\mathfrak{A} = (A, \leq)$ such that for each $a \in A$, the set $\{a' \in A \mid a \leq a'\}$ is a finite linear order.*

Later we study the rank (also called ordinal height) of $\mathfrak{L}$-automatic well-founded forests. For this purpose we recall the definition of rank. Let $\mathfrak{A} = (A, \leq)$ be a well-founded partial order. Setting $\sup(\emptyset) = 0$ we define the *rank* of $\mathfrak{A}$ by $\mathsf{rank}(a, \mathfrak{A}) = \sup\{\mathsf{rank}(a', \mathfrak{A}) + 1 \mid a' < a \in A\}$ and $\mathsf{rank}(\mathfrak{A}) = \sup\{\mathsf{rank}(a, \mathfrak{A}) + 1 \mid a \in A\}$.

3 Sum- and Box-Augmentation Technique

Delhommé [4] characterised the set of ordinals that can be represented by finite tree-automata. His results relies on a decomposition of definable substructures into *sum-* and *box-augmentations*. Huschenbett [6] and Kartzow et al. [7] introduced a refined notion of *tamely colourable* box-augmentations in order to bound the ranks of tree-automatic linear orders and well-founded order trees, respectively. We first recall the definitions and then show that the decomposition technique also applies to finite word scattered-oracle-automatic structures.

Before we go into details, let us sketch the ideas underlying the sum- and box-augmentation technique. Given an $\mathfrak{L}$-o-automatic structure $\mathfrak{A}$ with domain A and some automaton $\mathcal{A}$ (called *parameter automaton*) that recognises a subset of $A \times W(\mathfrak{L})$, let us denote by $\mathfrak{A}_p$ the substructure of $\mathfrak{A}$ induced by $\mathcal{A}$ and p, i.e., with domain $\{a \in A \mid a \otimes p \in L(\mathcal{A})\}$. The main proposition of this section says that there is a certain class $\mathcal{C}$ of structures (independent of p) such that each $\mathfrak{A}_p$

is a tamely colourable sum-of-box augmentation of structures from $\mathcal{C}$. $\mathcal{C}$ consists of finitely many $\mathfrak{L}$-oracle-automatic structures and scattered-oracle-automatic structures where the underlying scattered linear order has finite condensation rank strictly below that of $\mathfrak{L}$. This allows to compute bounds on structural parameters (like finite condensation rank of linear orders or ordinal height of well-founded partial orders) by induction on the rank of $\mathfrak{L}$. We say a structural parameter φ is compatible with sum-of-box augmentations if for $\mathfrak{A}$ a sum-of-box augmentation of $\mathfrak{A}_1, \dots, \mathfrak{A}_n$, there is a bound on $\varphi(\mathfrak{A})$ in terms of $\varphi(\mathfrak{A}_1), \dots, \varphi(\mathfrak{A}_n)$. The decomposition result tells us that some $\mathfrak{L}$-automatic structure $\mathfrak{A}$ is (mainly) a sum of boxes of scattered-automatic structures where the underlying orders have lower ranks. Thus, by induction hypothesis φ is bounded on these building blocks of $\mathfrak{A}$. Thus, $\varphi(\mathfrak{A})$ is also bounded if φ is compatible with sum- and box-augmentations.

3.1 Sums and Boxes

The next definition recalls the notion of sum- and box-augmentations. We restrict the presentation to structures with one binary relation (but the general case is analogous).

Definition 10. – *A structure* $\mathfrak{A}$ *is a* sum-augmentation *of structures* $\mathfrak{A}_1, \dots, \mathfrak{A}_n$ *if the domain of* $\mathfrak{A}$ *can be partitioned into* n *pairwise disjoint sets such that the substructure induced by the* i*-th set is isomorphic to* $\mathfrak{A}_i$.

– *A structure* $\mathfrak{A} = (A, \leq^A)$ *is a* box-augmentation *of structures* $\mathfrak{B}_1 = (B_1, \leq^{B_1}), \dots, \mathfrak{B}_n = (B_n, \leq^{B_n})$ *if there is a bijection* $\eta : \prod_{i=1}^n B_i \to A$ *such that for all* $1 \leq j \leq n$ *and all* $\bar{b} = (b_1, \dots, b_n) \in B_1 \times \cdots \times B_n$

$$\mathfrak{B}_j \simeq \mathfrak{A}\restriction_{\eta(\{b_1\} \times \cdots \times \{b_{j-1}\} \times B_j \times \{b_{j+1}\} \times \cdots \times \{b_n\})}.$$

– *Let* $\mathcal{C}_1, \dots, \mathcal{C}_n$ *be classes of structures. A structure* $\mathfrak{A}$ *is a* sum-of-box augmentation *of* $(\mathcal{C}_1, \dots, \mathcal{C}_n)$ *if* $\mathfrak{A}$ *is a sum-augmentation of structures* $\mathfrak{B}_1, \dots, \mathfrak{B}_k$ *such that each* $\mathfrak{B}_j$ *is a box-augmentation of structures* $\mathfrak{C}_{j,1}, \dots, \mathfrak{C}_{j,n}$ *with* $\mathfrak{C}_{j,i} \in \mathcal{C}_i$.

Definition 11. *Let* $\mathfrak{A} = (A, \leq)$ *be a sum-of-box augmentation of structures* $\mathfrak{B}_{i,j} = (B_{i,j}, \leq_{i,j})$ *via the map* $\eta : \bigsqcup_{i=1}^n \prod_{j=1}^k B_{i,j} \to A$. *This sum-of-box augmentation is called* tamely colourable *if for each* $1 \leq j \leq k$ *there is a function* $\varphi_j : (\bigsqcup_{i=1}^n B_{i,j})^2 \to C_j$ *with a finite range* C_j *such that the* $(\varphi_j)_{1 \leq j \leq k}$ *determine the edges of* $\mathfrak{A}$ *in the sense that there is a set* $M \subseteq \prod_{j=1}^k C_j$ *such that* $\eta(b_1, \dots, b_k) \leq \eta(b'_1, \dots, b'_k)$ *iff* $(\varphi_1(b_1, b'_1), \dots, \varphi_k(b_k, b'_k)) \in M$.

3.2 Decomposition of Scattered-Automatic-Structures

In this section, we prove that the sum- and box-augmentation technique applies to finite word scattered-oracle-automatic structures. Fix an arbitrary scattered order $\mathfrak{L}$ with $\mathsf{FC}(\mathfrak{L}) = \alpha \geq 1$. Assume that $\mathfrak{L} = \sum_{z \in \mathbb{Z}} \mathfrak{L}_z$ where each $\mathfrak{L}_z$ is a (possibly empty) suborder with $\mathsf{FC}(\mathfrak{L}_z) < \alpha$. We first introduce notation concerning definable subgraphs.

Definition 12. *Let $o \in W(\mathfrak{L})$ be some oracle. Let $\mathfrak{G} = (V, E)$ be a finite word $\mathfrak{L}$-o-automatic graph. For each parameter automaton $\mathcal{A}$ and parameter $p \in W(\mathfrak{L})$, we write $\mathfrak{G}_p^{\mathcal{A}}$ for the induced subgraph of $\mathfrak{G}$ with domain $V_p^{\mathcal{A}} := \{w \in V \mid w \otimes p \in L(\mathcal{A})\}$.*

We write $\mathfrak{G}_p$ and V_p for $\mathfrak{G}_p^{\mathcal{A}}$ and $V_p^{\mathcal{A}}$ if $\mathcal{A}$ is clear from the context.

Definition 13. *Let $c_0 = (C_0, D_0)$ and $c_1 = (C_1, D_1)$ be cuts of $\mathfrak{L}$. For a finite $\mathfrak{L}$-word w we say w is a (c_0, c_1)-parameter if $\mathsf{supp}(w) \subseteq D_0 \cap C_1$, i.e., the support of w is completely between c_0 and c_1.*

For the rest of this section, we fix two numbers $z_0 < z_1 \in \mathbb{Z}$ and define the cuts $c_0 := (\sum_{z<z_0} \mathfrak{L}_z, \sum_{z\geq z_0} \mathfrak{L}_z)$ and $c_1 := (\sum_{z\leq z_1} \mathfrak{L}_z, \sum_{z>z_1} \mathfrak{L}_z)$. We also define the scattered orders $\mathfrak{L}_{\mathbf{L}} := \sum_{z<z_0} \mathfrak{L}_z$ and $\mathfrak{L}_{\mathbf{R}} := \sum_{z>z_1} \mathfrak{L}_z$. The main result of this section is a uniform sum-of-box decomposition of all substructures defined by a given parameter automaton.

Theorem 14. *Let $\mathfrak{G}$ be some finite word $\mathfrak{L}$-oracle-automatic graph (V, E) where E is recognised by some automaton $\mathcal{A}_E$ with state set Q_E and let $\mathcal{A}$ be a parameter automaton with state set Q. There are*

- *a set $\mathcal{C}_{\mathbf{L}}$ of $\exp(|Q|^2 + 2|Q_E|^2)$ many $\mathfrak{L}_{\mathbf{L}}$-oracle-automatic graphs, and*
- *a set $\mathcal{C}_{\mathbf{R}}$ of $\exp(|Q|^2 + 2|Q_E|^2)$ many $\mathfrak{L}_{\mathbf{R}}$-oracle-automatic graphs,*

such that for each (c_0, c_1)-parameter p the subgraph $\mathfrak{G}_p^{\mathcal{A}}$ is a tamely-colourable sum-of-box-augmentation of $(\mathcal{C}_{\mathbf{L}}, \mathcal{F}_{\mathfrak{L}_{z_0}}, \mathcal{F}_{\mathfrak{L}_{z_0+1}}, \ldots, \mathcal{F}_{\mathfrak{L}_{z_1}}, \mathcal{C}_{\mathbf{R}})$.[1]

Proof. Let o be the oracle such that $\mathfrak{G}$ is finite word $\mathfrak{L}$-o-automatic. By definition, we can write $\mathfrak{L}$ as the sum $\mathfrak{L}_{\mathbf{L}} + \mathfrak{L}_{z_0} + \mathfrak{L}_{z_0+1} + \cdots + \mathfrak{L}_{z_1} + \mathfrak{L}_{\mathbf{R}}$. Induced by this decomposition there is a decomposition of any $\mathfrak{L}$-word w as $w = w_{\mathbf{L}} w_{z_0} w_{z_0+1} \ldots w_{z_1} w_{\mathbf{R}}$ such that w_j is an $\mathfrak{L}_j$-word. In particular, our parameter and oracle decompose as

$$p = p_{\mathbf{L}} p_{z_0} p_{z_0+1} \ldots p_{z_1} p_{\mathbf{R}} \quad \text{and} \quad o = o_{\mathbf{L}} o_{z_0} o_{z_0+1} \ldots o_{z_1} o_{\mathbf{R}}.$$

Independently of the choice of the (c_0, c_1)-parameter p, $p_{\mathbf{L}}$ and $p_{\mathbf{R}}$ are constant functions (with value $\diamond$).

In order to construct a sum-of-box decomposition of $\mathfrak{G}_p$, we first define the building blocks of this decomposition. For this purpose, we define equivalence relations $\sim^i_{p\otimes o}$ for each $i \in \{\mathbf{L}, \mathbf{R}, z_0, z_0+1, \ldots, z_1\}$ on $\mathfrak{L}_i$-words as follows. For $\mathfrak{L}_i$-words w, w' set $w \sim^i_{p\otimes o} w'$ if and only if

1. for all $q, q' \in Q$ $\quad q \xrightarrow[\mathcal{A}]{w\otimes p_i \otimes o_i} q' \iff q \xrightarrow[\mathcal{A}]{w'\otimes p_i \otimes o_i} q'$ and
2. for all $q, q' \in Q_E$ $\quad q \xrightarrow[\mathcal{A}_E]{w\otimes w\otimes o_i} q' \iff q \xrightarrow[\mathcal{A}_E]{w'\otimes w'\otimes o_i} q'$.

[1] Recall that $\mathcal{F}_{\mathfrak{L}}$ is the class of all finite word $\mathfrak{L}$-oracle-automatic graphs, see Definition 7.

Note that for fixed i, p, o there are at most $\exp(|Q \times Q| + |Q_E \times Q_E|)$ many $\sim^i_{p\otimes o}$ equivalence classes. As domains of the α_i-oracle-automatic building blocks of our decomposition we use the sets $K(i, w, p, o) := \{x \mid x \sim^i_{p\otimes o} w\}$ for each $\mathfrak{L}_i$-word w. We augment this notation by writing $K(i, v, p, o) := K(i, w, p, o)$ for $\mathfrak{L}$-words v, where w is the restriction of v to $\mathfrak{L}_i$. Now for each $M \subseteq Q_E \times Q_E$ we define a structure $\mathfrak{K}^M(i, w, p, o) = (K(i, w, p, o), E^M)$ where $(w_1, w_2) \in E^M$ if $w_1, w_2 \in K(i, w, p, o)$ and there is a $(q, q') \in M$ such that $q \xrightarrow[\mathcal{A}_E]{w_1 \otimes w_2 \otimes o} q'$. Recall that $p_{\mathbf{L}}$ and $p_{\mathbf{R}}$ are independent of the concrete choice of the (c_0, c_1)-parameter p whence (for fixed o) the sets

$$\mathcal{C}_{\mathbf{L}} := \{\mathfrak{K}^M(\mathbf{L}, w, p, o) \mid M \subseteq Q_E \times Q_E, p \text{ a } (c_0, c_1)\text{-parameter}\}$$
$$\mathcal{C}_{\mathbf{R}} := \{\mathfrak{K}^M(\mathbf{R}, w, p, o) \mid M \subseteq Q_E \times Q_E, p \text{ a } (c_0, c_1)\text{-parameter}\}$$

have each at most $\exp(|Q|^2 + 2|Q_E|^2)$ many elements (up to isomorphisms).

Our next goal is the definition of the function η that witnesses the decomposition claimed in this theorem. For this purpose, let $\sim_{p\otimes o}$ denote the equivalence on $\mathfrak{L}$-words that is the product of the $\sim^i_{p\otimes o}$.[2] Let

$$\eta : \bigsqcup_{[w]\in V_p/\sim_{p\otimes o}} K(\mathbf{L}, w, p, o) \times \left(\prod_{i=z_0}^{z_1} K(i, w, p, o)\right) \times K(\mathbf{R}, w, p, o) \longrightarrow V_p$$
$$(x_{\mathbf{L}}, x_{z_0}, x_{z_0+1}, \ldots, x_{z_1}, x_{\mathbf{R}}) \mapsto x := x_{\mathbf{L}} x_{z_0} x_{z_0+1} \ldots x_{z_1} x_{\mathbf{R}}.$$

It follows from the definitions that η is a well-defined bijection (using the fact that some $\mathfrak{L}$-word x belongs to V_p iff there is a run

$$q_I \xrightarrow[\mathcal{A}]{x_{\mathbf{L}} \otimes p_{\mathbf{L}} \otimes o_{\mathbf{L}}} q_{z_0} \xrightarrow[\mathcal{A}]{x_{z_0} \otimes p_{z_0} \otimes o_{z_0}} q_{z_0+1} \ldots q_{z_1} \xrightarrow[\mathcal{A}]{x_{\mathbf{R}} \otimes p_{\mathbf{R}} \otimes o_{\mathbf{R}}} q_F$$

for some initial state q_I and a final state q_F).

In order to finish the proof, we show that $\mathfrak{G}_p$ is a tamely-colourable sum-of-box-augmentation of $(\mathcal{C}_{\mathbf{L}}, \mathcal{F}_{\mathfrak{L}_{z_0}}, \mathcal{F}_{\mathfrak{L}_{z_0+1}}, \ldots, \mathcal{F}_{\mathfrak{L}_{z_1}}, \mathcal{C}_{\mathbf{R}})$ via η. For any $w \in V_p$, let $\mathfrak{F}_w$ be the restriction of $\mathfrak{G}_p$ to $\eta\big(K(\mathbf{L}, w, p, o) \times \big(\prod_{i=z_0}^{z_1} K(i, w, p, o)\big) \times K(\mathbf{R}, w, p, o)\big)$. It is clear that $\mathfrak{G}_p$ is a sum augmentation of $(\mathfrak{F}_{w_1}, \mathfrak{F}_{w_2}, \ldots, \mathfrak{F}_{w_k})$ for w_i representatives of the $\sim_{p\otimes o}$-classes. From now on let $I_E(F_E)$ denote the initial (final) states of $\mathcal{A}_E$.

1. Fix $w = w_{\mathbf{L}} w_{z_0} w_{z_0+1} \ldots w_{z_1} w_{\mathbf{R}} \in V_p$. We show that $\mathfrak{F}_w$ is a box-augmentation of $(\mathcal{C}_{\mathbf{L}}, \mathcal{F}_{\mathfrak{L}_{z_0}}, \mathcal{F}_{\mathfrak{L}_{z_0+1}}, \ldots, \mathcal{F}_{\mathfrak{L}_{z_1}}, \mathcal{C}_{\mathbf{R}})$. For this purpose, fix $i \in \{\mathbf{L}, \mathbf{R}, z_0, z_0 + 1, \ldots, z_1\}$ and let $\overleftarrow{w} := w_{\mathbf{L}} \ldots w_{i-1}$, $\overleftarrow{o} := o_{\mathbf{L}} \ldots o_{i-1}$, $\overrightarrow{w} := w_{i+1} \ldots w_{\mathbf{R}}$, and $\overrightarrow{o} := o_{i+1} \ldots o_{\mathbf{R}}$. Let M_i be the set defined by

$$(q_1, q_2) \in M_i \iff \exists q_I \in I_E, q_F \in F_E \quad q_I \xrightarrow[\mathcal{A}_E]{\overleftarrow{w} \otimes \overleftarrow{w} \otimes \overleftarrow{o}} q_1 \text{ and } q_2 \xrightarrow[\mathcal{A}_E]{\overrightarrow{w} \otimes \overrightarrow{w} \otimes \overrightarrow{o}} q_F. \tag{1}$$

[2] Thus, for $w = w_{\mathbf{L}} w_{z_0} w_{z_0+1} \ldots w_{z_1} w_{\mathbf{R}}$ and $v = v_{\mathbf{L}} v_{z_0} v_{z_0+1} \ldots v_{z_1} v_{\mathbf{R}}$ we have $w \sim_{p\otimes o} v$ iff $w_i \sim^i_{p\otimes o} v_i$ for all $i \in \{\mathbf{L}, \mathbf{R}, z_0, z_0 + 1, \ldots, z_1\}$.

The function

$$\eta_i^w : K(i,w,p,o) \to V_p, \quad x_i \mapsto w_{\mathbf{L}} w_{z_0} w_{z_0+1} \dots w_{i-1} x_i w_{i+1} \dots w_{z_1} w_{\mathbf{R}}$$

embeds $\mathfrak{K}^{M_i}(i,w,p,o)$ into $\mathfrak{G}_p$ because

$$\begin{aligned}
&\forall x_i, y_i \in K(i,w,p,o) \quad (x_i,y_i) \in E^{M_i} \\
\Leftrightarrow &\exists (q_1,q_2) \in M_i \quad q_1 \xrightarrow[\mathcal{A}_E]{x_i \otimes y_i \otimes o_i} q_2 \\
\overset{(1)}{\Leftrightarrow} &\exists q_I \in I_E, q_F \in F_E \quad q_I \xrightarrow[\mathcal{A}_E]{\overleftarrow{w} \otimes \overleftarrow{w} \otimes \overleftarrow{o}} q_1 \xrightarrow[\mathcal{A}_E]{x_i \otimes y_i \otimes o_i} q_2 \xrightarrow[\mathcal{A}_E]{\vec{w} \otimes \vec{w} \otimes \vec{o}} q_F \\
\Leftrightarrow &(\eta_i^w(x_i), \eta_i^w(y_i)) \in E.
\end{aligned}$$

2. We show that the decomposition is tamely colourable. For all $j \in \{\mathbf{L}, \mathbf{R}, z_0, z_0+1, \dots, z_1\}$, let $c_j : (\bigsqcup_{[w] \in V_p/\sim_{p \otimes o}} K(j,w,p,o))^2 \to Q_E^2$ be the colouring function satisfying $c_j(x_j,y_j) := \{(q,q') \in A_E \mid q \xrightarrow[\mathcal{A}_E]{x_j \otimes y_j \otimes o_j} q'\}$. The colour functions $(c_j)_{j \in \{\mathbf{L},\mathbf{R},z_0,z_0+1,\dots,z_1)}$ determine E because for $w = w_{\mathbf{L}} w_{z_0} w_{z_0+1} \dots w_{z_1} w_{\mathbf{R}}$ and $v = v_{\mathbf{L}} v_{z_0} v_{z_0+1} \dots v_{z_1} v_{\mathbf{R}}$,

$$\begin{aligned}
&(w_{\mathbf{L}} w_{z_0} w_{z_0+1} \dots w_{z_1} w_{\mathbf{R}}, v_{\mathbf{L}} v_{z_0} v_{z_0+1} \dots v_{z_1} v_{\mathbf{R}}) \in E \\
\Longleftrightarrow &\exists q_0, \dots, q_k \in Q_E \left(\begin{array}{l} q_0 \in I_E, q_k \in F_E, \text{ and} \\ q_0 \xrightarrow[\mathcal{A}_E]{w_{\mathbf{L}} \otimes v_{\mathbf{L}} \otimes o_{\mathbf{L}}} q_1 \xrightarrow[\mathcal{A}_E]{w_{z_0} \otimes v_{z_0} \otimes o_{z_0}} q_2 \dots q_{k-1} \xrightarrow[\mathcal{A}_E]{w_{\mathbf{R}} \otimes v_{\mathbf{R}} \otimes o_{\mathbf{R}}} q_k \end{array} \right) \\
\Longleftrightarrow &\exists q_0, \dots, q_k \in Q_E \left(\begin{array}{l} q_0 \in I_E, q_k \in F_E, \text{ and} \\ (q_{i-1}, q_i) \in c_j(w_j, v_j) \text{ with } j = \begin{cases} \mathbf{L} & \text{if } i = 1, \\ \mathbf{R} & \text{if } i = k, \\ z_0 + m & \text{if } i = m \end{cases} \end{array} \right).
\end{aligned}$$

4 Bounds on Scattered-Oracle-Automatic Structures

4.1 FC-Ranks of Linear-Orders

In this section, we first study the question which scattered linear orders are $\mathfrak{L}$-oracle-automatic for a fixed order $\mathfrak{L}$. We provide a sharp bound on the FC-rank. For the upper bound we lift Schlicht and Stephan's result [16] using our new sum- and box-decomposition from the case where $\mathfrak{L}$ is an ordinal:

Theorem 15. *Let $\mathfrak{L}$ be a scattered order of FC_* rank $1+\alpha$ (0, respectively) for some ordinal α. Then every finite word $\mathfrak{L}$-oracle-automatic scattered linear order $\mathfrak{A}$ satisfies $\mathsf{FC}_*(\mathfrak{A}) < \omega^{\alpha+1}$ ($\mathsf{FC}_*(\mathfrak{A}) < \omega^0 = 1$, respectively).*

If $\mathfrak{L}$ is an ordinal of the form $\omega^{1+\alpha}$, Schlicht and Stephan [16] showed that the supremum of the $\mathfrak{L}$-automatic ordinals is exactly $\omega^{\omega^{\alpha+1}}$ whence Theorem 15 is optimal. From our theorem we can also derive the following characterisation of finite FC-rank presentable ordinals.

Corollary 16. *Let $\mathfrak{L}$ be a scattered linear order with $\mathsf{FC}(\mathfrak{L}) < \omega$. The finite word $\mathfrak{L}$-oracle-automatic ordinals are exactly those below $\omega^{\omega^{\mathsf{FC}(\mathfrak{L})+1}}$.*

Here, the oracle is crucial: 0 and 1 are the only finite word $\mathbb{Z}^n$-automatic ordinals if $n \geq 1$ (any $\mathbb{Z}^n$-automatic linear order with 2 elements contains a copy of $\mathbb{Z}$).

4.2 Ranks of Well-Founded Automatic Order Forests

We next study scattered-oracle-automatic well-founded order forests. Kartzow et al. [7] proved compatibility of the ordinal height with sum- and box-augmentations. Together with our decomposition theorem this yields a bound on the height of an $\mathfrak{L}$-oracle-automatic well-founded order forest in terms of $\mathsf{FC}(\mathfrak{L})$. Unfortunately, in important cases these bounds are not optimal. For scattered orders $\mathfrak{L}$ where the set of finite $\mathfrak{L}$-words allow an $\mathfrak{L}$-oracle-automatic order which is scattered, we can obtain better bounds. If $\mathfrak{L}$ is an ordinal or has finite FC-rank, the set of $\mathfrak{L}$-words allows such a scattered ordering. If the finite $\mathfrak{L}$-words admit an $\mathfrak{L}$-automatic scattered order $\leq$, the Kleene-Brouwer ordering of an $\mathfrak{L}$-oracle-automatic well-founded order forest with respect to $\leq$ is $\mathfrak{L}$-oracle-automatic again. Thus, its FC-rank is bounded by our previous result. Adapting a result of Kuske et al.[13] relating the FC-rank of the Kleene-Brouwer ordering with the height of the forest, we derive a bound on the height. Our main result on forests is as follows.

Theorem 17.
- *Let $\mathfrak{L}$ be an ordinal or a scattered linear order with $\mathsf{FC}(\mathfrak{L}) < \omega$. Each $\mathfrak{L}$-oracle-automatic forest $\mathfrak{F} = (F, \leq)$ has rank strictly below $\omega^{\mathsf{FC}(\mathfrak{L})+1}$.*
- *Let $\mathfrak{L}$ be some scattered linear order. Each $\mathfrak{L}$-oracle-automatic forest $\mathfrak{F} = (F, \leq)$ has rank strictly below $\omega^{\omega\cdot(\mathsf{FC}(\mathfrak{L})+1)}$.*

Remark 18. The bounds in the first part are optimal: for each ordinal $\mathfrak{L}$ and each $c \in \mathbb{N}$, we can construct an $\mathfrak{L}$-automatic tree of height $\omega^{\mathsf{FC}(\mathfrak{L})} \cdot c$.

5 Separation of Tree- and Ordinal-Automatic Structures

Theorem 19. *The countable atomless Boolean algebra is not finite word $\mathfrak{L}$-automatic for any ordinal $\mathfrak{L}$.*

This theorem is proved by first showing that, if the atomless Boolean algebra is finite word $\mathfrak{L}$-automatic for some ordinal $\mathfrak{L}$, then it already is ω^n-automatic for some $n \in \mathbb{N}$. This follows because any finite word $\mathfrak{L}$-automatic structure for $\mathfrak{L}$ an ordinal above ω^ω has a sufficiently elementary substructure that has a ω^n-automatic presentation for some $n \in \mathbb{N}$. In the case of the countable atomless Boolean algebra any Σ_3-elementary substructure is isomorphic to the whole algebra. Extending Khoussainov et al.'s monoid growth rate argument for automatic structures (cf. [11]) to the ω^n-setting, we can reject this assumption. This answers a question of Frank Stephan.

References

1. Bárány, V., Grädel, E., Rubin, S.: Automata-based presentations of infinite structures. In: Esparza, J., Michaux, C., Steinhorn, C. (eds.) Finite and Algorithmic Model Theory, pp. 1–76. Cambridge University Press (March 2011)
2. Blumensath, A.: Automatic structures. Diploma thesis, RWTH Aachen (1999)
3. Bruyère, V., Carton, O.: Automata on linear orderings. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 236–247. Springer, Heidelberg (2001)
4. Delhommé, C.: Automaticité des ordinaux et des graphes homogènes. C.R. Acad. Sci. Paris Ser. I 339, 5–10 (2004)
5. Finkel, O., Todorcevic, S.: Automatic ordinals. CoRR, abs/1205.1775 (2012)
6. Huschenbett, M.: The rank of tree-automatic linear orderings. In: Portier, N., Wilke, T. (eds.) STACS. LIPIcs, vol. 20, pp. 586–597. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
7. Kartzow, A., Liu, J., Lohrey, M.: Tree-automatic well-founded trees. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 363–373. Springer, Heidelberg (2012)
8. Kartzow, A., Schlicht, P.: Structures without scattered-automatic presentation. CoRR, arXiv:1304.0912 (2013)
9. Khoussainov, B., Minnes, M.: Model-theoretic complexity of automatic structures. Ann. Pure Appl. Logic 161(3), 416–426 (2009)
10. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: LCC, pp. 367–392 (1994)
11. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: Richness and limitations. Logical Methods in Computer Science 3(2) (2007)
12. Khoussainov, B., Rubin, S., Stephan, F.: Automatic linear orders and trees. ACM Trans. Comput. Log. 6(4), 675–700 (2005)
13. Kuske, D., Liu, J., Lohrey, M.: The isomorphism problem on classes of automatic structures with transitive relations. Transactions of the American Mathematical Society (2011) (to appear)
14. Rispal, C., Carton, O.: Complementation of rational sets on countable scattered linear orderings. Int. J. Found. Comput. Sci. 16(4), 767–786 (2005)
15. Rosenstein, J.G.: Linear Ordering. Academic Press (1982)
16. Schlicht, P., Stephan, F.: Automata on ordinals and automaticity of linear orders. Annals of Pure and Applied Logic 164(5), 523–527 (2013)

Some Classes of Generalised Communicating P Systems and Simple Kernel P Systems

Shankara Narayanan Krishna[1], Marian Gheorghe[2], and Ciprian Dragomir[2]

[1] Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Powai, Mumbai 400076, India
krishnas@cse.iitb.ac.in
[2] Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP, United Kingdom
{m.gheorghe,c.dragomir}@sheffield.ac.uk

Abstract. In this paper we consider four restricted cases of the generalised communicating P systems and study their computational power. In all these cases better results are produced, with respect to the number of cells involved, than those provided so far in the literature. Only one of these results is fully presented, whereas the others are shortly and informally described. Connections between the variants considered and recently introduced kernel P systems are investigated.

1 Introduction

Membrane computing represents a branch of natural computing that brings from cellular biology to computer science a set of concepts, principles and computing mechanisms, with the aim of producing a family of coherent, powerful and efficient computational models, called *membrane systems* or (*P systems*), that are inspired by the behaviour of some cellular processes. The model includes non-deterministic, parallel and distributed processes that abstractly mimic, through a set of evolution rules applied in different compartments, the behaviour of various bio-chemical transformations occurring in living cells. The main model of a *cell-like* P system contains a *hierarchical structure of membranes* (tree structure) delimiting *compartments* (or *regions*); each compartment has a *multiset of objects* that react according to some local *rules* (specific to each compartment). The rules of the most basic cell-like P systems are of the form $u \to v$, where u and v are multisets of objects. The multiset v of such a rule belonging to a compartment consists of objects that remain in this compartment and others that will go into the compartment that contains the current one (the parent compartment) or to compartments contained in it (child compartments). Such a rule replaces the multiset u by v. Many other types of rules and links between compartments have been considered [6, Chapters 4-5, 7-11, & 13-14]. Some of these models replace the hierarchical structure of membranes with a graph structure and the model is called *tissue-like*. Most of these systems, both cellular or tissue models, are computationally complete and when membranes can be multiplied they are able to provide efficient solutions to complex problems [6, Chapters 12 & 21].

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 284–293, 2013.

Membrane computing has introduced a plethora of variants of P systems and the above mentioned classification of such systems is only one of many possible ones.

A special type of P systems emphasises the communicating aspects of these models by using different rules to transport objects across membranes [6, Chapter 5]. One of these models uses the biological metaphor of exchanging pairs of bio-chemical elements between compartments, symport/antiport phenomenon. A special case of such systems, called *generalised communicating P systems*, has been introduced in [7] where only communication rules are used in a very general way, by moving simultaneously symbols from two membranes into other two. This model is inspired by both the symport/antiport paradigm and the way transitions of the Petri nets fire tokens coming from various input places and then sent them to other output places. Some particular cases of such systems, defined in [7], have been investigated in [2].

A different trend of research is presented in [3] where a new class of systems, called *kernel P systems*, has been introduced in order to generalise various features, like the structure of the model, the type of rules and the execution strategy, of previously introduced P systems and to obtain a more versatile and sometimes more expressive model. This class of P systems is also supported by tools providing a rigorous way to formally check various properties of the associated models.

In this paper we further study the computational power of the P systems introduced in [2] and provide better results with respect to the number of membranes. One of these systems is fully investigated, whereas the others are only informally described. Details about the proofs of these results are provided in [4] We also study the relationships between these classes of P systems and kernel P systems [3], showing that all these classes of generalised communicating P systems can be simulated by kernel P systems. This is important as it provides a generic algorithm for translating the former models into the later ones. Finally, an example of a model defined initially by using generalised P systems is transformed into a kernel P system which is then formally verified using Spin [1]. The Promela code for this application is available from [4].

2 Definitions

In this section we introduce the definitions of the main models utilised in this paper, generalised communicating P systems, kernel P systems and register machine.

A *Generalised Communicating P System of degree* n (a GCPS *of degree* n) is a P system consisting of n compartments, called *cells*, linked in a tissue-like manner. These links are implicitly specified by the rules. Formally, a GCPS of degree n is a construct, $\Pi = (O, E, w_1, \ldots, w_n, R_\Pi, i_0)$, where O is a *finite alphabet*; $E \subseteq O$ is the set of *environment symbols*; $w_i \in O^*$, $1 \leq i \leq n$, is the *initial multiset* associated with cell i; R_Π is a finite set of *interaction rules* of the form $r : (a, i \to k;\ b, j \to l)$ (also written as $r : (a, i)(b, j) \to (a, k)(b, l)$),

with $a, b \in O$ and $0 \leq i, j, k, l \leq n$ (0 denotes the environment), and such that if $i = 0$ or $j = 0$ then $a \notin E$ or $b \notin E$; $i_0 \in \{1, \ldots, n\}$ is the *output cell*.

The P system Π consists of n cells, labelled $1, \ldots, n$, containing multisets of objects over O. The environment contains an unbounded number of copies of symbols from E. The cells, and the environment, are supposed to interact through rules from R_Π. A rule $r : (a, i)(b, j) \rightarrow (a, k)(b, l)$, moves a from cell i to k and b from cell j to l. The rules are applied in each step of the computation in the usual non-deterministic and maximally parallel manner. According to maximal parallelism principle, after associating objects to rules, no rule can be applied to any of the remaining objects, if any (see [6]). A *computation* is a sequence of steps where rules are applied as mentioned above. Only halting computations are considered. The result of any computation is obtained in cell i_0 and is given by the number of symbols from this cell at the end of a computation.

The functionality described by each rule $r : (a, i \rightarrow k; b, j \rightarrow l)$, is similar to that of transition in a Petri net model where tokens a and b from the input places are then moved to output places. In such a model the objects are simply moved between compartments. In order to increase or decrease the number of objects in the system, rules involving the environment are used.

A set of restricted variants of GCPS models has been defined in [7] and studied in [2]. These variants utilise a set of particular rules and their computational power has been established in [2]. The specific rules considered in this paper are: (a) *join rules* $(k = l, i \neq k, j \neq k, i \neq j)$; (b) *split rules* $(i = j, i \neq k, i \neq l, k \neq l)$; (c) *presence_move rules* $(i = k, i \neq l, i \neq j, j \neq l)$; and (d) *parallel_shift rules* $(i \neq k, i \neq l, i \neq j, j \neq l)$. We call these classes of systems, *GCPS with minimal interaction*.

The GCPS class with join rules, that will be fully investigated in this paper, describes systems with rules that send always the input symbols, a from i and b from j into the same output cell.

The set of non-negative integer numbers computed by the GCPS Π, as the number of symbols obtained in the output cell i_0, is denoted by $N(\Pi)$. The family of sets of numbers generated by GCPS with at most n cells is denoted by $NGCPS_n$.

If only rules of type (a), (b), (c) or (d) are used then the corresponding family is denoted by $NGCPS(t)_n$, $t \in \{\mathsf{join}, \mathsf{split}, \mathsf{presence\_move}, \mathsf{parallel\_shift}\}$.

A *simple kernel P* (skP) *system of degree n* is a tissue-like P system consisting of a tuple, $sk\Pi = (O, E, C_1, \ldots, C_n, \mu, i_0)$, where O is a *finite alphabet*; $E \subseteq O$ is the set of *environment symbols*; $C_1, \ldots, C_n$ are *compartments*; $\mu = (V, N)$ is an undirected graph, where $V \subseteq \{1, \ldots, n\}$ are vertices and N the edges, defining the *structure of* $sk\Pi$, i.e., the links between compartments, and $i_0 \in \{1, \ldots, n\}$ is the *output compartment*.

An skP system, $sk\Pi$, as above, can be viewed as a set of n compartments, $C_1, \ldots, C_n$, interconnected by edges from N, of an undirected graph μ. Each compartment, C_i, $1 \leq i \leq n$, consists of an initial multiset, w_i, a set of *rewriting and communication rules with guards*, R_i, and an execution strategy, σ_i, $C_i = (w_i, R_i, \sigma_i)$. A rewriting and communication rule with guard has the form

$r : x \to y\ \{\gamma\}$, where $x \in O^+$, y is a string, $y = (a_1, t_1) \cdots (a_h, t_h)$, $h \geq 0$, with $a_j \in O$ and $t_j \in \{0, 1, \ldots, n\}$, $1 \leq j \leq h$, (0 stands for environment) and γ is a *guard*. The guard and the way it is used when the rule r is applied is described below. If r is a rule with guard γ in a compartment C, then γ is considered true wrt the current multiset z of C, if and only if the following happens:

(i) if $\gamma = \theta_1 a_1^{n_1} \ldots \theta_k a_k^{n_k}$, where $a_j \in O$, $\theta_j \in Rel$, and $Rel = \{<, \leqslant, =, \neq, \geqslant, >\}$, then for every j, $1 \leq j \leq k$, $|z|_{a_j} \theta_j\ n_j$ holds ($|z|_a$ means the number of occurrences of a in z);

(ii) if γ is a finite nonempty disjunction of multisets over O with relational operators from Rel, $\gamma = w_1 \mid \cdots \mid w_p$, then there exists j, $1 \leq j \leq p$, such that w_j is true, according to (i).

If γ is not present, then the rule is applied when its left hand side is contained in the current multiset z.

A rule $r : x \to y\ \{\gamma\}$ is applicable to a multiset z if and only if γ is true wrt z and x is a submultiset of z. When r is applied to z, then x will be removed from it and a_i is sent to C_{t_i}, $1 \leq t_i \leq h$. In each computation step the rules of a compartment C_i are applied in accordance with the execution strategy, σ_i. This might be a non-deterministic and maximally parallel way, as usual in membrane computing, or any of the other studied strategies [6], a combination of them or a totally new one. The set of non-negative integer numbers computed by the skP system $skP\!I\!I$ as the number of symbols obtained in the output membrane i_0 is denoted by $N(skP\!I\!I)$. The family of sets of numbers computed by an skP system with at most n compartments is denoted by $NskP_n$. More details about these systems are available in [3].

A *register machine* with k registers is a 5-tuple, $M = (Q, R, q_0, q_f, P)$, where Q is a finite non-empty set of *states*, $R = \{A_1, \ldots, A_k\}$, $k \geq 1$, is a set of *registers*, $q_0 \in Q$ is the *initial state*, $q_f \in Q$ is the *final state*, P is the *set of instructions* of the following forms: (a) (p, A_i^+, q, s) where $p, q, s \in Q$, $p \neq q_f$, $A_i \in R$ (an *increment instruction*, which increments A_i by 1 and moves non-deterministically to either q or s); (b) (p, A_i^-, q, s) where $p, q, s \in Q$, $p \neq q_f$, $A_i \in R$ (a *decrement instruction*, which decrements A_i by 1 and moves to q, if strictly positive, otherwise left it unchanged and jumps to state s).

For every $p \in Q, (p \neq q_f)$, there is exactly one instruction of the form either (p, A_i^+, q, s) or (p, A_i^-, q, s). A configuration of a register machine M, defined above, is given by a $(k+1)$-tuple $(q, m_1, \ldots, m_k)$, where $q \in Q$ and $m_1, \ldots, m_k$ are non-negative integers; q corresponds to the current state of M and $m_1, \ldots, m_k$ are the current numbers stored in the registers $A_1, \ldots, A_k$, respectively. We say that a register machine M with k registers, given as above, *generates* a non-negative integer u if starting from the initial configuration $(q_0, 0, 0, \ldots, 0)$ it enters the final configuration $(q_f, u, 0, \ldots, 0)$. The set of non-negative integers generated by M is denoted by $N(M)$. It is known that register machines are able to generate all recursively enumerable sets of non-negative integers [5]; the family of these sets of numbers is denoted by NRE.

3 Main Results

In [2] it is proved that GCPS models with minimal interaction achieve universality for systems using: (a) 7 cells and *join rules*; (b) 9 cells and *split rules*; (c) 19 cells and *parallel_shift rules*; and (d) 36 cells and *presence_move rules*. In Section 3.1 we improve all these results, by showing that universality can be achieved for these systems when 4, 5, 5 and 6 cells are, respectively, used. The formal proof of these results is only provided for the join rules case, the others are informally described; the proofs of these results are provided in [4]. In Section 3.2 the relationship between GCPS and skP systems is established and an example of a problem solved using these two formalisms is discussed.

3.1 GCPS with Minimal Interaction

The proof below uses for an arbitrary set in NRE a register machine which will be simulated by a GCPS with at most 4 cells.

Theorem 1. $NGCPS(\mathsf{join})_4 = NRE$.

Proof. Consider a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \ldots, A_k\}$, $k \geq 1$, as given in Section 2. To prove the theorem, we construct a GCPS with minimal interaction, $\Pi = (O, E, w_1, w_2, w_3, w_4, R_\Pi, 2)$, using only *join rules*, such that $N(M) = N(\Pi)$. We shall show that for any successful generation of a non-negative number, u, in M, there is a successful computation in Π that leads to u symbols in the output cell, and conversely. Define first $Q^+ = \{p^+ \mid p$ to be the state corresponding to an increment instruction and $Q^- = \{p^- \mid p$ to be the state corresponding to a decrement instruction. Thiswill help in making the proof clearer.

Given M, the GCPS with minimal interaction, Π, is constructed as

(i) $O = Q^+ \cup Q^- \cup \{\overline{q} \mid q \in Q\} \cup \{q_f, c_1, \ldots, c_k\} \cup \{f, g, a, b, e, Y_1, Y_2, Y_3, \dagger, \chi\}$; the number of symbols c_i, $1 \leq i \leq k$, in Π represents the content of register A_i at any given point of time, the symbols $f, a, b, e, Y_1, Y_2, Y_3, \dagger, \chi$ are auxiliary and help in computation.

(ii) $E = Q^+ \cup Q^- \cup \{\overline{q} \mid q \in Q\} \cup \{\dagger, g, q_f\} \cup \{c_i \mid 1 \leq i \leq k\}$ is the set of symbols present in the environment in infinitely many copies.

(iii) $w_1 = fab\chi, w_2 = \lambda, w_3 = Y_2Y_3, w_4 = eY_1$ are the initial multisets of Π; the initial configuration is described by $[\,fab\chi]_1[\,]_2[Y_2Y_3]_3[eY_1]_4E$ where E denotes the environment content.

(iv) Cell 2 is the output cell.

The rules in R_Π are as follows:

I. *Initialisation.*

1. $(a,1)(Y_2,3) \to (a,0)(Y_2,0)$, $(b,1)(Y_3,3) \to (b,0)(Y_3,0)$ and $(q_0^+,0)(e,4) \to (q_0^+,1)(e,1)$.

Rules I.1 are used first. The objects Y_2, Y_3 from cell 3 and a, b from cell 1 go to the environment, while the state of the initial instruction, q_0^+, from the environment and e from cell 4 come to membrane 1. Formally, the new configuration of Π is $[\,efq_0^+\chi]_1[\,]_2[\,]_3[Y_1]_4E \cup \{abY_2Y_3\}$. Symbols $Y_1, Y_2, Y_3, e, f, a, b$ are present only in one copy in Π. This configuration appears before any of the instructions of the register machine is simulated. In the sequel, symbols a and b from the environment will be ignored as these will not be used anymore.

II. *Simulation of an increment instruction* (p, A_i^+, q, s). Let $r \in \{q, s\}$, and let r' stand for r^+ or r^-, depending on whether r is an increment instruction or a decrement instruction.

1. $(p^+, 1)(c_i, 0) \to (p^+, 2)(c_i, 2)$, $1 \le i \le k$, and $(Y_1, 4)(Y_2, 0) \to (Y_1, 2)(Y_2, 2)$;
2. $(p^+, 2)(\overline{r}, 0) \to (p^+, 3)(\overline{r}, 3)$ and $(Y_1, 2)(Y_3, 0) \to (Y_1, 4)(Y_3, 4)$;
3. $(p^+, 3)(Y_2, 2) \to (p^+, 0)(Y_2, 0)$ and $(\overline{r}, 3)(r', 0) \to (\overline{r}, 1)(r', 1)$;
4. $(\overline{r}, 1)(Y_3, 4) \to (\overline{r}, 0)(Y_3, 0)$.

Before we begin the simulation of an increment instruction (p, A_i^+, q, s), let us observe that the current configuration contains: (i) p^+ in cell 1, (ii) the symbols c_i in cell 2 (in zero or more copies), (iii) the symbols Y_2, Y_3 in the environment and (iv) Y_1 in cell 4. If $p = q_0$, then, we formally describe the configuration as $[\,efp^+\chi]_1[\,]_2[\,]_3[Y_1]_4E \cup \{x\}$, where x consists of Y_2Y_3. If $p \neq q_0$, then we have the configuration $[\,ef\overline{p}p^+\chi]_1[w]_2[\,]_3[Y_3Y_1]_4E \cup \{Y_2\}$ where $w = c_1^{l_1} \dots c_k^{l_k}$, $l_i \ge 0$, $1 \le i \le k$. In the sequel we show the sequence of steps in Π while simulating (p, A_i^+, q, s). Without loss of generality, assume that the next state selected is q, corresponding to (q, A_j^+, l, y). The configuration $[\,ef\overline{p}p^+\chi]_1[w]_2[\,]_3[Y_3Y_1]_4E \cup \{Y_2\}$ evolves into $[\,ef\chi]_1[p^+c_iwY_1Y_2]_2[\,]_3[\,]_4 E \cup \{Y_3\}$, which leads to $[\,ef\chi]_1[c_iwY_2]_2[p^+\overline{q}]_3[Y_1Y_3]_4E$, and this one in turn, evolves into $[\,ef\overline{q}q^+\chi]_1[c_iw]_2[\,]_3[Y_1Y_3]_4E \cup \{Y_2\}$. This one is further transformed into $[\,ef\chi]_1[q^+c_jc_iwY_1Y_2]_2[\,]_3[\,]_4E \cup \{Y_3\}$.

III. *Simulation of a decrement instruction* (p, A_i^-, q, s).

1. $(p^-, 1)(c_i, 2) \to (p^-, 4)(c_i, 4)$, $1 \le i \le k$, and $(Y_1, 4)(Y_2, 0) \to (Y_1, 2)(Y_2, 2)$;
2. $(p^-, 4)(\overline{q}, 0) \to (p^-, 3)(\overline{q}, 3)$ and $(Y_1, 2)(Y_3, 0) \to (Y_1, 4)(Y_3, 4)$;
3. $(p^-, 3)(Y_2, 2) \to (p^-, 0)(Y_2, 0)$ and $(\overline{q}, 3)(q', 0) \to (\overline{q}, 1)(q', 1)$;
4. $(\overline{q}, 1)(Y_3, 4) \to (\overline{q}, 0)(Y_3, 0)$;
5. $(p^-, 1)(Y_2, 2) \to (p^-, 3)(Y_2, 3)$;
6. $(p^-, 3)(f, 1) \to (p^-, 2)(f, 2)$;
7. $(p^-, 2)(\overline{s}, 0) \to (p^-, 3)(\overline{s}, 3)$ and $(f, 2)(Y_2, 3) \to (f, 4)(Y_2, 4)$;
8. $(p^-, 3)(Y_2, 4) \to (p^-, 0)(Y_2, 0)$, $(\overline{s}, 3)(s', 0) \to (\overline{s}, 1)(s', 1)$ and $(g, 0)(f, 4) \to (g, 1)(f, 1)$;
9. $(Y_3, 4)(\overline{s}, 1) \to (Y_3, 0)(\overline{s}, 0)$;
10. $(f, 2)(\dagger, 0) \to (f, 3)(\dagger, 3)$, $(\chi, 1)(\dagger, 3) \to (\chi, 2)(\dagger, 2)$, $(\chi, 2)(\dagger, 0) \to (\chi, 3)(\dagger, 3)$ and $(\chi, 3)(\dagger, 0) \to (\chi, 2)(\dagger, 2)$.

Rules III.1 to III.4 are used when the contents of register A_i is greater than 0, while rules III.5 to III.9 are used when register A_i is 0; rules III.10 handle exceptions.

Before we begin the simulation of a decrement instruction (p, A_i^-, q, s), the configuration we have is: (i) $\overline{p}, p^-, \chi, f$ are present in cell 1; (ii) the symbols

c_i are present in cell 2 (in zero or more copies); (iii) the symbol Y_2 is in the environment; while (iv) Y_1, Y_3 are in cell 4.

Case 1. Content of register A_i is non-zero. In this case, we start with rule III.1. Using this, p^- removes a copy of c_i from cell 2; both p^- and c_i move to cell 4; in parallel, Y_2 comes to cell 2 along with Y_1. This is followed by p^- getting the symbol $\overline{q}$ from the environment into cell 3; in parallel, the symbols Y_1, Y_3 enter cell 4. Next, the symbols p^-, Y_2 return to the environment using rule III.3 : $(p^-, 3)(Y_2, 2) \to (p^-, 0)(Y_2, 0)$, while $\overline{q}$ gets q' from the environment into membrane 1; finally, $\overline{q}$ and Y_3 return to the environment.

Case 2. Content of register A_i is zero. In this case, when p^- is in cell 1, the rule $(p^-, 1)(c_i, 2) \to (p^-, 4)(c_i, 4)$ in III.1 cannot be used. However, symbols Y_1, Y_2 move to cell 2 using the other rule in III.1. In this case, we have the symbol p^- in cell 1, and Y_2 in cell 2. Then we use rule III.5, moving both p^-, Y_2 to cell 3; in parallel, symbols Y_1, Y_3 move to cell 4 by III.2. This is followed by p^- moving to cell 2 along with f using rule III.6. p^- in cell 2 gets $\overline{s}$ from the environment, both symbols move to cell 3. In parallel, symbols f, Y_2 move to cell 4. Next, p^-, Y_2 return to the environment from cells 3 and 4, respectively, while $\overline{s}$ obtains the symbol s' from the environment; the symbol f goes back to cell 1 using the symbol g from the environment. Finally, $\overline{s}$ and Y_3 return to the environment.

Exception Handling. Note that in Case 1, instead of using III.3: $(p^-, 3)(Y_2, 2) \to (p^-, 0)(Y_2, 0)$, one could use the rule III.6 : $(p^-, 3)(f, 1) \to (p^-, 2)(f, 2)$. If this is done, we have f in cell 2, and Y_2 also in cell 2. Hence, the rule III.7 : $(f, 2)(Y_2, 3) \to (f, 4)(Y_2, 4)$ is not applicable in the next step; and, rule III.10 is used, giving a non-halting computation with no output. Assume we are at $[\,ef\overline{p}p^-\chi]_1[\,c_iw]_2[\,\,]_3[Y_3Y_1]_4E \cup \{Y_2\}$. Using III.1, this evolves into $[\,ef\chi]_1[\,wY_1Y_2]_2[\,\,]_3[p^-c_i]_4E \cup \{Y_3\}$, followed by use of III.2 obtaining $[\,ef\chi]_1[\,wY_2]_2[p^-\overline{q}]_3[c_iY_1Y_3]_4E$. To this, if we use the second rule of III.3 and III.6, we obtain $[\,e\chi\overline{q}q']_1[\,wY_2p^-f]_2[\,]_3[c_iY_1Y_3]_4E$. Since there is no Y_2 in cell 3, we use III.10, III.4, obtaining $[\,e\chi q']_1[\,wY_2]_2[p^-\overline{s}f\,\dagger]_3[c_iY_1]_4E \cup \{Y_3\}$. Irrespective of any other rule, we now obtain $\chi, \dagger$ in cell 2, then $\chi, \dagger$ in cell 3 and so on.

IV. *Halting* $(q_f, 1)(Y_1, 4) \to (q_f, 0)(Y_1, 0)$. Once we obtain q_f in cell 1, the symbol Y_1 is removed from the system to the environment. This way, the chain of actions $(Y_1, 4)(Y_2, 0) \to (Y_1, 2)(Y_2, 2)$, $(Y_1, 2)(Y_3, 0) \to (Y_1, 4)(Y_3, 4)$ can be stopped. The number of symbols $c_i, 1 \leq i \leq k$, in cell 2 is the output.

It is also clear that every set of numbers generated by a register machine can be simulated by system Π, belonging to $NGCPS(\mathsf{join})_4$. Hence, the statement of the theorem follows. □

Similar techniques can be applied for the other types of GCPS model with minimal interaction in order to achieve Turing completeness.

The following results are proved in [4]:

(a) $NGCPS(\mathsf{split})_5 = NRE$,
(b) $NGCPS(\mathsf{parallel\_shift})_5 = NRE$, and
(c) $NGCPS(\mathsf{presence\_move})_6 = NRE$.

3.2 GCPS with Minimal Interaction and skP Systems

In this section we show how a GCPS with n cells is simulated by an skP system with two compartments. The proof employs a standard procedure of mapping communication rules of a P system with a fixed number of compartments into one compartment P system using multiset rewriting rules. Three additional multiset rewriting rules with guards are considered for controlling the end of the computation and an additional compartment is introduced in order to collect the result.

Theorem 2. *For any GCPS with n cells, Π, there is an skP system, $sk\Pi$, with two compartments such that $N(\Pi) = N(sk\Pi)$.*

Proof. Given a GCPS of degree n $\Pi = (O, E, w_1, \ldots, w_n, R, i_0)$, where R contains rules of the form $r : (a,i)(b,j) \to (a,k)(b,l)$, with $a, b \in O$ and $0 \leq i, j, k, l \leq n$, and such that if $i = 0$ or $j = 0$, then $a \notin E$ or $b \notin E$, then the following skP system with two compartments is built, $sk\Pi = (O', E', C_1, C_2, \mu, 2)$, where $O' = O \cup \{[a,i] \mid a \in O, 1 \leq i \leq n\} \cup \{[a,0] \mid a \in O \setminus E\} \cup \{\alpha, \alpha', \alpha''\}$, where $\alpha, \alpha', \alpha''$ are new symbols; $E' = \emptyset$—there is no need for environment symbols; $C_1 = (w_1', R_1)$, will contain all the necessary multisets and rules to simulate the behaviour of the GCPS Π and $C_2 = (w_2', R_2)$ is used only to collect the result—the initial multisets and the rules will be defined later; $\mu = (V, N)$ is the graph, $V = \{1, 2\}$, $N = \{\{1, 2\}\}$, showing one single link between the above compartments.

For each rule $r : (a,i)(b,j) \to (a,k)(b,l)$, $1 \leq i, j, k, l \leq n$, a rule $r' : [a,i][b,j] \to [a,k][b,l]$ is constructed. If $i = 0$, i.e., a is brought from the environment, then ($j \neq 0$) $r' : [b,j] \to [a,k][b,l]$, for $a \in E$, or $r' : [a,0][b,j] \to [a,k][b,l]$, for if $a \in O \setminus E$ (similarly when $j = 0$ and $i \neq 0$). When $k = 0$ then $r' : [a,i][b,j] \to [b,l]$ (similarly for $l = 0$); in these cases one of i or j can be 0 as well. When both k and l are 0 then the right hand side of r' is λ. For each rule $r' : x \to y$ defined above, a rule $r'' : x \to y \; \{= \alpha\}$ is constructed. If $x_1, \ldots, x_p$ are all the distinct left hand side parts of the rules r'', and if $x_h = a_1^{k_1} \ldots a_{i_h}^{k_{i_h}}$, then the guard notation $\geq x_h$ ($\neq x_h$) stands for the guard $\geq a_1^{k_1} \cdots \geq a_{i_h}^{k_{i_h}}$ ($\neq a_1^{k_1} \cdots \neq a_{i_h}^{k_{i_h}}$). The rules $r_{\alpha'} : \alpha' \to \alpha \; \{\geq x_1 \mid \cdots \mid \geq x_p\}$, $r_{\alpha''} : \alpha' \to \alpha'' \; \{\neq x_1 \cdots \neq x_p\}$, $r_a : [a, i_0] \to (a, 2) \; \{= \alpha''\}$, $a \in O$, and $r_\alpha : \alpha \to \alpha'$ are also considered.

The compartment C_1 consists of the initial multiset
$w_1' = [a_{1,1}, 1] \ldots [a_{1,p_1}, 1] \ldots [a_{n,1}, n] \ldots [a_{n,p_n}, n]\alpha$, given that $w_i = a_{i,1} \ldots a_{i,p_i}$ is the initial multiset of the cell i, $1 \leq i \leq n$, and the rule set
$R_1 = \{r'' \mid r \in R\} \cup \{r_\alpha, r_{\alpha'}, r_{\alpha''}\} \cup \{r_a \mid a \in O\}$.

The second compartment is $C_2 = (\lambda, \emptyset)$.

The skP system $skΠ$ simulates the GCPS $Π$ as follows: each computation step using rules $r_1, \ldots, r_t$ from R in $Π$ is performed in two stages in $skΠ$: in the first one the rules $r_1'', \ldots, r_t''$ corresponding to $r_1, \ldots, r_t$, respectively, and r_α are used, and in the second stage either $r_{\alpha'}$ or $r_{\alpha''}$ is used; $r_{\alpha'}$ is used when another computation step will be executed and $r_{\alpha''}$ is used when the computation stops. In this final situation the rules r_a, $a \in O$, are used to transfer all the objects corresponding to the output cell, i_0, into compartment 2. □

For the GCPS with minimal interaction we can get the following result:

Corollary 1. *For any GCPS with minimal interaction having n cells, $Π$, there is an skP system, $skΠ$, with two compartments such that $N(Π) = N(skΠ)$.*

The above result provides a general algorithm to transform a GCPS (with minimal interaction) into an skP system computing the same set of numbers as the GCPS. As it can be observed from Theorem 2 the number of rules of $skΠ$ is equal to the number of rules plus the size of the alphabet of $Π$ plus 3 and each computation step in $Π$ is simulated by two steps in $skΠ$. In many examples this generic procedure may be replaced by a more efficient way of solving them.

In the sequel it will be presented an example where the skP system provided is much more efficient than the GCPS.

Example. We show how to compute n^2, $n \geq 1$, by using an skP system. Let us first recall that a solution to this problem has been provided in [7] where a GCPS model, $Π$, is used. Although the solution based on a GCPS model is elegant and modular, it uses 5 modules of 2 types and another simple rule. One of these modules is implemented with 2 GCPS rules (and is used 3 times) and the other one has 6 (and appears twice)—see [7]. Unless we can make some optimisations in the final GCPSl obtained from these modules, the current solution uses 19 ($3 \times 2 + 2 \times 6 + 1$) rules. Using the procedure described in Theorem 2 we obtain an skP system with $22 + s$ rules, where s is the size of the alphabet of $Π$. A more efficient solution using an skP system having only 3 rules is provided below.

According to [7], n^2 is computed by using the two recursive definitions:

$$c_i = c_{i-1} + b_{i-1} \text{ (i), and } b_i = b_{i-1} + 2 \text{ (ii)},$$

with $1 \leq i \leq n$ and initial values $b_0 = 1$ and $c_0 = 0$, we obtain $c_n = n^2$.

The skP system for this problem is $skΠ = (O, E, C_1, C_2, \mu, 2)$, where $O = \{a, b, c\}$; $E = \emptyset$; $C_1 = (w_1, R_1, \sigma_1)$, where $w_1 = a^n b$, $R_1 = \{r_1 : a \to bb$; $r_2 : b \to b(c, 2)$ $\{\geq a\}\}$ and the execution strategy, σ_1, requires that in each step, r_2 is executed in a maximally parallel manner and then r_1; $C_2 = (\lambda, \emptyset, \sigma_2)$—is used only to collect the result, with no initial multiset, no rules and execution strategy; $\mu = (V, N)$ is the graph, $V = \{1, 2\}$, $N = \{\{1, 2\}\}$, showing one single link between the above membranes. It is clear that for obtaining n^2, $n \geq 1$, it is sufficient to get c^{n^2} in C_2. This is computed as follows: the initial multiset, w_1 of C_1 contains a^n, similarly to the input value n inserted into the GCPS in [7] for computing finally n^2; b stands for $b_0 = 1$ and $w_2 = \lambda$ corresponds to $c_0 = 0$; in each step j, $1 \leq j \leq n$, the rule r_2 is applied in a maximally parallel manner,

i.e., $2j - 1$ times, followed by one application of r_1; in step n, r_1 rewrites the last symbol a and stops the computation. It is clear that the rule r_1 implements the recursion (ii) and r_2 works to get (i). These observations help to show that b's in C_1 and c's in C_2 in any two consecutive steps verify (i) and (ii). This has been checked for our Spin implementation by using LTL queries. Some other properties of this system can be checked, like relationships between occurrences of a's in consecutive steps or between a's and b's in the same step. Details about the codification of the problem and the verification of its invariants in Spin are provided in [4].

4 Conclusions

In this paper we have further improved the completeness results for GCPS models with minimal interaction, which was formulated as an open problem in [7], and have established a way to simulate them with skP systems. Finally an example has shown that in some cases much better solutions can be provided by codifying a problem directly into the skP formalism. For the example chosen, a formal verification procedure, in terms of model checking using Spin, is also applied.

Acknowledgements. MG was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0688. CD's work was funded by a DTA studentship. The authors would like to thank Gh. Păun for reading a first draft of this paper and making valuable comments and to the anonymous reviewers for their useful observations that helped improving the paper.

References

1. Ben-Ari, M.: Principles of the Spin model checker. Springer (2008)
2. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. Theoretical Computer Science 412, 124–135 (2011)
3. Gheorghe, M., Ipate, F., Lefticaru, R., Pérez-Jiménez, M.J., Ţurcanu, A., Valencia Cabrera, L., García-Quismondo, M., Mierlă, L.: 3-Col problem modelling using simple kernel P systems. International Journal of Computer Mathematics (accepted)
4. Krishna, S.N., Gheorghe, M., Dragomir, C.: Some classes of generalised communicating P systems and simple kernel P systems. Technical report, CS-12-03, University of Sheffield, (2013), `http://staffwww.dcs.shef.ac.uk/people/M.Gheorghe/research/paperlist.html`
5. Minsky, M.: Computation: Finite and infinite machines. Prentice Hall, Englewood Cliffs (1967)
6. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford handbook of membrane computing. Oxford University Press (2010)
7. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: Generalized communicating P systems. Theoretical Computer Science 404(1-2), 170–184 (2008)

Closed Choice for Finite and for Convex Sets*

Stéphane Le Roux[1] and Arno Pauly[2]

[1] Department of Mathematics, Technische Universität Darmstadt, Germany
leroux@mathematik.tu-darmstadt.de
[2] Clare College, University of Cambridge, United Kingdom
Arno.Pauly@cl.cam.ac.uk

Abstract. We investigate choice principles in the Weihrauch lattice for finite sets on the one hand, and convex sets on the other hand. Increasing cardinality and increasing dimension both correspond to increasing Weihrauch degrees. Moreover, we demonstrate that the dimension of convex sets can be characterized by the cardinality of finite sets encodable into them. Precisely, choice from an $n+1$ point set is reducible to choice from a convex set of dimension n, but not reducible to choice from a convex set of dimension $n-1$.

1 Introduction

In the investigation of the computational content of mathematical theorems in the Weihrauch lattice, variations of closed choice principles have emerged as useful canonic characterizations [1,3,6]. Closed choice principles are multivalued functions taking as input a non-empty closed subset of some fixed space, and have to provide some element of the closed set as output. In [1,3] the influence of the space on the computational difficulty of (full) closed choice was investigated, whereas in [6] it turned out that the restriction of choice to connected closed subsets of the unit hypercube is equivalent to Brouwer's Fixed Point theorem for the same space.

Here the restrictions of closed choice to convex subsets (of the unit hypercube of dimension n), and to finite subsets (of a compact metric space) are the foci of our investigations. Via the connection between closed choice and non-deterministic computation [1, 7, 20, 25], in particular the latter problem is prototypic for those problems having only finitely many correct solutions where wrong solutions are identifiable. As such, some parts may be reminiscent of some ideas from [8, 15].

One of our main results shows that choice for finite sets of cardinality $n+1$ can be reduced to choice for convex sets of dimension n, but not to convex choice of dimension $n-1$. This demonstrates a computational aspect in which convex sets get more complicated with increasing dimension. As such, our work also

* This work benefited from the Royal Society International Exchange Grant IE111233 and the Marie Curie International Research Staff Exchange Scheme *Computable Analysis*, PIRSES-GA-2011- 294962.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 294–305, 2013.

continues the study of the structural complexity of various classes of subsets of the unit hypercubes done in [12, 14].

Some of the techniques used to establish our main results are promising with regards to further applicability to other classes of choice principles, or to even more general Weihrauch degrees. These techniques are presented in Section 2.

Due to lack of space, some of the proofs had to be omitted. A version including proofs and additional results is as available as [13].

1.1 Weihrauch Reducibility

We briefly recall some basic results and definitions regarding the Weihrauch lattice. The original definition of Weihrauch reducibility is due to Weihrauch and has been studied for many years (see [10, 16, 17, 21–23]). Rather recently it has been noticed that a certain variant of this reducibility yields a lattice that is very suitable for the classification of mathematical theorems (see [1,3–5,9,11,18,19]). A basic reference for notions from computable analysis is [24]. The Weihrauch lattice is a lattice of multi-valued functions on represented spaces. A represented space is a pair (X, δ_X) where $\delta_X :\subseteq \mathbb{N}^{\mathbb{N}} \to X$ is a partial surjection, called representation. In general we use the symbol "$\subseteq$" in order to indicate that a function is potentially partial. Using represented spaces we can define the concept of a realizer. We denote the composition of two (multi-valued) functions f and g either by $f \circ g$ or by fg.

Definition 1 (Realizer). *Let $f :\subseteq (X, \delta_X) \rightrightarrows (Y, \delta_Y)$ be a multi-valued function on represented spaces. A function $F :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ is called a* realizer *of f, in symbols $F \vdash f$, if $\delta_Y F(p) \in f\delta_X(p)$ for all $p \in \mathrm{dom}(f\delta_X)$.*

Realizers allow us to transfer the notions of computability and continuity and other notions available for Baire space to any represented space; a function between represented spaces will be called *computable*, if it has a computable realizer, etc. Now we can define Weihrauch reducibility.

Definition 2 (Weihrauch reducibility). *Let f, g be multi-valued functions on represented spaces. Then f is said to be* Weihrauch reducible *to g, in symbols $f \leq_W g$, if there are computable functions $K, H :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ such that $K\langle \mathrm{id}, GH\rangle \vdash f$ for all $G \vdash g$. Moreover, f is said to be* strongly Weihrauch reducible *to g, in symbols $f \leq_{sW} g$, if there are computable functions K, H such that $KGH \vdash f$ for all $G \vdash g$.*

Here $\langle , \rangle$ denotes some standard pairing on Baire space.

There are two operations defined on Weihrauch degrees that are used in the present paper, product $\times$ and composition $\star$. The former operation was originally introduced in [4], the latter in [5]. Informally, the product allows using both involved operations independently, whereas for the composition the call to the first operation may depend on the answer received from the second.

Definition 3. *Given $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$, $g :\subseteq \mathbf{U} \rightrightarrows \mathbf{V}$, define $f \times g :\subseteq (\mathbf{X} \times \mathbf{U}) \rightrightarrows (\mathbf{Y} \times \mathbf{V})$ via $(y, v) \in (f \times g)(x, u)$ iff $y \in f(x)$ and $v \in g(u)$.*

Definition 4. *Given* $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$, $g :\subseteq \mathbf{U} \rightrightarrows \mathbf{V}$, *let*

$$f \star g := \sup_{\leq_W} \{f' \circ g' \mid f' \leq_W f \wedge g' \leq_W g\}$$

where f', g' *are understood to range over all those multivalued functions where the composition is defined.*

Both $\times$ and $\star$ are associative, but only $\times$ is commutative. We point out that while it is not obvious that the supremum in the definition of $\star$ always exists, this is indeed the case, hence $\star$ is actually a total operation.

1.2 Closed Choice and Variations Thereof

The space of continuous functions from a represented space $\mathbf{X}$ to $\mathbf{Y}$ has a natural representation itself, as a consequence of the UTM-theorem. This represented space is denoted by $\mathcal{C}(\mathbf{X}, \mathbf{Y})$. A special represented space of utmost importance is Sierpiński space $\mathbb{S}$ containing two elements $\{\top, \bot\}$ represented by $\delta_{\mathbb{S}} : \mathbb{N}^{\mathbb{N}} \to \mathbb{S}$ where $\delta_{\mathbb{S}}(0^{\mathbb{N}}) = \bot$ and $\delta_{\mathbb{S}}(p) = \top$, iff $p \neq 0^{\mathbb{N}}$. The space $\mathcal{A}(\mathbf{X})$ of closed subsets of $\mathbf{X}$ is obtained from $\mathcal{C}(\mathbf{X}, \mathbb{S})$ by identifying a set $A \subseteq \mathbf{X}$ with the characteristic function $\chi_{X \setminus A} : \mathbf{X} \to \mathbb{S}$ of its complement.

For a computable metric space $\mathbf{X}$, an equivalent representation $\psi_- : \mathbb{N}^{\mathbb{N}} \to \mathcal{A}(\mathbf{X})$, can be defined by $\psi_-(p) := X \setminus \bigcup_{i=0}^{\infty} B_{p(i)}$, where B_n is some standard enumeration of the open balls of X with center in the dense subset and rational radius (possibly 0). The computable points in $\mathcal{A}(X)$ are called *co-c.e. closed sets*. We are primarily interested in closed choice on computable metric spaces; additionally, most of our considerations pertain to compact spaces.

Definition 5 (Closed Choice, [3]). *Let* $\mathbf{X}$ *be a represented space. Then the* closed choice *operation of this space is defined by* $C_{\mathbf{X}} :\subseteq \mathcal{A}(\mathbf{X}) \rightrightarrows \mathbf{X}, A \mapsto A$ *with* $\mathrm{dom}(C_X) := \{A \in \mathcal{A}(\mathbf{X}) : A \neq \emptyset\}$.

Intuitively, $\mathrm{C}_{\mathbf{X}}$ takes as input a non-empty closed set and it produces an arbitrary point of this set as output. Hence, $A \mapsto A$ means that the multi-valued map $\mathrm{C}_{\mathbf{X}}$ maps the input set $A \in \mathcal{A}(\mathbf{X})$ to the points in $A \subseteq \mathbf{X}$ as possible outputs.

Definition 6. *For a represented space* $\mathbf{X}$ *and* $1 \leq n \in \mathbb{N}$, *let* $C_{\mathbf{X},\sharp=n} := C_{\mathbf{X}}|_{\{A \in \mathcal{A}(\mathbf{X}) \mid |A|=n\}}$ *and* $C_{\mathbf{X},\sharp\leq n} := C_{\mathbf{X}}|_{\{A \in \mathcal{A}(\mathbf{X}) \mid 1 \leq |A| \leq n\}}$.

More generally, for any choice principle the subscript $\sharp = n$ denotes the restriction to sets of cardinality n, and the subscript $\sharp \leq n$ to sets of cardinality less or equal than n. In the same spirit, the subscript $\lambda > \epsilon$ denotes the restriction to sets of outer diameter greater than ϵ, and $\mu > \epsilon$ the restriction to those sets where some value μ is greater than ε.

Definition 7. *Let* $XC_n := C_{[0,1]^n}|_{\{A \in \mathcal{A}([0,1]^n) \mid A \text{ is convex }\}}$.

The proof of the following proposition has been inspired by the proof of [15, Theorem 3.1] by Longpré et al. , which the proposition generalizes in some sort. In fact, the study of $\mathrm{C}_{[0,1],\sharp=m}$ is quite closely related to the theme of [15].

Proposition 1. *Let* $\mathbf{X}$ *be a computably compact computable metric space. Then* $C_{\mathbf{X},\sharp=n} \leq_{sW} C_{\{0,1\}^{\mathbb{N}},\sharp=n}$ *and* $C_{\mathbf{X},\sharp\leq n} \leq_{sW} C_{\{0,1\}^{\mathbb{N}},\sharp\leq n}$.

Proof. We associate a labeled finitely-branching infinite tree (with given bounds) with the space $\mathbf{X}$, where each vertex is labeled by an open subset of $\mathbf{X}$. The root is labeled by $\mathbf{X}$. Then we find a finite open cover of $\mathbf{X}$ by open balls $B(x_1, 2^{-1}), \ldots, B(x_n, 2^{-1})$ using the computable dense sequence and the computable compactness provided by $\mathbf{X}$. The $B(x_i, 2^{-1})$ form the second layer of the tree. For the third layer, each $B(x_i, 2^{-1})$ (whose closure is computably compact) is covered by finitely many $B(x_{i,j}, 2^{-2})$, and we then use $B(x_i, 2^{-1}) \cap B(x_{i,j}, 2^{-2})$ as labels. This process is iterated indefinitely, yielding finer and finer coverings of the space at each layer.

Any closed subset of a computably compact space is compact (in a uniform way), so we can assume the input to $\mathrm{C}_{\mathbf{X},\sharp=n}$ ($\mathrm{C}_{\mathbf{X},\sharp\leq n}$) to be a compact set A of cardinality n (less-or-equal n). On any layer of the tree, there are n vertices such that the union of their labels covers A. It is recognizable when an open set includes a compact set, so we shall find suitable n vertices eventually. Also, we can require that the vertices chosen on one level are actually below those chosen on the previous level.

Any finitely-branching tree with at most n nodes per layer can be encoded in a binary tree with at most n nodes per layer, and this just represents a closed subset of Cantor space with cardinality less-or-equal n. If the initial set A has exactly n elements, at some finite stage in the process any of the n open sets used to cover it will actually contain a point of A. Hence, from that stage onwards no path through the finitely-branching tree dies out, which translates to no path through the binary tree dying out. But then, the closed subset of Cantor space has cardinality exactly n.

Any point from the subset of Cantor space is an infinite path through the two trees constructed, hence, gives us a sequence of rational balls with diameter shrinking to 0 and non-empty intersection. This provides us with a name of a point in the original set, completing the reduction.

It is rather obvious that if $\mathbf{X}$ is a co-c.e. closed subspace of $\mathbf{Y}$, then $\mathrm{C}_{\mathbf{X},\sharp=n} \leq_{sW} \mathrm{C}_{\mathbf{Y},\sharp=n}$ and $\mathrm{C}_{\mathbf{X},\sharp\leq n} \leq_{sW} \mathrm{C}_{\mathbf{Y},\sharp\leq n}$ (compare [1, Section 4]). We recall that a computable metric space $\mathbf{X}$ is called *rich*, if it has Cantor space as computably isomorphic to a subspace (then this subspace automatically is co-c.e. closed). [2, Proposition 6.2] states that any non-empty computable metric space without isolated points is rich.

Corollary 1. *Let* $\mathbf{X}$ *be a rich computably compact computable metric space. Then* $C_{\mathbf{X},\sharp=n} \equiv_{sW} C_{\{0,1\}^{\mathbb{N}},\sharp=n}$ *and* $C_{\mathbf{X},\sharp\leq n} \equiv_{sW} C_{\{0,1\}^{\mathbb{N}},\sharp\leq n}$.

By inspection of the proof of Proposition 1, we notice that the names produced there as inputs to $\mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp=n}$ or $\mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp\leq n}$ have a specific form: If we consider the closed subsets of Cantor space to be represented as the sets of paths of infinite binary trees, the trees involved will have exactly n vertices on all layers admitting at least n vertices in a complete binary tree. The names used for

$\mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp=n}$ moreover have the property that from some finite depths onwards, all vertices have exactly one child. The restrictions of $\mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp=n}$ and $\mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp\leq n}$ to inputs of the described type shall be denoted by $\mathrm{C}_{\sharp=n}$ and $\mathrm{C}_{\sharp\leq n}$. We directly conclude $\mathrm{C}_{\sharp=n} \equiv_{sW} \mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp=n} \equiv_{sW} \mathrm{C}_{[0,1],\sharp=n}$ and $\mathrm{C}_{\sharp\leq n} \equiv_{sW} \mathrm{C}_{\{0,1\}^{\mathbb{N}},\sharp\leq n} \equiv_{sW} \mathrm{C}_{[0,1]^k,\sharp\leq n}$.

2 Relative Separation Techniques

The relative separation techniques to be developed in this section do not enable us to prove separation results just on their own; instead they constitute statements that some reduction $f \leq_W g$ implies some reduction $f' \leq_W g'$, so by contraposition $f' \not\leq_W g'$ (which may be easier to prove) implies $f \not\leq_W g$. A particular form of these implications are absorption theorems. These show that for special degrees h, whenever f has a certain property, then $f \leq_W g \star h$ (or $f \leq_W h \star g$) implies $f \leq_W g$. A known result of this form is the following:

Theorem 1 (Brattka, de Brecht & Pauly [1, Theorem 5.1][1]). *Let $f : \mathbf{X} \to \mathbf{Y}$ be single-valued and $\mathbf{Y}$ admissible. Then $f \leq_W \mathrm{C}_{\{0,1\}^{\mathbb{N}}} \star g$ implies $f \leq_W g$.*

We call a Weihrauch-degree a *fractal*, if each of its parts is again the whole. The concept was introduced by Brattka, de Brecht and Pauly in [1] as a criterion for a degree to be join-irreducible (all fractals are join-irreducible, cf. Lemma 1). The formalization uses the operation $f \mapsto f_A$ introduced next.

For some represented space $\mathbf{X} = (X, \delta_X)$ and $A \subseteq \mathbb{N}^{\mathbb{N}}$, we use the notation $\mathbf{X}_A$ for the represented space $(\delta_X[A], (\delta_X)|_A)$. This is a proper generalization of the notion of a subspace. Given $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ and $A \subseteq \mathbb{N}^{\mathbb{N}}$, then f_A is the induced map $f_A :\subseteq \mathbf{X}_A \rightrightarrows \mathbf{Y}$.

Definition 8. *We call f a* fractal *iff there is some $g : \mathbf{U} \rightrightarrows \mathbf{V}$, $\mathbf{U} \neq \emptyset$ such that for any clopen $A \subseteq \mathbb{N}^{\mathbb{N}}$, either $g_A \equiv_W f$ or $g_A \equiv_W 0$. If we can choose $\mathbf{U}$ to be represented by a total representation $\delta_{\mathbf{U}} : \mathbb{N}^{\mathbb{N}} \to \mathbf{U}$, we call f a* closed fractal.

We shall prove two absorption theorems, one for fractals and one for closed fractals. These essentially state that certain Weihrauch degrees are useless in solving a (closed) fractal.

Theorem 2 (Fractal absorption). *For fractal f, $f \leq_W g \star \mathrm{C}_{\{1,\ldots,n\}}$ implies $f \leq_W g$.*

2.1 Baire Category Theorem as Separation Technique

The absorption theorem for closed fractals is a consequence of the Baire Category Theorem, and was first employed as a special case in [3, Proposition 4.9] by Brattka and Gherardi.

[1] The precise statement of [1, Theorem 5.1] is weaker than the one given here, but a small modification of the proof suffices to obtain the present form.

Theorem 3 (Closed fractal absorption). *For a closed fractal f, $f \leq_W g \star C_{\mathbb{N}}$ implies $f \leq_W g$.*

Proof. The closed sets $A_n = \{p \mid n \in \psi_-^{\mathbb{N}}(p)\}$ cover $\mathrm{dom}(\mathrm{C}_{\mathbb{N}} \circ \psi_-^{\mathbb{N}}) \subseteq \mathbb{N}^{\mathbb{N}}$, and the corresponding restrictions $(\mathrm{C}_{\mathbb{N}})_{A_n}$ is computable for each $n \in \mathbb{N}$. Let δ be the representation used on the domain of $g \star \mathrm{C}_{\mathbb{N}}$, and $B_n \subseteq \mathrm{dom}\left((g \star \mathrm{C}_{\mathbb{N}}) \circ \delta\right) \subseteq \mathbb{N}^{\mathbb{N}}$ be the closed set of those names of inputs to $g \star \mathrm{C}_{\mathbb{N}}$ such that the call to $\mathrm{C}_{\mathbb{N}}$ involved is an element of A_n. The sets $(B_n)_{n\in\mathbb{N}}$ cover $\mathrm{dom}\left((g \star \mathrm{C}_{\mathbb{N}}) \circ \delta\right)$, and we find $(g \star \mathrm{C}_{\mathbb{N}})_{B_n} \leq_W g$.

W.l.o.g. assume that f witnesses its own fractality. Now let $f \leq_W g \star \mathrm{C}_{\mathbb{N}}$ be witnessed by computable K, H, and let ρ be the representation on the domain of f. Then the closed sets $H^{-1}(B_n)$ cover $\mathrm{dom}(f \circ \rho) = \mathbb{N}^{\mathbb{N}}$. We can apply the Baire Category Theorem, and find that there exists some n_0 such that $H^{-1}(B_{n_0})$ contains some non-empty clopen ball. As f is a fractal, we know:

$$f \leq_W f_{H^{-1}(B_{n_0})} \leq_W (g \star \mathrm{C}_{\mathbb{N}})_{B_{n_0}} \leq_W g$$

The preceding result occasionally is more useful in a variant adapted directly to choice principles in the rôle of g. For this, we recall the represented space $\mathbb{R}_>$, in which decreasing sequences of rational numbers are used to represent their limits as real numbers. Note that $\mathrm{id} : \mathbb{R} \to \mathbb{R}_>$ is computable but lacks a computable inverse. A generalized measure on some space $\mathbf{X}$ is a continuous function $\mu : \mathcal{A}(\mathbf{X}) \to \mathbb{R}_>$ taking only non-negative values. The two variants are connected by the following result:

Proposition 2. *Define* $\mathrm{LB} : \{x \in \mathbb{R}_> \mid x > 0\} \to \mathbb{N}$ *via* $\mathrm{LB}(x) = \min\{n \in \mathbb{N} \mid n^{-1} \leq x\}$. *Then* $\mathrm{LB} \equiv_{sW} C_{\mathbb{N}}$.

The preceding result indirectly shows how a closed choice principle for some class $\mathfrak{A} \subseteq \mathcal{A}(\mathbf{X})$ of closed sets with positive generalized measure μ can be decomposed into the slices with fixed lower bounds $\mu > n^{-1}$. For this, we recall the infinitary coproduct (i.e., disjoint union) $\coprod_{n\in\mathbb{N}}$ defined both for represented spaces and multivalued functions between them via $\left(\coprod_{n\in\mathbb{N}} f_n\right)(i, x) = (i, f_i(x))$.

Corollary 2. $\mathrm{C}_{\mathbf{X}}|_{\mathfrak{A},\mu>0} \leq_W \left(\coprod_{n\in\mathbb{N}} \mathrm{C}_{\mathbf{X}}|_{\mathfrak{A},\mu>n^{-1}}\right) \star C_{\mathbb{N}}$

Lemma 1 (σ-join irreducibility of fractals [1, Lemma 5.5]). *Let f be a fractal and satisfy $f \leq_W \coprod_{n\in\mathbb{N}} g_n$. Then there is some $n_0 \in \mathbb{N}$ such that $f \leq_W g_{n_0}$.*

Theorem 4. *Let f be a closed fractal such that $f \leq_W C_{\mathbf{X}}|_{\mathfrak{A},\mu>0}$. Then there is some $n \in \mathbb{N}$ such that $f \leq_W C_{\mathbf{X}}|_{\mathfrak{A},\mu>n^{-1}}$.*

Proof. By Corollary 2 we find $f \leq_W \left(\coprod_{n\in\mathbb{N}} \mathrm{C}_{\mathbf{X}}|_{\mathfrak{A},\mu>n^{-1}}\right) \star \mathrm{C}_{\mathbb{N}}$. Then Theorem 3 implies $f \leq_W \left(\coprod_{n\in\mathbb{N}} \mathrm{C}_{\mathbf{X}}|_{\mathfrak{A},\mu>n^{-1}}\right)$. By Lemma 1 there has to be some n_0 with $f \leq_W \mathrm{C}_{\mathbf{X}}|_{\mathfrak{A},\mu>n_0^{-1}}$.

2.2 Large Diameter Technique

Whereas Theorem 4 allows us to bound any positive generalized measure on the closed sets used to compute a function f away from 0, provided f is a closed fractal, the separation technique to be developed next bounds away only a specific generalized measure—the outer diameter—yet needs neither positivity nor the closed fractal property.

For $\varepsilon > 0$ and some class $\mathfrak{A} \subseteq \mathcal{A}(\mathbf{X})$, we introduce:

$$X_\varepsilon(\mathfrak{A}) = \overline{\psi_-^{-1}\left(\{A \in \mathfrak{A} \mid \forall x \in \mathbf{X} \exists B \in \mathfrak{A}\ B \subseteq A \setminus B(x,\varepsilon)\}\right)} \subseteq \mathbb{N}^{\mathbb{N}}$$

This means that the names in $X_\varepsilon(\mathfrak{A})$ are for sets large enough such that arbitrarily late an arbitrary ball of radius ε can be removed from them, and still a closed set in the class $\mathfrak{A}$ remains as a subset.

We proceed to show that a reduction between choice principles has to map sets large in this sense to sets with large outer diameter (denoted by λ).

Lemma 2 (Large Diameter Principle). *Let H and K witness a reduction $C_{\mathbf{X}}|_{\mathfrak{A}} \leq_W C_{\mathbf{Y}}|_{\mathfrak{B}}$, where $\mathbf{Y}$ is compact and $\mathfrak{A} \subseteq \mathcal{A}(\mathbf{X})$, $\mathfrak{B} \subseteq \mathcal{A}(\mathbf{Y})$. Then*

$$\forall p \in \mathrm{dom}(C_{\mathbf{X}}|_{\mathfrak{A}}\psi)\ \forall \varepsilon > 0\ \exists n \in \mathbb{N}\ \exists \delta > 0,\ q \in X_\varepsilon(\mathfrak{A}) \cap B(p, 2^{-n}) \Rightarrow \lambda\psi K(q) > \delta$$

Proof. Assume the claim were false, and let $p \in \mathrm{dom}(\mathrm{C}_{\mathbf{X}}|_{\mathfrak{A}}\psi)$ and $\varepsilon > 0$ be witness for the negation. There has to be a sequence $(p_n)_{n\in\mathbb{N}}$ such that $p_n \in X_\varepsilon(\mathfrak{A})$, $d(p, p_n) < 2^{-n}$ and $\lambda\psi H(p_n) < 2^{-n}$. As the p_n converge to p and H is continuous, we find that $\lim_{n\to\infty} H(p_n) = H(p)$. For the closed sets represented by these sequences, this implies $\left(\bigcap_{n\in\mathbb{N}} \overline{\bigcup_{i\geq n} \psi H(p_i)}\right) \subseteq \psi H(p)$. As $\mathbf{Y}$ is compact, the left hand side contains some point x.

As $x \in \psi H(p)$, for any $q \in \delta_{\mathbf{Y}}^{-1}(\{x\})$ we find $\langle p, q\rangle \in \mathrm{dom}(K)$. We fix such a q and $y = \delta_{\mathbf{X}}(K(\langle p, q\rangle))$. By continuity, there is some $N \in \mathbb{N}$ such that for any $\langle p', q'\rangle \in (B(p, 2^{-N}) \times B(q, 2^{-N})) \cap \mathrm{dom}(\delta_{\mathbf{X}}K)$ we find $\delta_{\mathbf{X}}K(\langle p', q'\rangle) \in B(y, \varepsilon)$.

By choice of x, for any $i \in \mathbb{N}$ there is some $k_i \geq i$ such that $d(x, \psi H(p_{k_i})) < 2^{-i}$. By choice of the p_n, this in turn implies $\psi H(p_{k_i}) \subseteq B(x, 2^{-i} + 2^{-k_i})$. Let $I \in \mathbb{N}$ be large enough, such that for any $x' \in B(x, 2^{-I} + 2^{-k_I})$ we find $\delta_{\mathbf{Y}}^{-1}(x') \cap B(q, 2^{-N}) \neq \emptyset$. The inclusion $\psi H(p_{k_I}) \subseteq B(x, 2^{-I} + 2^{-k_I})$ of a compact set in an open set implies that there is some $L > k_I$ such that for all $p' \in B(p_{k_I}, 2^{-L}) \cap \mathrm{dom}(\mathrm{C}_{\mathbf{X}}|_{\mathfrak{A}}\psi)$ we find $\psi Hp' \subseteq B(x, 2^{-I} + 2^{-k_I})$.

The choice of p_{k_I}, L and the point $y \in \mathbf{X}$ ensures that our reduction may answer any valid input to $\mathrm{C}_{\mathbf{X}}|_{\mathfrak{A}}$ sharing a prefix of length L with p_{k_I} with a name of some point $y' \in B(y, \varepsilon)$. However, as we have $p_{k_I} \in X_\varepsilon(\mathfrak{A})$, we can extend any long prefix of p_{k_I} to a name of a set not intersecting the ball $B(y, \varepsilon)$—this means, our reduction would answer wrong, and we have found the desired contradiction.

Corollary 3 (Large Diameter Principle for fractals). *Let $C_{\mathbf{X}}|_{\mathfrak{A}}$ be a fractal, $\mathbf{Y}$ be compact and $C_{\mathbf{X}}|_{\mathfrak{A}} \leq_W C_{\mathbf{Y}}|_{\mathfrak{B}}$. Then for any $\varepsilon > 0$ there is a $\delta > 0$ such that*

$$(C_{\mathbf{X}})_{X_\varepsilon(\mathfrak{A})} \leq_W C_{\mathbf{Y}}|_{\mathfrak{B},\lambda>\delta}$$

3 Separation Results for Finite and Convex Choice

We now have the tools available to completely characterize the valid reductions between $\mathrm{C}_{\{0,\ldots,n\}}$, XC_m, $\mathrm{C}_{\sharp\leq i}$ and $\mathrm{C}_{\sharp=j}$. Figure 1 provides an overview—the absence of an arrow (up to transitivity) indicates a proof of irreducibility. Besides an application of the general techniques of the preceding section, more specialized proof methods are employed, some with a rather combinatorial character, others based on the properties of simplices. We also exhibit a technique suitable to transfer results from the compact case to the locally compact case.

Observation 5. *$C_{\sharp=n}$ is a fractal. XC_n and $C_{\sharp\leq n}$ are even closed fractals.*

Corollary 4. *$C_{\sharp=n} \not\leq_W C_{\{0,\ldots,m\}}$ for all $n > 1, m \in \mathbb{N}$.*

Proof. Assume the reduction would hold for some $n, m \in \mathbb{N}$. Observation 5 allows us to use Theorem 2 to conclude $\mathrm{C}_{\sharp=n}$ to be computable—a contradiction for $n > 1$.

Proposition 3. *$C_{\sharp=n} \leq_W C_{\mathbb{N}}$*

Corollary 5. *$C_{\sharp\leq 2} \not\leq_W C_{\sharp=n}$*

Proof. Assume $\mathrm{C}_{\sharp\leq 2} \leq_W \mathrm{C}_{\sharp=n}$ for some $n \in \mathbb{N}$. By Proposition 3, this implies $\mathrm{C}_{\sharp\leq 2} \leq_W \mathrm{C}_{\mathbb{N}}$. Observation 5 together with Theorem 3 would show $\mathrm{C}_{\sharp\leq 2}$ to be computable, contradiction.

3.1 Combinatorial Arguments

Proposition 4. *$C_{\{0,\ldots,n\}} <_{sW} C_{\sharp=n+1}$.*

Proposition 5 (Pigeonhole principle). *$C_{\{0,\ldots,n\}} \not\leq_W C_{\sharp\leq n}$*

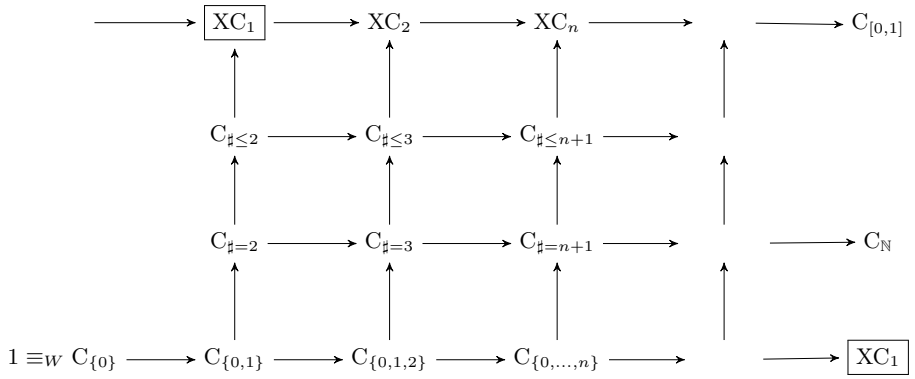


Fig. 1. The reducibilities

Proposition 6. *$C_{\sharp=n+1} \leq_W C^n_{\sharp=2}$ and $C_{\sharp\leq n+1} \leq_W C^n_{\sharp\leq 2}$*

Proof (Sketch). When trying to find a path through an infinite tree with exactly $n+1$ vertices per level, there are at any moment n vertices where both the left and the right successor could potentially lead to an infinite path. The difficulty solely lies in picking a suitable successor at each of these vertices in each stage. It is possible to disentangle these decisions, yielding n trees with just two vertices per level such that any problematic vertex in the original tree is mapped to a problematic vertex in one of the new trees in a way that preserves correct choices of successors.

As a consequence from the independent choice theorem in [1] together with Proposition 1 we obtain the following, showing ultimately that picking an element from a finite number of 2-element sets in parallel is just as hard as picking finitely many times from finite sets, with the later questions depending on the answers given so far:

Observation 6. *$C_{\sharp=n} \times C_{\sharp=m} \leq_W C_{\sharp=n} \star C_{\sharp=m} \leq_W C_{\sharp=(nm)}$ and $C_{\sharp\leq n} \times C_{\sharp\leq m} \leq_W C_{\sharp\leq n} \star C_{\sharp\leq m} \leq_W C_{\sharp\leq(nm)}$*

Corollary 6. $C^*_{\sharp=2} \equiv_W \left(\coprod_{n\in\mathbb{N}} C_{\sharp=n}\right) \equiv_W \left(\coprod_{n,k\in\mathbb{N}} C^{(k)}_{\sharp=n}\right)$

Corollary 7. $C^*_{\sharp\leq 2} \equiv_W \left(\coprod_{n\in\mathbb{N}} C_{\sharp\leq n}\right) \equiv_W \left(\coprod_{n,k\in\mathbb{N}} C^{(k)}_{\sharp\leq n}\right)$

Whether this property (that sequential uses of some closed choice principle are equivalent to parallel uses) also applies to convex choice XC_1 remains open at this stage. As we do have $\mathrm{XC}^k_1 \leq_W \mathrm{XC}_k \leq_W \mathrm{XC}^{(k)}_1$, a positive answer would also imply that increasing the dimension for convex choice means climbing the same hierarchy, again. We point out that the question is related to the open question in [6] whether connected choice in two dimensions is equivalent to connected choice in three dimensions.

Question 1. Is there some $k \in \mathbb{N}$ such that $\mathrm{XC}_1 \star \mathrm{XC}_1 \leq_W \mathrm{XC}^k_1$?

3.2 Simplex Choice

Proposition 7. *Given a closed set $A \subseteq [0,1]$ with $|A| \leq n$, we can compute a closed set $B \subseteq [0,1]^{n-1}$ with $|A| = |B|$, $\pi_1(B) = A$, and such that the points in B are affinely independent.*

Proposition 8. *Given a closed set $A \subseteq [0,1]^n$ with $|A| = n+1$ such that the points in A are affinely independent, we can compute a set $A \cup \{c\}$, where c is a point in the interior of the convex hull of A.*

Proposition 9. *Given a finite closed set $A \subseteq [0,1]^n$, such that the points in A are affinely independent, as well as a point x in the convex hull of A, we can compute a point in A.*

Proof (Sketch). We can obtain positive information about A by noticing that removing some area from A makes x fall out of the convex hull of the remaining set. But then this area must actually contain a point. Iteratively refining this information yields some point in A.

Corollary 8. $C_{\sharp\leq n} \leq_W XC_{n-1}$

Theorem 7. $C_{\sharp=n} <_W C_{\sharp=n+1}$

Proof. By Corollary 1, we can freely change the space we are working in among any rich computably compact computable metric space. We start with an n-point subset of $[0,1]$ and apply Proposition 7 to obtain n affinely independent points. Then we use Proposition 8 to obtain a set of cardinality $n+1$ containing the n previous points and some additional point in the interior of their convex hull. This is a valid input to $\mathrm{C}_{\sharp=n+1}$ (using Corollary 1 again), and we obtain one of the points, which certainly is contained in the convex hull. Hence, Proposition 9 allows us to find one of the vertices, which by Proposition 7 is sufficient to compute one of the points in the original set.

That the reduction is strict follows from Propositions 4 and 5.

Note that while $\mathrm{C}_{\sharp\leq n} \leq_W \mathrm{C}_{\sharp\leq n+1}$ is trivially true, the positive part of the preceding result is not obvious.

3.3 Application of the Large Diameter Technique

The usefulness of the large diameter technique for disproving reducibility to convex choice lies in the observation that convex sets with large outer diameter are simpler, as we can then cut by a hyperplane and obtain another convex set of smaller dimension:

Proposition 10 (Cutting). $XC_{n,\lambda>m^{-1}} \leq_W XC_{n-1} \star C_{\{1,\ldots,(m-1)n\}}$

Proof (Sketch). Because the input set has a large outer diameter, we can find $(m-1)n$ hyperplanes such that at least one of them intersects the input set. Picking a suitable hyperplane is done with $\mathrm{C}_{\{1,\ldots,(m-1)n\}}$, and the intersection then is a convex set of lower dimension, hence a valid input for XC_{n-1}.

Corollary 9. *Let* $C_{\mathbf{X}}|_{\mathfrak{A}}$ *and* $(C_{\mathbf{X}})_{X_\varepsilon(\mathfrak{A})}$ *be fractals and* $C_{\mathbf{X}}|_{\mathfrak{A}} \leq_W XC_{n+1}$. *Then we find* $(C_{\mathbf{X}})_{X_\varepsilon(\mathfrak{A})} \leq_W XC_n$.

Proof. Corollary 3 gives us $(\mathrm{C}_{\mathbf{X}})_{X_\varepsilon(\mathfrak{A})} \leq_W \mathrm{XC}_{n+1,\lambda>m^{-1}}$ for some $m \in \mathbb{N}$, then Proposition 10 implies $(\mathrm{C}_{\mathbf{X}})_{X_\varepsilon(\mathfrak{A})} \leq_W \mathrm{XC}_n \star \mathrm{C}_{\{1,\ldots,(m-1)(n+1)\}}$ and finally Theorem 2 fills the gap to $(\mathrm{C}_{\mathbf{X}})_{X_\varepsilon(\mathfrak{A})} \leq_W \mathrm{XC}_n$.

For $n \geq k \geq 1$ let $\mathrm{C}_{\sharp=n\rhd k} := \mathrm{C}_{[0,1]}\big|_{\{A\in\mathcal{A}([0,1])\,||A|=n\wedge|\{i<n\,|\,[\frac{2i}{2n},\frac{2i+1}{2n}]\cap A\neq\emptyset\}|\geq k\}}$. So $\mathrm{C}_{\sharp=n\rhd k}$ is choice for n element sets, where we know that our set intersects at least k of a collection of fixed distinct regions. We shall need three properties of these choice principles:

Proposition 11. *1.* $C_{\sharp=n\rhd(k+1)} \leq_W (C_{[0,1]})_{X_{(5n)^{-1}}(\mathrm{dom}(C_{\sharp=n\rhd k}))}$.
2. $C_{\sharp=n+1\rhd n}$ *is not computable.*
3. Any $C_{\sharp=n\rhd k}$ *is a fractal.*

Corollary 10. $C_{\sharp=n\rhd k} \leq_W XC_{m+1}$ *implies* $C_{\sharp=n\rhd(k+1)} \leq_W XC_m$.

Theorem 8. $C_{\sharp n+2} \not\leq_W XC_n$.

Proof. Assume $\mathrm{C}_{\sharp=n+2} = \mathrm{C}_{\sharp=(n+2)\rhd 1} \leq_W \mathrm{XC}_n$. Iterated use of Corollary 10 allows us to conclude that $\mathrm{C}_{\sharp=(n+2)\rhd(n+1)}$ is computable, which contradicts Proposition 11 (2).

Acknowledgements. We would like to thank Vladik Kreinovich for a question asked at CCA 2011 that spawned our investigation of the choice principles $\mathrm{C}_{\sharp\leq n}$ and $\mathrm{C}_{\sharp=n}$. We are indebted to Vasco Brattka for the formation of this line of research, and for providing the one dimensional cases of Propositions 8 and 9.

References

1. Brattka, V., de Brecht, M., Pauly, A.: Closed choice and a uniform low basis theorem. Annals of Pure and Applied Logic 163(8), 968–1008 (2012)
2. Brattka, V., Gherardi, G.: Borel complexity of topological operations on computable metric spaces. Journal of Logic and Computation 19(1), 45–76 (2009)
3. Brattka, V., Gherardi, G.: Effective choice and boundedness principles in computable analysis. Bulletin of Symbolic Logic 17(1), 73–117 (2011), arXiv:0905.4685
4. Brattka, V., Gherardi, G.: Weihrauch degrees, omniscience principles and weak computability. Journal of Symbolic Logic 76, 143–176 (2011), arXiv:0905.4679
5. Brattka, V., Gherardi, G., Marcone, A.: The Bolzano-Weierstrass Theorem is the jump of Weak König's Lemma. Annals of Pure and Applied Logic 163(6), 623–625 (2012), also arXiv:1101.0792
6. Brattka, V., Le Roux, S., Pauly, A.: On the computational content of the brouwer fixed point theorem. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 56–67. Springer, Heidelberg (2012)
7. Brattka, V., Pauly, A.: Computation with advice. Electronic Proceedings in Theoretical Computer Science 24 (2010), `http://arxiv.org/html/1006.0551` cCA 2010
8. Gasarch, W., Martin, G.: Bounded Queries in Recursion Theory. In: Progress in Computer Science & Applied Logic, vol. 16. Birkhäuser (1999)
9. Gherardi, G., Marcone, A.: How incomputable is the separable Hahn-Banach theorem? Notre Dame Journal of Formal Logic 50(4), 393–425 (2009)
10. Hertling, P.: Unstetigkeitsgrade von Funktionen in der effektiven Analysis. Ph.D. thesis, Fernuniversität, Gesamthochschule in Hagen (Oktober 1996)
11. Higuchi, K., Pauly, A.: The degree-structure of Weihrauch-reducibility. Logical Methods in Computer Science 9(2) (2013)
12. Kihara, T.: Incomputability of simply connected planar continua. Computability (2), 131–152 (2012)
13. Le Roux, S., Pauly, A.: Closed choice: Cardinality vs convex dimension. arXiv (2013)

14. Le Roux, S., Ziegler, M.: Singular coverings and non-uniform notions of closed set computability. Mathematical Logic Quarterly 54(5), 545–560 (2008)
15. Longpré, L., Kreinovich, V., Gasarch, W., Walster, W.: m solutions good, $m-1$ solutions better. Applied Mathematical Sciences 2, 223–239 (2008)
16. Mylatz, U.: Vergleich unstetiger Funktionen in der Analysis. Master's thesis, Fachbereich Informatik, FernUniversität Hagen (May 1992)
17. Mylatz, U.: Vergleich unstetiger Funktionen : "Principle of Omniscience" und Vollständigkeit in der C-Hierarchie. Ph.D. thesis, Fernuniversität, Gesamthochschule in Hagen (May 2006)
18. Pauly, A.: How incomputable is finding Nash equilibria? Journal of Universal Computer Science 16(18), 2686–2710 (2010)
19. Pauly, A.: On the (semi)lattices induced by continuous reducibilities. Mathematical Logic Quarterly 56(5), 488–502 (2010)
20. Pauly, A., de Brecht, M.: Non-deterministic computation and the Jayne Rogers theorem. Electronic Proceedings in Theoretical Computer Science, DCM 2012 (2012) (to appear)
21. Stein, T.v.: Vergleich nicht konstruktiv lösbarer Probleme in der Analysis. Master's thesis, Fachbereich Informatik, FernUniversität Hagen (1989)
22. Weihrauch, K.: The degrees of discontinuity of some translators between representations of the real numbers. Informatik Berichte 129, FernUniversität Hagen, Hagen (July 1992)
23. Weihrauch, K.: The TTE-interpretation of three hierarchies of omniscience principles. Informatik Berichte 130, FernUniversität Hagen, Hagen (September 1992)
24. Weihrauch, K.: Computable Analysis. Springer (2000)
25. Ziegler, M.: Real hypercomputation and continuity. Theory of Computing Systems 41, 177–206 (2007)

Realizability Models Separating Various Fan Theorems

Robert S. Lubarsky[1,⋆] and Michael Rathjen[2]

[1] Department of Mathematical Sciences, Florida Atlantic University, Boca Raton, FL 33431, U.S.A.
Robert.Lubarsky@alum.mit.edu
[2] Department of Pure Mathematics, University of Leeds, Leeds LS2 9JT, England
rathjen@maths.leeds.ac.uk

Abstract. We develop a realizability model in which the realizers are the reals not just Turing computable in a fixed real but rather the reals in a countable ideal of Turing degrees. This is then applied to prove several separation results involving variants of the Fan Theorem.

1 Introduction

Certain constructions in computability theory lend themselves well to realizability, the latter being based on an abstract notion of computation. A coarse example of this is the notion of a Turing computable function itself, as the collection of Turing machines makes an applicative structure and so provides an example of realizability. This model is closely tied to Turing computation, naturally enough, and so provides finer examples. Consider Weak König's Lemma, WKL, which is among constructivists more commonly studied in its contrapositive form, the Fan Theorem FAN. (For background on realizability, the Fan Theorem, and constructive mathematics in general, there are any of a number of standard texts, such as [3,23,25].) Kleene's well-known eponymous tree is a computable, infinite tree of binary sequences with no computable path. In the context of reverse mathematics, this shows that RCA_0 does not prove WKL. Within the realizability model, the same example shows IZF does not prove FAN.

Perhaps a word should be said on the choice of the ground theory. For the classical theory, it's RCA_0, while for the constructive, it's IZF. The former is notably weak, the latter strong. Why is that? And why those theories in particular? This is not the place to discuss the particular choice of RCA_0. As for IZF, its use is of secondary importance. The point is that much of reverse classical mathematics is to show the equivalence of various principles, for which a weaker base theory provides a stronger theorem. Even for independence results, which on the surface would be better over a stronger base, are often of the form that a weaker theory does not imply a stronger one, where the latter easily implies

⋆ The first author would like to thank Wouter Stekelenburg, Thomas Streicher, and Jaap van Oosten for useful discussion during the development of this work.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 306–315, 2013.

the former; clearly here, the base theory is the weaker of the two. Also for those cases where the independence result desired is an incomparability, the principles in question are all weak set existence principles, weak in the sense that they use a tiny fragment of ZF, and hence a weak base theory is needed, to keep the theories in question from being outright provable. In contrast, reverse constructive mathematics studies not set existence principles, but rather logical principles. Instead of fragments of ZF, the subject is fragments of Excluded Middle. Especially when discussing independence, when a stronger base theory gives a stronger theorem, in order to highlight that it really is the logic that is up for grabs, and not set existence, strong principles of set theory are taken as the base theory. IZF is used here, since it is the simplest and most common constructive correlate to ZF, the classical standard. Even for equivalence theorems, there would still be a tendency to work over IZF, since the degree to which a result depended on the IZF axioms is the degree to which the result is ultimately classical. In practice, if any theorem needs less than full IZF, what is actually used could be read off from the proof anyway.

Returning to realizability, the picture is not quite so rosy when it comes to other constructions. A case in point is the distinction between WKL and WWKL, Weak Weak König's Lemma. WWKL states that for every binary tree (of finite, 0-1 sequences) with no path there is a natural number n such that at least half the sequences of length n are not in the tree. This principle has been studied in reverse mathematics, both classical [22] and constructive [20]. Yu and Simpson [26] showed that WWKL does not imply WKL (over RCA_0). That's not so simple as merely taking the computable sets; while that would falsify WKL, as discussed above, it invalidates WWKL too. What they do is to extend the computable sets by a carefully selected real R_0 (implicitly closing under Turing reducibility) which provides a path through all the "bad" trees while not destroying the Kleene tree counter-example. That's not enough, though, because the construction of a counter-example to WWKL relativizes. So while R_0 kills off the bad computable trees, it introduces new bad trees of its own. Hence the construction must be iterated: $R_1, R_2, ...$ In the end, the union of the (reals computable in the) R_n's suffices.

The statement of WWKL carries over just fine to a constructive setting, where we shall call it the Weak Fan Theorem W-FAN, as well as the question of whether W-FAN implies FAN. For better or worse, the construction though doesn't. One might first think to use the Yu-Simpson set of reals as the set of realizers. An immediate problem is that we need an applicative structure: the realizers need to act on themselves. It is immediate and routine to view these reals as functions from the naturals to the naturals—that's a trivial identification these days. It doesn't help though. If the realizers are those functions, well, those functions act on naturals, not on functions. What we would need would be integer codes for those functions. The realizers from this point of view would be that set of naturals, which as need be could be taken to be functions. But here's the problem: what code do you give a function which showed up in some increasing tower? If you were looking at those functions computable in some fixed oracle,

you could consider all the naturals as each coding such a machine. If instead your oracle is continually changing, it's not clear what to do and still maintain an applicative structure. The fixes we tried did not work, as we were warned.

The same issue comes up with another distinction around FAN, namely the distinction between FAN and WKL. In order to show their inequivalence, one might want to come up with a model of FAN falsifying WKL. This has already been done using K_2 realizability, Kleene's notion of functional realizability. It is another matter to prove this theorem via K_1 realizability. The problem is as above: every Turing degree has an infinite binary tree with no path of the same or lesser degree (the Kleene tree relativized to that degree). One could take a set of degrees, any of the trees of which have a path in some other degree. The problem here is how to turn that into a realizability structure.

The goal of this paper is a realizability model in which the realizers code functions from the natural numbers to themselves with no highest Turing degree among them. As corollaries to this method we get the two results just cited.

2 The Main Construction

The main idea here is to build a Kripke model, and then within that a realizability model, which has sometimes been called relative realizability [25]. This kind of construction was apparently first suggested by de Jongh [10]. Variants have been used by several people: having the Kripke partial order consist of only two points and the realizers be at $\perp$ certain computable objects which are then injected into a full set of realizers [1,2], or using instead of Kripke semantics either double-negation [14] or a kind of Beth [24] semantics. For more detail on all of this, see the last two sections of [25].

To help keep things simple, we assume ZFC in the meta-theory. For most of this work, neither classical logic nor the Axiom of Choice is necessary, but we shall not be careful about this.

Let the underlying Kripke partial order be $\omega^{<\omega}$. Let M be the full Kripke model built on that p.o. Intuitively, that means throw in all possible sets. More formally, a set in the model is a function f from $\omega^{<\omega}$ to the sets of the model (inductively) which is non-decreasing (i.e., if $\sigma \subseteq \tau$ then $f(\sigma) \subseteq f(\tau)$). Equality and membership are defined by a mutual induction. On general principles, $M \models$ IZF. Moreover, the ground model V has a canonical image in M: given $x \in V$, let $\check{x}$ be such that $\check{x}(\sigma) = \{\check{y} \mid y \in x\}$. We often identify x with $\check{x}$, the context hopefully making clear whether we're in V or in M. For slightly more detail on the full model, defined over any partial order, see for instance [17].

Within M, we shall identify a special set R of natural numbers, based on a prior sequence R_n of reals ($n \in \omega$). We assume the R_n are of strictly increasing Turing degree. At node $\perp = \langle\rangle$, R looks empty: $\perp \not\models s \in R$; equivalently, $R(\langle\rangle) = \varnothing$. Suppose inductively $R(\sigma)$ is defined, where σ has length n. Let R_n^i list all the reals that differ from R_n in finitely many places. Let $R(\sigma^\frown i)$ be $R(\sigma) \cup \{\langle n, s\rangle \mid s \in R_n^i\}$. In words, at level $n+1$, beneath each node on level n, put into the nth slice of R all of the finite variations of R_n, spread out among all the successors. So R is a kind of join of the R_n's, just not all at once.

Because R is (in M) a real, it makes sense to use R as an oracle for Turing computation. At $\perp$, if a computation makes any query $s \in R$ of R, there are some nodes at which s is in R and others where it is not, so the oracle cannot answer and the computation cannot continue. This follows from the formal model of oracle computability: a run of an oracle machine is a tuple of natural numbers coding a correct computation; the rule for extending a tuple when the last entry is an oracle call is that the next entry must contain the right answer; if there is, at a node, no right answer, then there can be no extending tuple. Hence the only convergent computations are those that make no oracle calls, and the only R-computable functions are the computable ones. More generally, at node σ of length n, any query of the form $\langle k, s\rangle \in R$ with $k \geq n$ will be true at some future nodes and false at others, hence unanswerable at σ. The computable functions at σ are those computable in R_{n-1}.

In M, let App be the applicative structure of the indices of functions computable in R (using, of course, the standard way of turning such indices into an applicative structure). In M, let $M[\mathrm{App}]$ be the induced realizability model. On general principles, $M[\mathrm{App}] \models \mathrm{IZF}$. The natural numbers of $M[\mathrm{App}]$ can be identified with those of M, so any set of such in $M[\mathrm{App}]$ can be identified with one in M. Furthermore, at any node, a decidable real in one structure corresponds to a decidable real in the other, and that can be identified with a real in the ambient classical universe. Henceforth these various reals will not even be distinguished notationally. For instance, if $\sigma \models$ "$M[\mathrm{App}] \models$ "$T \subseteq \omega$ is decidable" " then we might refer to the real T in V.

For notational convenience, we shall abbreviate $\sigma \models$ "$M[\mathrm{App}] \models \phi$" as $\sigma \models_{\mathrm{App}} \phi$.

Lemma 1. *For σ of length n and R a real, $\sigma \models_{\mathrm{App}}$ "X is decidable" iff X is Turing computable in R_{n-1}.*

Proof. The statement "X is decidable" is $\forall m \in \omega\ m \in X \vee m \notin X$. A realizer r of the latter would be a function that, on input m, decides whether m is in or out of X. If in the course of its computation r asked the oracle any question of the form $\langle k, s\rangle$ with $k \geq n$ then the computation would not terminate at σ. So r can access only R_{n-1}, making X computable in R_n. The converse is immediate.

Lemma 2. *If $\sigma \models_{\mathrm{App}}$ "X is an infinite branch through the binary tree" then $\sigma \models_{\mathrm{App}}$ "X is decidable."*

Proof. To be an infinite branch means for every natural number m there is a unique node of length m. The realizer that X is an infinite branch has to produce that node given m.

Often people are concerned about the use of various choice principles. The independence results presented here are that much stronger because Dependent Choice holds in our models.

Proposition 1. $\langle\rangle \models_{\mathrm{App}} DC$

Proof. The same proof that DC holds in standard Kleene K_1 realizability works here.

3 D-FAN and W-D-FAN

When adapting the classical results to the current setting, we need an additional stipulation. All of the trees, and hence principles, we consider will be decidable: for all binary sequences b and trees T, either $b \in T$ or $b \notin T$. So, for instance, instead of the full Fan Theorem FAN, we shall be considering the Decidable Fan Theorem: if a decidable tree in $\{0,1\}^{\mathbb{N}}$ has no infinite path, then the tree is finite. Also, Weak FAN, also known as WWKL, when applied to decidable trees, would read: if a decidable tree in $\{0,1\}^{\mathbb{N}}$ has no infinite path, then there is an n such that at least half of $\{0,1\}^n$ is not in the tree.

This brings us to an annoying point about notation. Decidable FAN has been referred to in various places as D-FAN, FAN_D, Δ-FAN, and FAN_Δ. So notation for Weak Decidable FAN could be any of those, with a "W" stuck in somewhere. To make matters worse, even though the statement of Weak FAN is a weakening of FAN and not of WKL, the name WWKL for it is already established in the classical literature, and so one could make a case to stick with it, and insert decidability ("D" or "Δ") somewhere in there. These same considerations apply to other variants of FAN, whether already identified (c-FAN, Π^0_1-FAN) or not. Whatever we do here will likely not settle the matter. Still, we have to choose something. It strikes us as confusing to distinguish between FAN and WKL, and then call a variant of FAN by a variant of WKL. Also, what if somebody someday wants to study the contrapositive of "WWKL"? Hence we stick with the name W-FAN. As for how to get in the decidability part, we choose the option that's the easiest to type: W-D-FAN.

Returning to the matter at hand, Yu-Simpson [26] construct a sequence X^n of reals of increasing Turing degree such that:

i) if T is a tree computable in X^n the branches through which form a set of positive measure, then a path through T is computable in X^{n+1}, and

ii) no path through the Kleene tree is computable in any X^n.

We apply the construction from the previous section, with R_n set to X^n. From this, the following lemmas are pretty much immediate.

Lemma 3. $\langle\rangle \models_{\text{App}}$ *W-D-FAN*

Proof. At any node, a decidable tree T is computable in some X^n. If in V the measure of the branches through T were positive, then there would be a branch computable through X^{n+1}. So no node could force that there are no branches through T. To compute a level at which half the nodes are not in T, just go through T level by level until this is found.

Lemma 4. $\langle\rangle \models_{\text{App}} \neg$*D-FAN*

Proof. The Kleene tree provides a counter-example.

While we expect that even full W-FAN does not imply D-FAN, this model does not satisfy W-FAN. To see that, recall that any path through the binary tree is decidable, hence computable in some X^n. There are only countably many such

paths. It is easy in V to construct a tree avoiding those countably many paths with measure (of the paths) being as close to 1 as you'd like. The internalization of such a tree in $M[\mathrm{App}]$ will not be decidable, but will be internally a tree with no paths.

4 FAN and WKL

The distinction between FAN and WKL is a strange case. Their relation is that WKL implies FAN, but not the converse. With some exaggeration, it seems as though everyone knows that but no one has proven it.[1] At the very introduction of non-classical logic, Brouwer himself must have realized this distinction, as he made a conscious choice which variant of this class of principles he accepted. Moreover, while he accepted FAN (having proven it from Bar Induction), it is easy to see that WKL implies LLPO, which Brouwer rejected. So while Brouwer did not provide what we would today consider a model of FAN + $\neg$WKL, we would like to honor him in the style of the Pythagoreans by attributing this result to him, whatever may actually have been going through his mind.

Such models have since been provided, for instance by Kleene, using his functional realizability K_2 [16,21]. However, in neither of those sources is it mentioned that WKL fails. In [5], both FAN and WKL are analyzed into constituent principles, it is shown that WKL's components imply the corresponding FAN components, and it is nowhere stated that the converse does not hold. In [7], equivalents are given for what is there called FAN and WKL, although their FAN is actually D-FAN, and it is at least asked how much stronger WKL is than FAN. The one proof we have been able to find of some fan theorem not implying WKL is in [19], where once again the fan principle used is D-FAN. For what it's worth, that argument, like ours, uses relative realizability [8], albeit with K_2 realizability.

Below we give a full proof that FAN does not imply WKL. We would be interested in hearing of other extant proofs of such, and would find it amusing if there were none. What might be new here, if anything, is not the result itself, but rather the methodological point that this model is based on K_1. That is, while K_1 is usually used to falsify even D-FAN, its variant below validates full FAN.

Theorem 1. *(Brouwer) FAN does not imply WKL.*

Proof. Take a countable ω-standard model of WKL_0 [22, Chapter VIII]. There is a sequence X^n of reals of increasing Turing degree such that the reals in this model are exactly those computable in some X^n. This induces a model $M[\mathrm{App}]$ as in section 2 above (with R_n being set to X^n).

To see that WKL fails, suppose to the contrary $\sigma \models_{\mathrm{App}}$ "$n \Vdash_r$ WKL". So if $\sigma \models_{\mathrm{App}}$ "$e \Vdash_r T$ is an infinite binary tree" then, at σ, $\{n\}(e)$ must compute a

[1] Thanks are due here to Hannes Diener for first pointing this out to us and Thomas Streicher for useful discussion.

path through T. But a path through a tree is not computable in the tree, as is standard, by considering the Kleene tree. This shows moreover that WKL for decidable trees fails.

To show FAN, suppose at some node σ in the Kripke partial order r realizes that B is a bar. We must show how to compute a uniform bound on B. To do this, we shall build a decidable subset C of B. Inductively at stage n, suppose we have decided C on all binary sequences of length less than n. Consider each binary sequence $\bar{b}$ of length n (except if $\bar{b}$ extends something in C of shorter length, in which case what happens with $\bar{b}$ just doesn't matter anymore). Consider the path $P_{\bar{b}}$ which passes through $\bar{b}$ and is always 0 after that. Applying r to $P_{\bar{b}}$ produces a sequence b^+ in B on $P_{\bar{b}}$. If b^+ has length at most n, include in C all extensions of b^+ of length n, else just include b^+ in C. After doing this for all $\bar{b}$ of length n, anything of length n not put into C is out of C. This procedure terminates only when we have a uniform bound on C, hence on B. If this never terminates, we have a decidable, infinite set of binary sequences not in C computable from r. Hence at any child τ of σ there will be an infinite path P avoiding C. Applying r to P produces an initial segment of P in B, say b. This computation itself used only an initial segment of P, say c. Letting n be the maximum length of b and c, at stage n no initial segment of either has been put into C, by the choice of P. So the procedure would consider the path through b and c which is all 0s afterwards. This would then put the longer of b and c into C, contradicting the choice of P. So this procedure must terminate, producing a bound for B.

5 Questions

5.1

The second construction was developed for an entirely different purpose.

One way of stating FAN is that every bar is uniform. Weaker versions of FAN can be developed by restricting the bars to which the assertion applies. For instance, Decidable FAN, D-FAN, states that for every decidable set B (i.e., for all b either $b \in B$ or $b \notin B$), if B is a bar then B is uniform. Constructively, decidability is a very strong property; in fact, it is the strongest hypothesis on a bar yet to be identified. D-FAN is trivially implied by FAN; it has long been known that D-FAN is not provable in IZF (via the Kleene tree, described in any standard reference, such as [3,23]; see [18] for a different proof). A somewhat milder restriction on a bar B is that it be the intersection of countably many decidable sets; that is, B is Π^0_1 definable. Between decidable and Π^0_1 bars are c-bars: if there is a decidable set B$'$ such that $b \in B$ iff for every c extending $b\ c \in B'$, then B is called a c-set, and if it's a bar to boot then it's called a c-bar. Often this definition seems at first unnatural and rather technical. All that matters at the moment is that this is a weaker condition than decidability: every decidable bar is a c-bar. c-FAN is the assertion that every c-bar is uniform. Such principles occur naturally in reverse constructive mathematics [15,4,12], and are all inequivalent [18].

The first proof that D-FAN does not imply c-FAN, by Josef Berger [6], went as follows. Classically, for X any collection of bars, X-FAN and X-WKL are equivalent (as contrapositives). Furthermore, the Turing jump of a real R can be coded into a tree computable in R, so that c-WKL implies that jumps always exist. Hence over RCA_0 c-WKL implies ACA. D-FAN and WKL (which, in the setting of limited comprehension, is just D-WKL) are equivalent. So if D-FAN implied c-FAN, constructively or classically, then WKL would imply ACA, which is known not to be the case [22].

A limitation of this argument is that it works over a very weak base theory. It leaves open the question of whether D-FAN implies c-FAN over IZF. While this has been settled [18], a question of method still remains open. Could Berger's argument be re-cast to provide an independence results over IZF? The obvious place to look seemed to be a model of WKL + ¬ACA, using the functions there, which necessarily have no largest Turing degree, as realizers. Our analysis of such a model did not achieve that goal. Is there another way of turning such a model into a separation of D- and c-FAN?

More generally, could there be any realizability model separating D- and c-FAN? All of the realizability models we know about either falsify D-FAN or satisfy full FAN. Perhaps that's because of the difficulty of realizing that something is a bar. That is, to falsify any version of the FAN, one might well want to provide a counter-example, which would be a non-uniform bar. If a bar is not uniform, realizing the non-uniformity would typically be trivial, as nothing could realize that it is uniform, which suffices. Realizing that a set is a bar is different: given a binary path, you'd have to compute a place on that path and realize that that location is in the alleged bar. If this set is decidable, that's easy: continue along the path, checking each node on the way, until you're in it. If the set cannot be assumed decidable, it is unclear to us how to realize that it's a bar. This is something we would like to see: a way of realizing a non-decidable set being a bar.

5.2

The differences among D-FAN, c-FAN, Π^0_1-FAN, and FAN have to do with the hypothesis; they apply to different kinds of bars. In contrast, the difference between FAN and W-FAN has to do with the conclusion, with whether the bar is uniform or half-uniform, to coin a phrase. So these variants can be mixed and matched. There are D-FAN, c-FAN, Π^0_1-FAN, FAN, and also W-D-FAN, W-c-FAN, W-Π^0_1-FAN, and W-FAN. Clearly any version of FAN implies its weak cousin. Other than that, we conjecture there is complete independence between the variants of FAN and the variants of Weak FAN. This is, we conjecture W-FAN does not imply D-FAN, and conjoined with D-FAN does not imply c-FAN, and so on. Furthermore, we expect that D-FAN, while of course implying W-D-FAN, does not imply W-c-FAN, and c-FAN does not imply W-FAN, and so on for other variants that might appear.

References

1. Awodey, S., Birkedal, L.: Elementary axioms for local maps of toposes. Journal of Pure and Applied Algebra 177(3), 215–230 (2003)
2. Bauer, A.: The Realizability approach to computable analysis and topology. Ph.D. Thesis (2000)
3. Beeson, M.: Foundations of Constructive Mathematics. Springer (1985)
4. Berger, J.: The Logical strength of the uniform continuity theorem. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 35–39. Springer, Heidelberg (2006)
5. Berger, J.: A decomposition of Brouwers fan theorem. Journal of Logic and Analysis 1(6), 1–8 (2009)
6. Berger, J.: A separation result for varieties of Brouwer's fan theorem. In: Proceedings of the 10th Asian Logic Conference (alc 10), September 1-6, Kobe University in Kobe, hyogo (2008); Arai, et al (eds.) World Scientific, pp.85–92 (2010)
7. Berger, J., Ishihara, H.: Brouwers fan theorem and unique existence in constructive analysis. Mathematical Logic Quarterly 51(4), 360–364 (2005), doi:10.1002/malq.200410038
8. Birkedal, L., van Oosten, J.: Relative and modified relative realizability. Annals of Pure and Applied Logic 118(1-2), 115–132 (2002)
9. Bishop, E., Bridges, D.: Constructive Analysis. Springer (1985)
10. de Jongh, D.H.J.: The Maximality of the intuitionistic predicate calculus with respect to Heyting's Arithmetic. Typed manuscript (1969)
11. Diener, H.: Compactness under constructive scrutiny. Ph.D. Thesis (2008)
12. Diener, H., Loeb, I.: Sequences of real functions on [0, 1] in constructive reverse mathematics. Annals of Pure and Applied Logic 157(1), 50–61 (2009)
13. Fourman, M., Hyland, J.M.E.: Sheaf models for analysis. In: Fourman, M., Mulvey, C., Scott, D. (eds.) Applications of Sheaves. Lecture Notes in Mathematics, vol. 753, pp. 280–301. Springer, Heidelberg (1979), http://dx.doi.org/10.1007/BFb0061823
14. Goodman, N.D.: Relativized realizability in intuitionistic arithmetic of all finite types. Journal of Symbolic Logic 43, 23–44 (1978)
15. Julian, W., Richman, F.: A Uniformly continuous function on [0,1] that is everywhere different from its infimum. Pacific Journal of Mathematics 111(2), 333–340 (1984)
16. Kleene, S.C., Vesley, R.E.: The Foundations of Intuitionistic Mathematics. North-Holland (1965)
17. Lubarsky, R.: Independence results around Constructive ZF. Annals of Pure and Applied Logic 132(2-3), 209–225 (2005)
18. Lubarsky, R.S., Diener, H.: Separating the Fan Theorem and Its Weakenings. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 280–295. Springer, Heidelberg (2013)
19. Moschovakis, J.: Another Unique Weak König's Lemma. In: Berger, U., Diener, H., Schuster, P., Seisenberger, M. (eds.) Logic, Construction, Computation, Ontos (to appear, 2013)
20. Nemoto, T.: Weak weak König's Lemma in constructive reverse mathematics. In: Proceedings of the 10th Asian Logic Conference (alc 10), September 1-6, Kobe University in Kobe, hyogo (2008) Arai, et al (eds) World Scientific, pp. 263–270 (2010)

21. Rathjen, M.: Constructive Set Theory and Brouwerian Principles. Journal of Universal Computer Science 11, 2008–2033 (2005)
22. Simpson, S.: Subsystems of Second Order Arithmetic. ASL/Cambridge University Press (2009)
23. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics, vol. 1. North-Holland (1988)
24. van Oosten, J.: A Semantical proof of De Jongh's theorem. Archive for Mathematical Logic 31, 105–114 (1991)
25. van Oosten, J.: Realizability: An Introduction to its Categorical Side. Elsevier (2008)
26. Yu, X., Simpson, S.: Measure theory and weak Königs lemma. Archive for Mathematical Logic 30, 171–180 (1990)

Towards a Theory of Homomorphic Compression

Andrew McGregor*

Department of Computer Science, University of Massachusetts Amherst,
Amherst, MA 01002, U.S.A.
mcgregor@cs.umass.edu

Abstract. In this talk we survey recent progress on designing homomorphic fingerprints. Fingerprinting is a classic randomized technique for efficiently verifying whether two files are equal. We shall discuss two extensions of this basic functionality: a) verifying whether two text files are cyclic shifts of one another and b) when the files correspond to "address books", verifying whether the resulting social network is connected. Underlying both results is the idea of homomorphic lossy compression, i.e., lossy data compression that supports a range of operations on the compressed data that correspond directly to operations on the original data.

1 Introduction

Fingerprinting is a classic technique for verifying whether two large data sets are equal. Examples include the "rolling" fingerprint of Karp and Rabin [6] and cryptographic hash functions such as MD5 [7]. More generally, *linear sketching* is a form of lossy compression that also enables the "dissimilarity" of non-identical data sets to be estimated. If we represent the data as a vector $\mathbf{x} \in \mathbb{R}^n$, then a linear sketch is just a random projection $A\mathbf{x} \in \mathbb{R}^k$ where $k \ll n$ and we choose the random matrix $A \in \mathbb{R}^{k \times n}$ in such a way that the dissimilarity between files $\mathbf{x}$ and $\mathbf{y}$ can be estimated from $A\mathbf{x} - A\mathbf{y} = A(\mathbf{x} - \mathbf{y})$.

Linear sketches are trivially homomorphic with respect to linear operations in the sense that given sketches $A\mathbf{x}, A\mathbf{y}$, and $A\mathbf{z}$ we can also estimate the dissimilarity between $\mathbf{x}$ and $\mathbf{w} = \alpha\mathbf{y} + \beta\mathbf{z}$ for some arbitrary $\alpha, \beta \in \mathbb{R}$ since $A\mathbf{x} - \alpha A\mathbf{y} - \beta A\mathbf{z} = A(\mathbf{x} - \mathbf{w})$. In this talk, we shall survey recent work [1–3] on designing fingerprints and sketches that are also homomorphic with respect to other operations. These results have applications in data stream computation and communication complexity.

2 Homomorphic Fingerprints

2.1 Text Misalignment

Many sketches have been proposed for dissimilarity measures that decompose coordinate-wise such as the Hamming distance between alphanumeric strings,

* Supported by NSF CAREER Award CCF-0953754.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 316–319, 2013.

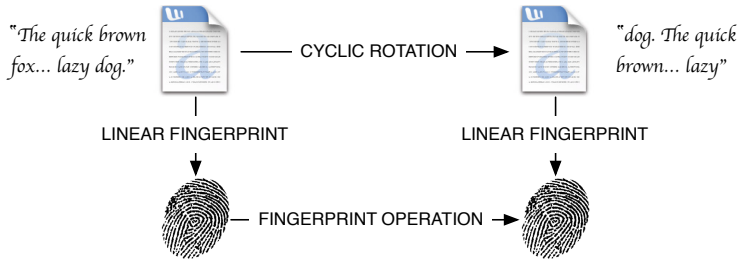


Fig. 1. Fingerprinting is an extreme form of compression that allows the Hamming distance between two files to be estimated given only the compressed form (the "fingerprint") of each file. However, traditional fingerprinting techniques perform poorly when edits result in misaligned characters. However, recent work shows that $n^{O(1/\log\log n)}$ bit fingerprints exist that are homomorphic with respect to both linear and rotation operations, i.e., given only the fingerprint of a file (and not the file itself), we can construct the fingerprint of any cyclic rotation of the file (i.e., the above diagram commutes). Such fingerprints enable us to test whether two files are within a small Hamming distance of being cyclic shifts of one another.

or the Euclidean distance between vectors. However, when editing textual data, the appropriate notions of dissimilarity do not decompose coordinate-wise. For example, adding a single character to the start of a file and deleting a character from the end of the file will result in a new file whose Hamming distance to the original file may be proportional to the length of the file. Hence we need fingerprints that are robust to misalignments.

In recent work [3], we designed a linear sketch $L : \mathbb{Z}_m^n \to \{0,1\}^s$ that randomly projected length-n files into $s = D(n) \cdot \operatorname{polylog} n$ dimensions where $D(n)$ is the number of divisors of n. The sketch has the following properties:

1. *Soundness:* Given $L(a)$ and $L(b)$ we can determine whether the file b is a cyclic shift of a (or more generally within a constant Hamming distance of a file that is a cyclic shift of a) with probability at least $2/3$.
2. *Shift homomorphism:* Given $L(a)$ and cyclic shift σ we can compute $L(\sigma(a))$;
3. *Linear homomorphism:* Given $L(a)$ and $L(b)$ we can compute $L(a+b)$.

Furthermore, we showed that the dependence on $D(n)$ was optimal. This is somewhat surprising in the sense that we wouldn't expect a problem that is ostensibly about lossy compression to be so sensitive to a number-theoretic quantity that isn't even monotonic with the size of the data being compressed. The algorithm is based on a modification of the Karp-Rabin fingerprinting technique [6] and analyzed by appealing to properties of cyclotomic polynomials.

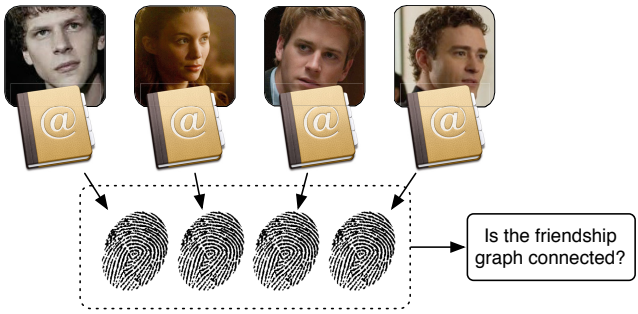


Fig. 2. Each person in a group of n people has an address book that lists their friends in the group. Without any inter-group communication, each person sends a "fingerprint" of their address book to a third party. How many bits must each fingerprint contain for the third party to determine whether the underlying friendship graph is connected with high probability? A trivial upper-bound is n bits and may appear tight. However, recent work shows that even for an arbitrary graph, it suffices for each fingerprint to contain $O(\text{polylog}\, n)$ bits and we extend the result to approximating the size of all cuts in the graph. An example application is the first sub-linear space data structures for processing dynamic graphs.

2.2 Graph Connectivity

Massive graphs arise in any application where there is data about both basic entities and the relationships between these entities, e.g., web-pages and hyperlinks; neurons and synapses; papers and citations; IP addresses and network flows; people and their friendships. Graphs have become the de facto standard for representing many types of highly-structured data. Relevant properties of these graphs could include dense subgraphs corresponding to tight-knit communities; shortest paths for network routing; hubs and authorities; sparse cuts and natural decompositions of the graph. However, the sheer size of some of these graphs can render classical algorithms useless when it comes to analyzing such graphs. For example, both the web graph and models of the human brain would use around 10^{10} nodes while IPv6 supports 2^{128} possible addresses. For such massive graphs we need efficient streaming and parallel algorithms.

In recent work [1,2], we designed a fingerprinting algorithm such that, given a $O(\epsilon^{-2} \cdot \text{polylog}\, n)$ bit fingerprint of each of the n rows of the adjacency matrix of a graph, we can approximate the size of every cut within a factor of $(1+\epsilon)$ with high probability. See Figure 2. Since these fingerprints are linear, this result immediately implies a $O(\epsilon^{-2} \cdot n \cdot \text{polylog}\, n)$-space stream algorithm for the dynamic graph connectivity problem (i.e., checking that the graph defined by a sequence of edge insertions and deletions is connected). Subsequently, similar ideas have also yielded data structures for dynamic connectivity with fast update and query time [5]. Underlying these results was the fact that the fingerprints developed were homomorphic to the operation of edge-contraction. The algorithm combines ℓ_0-sampling [4] with an encoding from matroid theory.

References

1. Ahn, K.J., Guha, S., McGregor, A.: Analyzing graph structure via linear measurements. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 459–467 (2012)
2. Ahn, K.J., Guha, S., McGregor, A.: Graph sketches: sparsification, spanners, and subgraphs. In: ACM Principles of Database Systems, pp. 5–14 (2012)
3. Andoni, A., Goldberger, A., McGregor, A., Porat, E.: Homomorphic fingerprints under misalignments. In: ACM Symposium on Theory of Computing (2013)
4. Jowhari, H., Saglam, M., Tardos, G.: Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In: PODS, pp. 49–58 (2011)
5. Kapron, B., King, V., Mountjoy, B.: Dynamic graph connectivity in polylogarithmic worst case time. In: ACM-SIAM Symposium on Discrete Algorithms (2013)
6. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development 31(2), 249–260 (1987)
7. Rivest, R.: The MD5 message-digest algorithm. RFC Editor (1992)

The Classification Problem for Compact Computable Metric Spaces

Alexander G. Melnikov[1] and André Nies[2]

[1] School of Mathematics, Statistics and Operations Research,
Victoria University of Wellington, Wellington, New Zealand
[2] Department of Computer Science, University of Auckland, Auckland, New Zealand

Abstract. We adjust methods of computable model theory to effective analysis. We use index sets and infinitary logic to obtain classification-type results for compact computable metric spaces. We show that every compact computable metric space can be uniquely described, up to isometry, by a computable Π_3 formula, and that orbits of elements are uniformly given by computable Π_2 formulas. We show that deciding if two compact computable metric spaces are isometric is a Π^0_2 complete problem *within* the class of compact computable spaces, which in itself is Π^0_3. On the other hand, if there is an isometry, then $\varnothing''$ can compute one. In fact, there is a set low relative to $\varnothing'$ which can compute an isometry. We show that the result can not be improved to $\varnothing'$. We also give further results for special classes of compact spaces, and for other related classes of Polish spaces.

1 Introduction

An equivalence relation on a standard Borel space is called *smooth* if it is Borel reducible to the equality relation on $\mathbb{R}$. By a result of Gromov (cf. [7, proof of 14.2.1]), the isometry relation on compact metric spaces is smooth. Thus, every compact metric space can be uniquely described, up to isometry, by a single real. In invariant descriptive set theory, a smooth equivalence relation E is considered trivial: by Silver's theorem, either E is Borel equivalent to equality on $\mathbb{R}$, or E has only countably many classes.

Recall that every compact metric space is separable and complete. Separable complete metric spaces occurring in mathematical practice are usually computable. For instance, $[0,1]^n$, the Hilbert cube, ℓ_2, $\mathcal{C}[0,1]$, and the Urysohn space are computable with any of the standard metrics [11,10]. In this paper, we adapt methods of computable model theory [2,6] to computable analysis [11,3,15] in order to study the classification problem for compact computable metric spaces. Although our paper is mostly restricted to the study of compact computable metric spaces up to isometry, we hope that our ideas and methods will find further applications to other topics of modern computable analysis, such as the study of computable Banach spaces and computable topological spaces.

In contrast to computable analysis, the main objects of computable algebra are countable algebraic structures. These are structures with domain $\mathbb{N}$ and in

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 320–328, 2013.

which the basic operations can be represented by computable functions on $\mathbb{N}$. In computable model theory and effective algebra there are several approaches to classification problems (cf., e.g., [8,9,4,5]). We focus on two approaches which use index sets and infinitary computable logic, respectively.

Index sets and isomorphism problems. The first approach uses the fact that all partially computable functions can be effectively listed. As a consequence, there exists an effective listing of all partial computable algebraic structures $(\mathcal{A}_e)_{e\in\mathbb{N}}$ which includes all infinite computable algebras. For a class $\mathcal{K}$ of computable algebras, the difficulty of the classification problems is reflected in the following sets:

1. *the index set* $I_{\mathcal{K}} = \{e : \mathcal{A}_e \in \mathcal{K}\}$ of $\mathcal{K}$, and
2. *the isomorphism problem* $E_{\mathcal{K}} = \{(e, j) \in I_{\mathcal{K}}^2 : \mathcal{A}_e \cong \mathcal{A}_j\}$ for $\mathcal{K}$.

The complexity of the index sets is measured using the arithmetical, hyperarithmetical, and analytical hierarchies [2]. Recall that the arithmetical hierarchy is defined via iterating quantifiers over computable predicates, and the hyprarithmetical hierarchy extends the arithmetical hierarchy to computable ordinals. Deciding if two algebras from $\mathcal{K}$ are isomorphic might be simpler than detecting whether an algebra belongs to this class. In this case one usually considers the complexity of $E_{\mathcal{K}}$ *within* $I_{\mathcal{K}}$. For example, $E_{\mathcal{K}}$ is Π_2^0 *within* a Π_3^0 set $I_{\mathcal{K}}$ if there exists a Π_2^0 set $S \subset \mathbb{N}^2$ such that $E_{\mathcal{K}} = S \cap (I_{\mathcal{K}} \times I_{\mathcal{K}})$.

A collection of computable models $\mathcal{K}$ is called *classifiable* if both $I_{\mathcal{K}}$ and $E_{\mathcal{K}}$ are hyperarithmetical. (Usually $\mathcal{K}$ will be closed under isomorphism on computable models.) See [8,9,4,5] for further background and results in this direction.

Infinitary computable logic. Ash [1] introduced computable infinitary formulas in the context of computable algebras. An infinitary computable language extends a first-order language by allowing infinite conjunctions and disjunctions over computably enumerable families of formulas. The definition [1,2] uses a recursion scheme. Computable formulas have proved to be of a great importance in computable algebra; cf. the book of Ash and Knight [2]. We say that a class $\mathcal{K}$ of computable structures closed under isomorphism *admits a syntactic description*, if there exists a computable infinitary sentence Φ such that, for any computable M, we have $M \models \Phi$ if and only if $M \in \mathcal{K}$. Note that this condition implies that the index set is hyperarithmetical [8]. The converse is known without the restriction to indices for computable structures. Vanden Boom [14] has shown that every hyperarithmetical invariant class can be described by a computable sentence.

There is also a syntactic counterpart of requiring that $E_{\mathcal{K}}$ is hyperarithmetical.

Definition 1. We say that a class $\mathcal{K}$ of computable structures *admits a syntactic classification* if there is a hyperarithmetical bound on the complexity of infinitary formulas which describe the orbits of tuples of elements in $M \in \mathcal{K}$ under the action of the automorphism group of M.

To adjust the effective classification methods to computable analysis, we need some basic definitions. Following the tradition rooted in the works of

Turing [12,13], we say that a real x is *computable* if for each k we can compute a rational within 2^{-k} of x.

Definition 2 ([3,11]). Let (M, d) be a complete separable metric space, and let $(q_i)_{i\in\mathbb{N}}$ be a dense sequence of points in M. The triple

$$\mathcal{M} = (M, d, (q_i)_{i\in\mathbb{N}})$$

is a *computable metric space* if $d(q_i, q_k)$ is a computable real uniformly in i, k. We say that $(q_i)_{i\in\mathbb{N}}$ is a *computable structure* on M, and that the q_i are the *special points* of $\mathcal{M}$. A *Cauchy name* for x is a sequence $(r_p)_{p\in\mathbb{N}}$ of special points converging rapidly to x in the sense that $d(r_p, r_{p+1}) < 2^{-p}$.

We introduce computable infinitary formulas in the context of computable metric spaces (cf. preliminaries). In Theorem 6 we prove that every computable compact metric space is uniquely described by a computable Π_3 infinitary sentence. Further, the orbits of special elements in a compact computable Polish space (under the action of its automorphism group) are given uniformly by computable infinitary Π_2 formulas. As a consequence, computable compact metric spaces admit a syntactic characterization. In Theorem 10 we shall apply Theorem 6 to show that the index set of compact computable metric spaces is Π^0_3-complete, and the isomorphism problem for compact computable metric spaces is Π^0_2-complete within this index set. Thus, the collection of compact computable metric spaces is classifiable in the sense given above.

2 Preliminaries

We view a metric space (X, d) as a structure in the signature $\mathcal{S} = \{R_{<q}, R_{>q} : q \in \mathbb{Q}^+\}$, where $R_{<q}$ and $R_{>q}$ are binary relation symbols. The intended meaning of $R_{<q}(x, y)$ is that $d(x, y) < q$. The intended meaning of $R_{>q}(x, y)$ is that $d(x, y) > q$. We denote the first-order language of $\mathcal{S}$ by $\mathcal{L}$.

For a tuple $\overline{x} \in X^n$ consider the $n \times n$ distance matrix $D_n(\overline{x}) = d(x_i, x_j)_{i,j<n}$. We often view this matrix as a tuple in $\mathbb{R}^{n^2}$ with the max norm $\|.\|_{\max}$. Sometimes we suppress the subscript n. Note that for any matrix $A \in \mathbb{Q}^{n^2}$ and any positive rational p, there is a quantifier free positive first-order formula $\phi_{A,n,p}(\overline{x})$ in the signature above expressing that $\|D_n(\overline{x}) - A\|_{\max} < p$.

In this paper, the main objects are computable metric spaces. Notice that, in the notations of Definition 2, a separable space is computable if and only if $R_{<r}(q_i, q_k)$ and $R_{>r}(q_i, q_k)$ are uniformly Σ^0_1.

Definition 3. Since all partial functions can be effectively listed, we obtain a uniformly computable sequence of partial computable structures $(M_e)_{e\in\mathbb{N}}$ so that *some* of these M_e are computable structures on metric spaces: we view M_e as a partial computable function ψ such that $r_p = \psi(i, j, p)_{p\in\mathbb{N}}$ converges rapidly (in the sense above) to $d(i, j)$. It is a Π^0_2 property of ψ to be total and describe a metric space. We denote the completion of M_e, after modding out by equivalent points, by $\mathtt{cp}(M_e)$.

Fact 4. *For $(M, d, (p_i)_{i\in\mathbb{N}})$ a computable metric space, and W a c.e. set, $(p_i)_{i\in W}$ is a computable structure on the space $\mathtt{cp}((p_i)_{i\in W}), d)$.*

Proof. If W is infinite, we use a computable bijection $f : \omega \to W$ to define a computable structure $(r_i)_{i\in\mathbb{N}}$ on $\mathtt{cp}((p_i)_{i\in W}), d)$ by the rule $r_i = p_{f(i)}$.

Infinitary Computable Formulas. The language $\mathcal{L}^c_{\omega_1\omega}$ is a countable fragment of $\mathcal{L}_{\omega_1\omega}$. The atomic formulas are (syntactically) open finitary formulas in the language of metric spaces introduced above, with $\neg$ but without $=$. We allow computably enumerable conjunctions, computably enumerable disjunctions, and quantification over a variable.

In contrast to computable model theory, a computable structure on a space is not the whole space but a dense subset of it. Thus, for a computable metric structure M_e and ϕ a computable infinitary formula, $\mathtt{cp}(M_e) \models \phi$ and $M_e \models \phi$ have different interpretations.

The hierarchy of such formulas is defined similarly to the countable case; cf. the book of Ash and Knight [2]. In our specific case, the important modification is that $D_{<q}(x, y)$, for a rational q and special points x and y, should be understood as a Σ_1 formula, and similarly for $D_{>q}(x, y)$.

Informally, in the calculation of the complexity of a formula we also count alternations of infinitary conjunctions and disjunctions. When we count these alternations, we do not distinguish the infinitary conjunction from $\forall$, and disjunction from $\exists$. So, for example, a prefix of the form $\exists \bigwedge \forall \bigvee \exists$ will have only 3 alternations. More formally, the complexity of $\bigvee_i \psi_i$ is determined using $\inf\{\beta : \psi_i \in \Sigma_\beta\}$, and similarly for conjunctions. See [2] for formal definitions. We shall omit the adjective "infinitary" when it is clear from the context.

Fact 5. *Let ψ be a computable formula of complexity Σ_n, where $n \in \omega$. Then the set $\{e : M_e \models \psi\}$ is Σ^0_n. (Similarly for Π_n.)*

Proof. By induction on the complexity of ψ we can show that, if M_e is a (partial) computable metric structure and $M_e \models \psi$, then $\varnothing^{(n-1)}$ will eventually recognize it.

3 Existential Theories and Infinitary Formulas

Theorem 6

(i) Within the class of computable Polish spaces, each compact member is uniquely described up to isometry by a computable Π_3 axiom.

(ii) The orbits of special elements in a compact computable metric space (under the action of its self-isometry group) are given uniformly by computable Π_2 formulas.

Proof. We shall need a result due to Friedman, Fokina, Körwien and Nies (2012) which itself is based on Gromov's work (cf. [7, proof of 14.2.1]).

Proposition 7. *Let X, Y be compact metric spaces. Suppose that tuples $\widetilde{a} \in X^k, \widetilde{b} \in Y^k$ satisfy the same existential positive finitary formulas. Then there is an isometry from X to Y mapping $\widetilde{a}$ to $\widetilde{b}$.*

Proof. It is well-known that any isometric self-embedding of a compact metric space is onto (cf. [7, proof of 14.2.1]). Thus, by symmetry, it suffices to find an isometric embedding of X into Y mapping $\widetilde{a}$ to $\widetilde{b}$. The following lemma from [7, Exercise 14.2.3] slightly extends the above-mentioned result of Gromov.

Lemma 8. *Suppose that for every $\epsilon > 0$, for any n and tuple $\overline{x} \in X^n$ there is a tuple $\overline{y} \in Y^n$ such that $\left\| D(\widetilde{a}, \overline{x}) - D(\widetilde{b}, \overline{y}) \right\|_{\max} < \epsilon$. Then there is an an isometric embedding of X to Y mapping $\widetilde{a}$ to $\widetilde{b}$.*

It now suffices to show that if $\widetilde{a} \in X^n, \widetilde{b} \in Y^n$ satisfy the same existential positive formulas, the hypothesis of the lemma is satisfied. For every $n \times n$ rational matrix A, there is a formula $\phi_{A,n,\epsilon}(\overline{x})$ saying that $\|D_n(\overline{x}) - A\|_{\max} < \epsilon/2$. Given $\overline{x} \in X^n$ choose a rational $(k+n) \times (k+n)$ matrix A such that

$$\|D(\widetilde{a}, \overline{x}) - A\|_{\max} < \epsilon/2.$$

Thus $\exists \overline{x}\, \phi_{A,n+k,\epsilon/2}(\widetilde{a}, \overline{x})$ holds in X. Hence there is $\overline{y} \in Y^n$ such that $\phi_{A,n+k,\epsilon/2}$ $(\widetilde{b}, \overline{y})$ holds in Y. This implies $\left\| D(\widetilde{a}, \overline{x}) - D(\widetilde{b}, \overline{y}) \right\|_{\max} < \epsilon$ as required.

We prove (i) of the theorem. Note that a complete metric space is compact iff it is totally bounded, namely, satisfies the computable sentence

$$\bigwedge_{q \in \mathbb{Q}^+} \bigvee_{n \in \mathbb{N}} \exists x_0 \dots x_{n-1} \forall y \bigvee_{i<n} d(x_i, y) < q. \tag{1}$$

We can replace each quantifier by a quantifier restricted to special points, and also replace $d(x_i, y) < q$ by $\neg(d(x_i, y) > q)$ with the meaning $d(x_i, y) \leq q$. Let θ be the resulting computable sentence. The quantifier $\bigvee_{i<n}$ is finitary and does not contribute any extra complexity to the formula. Thus, θ is computable Π_3. Clearly, $M_e \models \theta$ if and only if $\mathtt{cp}(M_e) \models \theta$.

We take M_e a computable structure on a Polish space. For the tuple $\widetilde{a} = \varnothing$ of special points we let ψ be a conjunction of all formulas $\exists \overline{x}\, \phi_{B,k,\epsilon}(\overline{x})$ (with quantification over special points, B a rational $k \times k$ matrix, ϵ a positive rational) which are true on M_e. Note that $\mathtt{cp}(M_e) \models \exists \overline{x}\, \phi_{B,k,\epsilon}(\overline{x})$ if and only if the corresponding restricted formula holds on M_e. Thus, the conjunction is in fact c.e. since we can enumerate all such sentences which are true on M_e. Therefore, ψ is computable Π_2. The desired computable axiom is $\mathcal{F} = \theta \wedge \psi$ which is of complexity Π_3.

We prove (ii). The orbit of a tuple $\widetilde{a}$ of special points in a compact computable Polish space is definable by the conjunction of $\exists \overline{x}\, \phi_{A,n+k,\epsilon/2}(\widetilde{a}, \overline{x})$ which hold on M_e. Given $\widetilde{a}$ we can effectively list all formulas $\phi_{A,n+k,\epsilon/2}(\widetilde{a}, \overline{x})$ such that $M_e \models \exists \overline{x}\, \phi_{A,n+k,\epsilon/2}(\widetilde{a}, \overline{x})$. Thus, the conjunction of all such formulas, with $\widetilde{a}$ replaced by a tuple of variables $\widetilde{y}$, is effective. Similarly to the proof of (1) above, we have $M_e \models \exists \overline{x}\, \phi_{A,n+k,\epsilon/2}(\widetilde{a}, \overline{x}) \Leftrightarrow \mathtt{cp}(M_e) \models \exists \overline{x}\, \phi_{A,n+k,\epsilon/2}(\widetilde{a}, \overline{x})$, for every $\widetilde{a} \in M_e$ and every parameters A, n, k and ϵ.

4 Descriptive Complexity of Index Sets

Recall from Definition 3 that $\mathtt{cp}(M_e)_{e\in\omega}$ is an effective listing which includes all computable metric spaces. Recall also that a metric space X is connected iff for each nonempty open sets U, V, we have $C = X - (U \cup V) = \varnothing \Rightarrow U \cap V \neq \varnothing$.

Proposition 9. *(i) The set* $\{e : \mathtt{cp}(M_e)$ *is locally compact*$\}$ *is* Π^1_1*–complete. (ii) The set* $\{e : \mathtt{cp}(M_e)$ *is connected*$\}$ *is* Π^1_1*–hard.*

Proof. A complete metric space X is locally compact iff for each $x \in X$, there is rational $\epsilon > 0$ such that the closed ball $K = K_\epsilon(x)$ is compact. If $X = \mathtt{cp}(M_e)$, then from a Cauchy name f for x we can compute a presentation of K as a computable metric space relative to f, where the special points are the special points p of M_e with $d(x, p) < \epsilon$. Then by (1) and the discussion thereafter, compactness of K is arithmetical in f. (On the other hand notice that being connected is merely Π^1_2.)

We now prove the Π^1_1–hardness. As usual let $[T]$ denote the set of infinite branches of a tree $T \subseteq \omega^{<\omega}$, and note that $[T]$, unless empty, is a metric space via $d(f, g) = 2^{-k}$, where k is minimal such that $f(k) \neq g(k)$. Also, $[T]$ is locally compact iff for each $f \in [T]$ there is n such that T with the dead ends removed is finitely branching above $f\restriction_n$.

We encode the problem whether a computable tree T has no infinite branch, which is well known to be Π^1_1–complete. Let

$$F(T) = \{\langle \sigma, \tau\rangle \colon\ \sigma \in T \ \wedge\ \tau \in \omega^{<\omega} \ \wedge\ |\sigma| = |\tau|\}.$$

Via the Cantor pairing function we can view $F(T)$ as a subtree of $\omega^{<\omega}$, and hence as a computable metric space. If $[F(T)]$ is nonempty, it is neither locally compact, nor connected. Now let M_T be the computable metric space obtained by adjoining an isolated point at distance 2 to $[F(T)]$. Then $[T] = \varnothing \Leftrightarrow M_T$ is locally compact $\Leftrightarrow M_T$ is connected.

Theorem 10. *(i) The index set* CSp *of compact computable metric spaces is* Π^0_3*-complete. (ii) The isomorphism problem for compact computable metric spaces is* Π^0_2*-complete within* Π^0_3*.*

Next we study the complexity of whether a computable metric space is a continuum.

Proposition 11. *The index set* CCSp *of compact and connected computable metric spaces is* Π^0_3*-complete.*

Proof. Suppose now we are given a compact computable metric space $X = \mathtt{cp}(M_e)$. For connectedness, we need to check if for each non-empty open U and V, we have $C = X - (U \cup V) = \varnothing \Rightarrow U \cap V \neq \varnothing$. We may restrict U and V to finite unions of basic open sets of the form $B_\epsilon(p)$ where $\epsilon \in \mathbb{Q}^+$ and p is a special point. We may effectively in e obtain a $\varnothing'$-computable map g from 2^ω onto X. Thus $C = \varnothing$ is equivalent to $g^{-1}(C) = \varnothing$. Since the latter is a $\Pi^0_1(\varnothing')$ class, this

condition is Σ^0_2. The condition $U \cap V \neq \varnothing$ is Σ^0_1 since this set contains a special point unless empty. Thus being connected is in fact Π^0_2 within the Π^0_3 set $\mathtt{CSp}$.

Let S be any Π^0_3–complete set, and choose a uniformly c.e. double sequence $(V_{i,n})$ of initial segments of ω such that $i \in S \leftrightarrow \forall n V_{i,n} \neq \omega$. Let $a_k = 1 - 2^{-k}$. Given i, we can compute an index e for the computable metric space the Cartesian product $\prod_{n\in\omega}[0, a_{|V_{i,n}|}]$ with the canonical computable structure obtained from the enumerations of the $V_{i,n}$, and the metric inherited from the standard metric on the Hilbert cube $[0,1]^\omega$. Clearly M_e is connected, and M_e is compact iff $i \in S$.

5 Δ^0_3 Categoricity

Definition 12. Let $S \subseteq \omega$ be an oracle. An isometry Φ from a computable metric space $(X, d, (q_i)_{i\in\mathbb{N}})$ to a computable metric space $(Y, d, (p_i)_{i\in\mathbb{N}})$ is *computable in* S if there is a Turing machine with oracle S which, on inputs i, k, outputs the k-th term of a Cauchy name for $\Phi(q_i)$.

We say that a computable metric space is Δ^0_n*–categorical* if between each of its computable presentations, there is an isometry computable relative to $\varnothing^{(n-1)}$.

Theorem 13. *Each compact computable metric space is Δ^0_3 categorical.*

Proof. Let $\mathcal{X} = (X, d, (p_i)_{i\in\mathbb{N}})$ and $\mathcal{Y} = (Y, d, (q_j)_{j\in\mathbb{N}})$ be compact computable metric spaces. Suppose that $\mathcal{X}$ can be isometrically embedded into $\mathcal{Y}$. We show that then there is a Δ^0_3 embedding; this is sufficient by symmetry.

Recall distance matrices D_n from Section 2. Let $\epsilon_n = 2^{-n}$. There is a computable triangular array of Y-special points $\langle y^n_i\rangle_{i<n}$ such that, where $\overline{y}^n = \langle y^n_0, \ldots, y^n_{n-1}\rangle$, we have

$$\|D_n(\langle p_0, \ldots, p_{n-1}\rangle) - D_n(\overline{y}^n)\|_{\max} < \epsilon_n.$$

We define a $\varnothing''$ computable triangular array of Y-special points $\langle w^n_i\rangle_{i\leq n, 0<n}$ such that for each n, where $\overline{w}^n = \langle w^n_0, \ldots, w^n_{n-1}\rangle$, we have

$$|\{k > n\colon\ d(\overline{y}^k\restriction_n, \overline{w}^n) < \epsilon_n\}| = \infty. \tag{2}$$

We use compactness of Y and its finite powers Y^n throughout. Let $w^1_0 \in Y$ be a special point such that $A_1 = \{k\colon\ d(y^k_0, w^1_0) < \epsilon_1\}$ is infinite. Then (2) holds for $n = 1$.

(a) Increasing the dimension. Let w^1_1 be a special point in Y such that $C_1 = \{k \in A_1\colon\ d(y^k_1, w^1_1) < \epsilon_1\}$ is infinite.

(b) Refining the sequence. Let $\overline{w}^2 \in Y^2$ be a special point in the ball $B_{\epsilon_1}(\langle w^1_0, w^1_1\rangle)$ such that $A_2 = \{k \in C_1\colon\ d(\overline{y}^k\restriction_2, \overline{w}^2) < \epsilon_2\}$ is infinite.

We continue this process. Suppose $\overline{w}^n$ (and hence A_n) has been defined
(a) Let w^n_n be a special point in Y such that

$$C_n = \{k \in A_n\colon\ k > n \wedge d(y^k_n, w^n_n) < \epsilon_n\}$$

is infinite.

(b) Let $\overline{w}^{n+1} \in Y^{n+1}$ be a special point in $B_{\epsilon_n}((\overline{w}^n)^\frown w_n^n)$ such that

$$A_{n+1} = \{k \in C_n \colon d(\overline{y}^k \restriction_{n+1}, \overline{w}^{n+1}) < \epsilon_{n+1}\}$$

is infinite. Then (2) holds for $n+1$.

Note that the sequence $\langle w_i^n \rangle_{i \leq n, 0<n}$ is indeed $\varnothing''$-computable because we uniformly in the previously defined special points obtain indices for the potential c.e. sets C_n, A_{n+1}. It takes $\varnothing''$ as an oracle to pick the next special points in such a way that the relevant set is infinite. Also note that $d(w_r^n, w_r^{n+1}) < \epsilon_n$ for each $n > r$. Thus, the sequence of points in Y $z_r = \lim_{n>r} w_r^n$ is computable in $\varnothing''$. It now suffices to show that the map $x_i \mapsto z_i$ preserves distances. Let $i < j$. Given n, by (2) pick $k > n$ such that $d(y^k \restriction_n, \overline{w}^n) < \epsilon_n$. Then, by the definitions,

$$\begin{aligned} |d(z_i, z_j) - d(w_i^n, w_j^n)| &\leq 2\epsilon_n \\ |d(w_i^n, w_j^n) - d(y_i^k, y_j^k)| &\leq \epsilon_n \\ |d(y_i^k, y_j^k) - d(x_i, x_j)| &\leq \epsilon_n. \end{aligned}$$

Therefore, $|d(z_i, z_j) - d(x_i, x_j)| \leq 4\epsilon_n$.

The bound on the complexity of an isomorphism we obtained in Theorem 13 is not optimal. We can prove the following strengthening saying that some isomorphism is low relative to $\varnothing'$.

Theorem 14. *Let $\mathcal{X} = (X, d, (p_i)_{i\in\mathbb{N}})$ and $\mathcal{Y} = (Y, d, (q_j)_{j\in\mathbb{N}})$ be isometric compact computable metric spaces. Then there is a set S with $S' \leq_T \varnothing''$ which computes an isometry.*

The proof is an extension of the previous argument in that we build a nonempty $\Pi_1^0(\varnothing')$ class of isometries. Since the space is compact, the level size of the corresponding tree is bounded by a $\varnothing'$–computable function. Then, by the low basis theorem relative to $\varnothing'$, we obtain an isometry as required. We have also shown that the bound in Theorem 13 can not be improved to Δ_2^0, by building a metric space with two computable presentations and no Δ_2^0 isometry between them. Proofs of these results will appear in a journal paper.

Acknowledgement. Nies was supported through the Marsden fund of New Zealand.

References

1. Ash, C.J.: Stability of recursive structures in arithmetical degrees. Ann. Pure Appl. Logic 32(2), 113–135 (1986)
2. Ash, C.J., Knight, J.: Computable structures and the hyperarithmetical hierarchy. Studies in Logic and the Foundations of Mathematics, vol. 144. North-Holland Publishing Co., Amsterdam (2000)
3. Brattka, V., Hertling, P., Weihrauch, K.: A tutorial on computable analysis. In: New Computational Paradigms, pp. 425–491. Springer, New York (2008)

4. Calvert, W., Knight, J.F.: Classification from a computable viewpoint. Bull. Symbolic Logic 12(2), 191–218 (2006)
5. Csima, B.F., Montalbán, A., Shore, R.A.: Boolean algebras, Tarski invariants, and index sets. Notre Dame J. Formal Logic 47(1), 1–23 (2006)
6. Ershov, Y.L., Goncharov, S.S.: Constructive models. In: Siberian School of Algebra and Logic. Consultants Bureau, New York (2000)
7. Gao, S.: *Invariant descriptive set theory*. Pure and Applied Mathematics (Boca Raton), vol. 293. CRC Press, Boca Raton (2009)
8. Goncharov, S.S.: Dzh. Naĭt. Computable structure and antistructure theorems. Algebra Logika 41(6), 639–681, 757 (2002)
9. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: richness and limitations. Log. Methods Comput. Sci. 3(2), 2:2, 18 (2007)
10. Melnikov, A.: Computably isometric spaces (preprint)
11. Pour-El, M.B., Richards, J.I.: Computability in analysis and physics. In: Perspectives in Mathematical Logic, Springer, Berlin (1989)
12. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proceedings of the London Mathematical Society 42, 230–265 (1936)
13. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. A correction. Proceedings of the London Mathematical Society 43, 544–546 (1937)
14. Vanden Boom, M.: The effective Borel hierarchy. Fund. Math. 195(3), 269–289 (2007)
15. Weihrauch, K.: Computable analysis. In: An introduction Texts in Theoretical Computer Science. An EATCS Series, Springer, Berlin (2000)

Exploiting Co-evolution across Protein Families for Predicting Native Contacts and Protein-Protein Interaction Surfaces

Andrea Pagnani[1,2]

[1] DISAT and and Center for Computational Studies, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy
[2] Human Genetics Foundation, Via Nizza 52, 10126 Turin, Italy

Correlated substitution patterns between residues of a protein family have been exploited to reveal information on the structures of proteins. However, such studies require a large number (e.g., the order of one thousand) of homologous yet variable protein sequences. So far, most studies have been limited to a few exemplary proteins for which a large number of such sequences happen to be available. Rapid advances in genome sequencing will soon be able to generate this many sequences for the majority of common bacterial proteins. Sequencing a large number of simple eukaryote such as yeast can in principle generate similar number of common eukaryotic protein sequences, beyond a collection of highly amplified protein domains which already reach the necessary numbers.

The heart of our approach is a novel, statistical-physics inspired analysis of residue co-evolution, which has recently been shown (i) to accurately predict inter- and intra-protein amino-acid spatial dependencies, (ii) to achieve structural predictions with experimental accuracy when integrated with molecular simulations.

A systematic evaluation of the information contained in correlated substitution patterns for predicting residue contacts will be given, as a first step towards a purely sequence-based approach to protein structure, and protein-protein interaction prediction.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, p. 329, 2013.

A Compositional Semantics of Reaction Systems with Restriction

Giovanni Pardini[1], Roberto Barbuti[1], Andrea Maggiolo-Schettini[1], Paolo Milazzo[1], and Simone Tini[2]

[1] Dipartimento di Informatica, Università di Pisa, Largo Pontecorvo 3, 56127 Pisa, Italy
{pardinig,barbuti,maggiolo,milazzo}@di.unipi.it
[2] Dipartimento di Scienza e Alta Tecnologia, Università dell'Insubria, Via Valleggio 11, 22100 Como, Italy
simone.tini@uninsubria.it

Abstract. Reaction systems are an abstract model of interactions among biochemical reactions, developed around two opposite mechanisms: facilitation and inhibition. The evolution of a Reaction System is driven by the external objects which are sent into the system by the environment at each step. In this paper, we propose the *Reaction Algebra*, a calculus resembling reaction systems extended with a restriction operator. Restriction increases the expressiveness of the calculus by allowing the modeling of hidden entities, such as those contained in membranes.

We define a compositional semantics and a behavioral equivalence for the Reaction Algebra, in order to enable the modular description of biological systems.

1 Introduction

Reaction systems [1, 2] have been introduced by Ehrenfeucht and Rozenberg as a novel model for the description of biochemical processes driven by the interaction among reactions in living cells. Reaction systems are based on two opposite mechanisms, namely *facilitation* and *inhibition*. Facilitation means that a reaction can occur only if all its reactants are present, while inhibition means that the reaction cannot occur if any of its inhibitors is present. Reactants play a role analogous to that of *promoters* in the setting of Membrane Computing [3].

The state of a reaction system is modeled as a finite set of entities, and evolves by means of the application of a set of reactions. The presence of an element in the state expresses the fact that the corresponding biological entity, in the real system being modeled, is present in a number of copies as high as needed.

The overall behavior of a reaction system model is driven by the (set of) contextual elements which are received from the external environment at each step. Moreover, reaction systems assume the *non permanency* of the elements, namely unused elements are never carried over to the next state. In particular, the next state consists only of the products of the reactions applied in the current step to a set composed of both the contextual elements and the current state.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 330–339, 2013.

In this paper, we define the *Reaction Algebra*, a calculus resembling reaction systems which also includes a restriction operator à la CCS. The purpose of the restriction operator is to increase the modeling expressiveness of the calculus, by allowing elements to be hidden. This, for example, would enable a direct modeling of membranes, by hiding internal elements and their interactions. Together with the behavioral equivalence that we also define, restriction allows comparing two systems only with respect to the "visible" elements. Restriction is analogous to compartmentalization operators provided by many bio-inspired calculi.

Due to the presence of the external environment, Reaction Algebra models are ultimately interactive systems. For this reason, we define a compositional semantics of Reaction Algebra by means of *Labeled Transition Systems* (LTS), where each transition captures the behavior with respect to the contextual elements received from the environment. Note that, in this setting, there is no distinction between the "external environment", as defined by reaction systems, and "the rest of the system", since both of them can actually be seen as supplying contextual elements to the system. We show that the compositional semantics is equivalent to that of reaction systems, for terms without restriction.

We propose bisimulation as a proper behavioral equivalence for Reaction Algebra terms, and show that it is a congruence, thus providing a formal ground to the modular composition of Reaction Algebra systems. Finally, we show that the behavioral equivalence of Reaction Algebra terms subsumes the *functional equivalence* between reaction systems proposed in [1].

Related work. Reaction systems have been used to model various features which are useful for the modeling of biological systems. For example, binary counters [1] form the basis for the inclusion of a notion of time [4]. In [5] an extension with duration of reaction systems is presented. Theoretical aspects of reaction systems have been studied in [6–9]. Compositional semantics and behavioral equivalences of variants of P systems have been proposed in [10–12]. An axiomatization of the behavioral equivalence from [10] is presented in [13].

The paper is structured as follows. In Section 2 we recall the definition of Reaction systems, and their related concepts. Section 3 contains the formal definition of the Reaction Algebra with restriction, the compositional semantics, and the behavioral equivalence. Finally, in Section 4 we draw the conclusions of our work, and discuss possible ideas for future work.

2 Background on Reaction Systems

In this section we recall the basic definition of reaction systems [1, 2]. Let S be a finite set of objects used in a reaction system. A *reaction* is formally a triple (R, I, P) with $R, I, P \subseteq S$, composed of *reactants* R, *inhibitors* I, and *products* P. We assume reactants and inhibitors to be disjoint ($R \cap I \neq \emptyset$), otherwise the reaction would never be applicable. The set of all possible reactions over a set S is denoted by $\mathrm{rac}(S)$. Finally, a *reaction system* is a pair $\mathcal{A} = (S, A)$, with S being the finite background set, and $A \subseteq \mathrm{rac}(S)$ being its set of reactions.

The state of a reaction system is described by a set of objects. Let $a = (R_a, I_a, P_a)$ be a reaction and T a set of objects. The result $\mathrm{res}_a(T)$ of the application of a to T is either P_a, if T separates R_a from I_a (i.e., $R_a \subseteq T$ and $I_a \cap T = \emptyset$), or the empty set $\emptyset$ otherwise. Because of the threshold supply assumption, the application of multiple reactions at the same time occurs without any competition for the used reactants. Therefore, each reaction which is not inhibited can be applied, and the result of application of multiple reactions is cumulative. Formally, given a reaction system $\mathcal{A} = (S, A)$, the results of application of $\mathcal{A}$ to a set $T \subseteq S$ is defined as $\mathrm{res}_{\mathcal{A}}(T) = \mathrm{res}_A(T) = \bigcup_{a \in A} \mathrm{res}_a(T)$.

The dynamics of a reaction system is driven by the *contextual* objects, namely the objects which are supplied to the system by the external environment at each step. Formally, the dynamics of $\mathcal{A} = (S, A)$ is defined as an *interactive process* $\pi = (\gamma, \delta)$, with γ and δ being finite sequences of sets of objects called the *context sequence* and the *result sequence*, respectively. The sequences are of the form $\gamma = C_0, C_1, \ldots, C_n$ and $\delta = D_0, D_1, \ldots, D_n$ for some $n \geq 1$, with $C_i, D_i \subseteq S$, and $D_0 = \emptyset$. Each set D_i, for $i \geq 1$, in the result sequence is obtained from the application of reactions A to a state composed of both the results of the previous step D_{i-1} and the objects C_{i-1} from the context; formally $D_i = res_{\mathcal{A}}(C_{i-1} \cup D_{i-1})$ for all $1 \leq i \leq n$. Note that, according to the *non-permanency* assumption, unused objects from D_{i-1} are not carried over to D_i. Finally, the *state sequence* of π is defined as the sequence $W_0, W_1, \ldots, W_n$, where $W_i = C_i \cup D_i$ for all $1 \leq i \leq n$.

We conclude this section by recalling from [1] the definition of *functional equivalence* between reaction systems. Given two reactions $a, b \in \mathrm{rac}(S)$ for some background set S, a and b are functionally equivalent (denoted $a \sim_S b$) iff $\forall T \subseteq S, \mathrm{res}_a(T) = \mathrm{res}_b(T)$ (this implies that all reactions of the form $(R, I, \emptyset)$ are functionally equivalent). This equivalence relation is extended to sets of reactions in the natural way: given $A, B \in \mathrm{rac}(S)$, then A and B are functionally equivalent ($A \sim_S B$) iff $\forall T \subseteq S$, $\mathrm{res}_A(T) = \mathrm{res}_B(T)$. Finally, two reaction systems $\mathcal{A}, \mathcal{B}$ defined over the same background set S are functionally equivalent ($\mathcal{A} \sim_S \mathcal{B}$) iff their sets of reactions are functionally equivalent. The problem of deciding the functional equivalence between two sets of reactions is co-NP-complete [1].

3 Reaction Algebra

In order to define a compositional semantics of reaction systems, we propose in this section the *Reaction Algebra*, a calculus resembling reaction systems. A term of the reaction algebra is formed by a parallel composition of *reactions* $R \overset{I}{\mapsto} P$ and single *elements* $c \in S$. The elements appearing in a term represent the elements occurring in the state in the current step which, according to the behavior of reactions systems, have been produced by the reactions at the previous step. With respect to the dynamics of reaction systems, at a generic step $i \geq 0$, the elements in the term correspond to the set D_i from the result sequence δ.

A major difference from reaction systems is the inclusion in the Reaction Algebra of a *restriction operator* $\cdot \setminus \cdot$, which allows a greater expressiveness of

the calculus. Conceptually, such an operator is akin to the restriction operator of process algebras, and allows a primitive, but powerful, way of modeling a hidden internal state for reaction systems.

Definition 1 (Syntax). *Let S be a finite and* totally ordered set *of symbols, with total order relation $\leq$. The syntax of Reaction Algebra terms is defined by the following grammar:*

$$\mathcal{A} \ ::= \ \mathbf{0} \ \mid \ R \overset{I}{\mapsto} P \ \mid \ c \ \mid \ \mathcal{A}\backslash c \ \mid \ \mathcal{A} \mid \mathcal{A}$$

where $R, I, P \subseteq S$, $c \in S$, $R \cap I = \emptyset$.

Symbols from the set S denote entities of the biological system, which can be used in the reactions. Without loss of generality, we assume that S is a totally ordered set; this assumption is exploited in the definition of the compositional semantics. In the following, we use the symbol r (possibly with a subscript) to denote a generic reaction $R \overset{I}{\mapsto} P$. Moreover, we denote the set of all terms over the set S as Θ_S.

The operator $\cdot \mid \cdot$ denotes the parallel composition of two terms. In order to avoid inserting too many parentheses, we assume the parallel operator to be left-associative. The restriction operator $\mathcal{A}\backslash c$ indicates that all free occurrences of symbol c in $\mathcal{A}$ are considered private, thus introducing a form of bound symbols into the calculus. We define the set of *free objects* of a term as follows:

$$\begin{array}{cccc} \mathrm{fo}(\mathbf{0}) = \emptyset & & \mathrm{fo}(R \overset{I}{\mapsto} P) = \emptyset & \mathrm{fo}(c) = \{c\} \\ \mathrm{fo}(\mathcal{A}\backslash c) = \mathrm{fo}(\mathcal{A}) \setminus \{c\} & & \mathrm{fo}(\mathcal{A}_1 \mid \mathcal{A}_2) = \mathrm{fo}(\mathcal{A}_1) \cup \mathrm{fo}(\mathcal{A}_2) & \end{array}$$

The motivation for including the restriction operator is to increase the modeling expressiveness of the calculus. Recall, from Section 2, the definition of functional equivalence $\cdot \sim \cdot$ between reaction systems: given two sets of reactions $A, B \in \mathrm{rac}(S)$, then $A \sim_S B$ iff $\forall T \subseteq S$, $\mathrm{res}_A(T) = \mathrm{res}_B(T)$. Therefore, functional equivalence captures the natural way of defining a behavioral equivalence between reaction systems, since two systems are considered equivalent if they behave in the same way over any possible set of elements $T \subseteq S$. In other words, two equivalent reaction systems $(S, A_1) \sim (S, A_2)$ may only differ in the sets of reactions A_1, A_2, while they need to produce the same elements at each step, given any possible context sequence γ. This is a very strong requirement, which rules out the possibility of modeling "private" elements, such as those occurring inside membranes. A standard reaction system does not allow the hiding of the internal elements appearing inside the membrane, which are never sent outside. Therefore it is not possible to compare two reaction systems only on the basis of the *externally observable* behavior, since any purportedly "internal" element is always observable.

Finally, recall that the initial state of a reaction system is always empty. In fact, the role of such an initial set of elements is played by the first set D_0 of the result sequence δ, which is assumed to be empty. In the reaction algebra, where the initial state conceptually corresponds to the elements appearing in the initial term, we do not have such a restriction.

3.1 Compositional Semantics

In this section we define a compositional semantics for the Reaction Algebra, formalized by means of a Labeled Transition System, where states are terms of the Reaction Algebra, and each transition represents a step of the evolution of the system. Informally, transition labels describe both the contextual elements that are required for the transition to occur, and those elements which prevent it. Precisely, from a state, there is a different transition for each possible set of contextual elements which are present/absent. Since reaction systems assume the background set to be finite, this ensures that the LTS is finitely branching.

Note that, in a compositional semantics for the Reaction Algebra, from the point of view of a term $\mathcal{A}$ there is no distinction between elements received from the *external environment* (as per reaction systems), and those available in the rest of the system, i.e., those elements which form a parallel composition with $\mathcal{A}$. In fact, according to the semantics of reaction systems, both of them contribute to the set of elements available to the reactions.

The semantics is described by an LTS with transitions of the form $\mathcal{A} \xrightarrow{x,y,z} \mathcal{A}'$, with $x, y, z \subseteq S$ being a partition of S. Label x denotes the elements visible at top-level, and it corresponds to the free elements of $\mathcal{A}$, i.e., $x = \mathrm{fo}(\mathcal{A})$. On the other hand, labels y and z describe the applicability of the transition with respect to the contextual elements; precisely, y denotes which elements must be present, while z denotes those which must be absent.

Definition 2. *The* compositional semantics *of Reaction Algebra is defined as the LTS* $(\Theta_S, (2^S)^3, \longrightarrow)$*, where* $\longrightarrow$ *is the least transition relation statisfying the inference rules of Figure 1.*

Rules (1) and (2) define the semantics of single reactions. In particular, the first one describes the application of the reaction, for which the products $p_1, \ldots, p_k$ are added to the resulting term. The condition on the ordering of the products with respect to the total order relation assumed for the elements in S ensures the uniqueness of the transition for a given label. The rule allows deriving a transition labeled by the triplet $(\emptyset, y, z)$, for each partition y, z of S such that $R \subseteq y$ and $I \subseteq z$. Conversely, a reaction cannot be applied in two cases: either at least one of its reactants is absent, or at least one of its inhibitors is present. Therefore, for each reactant $c \in R$, rule (2) allows deriving transitions labeled $(\emptyset, y, z)$ with y, z being a partition of S such that $c \in z$. Similarly, each inhibitor $c' \in I$ yields transitions with y, z being a partition of S such that $c' \in y$.

Rule (3) defines the semantics of a single element. In agreement with the non-permanency assumption, the resulting term is the nil term $\mathbf{0}$. This rule allows deriving transitions labeled $(\{c\}, y, z)$, where the first item correctly captures the available elements in the source term. In this case, the transition is independent of contextual elements; therefore the transition can be derived for any partition y, z of $S \setminus \{c\}$.

Rule (4), which defines the semantics of the nil term, is similar to the previous case, except that no elements are available, therefore label x is empty, and the transition is permitted by any partition y, z of S.

$$\frac{Z_1 \cup Z_2 = S \setminus (R \cup I) \quad Z_1 \cap Z_2 = \emptyset \quad P = \{p_1, \ldots, p_k\} \quad p_1 < \cdots < p_k}{R \overset{I}{\mapsto} P \xrightarrow{\emptyset,\, R \cup Z_1,\, I \cup Z_2} R \overset{I}{\mapsto} P \mid p_1 \mid \cdots \mid p_k} \quad (1)$$

$$\frac{I' \subseteq I \quad R' \subseteq R \quad |I' \cup R'| = 1 \qquad Z_1 \cup Z_2 = S \setminus (I' \cup R') \qquad Z_1 \cap Z_2 = \emptyset}{R \overset{I}{\mapsto} P \xrightarrow{\emptyset,\, I' \cup Z_1,\, R' \cup Z_2} R \overset{I}{\mapsto} P} \quad (2)$$

$$\frac{Z_1 \cup Z_2 = S \setminus \{c\} \qquad Z_1 \cap Z_2 = \emptyset}{c \xrightarrow{\{c\}, Z_1, Z_2} \mathbf{0}} \quad (3)$$

$$\frac{Z_1 \cup Z_2 = S \qquad Z_1 \cap Z_2 = \emptyset}{\mathbf{0} \xrightarrow{\emptyset, Z_1, Z_2} \mathbf{0}} \quad (4)$$

$$\frac{\mathcal{A}_1 \xrightarrow{x_1, y_1, z} \mathcal{A}_1' \qquad \mathcal{A}_2 \xrightarrow{x_2, y_2, z} \mathcal{A}_2' \qquad y' = y_1 \setminus x_2}{\mathcal{A}_1 \mid \mathcal{A}_2 \xrightarrow{x_1 \cup x_2,\, y',\, z} \mathcal{A}_1' \mid \mathcal{A}_2'} \quad (5)$$

$$\frac{\mathcal{A} \xrightarrow{x, y, z} \mathcal{A}' \qquad c \notin y \qquad Z_1 \cup Z_2 = \{c\} \qquad Z_1 \neq Z_2}{\mathcal{A} \setminus c \xrightarrow{x \setminus \{c\},\; y \cup Z_1,\; z \setminus \{c\} \cup Z_2} \mathcal{A}' \setminus c} \quad (6)$$

Fig. 1. Inference rules of the compositional semantics

As regards the parallel operator, rule (5) defines the transitions derivable from a term of the form $\mathcal{A}_1 \mid \mathcal{A}_2$. The premises of the rule consider transitions of the subterms $\mathcal{A}_1, \mathcal{A}_2$ having the same set of inhibitors z. This ensures that the two transitions are "compatible", i.e., they can be applied at the same time. In fact, since both (x_1, y_1, z) and (x_2, y_2, z) are partitions of S, this implies that the inhibitors of a transition are distinct from both the available elements and reactants of the other transition, and vice versa; that is $z \cap (x_1 \cup y_1) = z \cap (x_2 \cup y_2) = \emptyset$. The set of inhibitors of the resulting transition is the same z as for the subterms, while the elements available are obtained by the union of those available in both $\mathcal{A}_1$ and $\mathcal{A}_2$. Finally, the reactants y' of the resulting transition correspond to the reactants required by the subterms which are not already available in one of the two. Note that, since (x_1, y_1, z) and (x_2, y_2, z) are partitions, then $y' = y_1 \setminus x_2 = y_2 \setminus x_1$.

Lastly, rule (6) deals with the restriction operator, providing the semantics of terms of the form $\mathcal{A} \setminus c$. As regards the available elements in the resulting transition, they are obtained by removing c from x, because c is considered private to the restriction operator and thus it is not visible outside. On the other hand, given a transition for $\mathcal{A}$ labeled x, y, z with $c \notin y$, rule (6) allows deriving two different transitions in which c appears once in y, thus being present, and once in z, thus being absent. The idea is that any contextual element c denotes a different element than the one appearing within the scope of the restriction operator. The condition $c \notin y$ ensures that no transitions still requiring c to be present are considered, since such an element can never be provided from the outside.

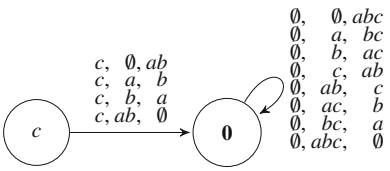


Fig. 2. LTS of term c

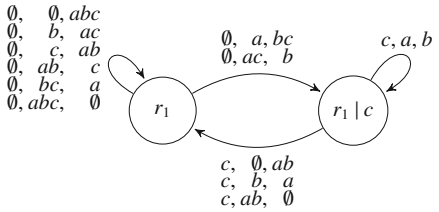


Fig. 3. LTS of term $r_1 = a \overset{b}{\mapsto} c$

Example Let us consider a single element c. Its semantics corresponds to the finite LTS shown in Figure 2, where different transitions having the same source and target are collapsed into a single arrow with multiple labels. Recall that the first label x of each transition corresponds to the free elements of the term. In each state, for each set y of contextual elements from $S \setminus x$ there is a transition describing the behavior when y is present, and $z = S \setminus (x \cup y)$ is absent. From state c all transitions lead to $\mathbf{0}$, since there are no reactions. Similarly, all transitions from $\mathbf{0}$ yield to $\mathbf{0}$ itself.

As regards the semantics of reactions, let us consider term $r_1 = a \overset{b}{\mapsto} c$. Since the term has no free objects, its behavior depends upon contextual elements. In particular, we expect c to be produced iff a is provided from the context and b is not; namely there is a transition $r_1 \xrightarrow{\emptyset,y,z} r_1 \mid c$ for each partition y, z of S such that $a \in y$ and $b \in z$.

Since c is neither a reactant nor an inhibitor of r_1, we expect the behavior of state $r_1 \mid c$ to be analogous to r_1, i.e., c is produced whenever $a \in y$ and $b \in z$. In such a case, there is the transition $r_1 \mid c \xrightarrow{\{c\},y,z} r_1 \mid c \mid \mathbf{0}$. On the other hand, whenever $a \notin y$ or $b \notin z$, there is a transition $r_1 \mid c \xrightarrow{\{c\},y,z} r_1 \mid \mathbf{0}$. Note that we cannot get rid of redundant occurrences of $\mathbf{0}$, therefore the length of the term increases at each step, giving rise to an infinite LTS. As we are going to show in the following of the paper, the semantics is invariant with respect to commutativity and associativity of the parallel operator, and $\mathbf{0}$ is an identity element for it. By assuming such properties, the LTS reduces to the finite LTS shown in Figure 3.

Finally, let us consider term $(r_1)\backslash a$, obtained by enclosing r_1 in a restriction over a. Since a is a reactant of r_1, the reaction is never applicable, thus we expect the semantics of $(r_1)\backslash a$ to be the same as $\mathbf{0}$. It is easy to see that rule 6 allows deriving transitions $(r_1)\backslash a \xrightarrow{\emptyset,Z_1,Z_2} (r_1)\backslash a$ for all partitions Z_1, Z_2 of S.

Proposition 1. *The semantics satisfies the following properties:*

1. *(determinism) whenever* $\mathcal{A} \xrightarrow{x,y,z} \mathcal{A}'$ *and* $\mathcal{A} \xrightarrow{x,y,z} \mathcal{A}''$, $\mathcal{A}' = \mathcal{A}''$;
2. *for all transitions* $\mathcal{A} \xrightarrow{x,y,z} \mathcal{A}'$, $x = \mathrm{fo}(\mathcal{A})$, *and sets* x, y, z *form a partition of the background set* S;

3. *for all terms* $\mathcal{A}$*, and sets* $\{y, z\}$ *being a partition of* $S \setminus \mathrm{fo}(\mathcal{A})$*, there exists a transition* $\mathcal{A} \xrightarrow{x,y,z} \mathcal{A}'$ *with* $x = \mathrm{fo}(\mathcal{A})$ *for some* $\mathcal{A}'$*.*

A consequence of the above proposition is that either one of labels y and z of the semantics is actually redundant and could be omitted. Nevertheless, we believe the definition using both labels y and z enjoys greater clarity.

We have already anticipated that the semantics satisfies some intuitive invariant properties. We now define a congruence which provides an account of them, then we show that the semantics is invariant with respect to it.

Definition 3. *Let* $\equiv \subseteq \Theta_S \times \Theta_S$ *denote the least congruence relation satisfying the following axioms:*

1. $\mathcal{A}_1 \mid \mathcal{A}_2 \equiv \mathcal{A}_2 \mid \mathcal{A}_1$;
2. $(\mathcal{A}_1 \mid \mathcal{A}_2) \mid \mathcal{A}_3 \equiv \mathcal{A}_1 \mid (\mathcal{A}_2 \mid \mathcal{A}_3)$;
3. $\mathcal{A} \mid \mathbf{0} \equiv \mathcal{A}$;
4. $\mathcal{A} \mid \mathcal{A} \equiv \mathcal{A}$;
5. $\mathcal{A} \backslash c \backslash d \equiv \mathcal{A} \backslash d \backslash c$;
6. $\mathcal{A} \backslash c \mid d \equiv (\mathcal{A} \mid d) \backslash c \quad$ *if* $c \neq d$.

Axioms 1 and 2 state the commutativity and associativity of the parallel composition operator, respectively. Axiom 3 states that $\mathbf{0}$ is an identity element of the parallel operator. Axiom 4 states that having more occurrences of the same term does not change the semantics. Intuitively, axioms 1–4 mean that a term can be seen as a *set* composed of reactions, single elements, and restricted subterms. In line with the definition of reaction systems, there is no counting involved.

Axiom 5 states that the order of multiple consecutive restrictions is not meaningful. Axiom 6 states a peculiar property of the semantics, as regards the possibility for an element to move between the inner subterm of a restriction and the external scope. Precisely, an element d can be moved inside the scope of a restriction operator only as long as it does not become bound. Conversely, if the element d appears within the scope of the restriction operator then it can be moved outside only if it is not actually a private symbol.

Lemma 1. *Semantics* $\longrightarrow$ *is closed under* $\equiv$*, namely* $\forall \mathcal{A}_1, \mathcal{A}_1', \mathcal{A}_2, x, y, z.\ \mathcal{A}_1 \equiv \mathcal{A}_1' \wedge \mathcal{A}_1 \xrightarrow{x,y,z} \mathcal{A}_2$ *implies* $\exists \mathcal{A}_2' \equiv \mathcal{A}_2.\ \mathcal{A}_1' \xrightarrow{x,y,z} \mathcal{A}_2'$*.*

The following theorem shows that a reaction system modeled as a Reaction Algebra term behave in the same way as the original system. In other words, we prove the correctness of compositional semantics of the Reaction Algebra w.r.t. the semantics of reaction systems. Note that the comparison deals only with the restriction-free fragment of the Reaction Algebra, since any reaction system can be directly encoded in a term without restriction operators.

Given a reaction system (S, A), we denote as $\eta(S, A)$ the Reaction algebra term associated with it; formally: $\eta(S, \{a_1, \ldots, a_n\}) = \eta(a_1) \mid \cdots \mid \eta(a_n)$, $\eta(R, I, P) = R \overset{I}{\mapsto} P$. Moreover, let $\mathrm{traces}(\mathcal{A}) = \{\langle (x_0, y_0, z_0), \ldots, (x_n, y_n, z_n) \rangle \mid \exists \mathcal{A}_0, \ldots, \mathcal{A}_n.\ \mathcal{A}_0 = \mathcal{A},\ \forall i < n.\ \mathcal{A}_i \xrightarrow{x_i, y_i, z_i} \mathcal{A}_{i+1}\}$.

Theorem 1. *Let* (S, A) *be a reaction system,* $\gamma = C_0, C_1, \ldots, C_n$ *be a context sequence for some* $n \geq 1$*. Then* $\delta = D_0, D_1, \ldots, D_n$ *is the result sequence corresponding to* γ*, as defined by the semantics of reaction systems, iff* $\langle (x_0, y_0, z_0),$

$(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\rangle \in \text{traces}(\eta(S, A))$, *with* $\forall i.\ x_i = D_i,\ y_i = C_i \setminus D_i$, *and* $z_i = S \setminus (C_i \cup D_i)$.

3.2 Behavioral Equivalence

The greater expressiveness of the Reaction Algebra stems from both the restriction operator, and the ability to consider an initial non-empty set of elements. We argue that *bisimulation*, the finest behavioral equivalence one may impose over LTSs, identifies all the terms that intuitively have the same behavior. In this setting, as we are going to show, bisimulation is also a congruence, thus enabling the modular composition of Reaction Algebra terms.

For example, let us consider the term $r_1 \mid r_2$, with $r_1 = a \overset{b}{\mapsto} c$, $r_2 = ab \mapsto c$. It is easy to see that such a term is able to produce c whenever a occurs, *independently* of the fact that b is present or absent. Therefore, if we consider reaction $r_3 = a \mapsto c$, we expects r_3 to be behaviorally equivalent to $r_1 \mid r_2$.

Formally, given the LTS $(\Theta_S, (2^S)^3, \longrightarrow)$, a binary relation $R \subseteq \Theta_S \times \Theta_S$ is a *bisimulation* if, whenever $(\mathcal{A}_1, \mathcal{A}_2) \in R$, then $\forall x, y, z \subseteq S$:

- $\mathcal{A}_1 \xrightarrow{x,y,z} \mathcal{A}_1' \implies \exists \mathcal{A}_2'.\ \mathcal{A}_2 \xrightarrow{x,y,z} \mathcal{A}_2' \wedge (\mathcal{A}_1', \mathcal{A}_2') \in R$;
- $\mathcal{A}_2 \xrightarrow{x,y,z} \mathcal{A}_2' \implies \exists \mathcal{A}_1'.\ \mathcal{A}_1 \xrightarrow{x,y,z} \mathcal{A}_1' \wedge (\mathcal{A}_1', \mathcal{A}_2') \in R$.

Two terms $\mathcal{A}_1, \mathcal{A}_2$ are *bisimilar*, denoted $\mathcal{A}_1 \leftrightarrows \mathcal{A}_2$, if there exists a bisimulation R such that $(\mathcal{A}_1, \mathcal{A}_2) \in R$.

Albeit bisimulation being the finest behavioral equivalence, the Reaction Algebra is actually not expressive enough to fully exploit it. In fact, since the semantics is deterministic (Proposition 1), all the usual behavioral equivalences coincide [14]; in particular, bisimulation is also the same as trace equivalence.

Theorem 2. *Bisimulation over Reaction Algebra terms is a congruence.*

The following theorem shows that the behavioral equivalence $\leftrightarrows$ defined for Reaction Algebra terms, when considering only restriction-free terms without free objects, coincides with the *functional equivalence* $\sim$ of reaction systems.

Theorem 3. $\forall A_1, A_2 \in \text{rac}(S) : (S, A_1) \sim_S (S, A_2) \iff \eta(S, A_1) \leftrightarrows \eta(S, A_2)$.

Example. Let us consider a system which keeps producing c until signal *off* is received from the environment. Given a background set $S \supseteq \{a, b, c, \mathit{off}\}$, it is easy to see that the following terms are behaviorally equivalent: $(a \mid a \overset{\mathit{off}}{\longmapsto} ac) \setminus a \leftrightarrows (a \mid a \overset{\mathit{off}}{\longmapsto} bc \mid b \overset{\mathit{off}}{\longmapsto} ac) \setminus a \setminus b$.

4 Conclusions

We have proposed the Reaction Algebra with restriction, a calculus resembling reaction systems, equipped with a compositional semantics formalized as a LTS.

Such a semantics has been shown to conform to the semantics of reaction systems. In order to enable the modular construction of Reaction Algebra systems, we have proposed a behavioral equivalence which is a congruence, and shown that it subsumes the *functional equivalence* of reaction systems.

As future work, we plan to revise the compositional semantics by exploiting the idea that transitions can be omitted if they fall within the scope of application of other transitions, and study behavioral equivalences which are suitable in such setting. Finally, as regards its modeling expressiveness, the Reaction Algebra could be extended to include nondeterministic behaviors (similarly to [15]).

References

1. Ehrenfeucht, A., Rozenberg, G.: Reaction Systems. Fundam. Inform. 75(1-4), 263–280 (2007)
2. Brijder, R., Ehrenfeucht, A., Main, M.G., Rozenberg, G.: A tour of reaction systems. Int. J. Found. Comput. Sci. 22(7), 1499–1517 (2011)
3. Păun, G.: Membrane Computing: An Introduction. Natural Computing Series. Springer, GmbH (2002)
4. Ehrenfeucht, A., Rozenberg, G.: Introducing time in reaction systems. Theor. Comput. Sci. 410(4-5), 310–322 (2009)
5. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction Systems with Duration. In: Kelemen, J., Kelemenová, A. (eds.) Păun Festschrift. LNCS, vol. 6610, pp. 191–202. Springer, Heidelberg (2011)
6. Ehrenfeucht, A., Rozenberg, G.: Events and modules in reaction systems. Theor. Comput. Sci. 376(1-2), 3–16 (2007)
7. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Functions Defined by Reaction Systems. Int. J. Found. Comput. Sci. 22(1), 167–178 (2011)
8. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Representing reaction systems by trees. In: Dinneen, M.J., Khoussainov, B., Nies, A. (eds.) WTCS 2012 (Calude Festschrift). LNCS, vol. 7160, pp. 330–342. Springer, Heidelberg (2012)
9. Ehrenfeucht, A., Main, M.G., Rozenberg, G., Brown, A.T.: Stability and Chaos in reaction Systems. Int. J. Found. Comput. Sci. 23(5), 1173–1184 (2012)
10. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Compositional semantics and behavioral equivalences for P Systems. Theor. Comput. Sci. 395(1), 77–100 (2008)
11. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Compositional semantics of spiking neural P systems. J. Log. Algebr. Program. 79(6), 304–316 (2010)
12. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: An Overview on Operational Semantics in Membrane Computing. Int. J. Found. Comput. Sci. 22(1), 119–131 (2011)
13. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: A P Systems Flat Form Preserving Step-by-step Behaviour. Fundam. Inform. 87(1), 1–34 (2008)
14. van Glabbeek, R.: The Linear Time–Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes. In: Handbook of Process Algebra, pp. 3–99. Elsevier (2001)
15. Păun, G., Pérez-Jiménez, M.J.: Towards bridging two cell-inspired models: P systems and R systems. Theor. Comput. Sci. 429(0), 258–264 (2012)

Using Random Graphs in Population Genomics

Laxmi Parida

Computational Genomics Group, IBM T.J. Watson Research Center, U.S.A.
parida@us.ibm.com

I shall discuss the application of algorithmic and combinatorial tools in the area of population genomics, which has not been the traditional stomping ground for algorithmicists.

The modeling of the evolutionary dynamics of evolving populations as random graphs offers a new methodology for analysis. This exploration begins as a quest for understanding the reconstructability of common evolutionary history of populations. It provides new insights including a purely topological (or graph theoretic definition) of traditional population genomic entity like the GMRCA (Grand Most Common Ancestor) of individuals under mutations as well as recombinations. Apart from giving interesting characterizations of another important structure called the ARG (Ancestral Recombinations Graph), it provides the basis for identifying a mathematical minimal nonredundant structure in the ARG and for adapting very naturally the coalescence theory (a well-studied notion in population genetics) in designing ARG sampling algorithms. This connection also opens the door for many interesting questions ranging from human migration paths, to genetic diversity study in plant (cacao) cultivars.

References

1. Utro, F., Pybus, M., Parida, L.: Sum of parts is greater than the whole: inference of common genetic history of populations. BMC Genomics (2013)
2. Utro, F., Cornejo, O.E., Livingstone, D., Motamayor, J.C., Parida, L.: ARG-based Genome-wide Analysis of Cacao Cultivars. BMC Bioinformatics (2012)
3. Javed, A., Pybus, M., Mele, M., Utro, F., Bertranpetit, J., Calafell, F., Parida, L.: IRiS: Construction of ARG networks at genomic scales. Bioinformatics (2011)
4. Javed, A., Mele, M., Pybus, M., Zalloua, P., Haber, M., Comas, D., Netea, M., Balanovsky, O., Balanovska, E., Jin, L., Yang, Y., Arunkumar, G., Pitchappan, R.M., Bertranpetit, J., Calafell, F., Parida, L.: The Genographic Consortium: Recombination networks as genetic markers: a human variation study of the Old World. Human Genetics (2011)
5. Mele, M., Javed, A., Pybus, M., Zalloua, P., Haber, M., Comas, D., Netea, M., Balanovsky, O., Balanovska, E., Jin, L., Yang, Y., Pitchappan, R.M., Arunkumar, G., Parida, L., Calafell, F., Bertranpetit, J.: The Genographic Consortium: The footprint of recombination gives a new insight in the effective population size and the history of the Old World human populations. Molecular Biology and Evolution (2011)

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 340–341, 2013.

6. Parida, L., Palamara, P.F., Javed, A.: A Minimal Descriptor of an Ancestral Recombinations Graph. BMC Bioinformatics 12(Suppl. 1) (2011)
7. Mele, M., Javed, A., Calafell, P.F., Parida, L., Bertranpetit, J.: A New Method to Reconstruct Recombination Events at a Genomic Scale. PLoS Computational Biology (2010)
8. Parida, L.: Ancestral Recombinations Graph: A Reconstructability Perspective using Random-Graphs Framework. Journal of Computational Biology (2010)
9. Parida, L., Javed, A., Mele, M., Calafell, P.F., Bertranpetit, J.: Minimizing Recombinations in Consensus Networks for Phylogeographic Studies. BMC Bioinformatics (2009)
10. Parida, L., Mele, M., Calafell, P.F., Bertranpetit, J.: The Genographic Consortium: Estimating the Ancestral Recombinations Graph (ARG) as Compatible Networks of SNP Patterns. Journal of Computational Biology (2008)

The Tarski-Lindenbaum Algebra of the Class of All Strongly Constructivizable Countable Saturated Models

Mikhail G. Peretyat'kin

Institute of Mathematics and Mathematical Modeling, 125 Pushkin Street, 050010 Almaty, Kazakhstan
m.g.peretyatkin@prericate-logic.org

Abstract. We study the class $S_{s.c}$ of all strongly constructivizable countable saturated models of a finite rich signature σ. We prove that the Tarski-Lindenbaum algebra $\mathcal{L}(S_{s.c})$ considered together with a Gödel numbering γ of the sentences is a Boolean Σ^1_1-algebra whose computable ultrafilters form a dense set in the set of all ultrafilters; moreover, the Boolean algebra $\mathcal{L}(S_{s.c})$ is universal relative to the class of all Boolean Σ^1_1-algebras. This gives an important characterization of the Tarski-Lindenbaum algebra $\mathcal{L}(S_{s.c})$.

Keywords: finitely axiomatizable theory, Tarski-Lindenbaum algebra, c.e. Boolean algebra, countable saturated model, computable family of types, Turing computability, hierarchy, m-complete set.

1 Introduction

The main result presented in Theorem 1 characterizes the Tarski-Lindenbaum algebra of the class $S_{s.c}$ of all strongly constructivizable (i.e., decidable) countable saturated models of a finite rich signature. Analogous result concerning class $P_{s.c}$ of all decidable prime models was obtained in [7]. Mention that, formulation of [7, Theorem 1] should be modified. Its statement must include Parts (a)–(d) similar to those available in Theorem 1 below.

Preliminaries

Theories in first-order predicate logic with equality are considered. General concepts of model theory, algorithm theory, Boolean algebras, and constructive models can be found in Hodges [3], Rogers [8], Goncharov and Ershov [2], and Goncharov [1].

A finite signature is called *rich* if it contains at least one n-ary predicate or function symbol for $n > 1$, or two unary function symbols. By $L(T)$, we denote the Tarski-Lindenbaum algebra of theory T over formulas without free variables, while $\mathcal{L}(T)$ is the Tarski-Lindenbaum algebra $L(T)$ considered together with a Gödel numbering γ; thereby, the concept of a computable isomorphism

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 342–352, 2013.

is applicable to such objects. The set of all finite tuples α of the form $\alpha = \langle \alpha_0, \alpha_1, \ldots, \alpha_n \rangle$, $\alpha_i \in \{0,1\}$, is denoted by $2^{<\omega}$. The empty tuple is denoted by $\varnothing$. A set $\mathfrak{D} \subseteq 2^{<\omega}$ is said to be a *tuple tree* if $\varnothing \in \mathfrak{D}$ and $\alpha \in \mathfrak{D} \Leftrightarrow \alpha 0 \in \mathfrak{D} \vee \alpha 1 \in \mathfrak{D}$ for all $\alpha \in 2^{<\omega}$. The *canonical* (*Gödel*) *index* of a tuple $\varepsilon = \langle \varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{n-1} \rangle$, $\varepsilon_i \in \{0,1\}$, is the number $\mathrm{Nom}(\varepsilon) = 2^n + \varepsilon_0 2^{n-1} + \varepsilon_1 2^{n-2} + \ldots + \varepsilon_{n-1} - 1$. We write shortly $\langle \varepsilon \rangle$ instead of $\mathrm{Nom}(\varepsilon)$. If $\mathcal{B}$ is a Boolean algebra and $a \in \mathcal{B}$, $\mathcal{B}[a]$ stands for the restriction of $\mathcal{B}$ on the set of all subelements of a counting that $\mathbf{1} = a$ and $-x$ is defined as $a \setminus x$ in $\mathcal{B}[a]$. If b is an element of a Boolean algebra and $\alpha \in \{0,1\}$, b^α means b for $\alpha = 1$ and $-b$ for $\alpha = 0$. Similarly, if Φ is a formula and $\alpha \in \{0,1\}$, Φ^α means Φ for $\alpha = 1$ and $\neg\Phi$ for $\alpha = 0$. Definition of the concept of a *compact binary tree* is found in [6, § 2.1] as well as in Preliminaries of [7]. If $\mathcal{D}$ is a compact binary tree, we denote by $\Pi(\mathcal{D})$ the set of all its maximal chains. We also use the following technical notations: $\mathfrak{P}_n$, is the table condition with the Gödel number n, $n \in \mathbb{N}$; $A \models \mathfrak{P}_i$, means that the table condition is satisfied in the set A, $A \subseteq \mathbb{N}$; $\Omega(m) = \{A \subseteq \mathbb{N} \mid (\forall i \in W_m)\, A \models \mathfrak{P}_i\}$, is parametric Stone space with index m; $\mathcal{D}_n^X$ = the closure of W_n^X up to a compact binary tree, where W_n is nth computably enumerable set, while W_n^X denotes computably enumerable set with c.e. index n relative an oracle $X \subseteq \mathbb{N}$, cf. [8].

2 Main Claim

Hereafter, we fix a finite rich signature σ. We denote by $S_{s.c}(\sigma)$ the class of all strongly constructivizable countable saturated models of signature σ.

Theorem 1. *The following assertions hold:*

(a) $\mathcal{L}(S_{s.c}(\sigma))$ *is a Boolean* Σ_1^1*-algebra,*
(b) computable ultrafilters of $\mathcal{L}(S_{s.c}(\sigma))$ *represent a dense set among arbitrary ultrafilters in the algebra,*
(c) for an arbitrary numerated Boolean Σ_1^1*-algebra* $(\mathcal{B}, \nu)$ *with a dense set of computable ultrafilters among arbitrary ultrafilters there is a sentence* Φ *of signature* σ*, such that* $(\mathcal{B}, \nu) \cong (\mathcal{L}\big(\mathrm{Th}(\mathrm{Mod}(\Phi) \cap S_{s.c}(\sigma))\big), \gamma)$*, where* γ *is a Gödel numbering of the sentences of signature* σ*,*
(d) for an arbitrary Boolean Σ_1^1*-algebra* $\mathcal{B}$ *there is a sentence* Φ *of signature* σ*, such that* $\mathcal{B} \cong \mathcal{L}\big(\mathrm{Th}(\mathrm{Mod}(\Phi) \cap S_{s.c}(\sigma))\big)$*.*

Proof. By criterion of Morley [5], also independently obtained by Millar in [4], a countable saturated model $\mathfrak{M}$ of a complete decidable theory T is strongly constructivizable if and only if the family of types realized in $\mathfrak{M}$ is computable. From this we obtain that a sentence Ψ has a strongly constructivizable countable saturated model if and only if there is a complete decidable theory T with a computable set $\mathfrak{T}$ of its types such that each type of T belongs to $\mathfrak{T}$; moreover, $\Psi \in T$. An immediate calculation gives prefix $\forall^1$. Finally, sentences Φ and Ψ are equivalent on the class $S_{s.c}(\sigma)$ of all strongly constructivizable countable saturated models if and only if $(\Phi \,\&\, \neg\Psi) \vee (\Psi \,\&\, \neg\Phi)$ does not have a model in this class. This gives prefix $\exists^1$ for (a).

Let T be an arbitrary complete theory extending $\mathrm{Th}(S_{s.c}(\sigma))$, and $\Psi \in T$. Obviously, Ψ has a model $\mathfrak{M} \in S_{s.c}(\sigma)$. From this we have that complete decidable theory $T' = \mathrm{Th}(\mathfrak{M})$ presenting a computable ultrafilter in $\mathrm{St}(\mathrm{Th}(S_{s.c}(\sigma)))$, is found in the neighborhood Ψ of the given arbitrary ultrafilter T of this Stone space. This gives the required density property posed in (b). As for Part (d), it is a simple consequence of Part (c) and Lemma 6 with $\Xi = \Sigma^1_1$.

The proof of Part (c) is given in Section 4.

3 A Property Concerning Numerated Boolean Algebras

The following assertion is intended to be used in proof of Part (c) of Theorem 1. Given a set $X \subseteq \mathbb{N}$ that is considered as an oracle.

Lemma 1. *For an arbitrary numerated Boolean Σ^X_1-algebra $(\mathcal{B}, \nu)$, there is a numeration υ of $\mathcal{B}$ such that $(\mathcal{B}, \upsilon)$ is a Boolean Σ^X_1-algebra whose computable ultrafilters form a dense set in the set of all ultrafilters of the algebra $(\mathcal{B}, \upsilon)$.*

Proof. Given a numerated Boolean Σ^X_1-algebra $(\mathcal{B}, \nu)$ defined by relativized computability with an oracle $X \subseteq \mathbb{N}$. We assume, that $\mathcal{B}$ is a nontrivial algebra. By definition, signature operations $\cup$, $\cap$, and $-$ in $\mathcal{B}$ are presentable by computable functions on ν-numbers, while the equality is a Σ^X_1-relation in numeration ν. This means that there is a computably enumerable in X set $B \subseteq 2^{<\omega}$ such that the following relation holds for any α in $2^{<\omega}$, $\alpha = \langle \alpha_0, ..., \alpha_k \rangle$:

$$\nu(0)^{\alpha_0} \cap ... \cap \nu(k)^{\alpha_k} = \mathbf{0} \;\Leftrightarrow\; (\exists \beta \in B)(\beta \preccurlyeq \alpha). \tag{3.1}$$

It is easily seen that the right-hand side expression in (3.1) represents a Σ^X_1-form. Equivalently, the latter relation can be transformed into a Π^X_1-form as follows for all α in $2^{<\omega}$:

$$\nu(0)^{\alpha_0} \cap ... \cap \nu(k)^{\alpha_k} \neq \mathbf{0} \;\Leftrightarrow\; (\forall \beta \in B)(\beta \not\preccurlyeq \alpha). \tag{3.2}$$

The case is trivial when B is finite in (3.1) and (3.2). Therefore, we assume that B is infinite. Let B^t be a finite part of B, which is enumerated for $\leqslant t$ steps in the computation with oracle X. We can assume that $B_0 = \varnothing$ and

$$|B^{t+1} \setminus B^t| = 1, \quad \textit{for all } t \in \mathbb{N}. \tag{3.3}$$

Denote

$$\begin{aligned} &\texttt{(a)}\ \mathcal{D}^t = \{\alpha \in 2^{<\omega} \mid (\forall \beta \in B^t)(\beta \not\preccurlyeq \alpha)\}, \\ &\texttt{(b)}\ \mathcal{D} = \textstyle\bigcap_{t \in \mathbb{N}} \mathcal{D}^t = \{\alpha \in 2^{<\omega} \mid (\forall \beta \in B)(\beta \not\preccurlyeq \alpha)\}. \end{aligned} \tag{3.4}$$

Since the set $\mathcal{D}$ represents nonzero expressions in the left-hand side part of (3.2), we obtain that $\mathcal{D}$ is a tuple tree; particularly, we have

$$\varnothing \in \mathcal{D}^t \quad \textit{for all } t \in \mathbb{N}. \tag{3.5}$$

We can assume that the enumeration of B in the computation with oracle X is organized in such a manner that the following condition is valid for all $t \in \mathbb{N}$:

$$\alpha \in \mathcal{D}^t \;\Leftrightarrow\; \alpha 0 \in \mathcal{D}^t \vee \alpha 1 \in \mathcal{D}^t, \quad \textit{for all } \alpha \in 2^{<\omega}, \tag{3.6}$$

i.e., each set $\mathcal{D}^t$ itself is a tuple tree.

Now, we are going to describe a step-by-step process of computation with oracle X whose result will be a new numeration υ of the same algebra $\mathcal{B}$. The following objects and relations are involved in the construction:

$$\begin{aligned}
&\texttt{(a)}\ q^t : \mathcal{D}^t \to 2^{<\omega},\ \textit{is a mapping satisfying}: \\
&\texttt{(b)}\ q^t(\varnothing) = \varnothing, \\
&\texttt{(c)}\ \alpha \preccurlyeq \beta \;\Leftrightarrow\; q^t(\alpha) \preccurlyeq q^t(\beta), \\
&\texttt{(d)}\ Q^t = \left\{\alpha \in 2^{<\omega} \;:\; |q^t(\alpha u)| > |\alpha| + 1 \textit{ for some } u \in \{0,1\}\right\} \textit{ is finite}, \\
&\texttt{(e)}\ (\forall \alpha \in 2^{<\omega})(\forall u \in \{0,1\}) \left[\, \alpha \in \mathcal{D}^t \setminus Q^t \;\Rightarrow q^t(\alpha u) = q^t(\alpha) u \,\right].
\end{aligned} \tag{3.7}$$

Notice that, having a pair of Gödel numbers for B^t and Q^t together with the values of $q^t(\alpha)$ for α in Q^t, we can uniquely determine all other parts involved in the listing (3.7) obtaining an effective calculation of all relations and inclusions needed in the construction.

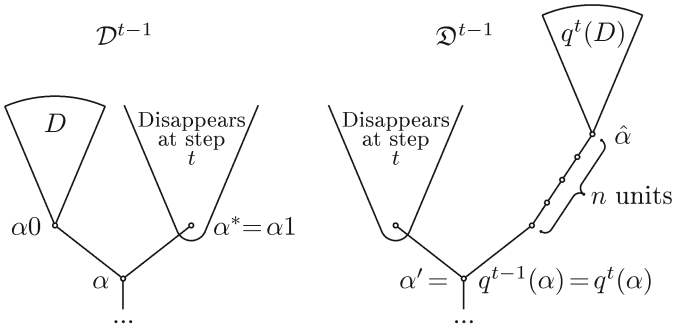


Fig. 1. General scheme of step $t > 0$ of the construction

Construction. Step $t = 0$. We put $q^0(x)$ to be the identical mapping $\mathrm{id} : 2^{<\omega} \to 2^{<\omega}$; furthermore, we put $Q^0 = \varnothing$.

Step $t > 0$. Consider the unique tuple $\alpha^* \in B^t \setminus B^{t-1}$. In the case when α^* has the form $\alpha 1$ for some α satisfying $\alpha 0 \in \mathcal{D}^{t-1}$, we find the least $n > 0$ such that there are no elements of B^t above the element

$$\hat{\alpha} = q^{t-1}(\alpha)\, \underbrace{11\ldots1}_{n \text{ times}}.$$

After that, we put for all δ in $2^{<\omega}$:

$$q^t(\delta) = \begin{cases} q^{t-1}(\delta), & \textit{if } \delta \not\succ \alpha,\ \delta \in \mathcal{D}^{t-1}, \\ \hat{\alpha}\beta, & \textit{if } \delta = \alpha 0 \beta,\ \beta \in 2^{<\omega},\ \textit{and}\ \ \alpha 0 \beta \in \mathcal{D}^{t-1}. \end{cases} \tag{3.8}$$

Notice that, we do not need to define values of $q^t(x)$ in the region $\{\delta \mid \alpha^* \preccurlyeq \delta\}$ since it disappears from the tree $\mathcal{D}^t$ at step t, cf. Fig. 1.

In the other case when either α^* has the form $\alpha 0$ for some α, or α^* is $\alpha 1$ with $\alpha 0 \notin \mathcal{D}^{t-1}$, we do nothing putting $q^t(x) = q^{t-1}(x)$ for all $x \in \mathcal{D}^t$, where, by definition, $\mathcal{D}^t = \mathcal{D}^{t-1} \setminus \{\delta \mid \alpha^* \preccurlyeq \delta\}$.

Notice that, in any case, we have $\mathcal{D}^t = \mathcal{D}^{t-1} \cap \{\beta \mid \alpha^* \not\preccurlyeq \beta\}$, since, by construction, the tuple α^* is a new "restricting" element from B occurred at step t. This finishes the construction.

Let us introduce the following notation:

$$\mathfrak{D}^t =_{dfn} q^t(\mathcal{D}^t) \cup (\textit{related elements marked as "n units"}), \tag{3.9}$$

$$\mathfrak{D} = \bigcap_{t \in \mathbb{N}} \mathfrak{D}^t. \tag{3.10}$$

The following simple properties hold:

$$\begin{aligned} &\texttt{(a)}\ (\forall \alpha \in \mathcal{D})(\exists t_0)(\forall t \geqslant t_0)\ \big[\ q^t(\alpha) = q^{t_0}(\alpha)\ \big], \\ &\texttt{(b)}\ (\forall \alpha \in 2^{<\omega})\ \big[\ \alpha \in \mathfrak{D}^t\ \Rightarrow\ \alpha 1 \in \mathfrak{D}^t\ \big], \\ &\texttt{(c)}\ (\forall \alpha \in 2^{<\omega})\ \big[\ \alpha \in \mathfrak{D}\ \Rightarrow\ \alpha 1 \in \mathfrak{D}\ \big]. \end{aligned} \tag{3.11}$$

Indeed, because of (3.3), for a fixed α in $\mathcal{D}$, for some t_0, the length of the element α^* from $B^t \setminus B^{t-1}$ becomes greater $|\alpha|$ for all $t \geqslant t_0$. Considering our construction, cf. Fig. 1, we can see that (3.11)(a) is satisfied. Furthermore, by construction, we have $B_0 = \varnothing$ and $Q^0 = \varnothing$ obtaining (3.11)(b) at step $t = 0$. Now, suppose that (3.11)(b) holds at step $t-1$. Analyzing the action performed by the construction at step t, we can easily check that (3.11)(b) is still to be valid at the step t. As for Part (c) of (3.11), it is a consequence of (3.11)(b) together with (3.10).

Thereby, all parts of (3.11) are checked. Based on (3.11)(a), we define the following limit function on tuples α in $\mathcal{D}$:

$$q(\alpha) = \beta\ \Leftrightarrow\ (\exists t_0)(\forall t \geqslant t_0)\ \big[\ q^t(\alpha) = \beta\ \big]. \tag{3.12}$$

By construction, we have

$$\mathrm{Dom}(q) = \mathcal{D},\ \ \textit{and}\ \ \mathrm{Val}(q) \subseteq \mathfrak{D}; \tag{3.13}$$

moreover, the mapping $q : \mathcal{D} \to \mathfrak{D}$ satisfies the following property for all $\alpha, \beta \in \mathcal{D}$:

$$\alpha \preccurlyeq \beta\ \Leftrightarrow\ q(\alpha) \preccurlyeq q(\beta); \tag{3.14}$$

in addition, the following technical property is provided by the construction:

$$\alpha \in \mathfrak{D} \setminus \mathrm{Val}(q)\ \Leftrightarrow\ \begin{array}{l}\alpha \textit{ is located at a non-brabching chain of "n units",} \\ \textit{cf. Fig. 1, excepting the top element of the chain.}\end{array} \tag{3.15}$$

Now, we turn to the main properties of the construction.

Claim 2. *For each $t > 0$, there exists a finite set $C^t \subseteq 2^{<\omega}$ such that we have*

$$\mathfrak{D}^t = \mathfrak{D}^{t-1} \cap \{\alpha \in 2^{<\omega} \mid (\forall \beta \in C^t)(\beta \not\preccurlyeq \alpha)\};$$

moreover, a Gödel number of the set C^t is found effectively from the complex of information available at step t of the construction (an exact form of the statement: the finite set C^t is found effectively in a Gödel number of the chain of finite sets $\varnothing = B^0 \subseteq B^1 \subseteq ... \subseteq B^t$).

Proof. First, we consider the case when the action is performed, as shown in Fig. 1. Consider the following sets of tuples (see position of α' in Fig. 1):

$$\begin{aligned} &C^t = C_0^t \cup C_1^t \cup C_2^t, \quad \textit{where} \\ &C_0^t = \{\alpha' 0\}, \quad \alpha' = q^{t-1}(\alpha), \\ &C_1^t = \{\alpha' 10\,,\ \alpha' 110\,,\ \ldots, \alpha' \underbrace{11..1}_{n-1 \text{ times}} 0\}, \\ &C_2^t = \{q^t(\beta) \mid \alpha 0 \preccurlyeq \beta \textit{ and } \beta \in B^t\}. \end{aligned}$$

The set C_t^0 restricts the region $\{\beta \mid \alpha' 0 \preccurlyeq \beta\}$ in $\mathfrak{D}$, the set C_1^t restricts outside parts along the line of "n units", while C_2^t represents the q^t-image of restricting elements available in the region D of tree $\mathcal{D}$ at moment $t-1$, thus, mapped to the region $q^t(D)$ of tree $\mathfrak{D}$ at moment t, see Fig. 1. In the other case when either α^* has the form $\alpha 0$ for some α, or α^* is $\alpha 1$ with $\alpha 0 \notin \mathcal{D}^{t-1}$, we simply put $C^t = \{q^{t-1}(\alpha^*)\}$ for the unique tuple $\alpha^* \in B^t \setminus B^{t-1}$.

It is easily checked that, in both cases, Claim 2 holds.

Claim 3. *Tree $\mathfrak{D}$ is a Π_1^X-set.*

Proof. Consider the set $C = \bigcup_{t \in \mathbb{N}} C^t$, where finite sets C^t are defined in Claim 2. By virtue of Claim 2, the set C must be a Σ_1^X-set. Furthermore, we have $\alpha \notin \mathfrak{D} \ \Leftrightarrow\ (\exists \beta \in C)\left[\ \beta \preccurlyeq \alpha\ \right]$ for all α in $2^{<\omega}$ obtaining that the complement of $\mathfrak{D}$ is a Σ_1^X-set; thus, $\mathfrak{D}$ itself is a Π_1^X-set.

Introduce the following notation for an arbitrary chain π in $\Pi(\mathcal{D})$:

$$q(\pi) = \left\{\ q(\alpha) \mid \alpha \in \pi\ \right\} \cup (\textit{related elements marked as "n units"}). \qquad (3.16)$$

Claim 4. *The mapping q defined by rule* (3.16) *determines a bijection $q : \Pi(\mathcal{D}) \to \Pi(\mathfrak{D})$ between the families of chains in these trees.*

Proof. According to the construction, at each step, we cut corresponding chains out of the trees $\mathcal{D}$ and $\mathfrak{D}$ sometimes relocating some chains of $\mathfrak{D}$ in another place. In the case of the action presented in Fig. 1, the set of chains passing through $\alpha' 0$ in $\mathfrak{D}$ is moved to another place; namely, it becomes passing through $\hat{\alpha}$ in $\mathfrak{D}$. Moreover, the interval of "n units" appeared at the step adds none extra chains. Based on the fact that stabilization (3.11)(a) takes place, we obtain that the limited function $q(x)$ indeed bijectively maps $\Pi(\mathcal{D})$ onto $\Pi(\mathfrak{D})$.

Now, we are in a position to construct a new numeration υ of $\mathcal{B}$. Consider two formal systems of generating elements b_α and d_α, $\alpha \in 2^{<\omega}$, for Boolean algebras. Introduce the following formal dependence relations for these sets:

$$\begin{array}{ll} \texttt{(a)} & \texttt{(b)} \\ b_\varnothing \neq \mathbf{0}, & d_\varnothing \neq \mathbf{0}, \\ b_\alpha = b_{\alpha 0} \cup b_{\alpha 1}, & d_\alpha = d_{\alpha 0} \cup d_{\alpha 1}, \\ b_{\alpha 0} \cap b_{\alpha 1} = \mathbf{0}, & d_{\alpha 0} \cap d_{\alpha 1} = \mathbf{0}, \\ b_\alpha \neq \mathbf{0},\ \textit{for } \alpha \in \mathcal{D}, & d_\alpha \neq \mathbf{0},\ \textit{for } \alpha \in \mathfrak{D}, \\ b_\alpha = \mathbf{0},\ \textit{for } \alpha \in 2^{<\omega} \setminus \mathcal{D}; & d_\alpha = \mathbf{0},\ \textit{for } \alpha \in 2^{<\omega} \setminus \mathfrak{D}. \end{array} \tag{3.17}$$

In view of (3.1) and (3.4)(b), system (3.17)(a) represents the algebra $(\mathcal{B}, \nu)$ in accordance with the following interpretation

$$b_\alpha = \nu(0)^{\alpha_0} \cap ... \cap \nu(k)^{\alpha_k}, \quad \alpha = \langle \alpha_0, ..., \alpha_k \rangle.$$

Claim 5. *System* (3.17)(*b*) *represents a countable Boolean algebra* $\mathcal{B}'$ *which is isomorphic to* $\mathcal{B}$.

Proof. For the formal dependence relations (3.17)(a) and (3.17)(b), there is a branch-preserving embedding q of $\mathcal{D}$ into $\mathfrak{D}$ satisfying the properties (3.13), (3.14), and (3.15). Moreover, by Claim 4, we have a bijective correspondence between the sets of infinite chains of these tuple trees. This is sufficient for existence of an isomorphism between the algebras $\mathcal{B}$ and $\mathcal{B}'$.

Now, we turn immediately to the proof of Lemma 1.

Let $\mathcal{B}'$ be the Boolean algebra uniquely determined by the system of generating relations (3.17)(b). Define a mapping $\tau : \mathbb{N} \to |\mathcal{B}'|$ by the following rule for all $k \in \mathbb{N}$:

$$\begin{array}{l} \tau(k) = d_\alpha \cup d_\beta \cup ... \cup d_\delta, \quad \textit{where} \\ k = \mathrm{Nom}\,\langle j_1, j_2, ..., j_t \rangle, \\ j_1 = \mathrm{Nom}(\alpha),\ j_2 = \mathrm{Nom}(\beta),\ ...\ ,\ j_t = \mathrm{Nom}(\delta). \end{array} \tag{3.18}$$

It can be easily checked that each element a in $\mathcal{B}'$ is presentable in the form (3.18); thus, τ is a numeration of the algebra $\mathcal{B}'$. Since the operations in $\mathcal{B}'$ are performed via some formal manipulations over the terms of the form (3.18), there are general computable functions presenting all Boolean operations relative to the numeration τ. Finally, by (3.17)(b) together with Claim 3, the condition $d_\alpha = \mathbf{0}$ is Σ_1^X showing that $(\mathcal{B}', \tau)$ is a numerated Boolean Σ_1^X-algebra.

Now, we check the condition of density of the set of all computable ultrafilters in $(\mathcal{B}', \tau)$. Let a be a nonempty element in $\mathcal{B}'$. From (3.18), we can find a nonzero element $d_\beta \subseteq a$ for some β in $\mathfrak{D}$, $\beta = \langle \beta_0 ... \beta_k \rangle$. Consider an infinite sequence of the form $\beta^* = \langle \beta_0\ , ...,\ \beta_k, 1, 1, ..., 1, ... \rangle$. By virtue of (3.11)(c), for an arbitrary initial finite segment δ of the tuple β^*, $\delta = \langle \delta_0 ... \delta_t \rangle$, we have $\tau(0)^{\delta_0} \cap ... \cap \tau(t)^{\delta_t} \neq \mathbf{0}$. Therefore, the infinite tuple β^* determines an ultrafilter

in the given neighborhood a of $\mathcal{B}'$. By construction, we obtain that this ultrafilter is computable. Thereby, the set of all computable ultrafilters is indeed dense in the set of all ultrafilters of the numerated Boolean algebra $(\mathcal{B}', \tau)$.

Let $\lambda : \mathcal{B}' \to \mathcal{B}$ be the isomorphism provided by Claim 5. We define a new numeration υ of the algebra $\mathcal{B}$ required to Lemma 1 by the following rule $\upsilon(n) = \lambda(\tau(n))$, for all $n \in \mathbb{N}$. We obtain a numerated Boolean algebra $(\mathcal{B}, \upsilon)$ that satisfies all the demands posed in the formulation of Lemma 1.

Lemma 1 is proven.

Lemma 6. *Given a class of hierarchy $\Xi \in \{\Sigma^0_n, \Sigma^1_n, \Pi^1_n, \Delta^1_n\}$ with $0 < n < \omega$. For an arbitrary numerated Boolean Ξ-algebra $(\mathcal{B}, \nu)$, there is a numeration υ of $\mathcal{B}$ such that $(\mathcal{B}, \upsilon)$ is a Boolean Ξ-algebra whose computable ultrafilters form a dense set in the set of all ultrafilters of the algebra $(\mathcal{B}, \upsilon)$.*

Proof. The case $\Xi = \Sigma^0_n$ is a consequence of Lemma 1 with $X = \varnothing^{(n-1)}$. The other cases are provided by simple properties of the hierarchy together with the fact that the set C in the proof of Claim 3 is Δ^X_1 whenever B in (3.1) is Δ^X_1.

4 Proof to Part (c) of Theorem 1

Given a numerated Boolean Σ^1_1-algebra $(\mathcal{B}, \nu)$ whose computable ultrafilters represent a dense set in the set of all ultrafilters. We assume, that $\mathcal{B}$ is a nontrivial algebra. By definition, signature operations $\cup$, $\cap$ and $-$ in $\mathcal{B}$ are presentable by computable functions on ν-numbers, and the equality relation is a Σ^1_1-relation in numeration ν. Therefore, there exists a unary relation H^* such that for any finite tuple of zeros and ones $\alpha = \langle \alpha_0, \alpha_1, \ldots, \alpha_n \rangle$, we have

$$\nu(0)^{\alpha_0} \cap \nu(1)^{\alpha_1} \cap \ldots \cap \nu(n)^{\alpha_n} = \mathbf{0} \;\Leftrightarrow\; \langle \alpha_0, \alpha_1, \ldots, \alpha_n \rangle \in H^*, \quad H^* \in \Sigma^1_1.$$

We shall use the following m-complete in class Π^1_1 set, [8, Ch.16,Cor.XX(b)]:

$$W = \{n \mid \varphi^{(2)}_n \textit{ represents a well-ordering on a set of natural numbers}\}.$$

where $\varphi^{(2)}_n$ is nth partially computable function with two arguments in Kleene's numbering, [8]. Since H^* is Σ^1_1, and thus, $H^* \leqslant_m \mathbb{N} \setminus W$, there is a computable function $f(x)$ satisfying the following properties for all $\alpha \in 2^{<\omega}$, $\alpha = \langle \alpha_0, \ldots, \alpha_n \rangle$:

$$\nu(0)^{\alpha_0} \cap \nu(1)^{\alpha_1} \cap \ldots \cap \nu(n)^{\alpha_n} \neq \mathbf{0} \;\Leftrightarrow\; f\big(\langle \alpha_0, \alpha_1, \ldots, \alpha_n \rangle\big) \in W. \tag{4.1}$$

Using the method of [6, Sec.2.3, p.44], we construct an effective sequence of constructive linear orders $(\mathcal{L}_s, \nu_s)$, $s \in \mathbb{N}$, so that the following relation is satisfied:

$$s \in W \Leftrightarrow \mathcal{L}_s \textit{ is a well ordering.} \tag{4.2}$$

Now, our goal is to choose some pair (m, s) of integer parameters.

Choice of m. We choose m such that $\Omega(m) = \mathcal{P}(\mathbb{N})$ (cf. Preliminaries). For this purpose, it is enough to take m such that $W_m = \varnothing$.

Choice of s. For this purpose, we describe a computable functional Ψ from $\mathcal{P}(\mathbb{N})$ to $\mathcal{P}(\mathbb{N})$. Given a set $A \subseteq \mathbb{N}$. Let $\alpha = \langle \alpha_0, \alpha_1, ..., \alpha_k, ... \rangle$ be the characteristic sequence for A, i.e., the following is satisfied:

$$\alpha_k = \begin{cases} 1, \textit{ if } k \in A, \\ 0, \textit{ if } k \notin A. \end{cases} \tag{4.3}$$

Taking A as an input parameter, we construct the following set

$$\Psi(A) = \mathcal{D}^\circ = \mathrm{Tree}(\mathcal{L}^\circ), \quad \mathcal{L}^\circ = \mathcal{L}_{f(\varnothing)} + \mathcal{L}_{f(\langle \alpha_0 \rangle)} + \mathcal{L}_{f(\langle \alpha_0, \alpha_1 \rangle)} + ..., \tag{4.4}$$

where $\mathcal{L} \mapsto \mathrm{Tree}(\mathcal{L})$ is an effective procedure performing the transformation from a constructive linear order to a c.e. binary tree, as it is described in [6, Sec.2.2]. It can easily be checked that the described transformation $A \mapsto \Psi(A)$ is realized by an algorithm using the set $A \subseteq \mathbb{N}$ as its input parameter. Thereby, the transformation $A \mapsto \Psi(A)$ can be considered as a computation by an algorithm $\mathcal{M}$ with an oracle A. Let s be a Gödel number of the algorithm $\mathcal{M}$. In accordance with the basic definitions of the theory of algorithms, we obtain the following form of the operator Ψ defined by an index $s \in \mathbb{N}$ (see Preliminaries):

$$\Psi(A) = \mathcal{D}_s^A. \tag{4.5}$$

On this, choice of the pair of parameters (m, s) is finished.

Lemma 7. *Let $A \subseteq \mathbb{N}$ be a computable set. The tree $\mathcal{D}^\circ = \mathrm{Tree}(\mathcal{L}^\circ)$ is superatomic and the family of chains $\Pi(\mathcal{D}^\circ)$ is computable if and only if the linear order $\mathcal{L}^\circ$ in (4.4) is a well ordering.*

Proof. Immediate; based on properties of the operation $\mathcal{L} \mapsto \mathrm{Tree}(\mathcal{L})$ formulated in [6, Lemma 2.2.1].

Now we immediately pass to the proof of Part (c) of Theorem 1.

First of all, we have to point out a sentence Φ of the given finite rich signature σ, as it is stated in Part (c) of Theorem 1. For this, we use the canonical construction, cf. [6, Theorem 3.1.1]. Apply this construction to the pair (m, s) specifying also signature σ. As a result, we obtain a finitely axiomatizable theory $F = \mathbb{FC}(m, s, \sigma)$ of signature σ. As Φ, we get a conjunction of axioms of the theory F. After that, our principal aim is to show that the sentence Φ satisfies all the requirements listed in Theorem 1 (c).

By main statement of the canonical construction, there is an effective sequence of sentences θ_n, $n \in \mathbb{N}$, of signature σ such that the family of all complete extensions of F is presented in the form

$$F[A] = F \cup \{\theta_i | i \in A\} \cup \{\neg\theta_j | j \in \mathbb{N} \setminus A\}, \ A \in \Omega(m), \tag{4.6}$$

moreover, the following relations are held for any $A \in \Omega(m)$:

$$F[A] \textit{ has a countable saturated model } \Leftrightarrow \textit{ } \mathcal{D}_s^A \textit{ is superatomic}, \tag{4.7}$$

$$\textit{a countable saturated model of } F[A], \textit{ if it exists, is strongly constructivizable } \Leftrightarrow A \textit{ is computable and } \Pi(\mathcal{D}_s^A) \textit{ is computable.} \tag{4.8}$$

Consider an arbitrary finite tuple of zeros and ones $\alpha = \langle \alpha_0, ..., \alpha_k \rangle$. Construct an elementary intersection of elements in $\mathcal{B}$ by the rule

$$b_\alpha = \nu(0)^{\alpha_0} \cap \nu(1)^{\alpha_1} \cap ... \cap \nu(k)^{\alpha_k}, \tag{4.9}$$

as well as an elementary conjunction of corresponding sentences by the rule

$$\beta_\alpha = \theta_0^{\alpha_0} \,\&\, \theta_1^{\alpha_1} \,\&\, ... \,\&\, \theta_k^{\alpha_k} \,. \tag{4.10}$$

The main idea behind the construction is to provide the following relation:

Lemma 8. *For any tuple $\alpha \in 2^{<\omega}$, $b_\alpha \neq \mathbf{0}$ if and only if $\Phi \,\&\, \beta_\alpha$ has a strongly constructivizable countable saturated model.*

Proof. Assume that $b_\alpha \neq \mathbf{0}$. Since computable ultrafilters form a dense set among arbitrary ultrafilters in the Boolean algebra $(\mathcal{B}, \nu)$, there is an infinite sequence $\alpha^* = \langle \alpha_i \mid i < \omega \rangle$ extending α such that the set A related to α^* by (4.3) is computable, and

$$\nu(0)^{\alpha_0} \cap ... \cap \nu(i)^{\alpha_i} \neq \mathbf{0}, \quad \textit{for all} \;\; i \in \mathbb{N}. \tag{4.11}$$

By (4.1), we obtain that $f(\langle \alpha_0, ..., \alpha_s \rangle) \in W$ for all $i \in \mathbb{N}$; thus, by (4.2), the sequence of orders in (4.4) consists of only well orderings. By Lemma 7, the tree $\Psi(A) = \mathcal{D}_s^A$ is superatomic and the family of chains $\Pi(\mathcal{D}_s^A)$ is computable. By main statement of the canonical construction providing (4.7) and (4.8), theory $F[A]$ is consistent, complete, and has a countable saturated model $\mathfrak{M}$, which is strongly constructivizable. This ensures that the formula $\Phi \,\&\, \beta_\alpha$ is satisfied in the strongly constructivizable model $\mathfrak{M}$ since it is provable from $F[A]$.

Now, we assume that the sentence $\Phi \,\&\, \beta_\alpha$ has a strongly constructivizable countable saturated model $\mathfrak{M}$. Consider the set

$$A = \{\theta_i \mid \mathfrak{M} \models \theta_i\}, \tag{4.12}$$

which is obviously computable. Build an infinite sequence $\alpha^* = \langle \alpha_i \mid i < \omega \rangle$ related to A by (4.3). Since $A \in \Omega(m)$, theory $F[A]$ is consistent and complete. Moreover, this theory is decidable by Janiczak theorem since it is computably axiomatizable. By (4.12), all axioms of $F[A]$ are satisfied in the model $\mathfrak{M}$. Thereby, we have that A is computable and $F[A]$ has a strongly constructivizable countable saturated model. By (4.7) and (4.8), we conclude that the tree $\mathcal{D}_s^A$ is superatomic and the family $\Pi(\mathcal{D}_s^A)$ is computable. By Lemma 7, linear order $\mathcal{L}^\circ$ in (4.4) is well ordering; in view of (4.2), we obtain that $f(\langle \alpha_0, ..., \alpha_s \rangle) \in W$ for all $s \in \mathbb{N}$. Applying (4.1), we finally obtain $b_\alpha \neq \mathbf{0}$.

Lemma 8 is proven.

Let us map elements $\nu(i)$, $i \in \mathbb{N}$, of $\mathcal{B}$ to sentences θ_i, $i \in \mathbb{N}$, by the rule:

$$\lambda^*(\nu(k)) = \theta_k, \quad k \in \mathbb{N}. \tag{4.13}$$

Now, we shall extend the partial mapping (4.13) up to a computable isomorphism of the algebras under consideration. Define a mapping

$$\lambda : \mathcal{B} \to \mathcal{L}\big(\mathrm{Th}\big(\mathrm{Mod}(\Phi) \cap S_{s.c}(\sigma)\big)\big) \tag{4.14}$$

by the following rule: for an arbitrary finite sequence of finite binary tuples $\alpha_0, \alpha_1, \ldots \alpha_n \in 2^{<\omega}$, we put

$$\lambda(b_{\alpha_0} \cup b_{\alpha_1} \cup \ldots \cup b_{\alpha_n}) = \beta_{\alpha_0} \cup \beta_{\alpha_1} \cup \ldots \cup \beta_{\alpha_n}. \tag{4.15}$$

The mapping λ is defined on all elements of $\mathcal{B}$ and is surjective since the set of expressions involved in (4.15) includes all elements of these algebras. Taking into consideration the property stated in Lemma 8 together with the fact that Boolean operations above unions of elements b_α of the form (4.9) and disjunctions of elements β_α of the form (4.10) are produced by same rules, we obtain finally a computable isomorphism: $\lambda : (\mathcal{B}, \nu) \to \big(\mathcal{L}\big(\mathrm{Th}(\mathrm{Mod}(\Phi) \cap S_{s.c}(\sigma))\big), \gamma\big)$.

Thereby, Part (c) of Theorem 1 is completely proven.

References

1. Goncharov, S.S.: Countable Boolean Algebras and Decidability, Plenum (1997)
2. Goncharov, S.S., Ershov, Y.L.: Constructive models. Plenum (1999)
3. Hodges, W.: A shorter model theory. Cambridge University Press, Cambridge (1997)
4. Millar, T.S.: Foundation of recursive model theory. Annals of Mathematical Logic 13, 45–72 (1978)
5. Morley, M.: Decidable models. Israel J. Math. 25(3–4), 233–240 (1976)
6. Peretyat'kin, M.G.: Finitely axiomatizable theories. Plenum, New York (1997)
7. Peretyat'kin, M.G.: On the Tarski-Lindenbaum algebra of the class of all strongly constructivizable prime models. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 590–599. Springer, Heidelberg (2012)
8. Rogers, H.J.: Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Co., New York (1967)

The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words

Giovanna Rosone and Marinella Sciortino

Dipartimento di Matematica e Informatica, University of Palermo
Via Archirafi, 34, 90123, Palermo, Italy
{giovanna,mari}@math.unipa.it

Abstract. The Burrows-Wheeler Transform (BWT) is a tool of fundamental importance in Data Compression and, recently, has found many applications well beyond its original purpose. The main goal of this paper is to highlight the mathematical and combinatorial properties on which the outstanding versatility of the BWT is based, i.e., its *reversibility* and the *clustering effect* on the output. Such properties have aroused curiosity and fervent interest in the scientific world both for theoretical aspects and for practical effects. In particular, in this paper we are interested both to survey the theoretical research issues which, by taking their cue from Data Compression, have been developed in the context of Combinatorics on Words, and to focus on those combinatorial results useful to explore the applicative potential of the Burrows-Wheeler Transform.

1 Introduction

Michael Burrows and David Wheeler introduced in 1994 a transformation, called BWT, that permutes the letters of a text in according to the sorting of its cyclic rotations, making the text more compressible (cf. [4]). In the last two decades, the BWT has become a tool of fundamental importance in Data Compression and, recently, has found many applications well beyond its original purpose (cf. [16,17,32,42,7,1]). Several implementations in external memory (cf. [14,2]) make the BWT also applicable to massive datasets.

The main goal of this paper is to highlight the mathematical and combinatorial properties that make the BWT a so versatile tool, i.e., its *reversibility* and the *clustering effect* on the produced output. Reversibility is guaranteed by the sorting of the cyclic rotations of the text performed by the BWT and it is a key feature for lossless compression. The clustering effect is due to the fact that the BWT tends to group together characters that occur in similar contexts in the input text, making the output more compressible even by simple compressors. Theoretical evaluations of the performance of the BWT-based compressors can be found in [35,12,5]. The mathematical and combinatorial properties of the BWT have aroused curiosity and fervent interest in the scientific world both for theoretical aspects and for practical effects. Several issues have been developed in the context of Combinatorics on Words. In particular, some studies have been carried on determining the words that are the output of the BWT

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 353–364, 2013.

(cf. [28]) and, more widely, many researches have been pursued on characterizing the words that become the most compressible after the application of the BWT (cf. [33,40,39,41,38,13]). Actually, the practical importance of the clustering effect led to two different research themes. From one hand, to consider the question of defining a combinatorial measure on words able to provide a quantitative estimate of the "degree of clustering" that the BWT could produce (cf. [40]). On the other hand, to investigate the problem of the optimal partitioning of the BWT output so that compressing individually its parts via a base-compressor gets a compressed output that is more compact than applying the compressor over the entire text at once (cf. [15,19]). The use of these results allows the definition of particularly efficient BWT-based compressors.

The combinatorial properties of BWT are also maintained when the transformation is extended to a multiset of words (cf. [31]). Such an extension has been used to define measures for comparing sequences (cf. [32,45,7]) and as preprocessing step of compressors that work on large collections of sequences (cf. [6]).

2 The Burrows-Wheeler Transform

Let Σ be a finite ordered alphabet. We denote by Σ^* the set of words over Σ. Given a finite word $w = w_1 w_2 \cdots w_n \in \Sigma^*$ with each $w_i \in \Sigma$, the length of w, denoted $|w|$, is equal to n.

We say that two words $x, y \in \Sigma^*$ are *conjugate*, if $x = uv$ and $y = vu$, where $u, v \in \Sigma^*$. Conjugacy between words is an equivalence relation over Σ^*. The *conjugacy class* $[w]$ of $w \in \Sigma^n$ is the set of all words $w_i w_{i+1} \cdots w_n w_1 \cdots w_{i-1}$, for $1 \leq i \leq n$. A conjugacy class can also be represented as a circular word. Hence in what follows we shall use "circular word" and "conjugacy class" as synonyms.

The Burrows-Wheeler Transform (BWT) [4] is described as follows: given a word $w \in \Sigma^*$, the output of BWT is the pair $(\mathtt{bwt}(w), I)$ obtained by lexicographically sorting the list of the conjugates of w. In particular, $\mathtt{bwt}(w)$ is the word obtained by concatenating the last symbol of each conjugate in the sorted list and I is the position of w in such a list.

For instance, if $w = mathematics$ then $\mathtt{bwt}(w) = mmihttsecaa$ and $I = 7$. The matrix obtained by lexicographically sorting all the conjugates of w is depicted in Figure 1. We denote by F the first column of the matrix, i.e., F is the word obtained by lexicographically sorting the characters of w. The column L contains the word $\mathtt{bwt}(w)$.

One of the most important characteristic of BWT is that it tends to group together characters that occur in similar contexts in the input text (occurrences of a given symbol tend to occur in clusters) by producing an output that turns out to be highly compressible. Several authors refer to such property as the *clustering effect* of BWT. For instance, in Figure 1 one can see that equal characters are consecutive in the column L.

Another important property of the BWT is its *reversibility*. In fact, given the pair $(\mathtt{bwt}(w), I)$ it is possible to recover the original word w. This fact can be

		F										L
		↓										↓
	1	*a*	*t*	*h*	*e*	*m*	*a*	*t*	*i*	*c*	*s*	*m*
	2	*a*	*t*	*i*	*c*	*s*	*m*	*a*	*t*	*h*	*e*	*m*
	3	*c*	*s*	*m*	*a*	*t*	*h*	*e*	*m*	*a*	*t*	*i*
	4	*e*	*m*	*a*	*t*	*i*	*c*	*s*	*m*	*a*	*t*	*h*
	5	*h*	*e*	*m*	*a*	*t*	*i*	*c*	*s*	*m*	*a*	*t*
	6	*i*	*c*	*s*	*m*	*a*	*t*	*h*	*e*	*m*	*a*	*t*
$I \rightarrow$	7	*m*	*a*	*t*	*h*	*e*	*m*	*a*	*t*	*i*	*c*	*s*
	8	*m*	*a*	*t*	*i*	*c*	*s*	*m*	*a*	*t*	*h*	*e*
	9	*s*	*m*	*a*	*t*	*h*	*e*	*m*	*a*	*t*	*i*	*c*
	10	*t*	*h*	*e*	*m*	*a*	*t*	*i*	*c*	*s*	*m*	*a*
	11	*t*	*i*	*c*	*s*	*m*	*a*	*t*	*h*	*e*	*m*	*a*

Fig. 1. The matrix of the lexicographically sorted conjugates of the word *mathematics*

deduced by the following properties proved in [4] and that can be easily verified on the matrix of the figure.

1. For all $i = 1, \ldots, n$ and $i \neq I$, the character $F[i]$ follows $L[i]$ in the original word;
2. for each character z, the ith occurrence of z in L corresponds to the ith occurrence of z in F.

According to Property 2, we can define a permutation $\tau : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$ giving the correspondence between the positions of characters of L and F. The function τ represents also the order in which we have to rearrange the elements of L to reconstruct the original word w. Hence, starting from the position I, the word w can be recovered as follows:

$$w_{n-i} = L[\tau^i(I)], \text{ where } \tau^0(x) = x, \text{ and } \tau^i(x) = \tau(\tau^{i-1}(x)).$$

The permutation τ correspondent to the example described in Figure 1 is the following.

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

Given the permutation τ, the set $\{i, \tau(i), \tau^2(i), \ldots\}$ (for $i \in \{1, \ldots, n\}$) is called *orbit* of τ. The orbits of τ form a partition of the set $\{1, \ldots, n\}$. For instance the unique orbit of the function τ obtained by the L to F mapping correspondent to the word *mathematics* is $\{1, 7, 9, 3, 6, 11, 2, 8, 4, 5, 10\}$.

From the definition of BWT, it is possible to deduce the following characterization of conjugate words.

Proposition 1. *Two words x and y are conjugate if and only if* $\texttt{bwt}(x) = \texttt{bwt}(y)$.

From the previous proposition one can easily see that the index I allows to univocally individuate w in its conjugacy class. If we do not care about the index I, the Burrows-Wheeler Transform defines a function `bwt` from the set of all conjugacy classes or circular words over the alphabet Σ to the language

Σ^* that maps each circular word $[w]$ into $\texttt{bwt}(w)$. Note that such a function is injective but is not surjective. In fact, one can verify that the word *abraca* is not image by `bwt` of any circular word over the alphabet $\{a, b, c, r\}$.

A nonempty word $w \in \Sigma^*$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$. Recall that every nonempty word $u \in \Sigma^*$ can be written in a unique way as a power of a primitive word, i.e., there exists a unique primitive word w, called the *root* of u, and a unique integer k such that $u = w^k$.

The following proposition shows that the study of `bwt` can be reduced to that of `bwt` of primitive words (cf. [33]) and that the `bwt` of a non-primitive word can be deduced by the `bwt` of the root of the word itself.

Proposition 2. *For $u, v \in \Sigma^*$ one has:*

1. *If $u = v^d$ and $\texttt{bwt}(v) = \alpha_1\alpha_2\cdots\alpha_n$ then $\texttt{bwt}(u) = \alpha_1^d\alpha_2^d\cdots\alpha_n^d$.*
2. *If $\texttt{bwt}(v) = \alpha_1\alpha_2\cdots\alpha_n$ and $\texttt{bwt}(u) = \alpha_1^d\alpha_2^d\cdots\alpha_n^d$ then there exists a conjugate u' of u such that $u' = v^d$.*

3 Combinatorial Issues on the BWT

As shown in previous section the Burrows-Wheeler Transform has two properties which are two key aspects for its application in Text Compression. In fact, it is *reversible*, i.e., the original string can be reconstructed from the output of BWT, and it produces a *clustering effect* or *high local similarity*. In general, the output contains many runs of equal symbols and therefore is more suitable for compression.

Very recently, several natural issues have been addressed in the field of Combinatorics on Words on the above mentioned properties of BWT. The first question consists in characterizing all the words in Σ^* that are images by `bwt` of some word in Σ^*. Another important question is to give a qualitative characterization of all the words of Σ^* that are the most compressible, i.e., the words such that the `bwt` produces a perfect clustering of the characters. The following subsections are devoted to the presentation of the main results obtained about these topics.

3.1 Characterization of `bwt` Images

The problem of characterizing `bwt` images has been faced in [28], where the authors prove two necessary and sufficient conditions. The first one describes the words that are `bwt` images, while the second one explains which words can be converted to `bwt` images using a natural "pumping" procedure. Both conditions can be checked in linear time. In particular, the following theorem is proved.

Theorem 1. *Given a word $u \in \Sigma^*$, $u = \texttt{bwt}(w)$ for some $w \in \Sigma^*$ if and only if the number of orbits of the permutation τ equals the greatest common divisor of the lengths of runs of equal characters in u.*

By using the theorem one can deduce that the word $u = bccaaab$ is not a bwt image because the orbits of τ are 2 (i.e., $\{1, 4\}$ and $\{2, 6, 3, 7, 5\}$), but the lengths $1, 2, 3, 1$ of the runs of equal symbols in u are coprime.

In the paper [28] the authors also characterize the words $v = c_1 \cdots c_h$ that can be transformed to bwt images by a sort of pumping: each letter c_i can be replaced by $c_i^{p_i}$ for an arbitrary positive number p_i. In particular, they prove the following theorem that uses the notion of global ascent[1] of a word.

Theorem 2. *Let $v = c_1 \cdots c_h$ be an arbitrary word with at least two different letters. A bwt image of the form $c_1^{p_1} c_2^{p_2} \cdots c_h^{p_h}$, where $p_i > 0$ for all i, exists if and only if v has no global ascents.*

By using the algorithm provided in their paper, starting from the word $v = dacba$ one can construct the word $w = daccbaa$ that is the bwt image of *aacbcad*. An interesting open question is to construct the shortest possible bwt image with given order of letters.

3.2 Perfectly Clustering Words

An interesting issue, inspired by Data Compression, consists in characterizing the *perfectly clustering* words by bwt, that are the words that after the bwt are transformed into expressions in which all the occurrences of the same characters are consecutive, such as $c^i b^j a^h$ or $d^i b^j c^h a^k$. Notice that, if we consider primitive words, we can suppose that the exponents are coprime. In the field of Data Compression perfectly clustering words represent the most compressible words via a BWT-based compressor.

In the case of words over binary alphabet $\{a, b\}$, the problem is to characterize the words u such that $\mathtt{bwt}(u) = b^p a^q$, for some $p, q > 0$, because one can not have a word $\mathtt{bwt}(u)$ starting with the smallest symbol. Such a problem has been solved in [33,37]. In fact, the authors prove the following theorem that also represents a characterization of a well known family of finite words, called *standard sturmian words* (cf. [29]). These words have several characterizations as, for instance, a special decomposition into palindrome words and an extremal property on the periods of the word that is closely related to Fine and Wilf's theorem (cf. [10,9]). Moreover they also appear as extremal case in the pattern matching algorithm of Knuth-Morris-Pratt (see [26]).

Theorem 3. *Given a word u over the alphabet $\{a, b\}$, $\mathtt{bwt}(u) = b^p a^q$ (with $\gcd(p, q) = 1$) if and only if u is a conjugate of a standard sturmian word.*

Very recently, the problem has been also faced for alphabet of size greater than 2. In this case, a special attention has been given to the words with "simple" bwt. We say that a word w over an ordered alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$ with $a_1 < a_2 < \ldots < a_k$, has a *simple* bwt, if $\mathtt{bwt}(w)$ is of the form $a_k^{n_k} a_{k-1}^{n_{k-1}} \cdots a_1^{n_1}$, for some positive integers $n_1, n_2, \ldots, n_k$.

[1] A word v has a *global ascent* if $v = xy$, where x, y are nonempty and the maximal letter of x is less than or equal to the minimal letter of y.

We observe that the set of simple bwt words is a proper subset of the perfectly clustering words. In the case of binary alphabets, such set coincides with the set of perfectly clustering words.

In the case of three letters alphabet, the following result has been proved in [41] (the necessary condition has been independently given in [38]). Note that the following result that involves the vector of the occurrences of the characters cannot be naturally extended for greater alphabets and the question is still open.

Theorem 4. *The word u is a primitive word having a simple* bwt *on the alphabet $\Sigma = \{a_1, a_2, a_3\}$, i.e., $\mathtt{bwt}(u) = a_3^{n_3} a_2^{n_2} a_1^{n_1}$, if and only if (n_1, n_2, n_3) is a triple of integers satisfying both the conditions $\gcd(n_1, n_2, n_3) = 1$ and $\gcd(n_1 + n_2, n_2 + n_3) = 1$.*

Moreover, the words having simple bwt are related with the notion of palindromic word[2]. In [39] the authors prove the following result that holds for words over a generic alphabet Σ.

Theorem 5. *If the word $w \in \Sigma^*$ of length n has a simple* bwt *then ww has $2n + 1$ distinct palindromic factors.*

Note that $2n + 1$ is the maximum number of distinct palindromic factors that ww can contain (cf. [11]). Remark also that the condition stated in the previous theorem is not a characterization. In fact, there exist words having the maximal number of distinct palindromic factors but that have not a simple bwt. For instance, if $w = ccaaccb$ then ww has the maximal number of distinct palindromic factors but $\mathtt{bwt}(w) = cacccba$, hence w is not a perfectly clustering word.

Finally, in [13] it is proved that perfectly clustering words are intrinsically related to k-discrete interval exchange transformations. Such transformations can be intuitively defined by partitioning the interval $\{1, \ldots, n\}$ into k distinct sub-intervals. If each position of the interval is labeled by a character of the alphabet Σ, the transformation produces a trajectory starting from a given position and is the infinite sequences of characters obtained by following the transformation. Formal definitions can be found in [13]. The authors prove that some discrete interval exchange transformations generate perfectly clustering words, as stated in the following theorem.

Theorem 6. *Perfectly clustering words are exactly those words $w \in \Sigma^*$ such that ww occurs in a trajectory of a k-discrete interval exchange transformation, where k is the size of Σ.*

4 Practical Effects of the Combinatorial Properties

BWT-based compression algorithms take advantage of the fact that BWT typically produces an output word with a high local similarity, i.e., the occurrences of the same symbol tend to occur in clusters.

[2] The word w is *palindrome* if $w = \tilde{w}$, where $\tilde{w}$ denotes the *reversal* of w.

Actually, the practical importance of the clustering effect led to two different research lines. From one hand, to consider the question of defining functions on words to measure those combinatorial regularities of a text that are able to produce high local similarities via the BWT. On the other hand, to investigate the problem of the optimal partitioning of the BWT output word so that compressing individually its parts via a base-compressor gets a compressed output that is more compact than applying the compressor over the entire text at once.

This section is devoted to describe the results related to these two research issues.

4.1 Balancing and Compressibility of a Text

The problem of finding which kind of combinatorial regularities determine high compressibility in the texts via BWT-based compressors, has been posed in [25]. Note that identifying such regularities and eventually quantifying them could allow to prefer a compression strategy over another one.

As already shown in previous section the perfect clustering words represent the optimal case even for simple BWT-based compressors. It is interesting to note that, in the case of binary alphabets, perfectly clustering words are exactly the binary circularly balanced words. In fact, they are precisely the finite circular sturmian words. Intuitively, a word is *circularly balanced* if the frequency of each character of the alphabet, in different blocks[3] of the same length, is almost the same. Although this exact correspondence between highly compressible words and balanced words is not extensible even to larger alphabets, it has been experimentally verified that the notion of balancing of a word seems to play an important role in the compression of texts. In fact, although it has not been theoretically demonstrated, experimental results in [40] support the idea that the more balanced the input word is, the more clustered the BWT output word is, and, as a consequence, the better the compression is. In the paper the authors introduced two functions to measure both the degree of balancing of a word w and the clustering effect (or local similarity) of the word $\texttt{bwt}(w)$. Both the measures are defined by using the notion of *local entropy*[4] of a text that is a statistic, studied in [24], that captures the compressibility of a text.

The authors prove that the local entropy reaches its maximum on *constant gap words*, i.e., particular balanced words in which the distance between any two consecutive occurrences of a given symbol is constant throughout the word. The minimum value is obtained for clustered words. In [40], the behavior of a BWT-based compressor and a dictionary-based compressor (in particular, a compressor based on "LZ" method) has been tested on real texts. The experiments have showed that when the input text has a high degree of balancing then the BWT produces a text

[3] Note that a block is considered as a circular factor of the text.

[4] The local entropy is based on the notion of *distance coding* (DC) that encodes each symbol of a word as the number of characters between the symbol itself and its circular previous occurrence. Given a word $w = w_1 w_2 \dots w_n$ of length n, the local entropy on DC is defined as $LE(w) = \frac{1}{n} \sum_{i=1}^{n} \log(DC(w_i) + 1)$, where $DC(w_i)$ denotes the encoding of the ith symbol of w.

that has a high degree of clustering, and the BWT-based compressor achieves a better compression ratio. However, the theoretical analysis of the intermediate cases and the precise relationship between the local entropy of a word w and of the respective $\texttt{bwt}(w)$ remains still open. Answering to these questions would provide a useful tool to determine a priori the efficacy of a BWT-based compressor.

4.2 Optimal Partitioning of the Output of BWT

Several authors have analyzed the performances of the BWT-based compression algorithms (cf. [35,12,5,24]). It was proved that the BWT improves the performance of the base compressor from 0th order empirical entropy[5] bounds to kth order empirical entropy[6] bounds, simultaneously over all $k \geq 0$ (cf. [35]). Thus, the BWT can be viewed as a key component of a "compression booster" (cf. [18]), since it makes possible for relatively simple compression algorithms to perform better on most input texts.

A recent and important paradigm in Data Compression consists in reorganizing data in order to improve the performance of a given compressor C. Such a paradigm is called PPC. The basic idea consists in *permuting* the input text w to produce a new text w' that is *partitioned* into factors $z_1 z_2 \cdots z_t$ that are individually *compressed* by the compressor C. Although such a paradigm has been introduced in the context of table compression, it assumes a central role when BWT-based text compressors are considered. In fact, the Burrows-Wheeler Transform is used as preprocessing and the transformed text is finally compressed via proper 0th order-entropy compressors (like Move to Front and Run Length Encoding combined with Huffman and Arithmetic coding [44]).

In terms of the PPC paradigm, the BWT seems to be a valid support for the permuting step because produces a highly compressible permutation which is fast to be computed and achieves effective compression bounds. An important and still open problem of Data Compression is finding the efficient computation of an optimal partitioning of the output produced by BWT for compression boosting. This is close to find an optimal partition of $\texttt{bwt}$ into factors that are "almost" clustered words, or with high degree of clustering. In [19] the authors give the first efficient approximation algorithm for the problem of finding an optimal partition of an input text w in blocks such that compress size achieved by compressing them individually with a base compressor C is better than that of the whole w. As shown in [21] the optimal partition can be computed by a dynamic-programming approach. Unfortunately, this solution applied to input of size n requires to run the compressor C over $\Theta(n^2)$ factors having average length $\Theta(n)$, for an overall $\Theta(n^3)$ time cost in the worst case. It is clearly not usable in practice even for texts longer than few Mbs.

[5] The 0th order empirical entropy of w of length n is defined as $H_0(w) = \sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n}{n_i}$, where n_i is the number of occurrences of $a_i \in \Sigma$ in w. It is assumed that all logarithms are taken to the base 2 and $0 \log 0 = 0$.

[6] For any word $w \in \Sigma^k$, let w_s denote the string consisting of the characters preceding all occurrences of w in a string s. The value $H_k(s) = \frac{1}{n} \sum_{w \in \Sigma^k} |w_s| H_0(w_s)$ is called the kth order empirical entropy of the string s.

In [19] the authors prove the following theorem.

Theorem 7. *Given a text w of length n drawn from an alphabet of size $|\Sigma| = poly(n)$, both with respect to a 0th order compressor and to a kth order compressor, an $(1+\epsilon)$-optimal partition of w can be found in $O(n \log_{1+\epsilon} n)$ time and $O(n)$ space, where ϵ is any positive constant.*

The authors also propose a solution to the optimal partitioning problem for BWT-based compressors that introduces a $\Theta(|\Sigma| \log n)$ slowdown in the time complexity, but with the advantage of computing the $(1+\epsilon)$-optimal solution with respect to the real compressed size, thus without any estimation by entropy-cost functions. If $|\Sigma|$ is small, this slowdown results negligible.

5 An Extension of BWT to a Multiset of Words

In [8] the authors pointed out the very interesting fact that the `bwt` coincides with a particular case of a bijection defined in [20] by Gessel and Reutenauer. This remark suggested the definition (cf. [30]) and the analysis (cf. [31]) of an extension of the BWT (called EBWT) to a multiset of primitive words, that is somehow an algorithmic presentation of the bijection introduced in [20].

The EBWT of a multiset $\mathcal{S}$ is a word (denoted by $\texttt{ebwt}(\mathcal{S})$) obtained by letter permutation of the words in $\mathcal{S}$ together a set of indexes used to recover the original multiset. In particular, $\texttt{ebwt}(\mathcal{S})$ is obtained by concatenating the last symbol of each element in the sorted list of the conjugates of $\mathcal{S}$. The sorting exploits an order relation defined by using lexicographic order between infinite words. It is interesting to note that the `ebwt` inherits both the mathematical and combinatorial properties of the `bwt` but, differently from `bwt`, the `ebwt` is surjective. Indeed, it can be also seen as a bijection from multisets of circular words over the alphabet Σ and the language Σ^*. A study of the combinatorial aspects connecting the BWT and the EBWT can be found in [3].

In the field of the Data Compression, the EBWT has been used in [30] where the authors have experimentally verified that when the single text is split into small blocks of equal length and the multiset of the blocks are simultaneously compressed by using EBWT as preprocessing, the compression ratio is better than compressing each block separately, and then concatenating them. A similar approach has been employed in [22,27] where a different factorization is used. Very recently, the EBWT has been used for the compression of large collections of DNA sequences (cf. [6]) by an external memory implementation of the algorithm (cf. [2]). The authors showed that, while `bzip2`[7] performs comparably to `gzip`[8] on DNA sequence reads, the compression achieved by EBWT-based methods improves by more than threefold if the EBWT of the entire read multiset is built, since this captures redundancy between reads that were widely spaced in the original file. It was shown that compression can be further boosted by pre-sorting the sequences or applying an implicit sorting strategy while the EBWT is being built, enabling compression of better than 0.5 bits-per-base to be achieved.

[7] `bzip2` is a BWT-based compressor (`www.bzip.org`)
[8] `gzip` is a LZ-based compressor (`www.gzip.org`)

The EBWT is also used in other applicative context as the circular pattern matching (cf. [23]) and the alignment-free methods for comparing sequences (cf. [43,34]). In particular, in [32,45,36] different distance measures have been defined and successfully used in several biological datasets, as for instance mitochondrial DNA genomes, expressed sequence tags and proteins.

The methods based on the EBWT rely on the following idea: when EBWT is applied to $\mathcal{S} = \{u, v\}$, if the same factor z occurs both in u and v, then the conjugates of u and v starting by z are likely to be close in the sorted list of conjugates. This implies that the greater the number of factors shared by u and v is, the greater the mixing of the conjugates of u and v in the sorted list is. The comparison method based on the EBWT measures how similar u and v are, by taking into account how much their conjugates are mixed. This intuition has different possible formalizations that are closely related to the different partitions of the word $\texttt{ebwt}(\mathcal{S})$. In particular, a different color can be assigned to each element of $\mathcal{S}$. The word $\texttt{ebwt}(\mathcal{S})$ contains a sequence of colors that depends on how the conjugates of u and v are mixed in the sorted list. See Figure 2 for an example. It is possible to define a distance measure by computing the number of the alternations in the sequence of colors (cf. [31]). A more general class of distance measures can be defined by using different partitioning of the colored output of the $\texttt{ebwt}(\mathcal{S})$ and by finally counting the difference of frequencies of colors into each block of the partition.

Conjugates	ebwt	*Colors*
ababccb	*b*	*R*
ababccc	*c*	*B*
abccbab	*b*	*R*
abcccab	*b*	*B*
bababcc	*c*	*R*
babccba	*a*	*R*
babccca	*a*	*B*
bccbaba	*a*	*R*
bcccaba	*a*	*B*
cababcc	*c*	*B*
cbababc	*c*	*R*
ccababc	*c*	*B*
ccbabab	*b*	*R*
cccabab	*b*	*B*

Fig. 2. Given the set $\mathcal{S} = \{u = ababcccb, v = ccccabab\}$ we associate the color *Red* to u and the color *Blue* to v. The columns of the matrix are the sorted list of the conjugates, the word $\texttt{ebwt}(\mathcal{S})$ and the sequences of colors, respectively.

References

1. Adjeroh, D., Bell, T., Mukherjee, A.: The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching, 1st edn. Springer Publishing Company, Incorporated (2008)
2. Bauer, M.J., Cox, A.J., Rosone, G.: Lightweight algorithms for constructing and inverting the BWT of string collections. Theoret. Comput. Sci. 483, 134–148 (2013)

3. Bonomo, S., Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: Suffixes, Conjugates and Lyndon words. In: Béal, M.-P., Carton, O. (eds.) DLT 2013. LNCS, vol. 7907, pp. 131–142. Springer, Heidelberg (2013)
4. Burrows, M., Wheeler, D.J.: A block sorting data compression algorithm. Technical report, DIGITAL System Research Center (1994)
5. Cai, H., Kulkarni, S.R., Verdú, S.: Universal entropy estimation via block sorting. IEEE Transactions on Information Theory 50(7), 1551–1561 (2004)
6. Cox, A.J., Bauer, M.J., Jakobi, T., Rosone, G.: Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. Bioinformatics 28(11), 1415–1419 (2012)
7. Cox, A.J., Jakobi, T., Rosone, G., Schulz-Trieglaff, O.B.: Comparing DNA sequence collections by direct comparison of compressed text indexes. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS (LNBI), vol. 7534, pp. 214–224. Springer, Heidelberg (2012)
8. Crochemore, M., Désarménien, J., Perrin, D.: A note on the Burrows-Wheeler transformation. Theoret. Comput. Sci. 332, 567–572 (2005)
9. de Luca, A.: Combinatorics of standard sturmian words. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) Structures in Logic and Computer Science. LNCS, vol. 1261, Springer, Heidelberg (1997)
10. de Luca, A., Mignosi, F.: Some combinatorial properties of sturmian words. Theoret. Comput. Sci. 136(2), 361–385 (1994)
11. Droubay, X., Justin, J., Pirillo, G.: Episturmian words and some constructions of de Luca and Rauzy. Theoret. Comput. Sci. 255(1-2), 539–553 (2001)
12. Effros, M., Visweswariah, K., Kulkarni, S.R., Verdú, S.: Universal lossless source coding with the Burrows Wheeler Transform. IEEE Transactions on Information Theory 48(5), 1061–1081 (2002)
13. Ferenczi, S., Zamboni, L.Q.: Clustering Words and Interval Exchanges. Journal of Integer Sequences 16(2), Article 13.2.1 (2013)
14. Ferragina, P., Gagie, T., Manzini, G.: Lightweight Data Indexing and Compression in External Memory. Algorithmica 63(3), 707–730 (2012)
15. Ferragina, P., Giancarlo, R., Manzini, G., Sciortino, M.: Boosting textual compression in optimal linear time. J. ACM 52(4), 688–713 (2005)
16. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: FOCS 2000, pp. 390–398. IEEE Computer Society (2000)
17. Ferragina, P., Manzini, G.: An experimental study of an opportunistic index. In: SODA 2001, pp. 269–278. SIAM (2001)
18. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM 52, 552–581 (2005)
19. Ferragina, P., Nitto, I., Venturini, R.: On optimally partitioning a text to improve its compression. Algorithmica 61, 51–74 (2011)
20. Gessel, I.M., Reutenauer, C.: Counting permutations with given cycle structure and descent set. J. Combin. Theory Ser. A 64(2), 189–215 (1993)
21. Giancarlo, R., Sciortino, M.: Optimal partitions of strings: A new class of Burrows-Wheeler compression algorithms. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 129–143. Springer, Heidelberg (2003)
22. Gil, J.Y., Scott, D.A.: A bijective string sorting transform. CoRR (2012); abs/1201.3077
23. Hon, W.-K., Ku, T.-H., Lu, C.-H., Shah, R., Thankachan, S.V.: Efficient Algorithm for Circular Burrows-Wheeler Transform. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 257–268. Springer, Heidelberg (2012)
24. Kaplan, H., Landau, S., Verbin, E.: A simpler analysis of Burrows-Wheeler-based compression. Theoret. Comput. Sci. 387(3), 220–235 (2007)

25. Kaplan, H., Verbin, E.: Most burrows-wheeler based compressors are not optimal. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 107–118. Springer, Heidelberg (2007)
26. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323–350 (1977)
27. Kufleitner, M.: On bijective variants of the Burrows-Wheeler transform, pp. 65–79 (2009)
28. Likhomanov, K.M., Shur, A.M.: Two combinatorial criteria for BWT images. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 385–396. Springer, Heidelberg (2011)
29. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge Univ. Press (2002)
30. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 178–189. Springer, Heidelberg (2005)
31. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: An extension of the Burrows-Wheeler Transform. Theoret. Comput. Sci. 387(3), 298–312 (2007)
32. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: A new combinatorial approach to sequence comparison. Theory Comput. Syst. 42(3), 411–429 (2008)
33. Mantaci, S., Restivo, A., Sciortino, M.: Burrows-Wheeler transform and Sturmian words. Information Processing Letters 86, 241–246 (2003)
34. Mantaci, S., Restivo, A., Sciortino, M.: Distance measures for biological sequences: Some recent approaches. Int. J. Approx. Reasoning 47(1), 109–124 (2008)
35. Manzini, G.: An analysis of the Burrows-Wheeler transform. J. ACM 48(3), 407–430 (2001)
36. Ng, K.-H., Ho, C.-K., Phon-Amnuaisuk, S.: A hybrid distance measure for clustering expressed sequence tags originating from the same gene family. PLoS ONE 7(10) (2012)
37. Jenkinson, O., Zamboni, L.Q.: Characterisations of balanced words via orderings. Theoret. Comput. Sci. 310(1), 247–271 (2004)
38. Pak, I., Redlich, A.: Long cycles in abc-permutations. Functional Analysis and Other Mathematics 2, 87–92 (2008)
39. Restivo, A., Rosone, G.: Burrows-Wheeler transform and palindromic richness. Theoret. Comput. Sci. 410(30-32), 3018–3026 (2009)
40. Restivo, A., Rosone, G.: Balancing and clustering of words in the Burrows-Wheeler transform. Theoret. Comput. Sci. 412(27), 3019–3032 (2011)
41. Simpson, J., Puglisi, S.J.: Words with simple Burrows-Wheeler transforms. Electronic Journal of Combinatorics 15 article R83 (2008)
42. Simpson, J.T., Durbin, R.: Efficient construction of an assembly string graph using the FM-index. Bioinformatics 26(12), i367–i373 (2010)
43. Vinga, S., Almeida, J.: Alignment-free sequence comparison a review. Bioinformatics 19(4), 513–523 (2003)
44. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann (1999)
45. Yang, L., Zhang, X., Wang, T.: The Burrows-Wheeler similarity distribution between biological sequences based on Burrows-Wheeler transform. Journal of Theoretical Biology 262(4), 742–749 (2010)

A Note on ω-Jump Inversion of Degree Spectra of Structures

Ivan N. Soskov*

Faculty of Mathematics and Computer Science, Sofia University,
5 James Bourchier Blvd., 1164 Sofia, Bulgaria
soskov@fmi.uni-sofia.bg

Abstract. In this note we provide a negative solution to the ω-jump inversion problem for degree spectra of structures.

1 Introduction

Let $\mathfrak{A}$ be a countable structure. The spectrum of $\mathfrak{A}$ is the set of Turing degrees defined by:

$$\mathrm{Sp}(\mathfrak{A}) := \{\mathbf{a}|\ \mathbf{a} \text{ computes an isomorphic copy of } \mathfrak{A}\}$$

For $\alpha < \omega_1^{\mathrm{CK}}$ the α-th jump spectrum of $\mathfrak{A}$ is the set $\mathrm{Sp}_\alpha(\mathfrak{A}) = \{\mathbf{a}^{(\alpha)}|\mathbf{a} \in \mathrm{Sp}(\mathfrak{A})\}$. The jump inversion problem for degree spectra of structures can be stated as follows: Let $\alpha < \omega_1^{\mathrm{CK}}$ and $\mathfrak{A}$ be a countable structure such that all elements of $\mathrm{Sp}(\mathfrak{A})$ are above $\mathbf{0}^{(\alpha)}$. Does there exist a structure $\mathfrak{M}$ such that $\mathrm{Sp}_\alpha(\mathfrak{M}) = \mathrm{Sp}(\mathfrak{A})$?

A positive solution to the problem for successor ordinals can be found in [1]. Though the problem is not explicitly stated there, the solution is a byproduct of the construction for all successor ordinals $\alpha < \omega_1^{\mathrm{CK}}$ of α-categorical structures which are not relatively α-categorical. Another solution for finite ordinals based on Marker's extensions is given in [2].

In this note we shall show that in the general case the ω-jump inversion problem for degree spectra has a negative solution. The proof can be easily adapted for all recursive limit ordinals.

In what follows we shall define a structure $\mathfrak{A}$ such that $\mathrm{Sp}(\mathfrak{A}) \subseteq \{\mathbf{a}|\mathbf{0}^{(\omega)} \leq \mathbf{a}\}$ and for all countable structures $\mathfrak{M}$, $\mathrm{Sp}_\omega(\mathfrak{M}) \neq \mathrm{Sp}(\mathfrak{A})$. The definition of $\mathfrak{A}$ is based on a special property of the ω co-spectra which can be expressed in terms of enumeration reducibility. Namely we shall show that if $\mathfrak{M}$ is a countable structure then every enumeration degree in the ω co-spectrum of $\mathfrak{M}$ is bounded by a total enumeration degree which also belongs to the ω co-spectrum of $\mathfrak{M}$.

The basic facts about enumeration reducibility and co-spectra of structures needed for the presentation are summarized below.

* This research was partially supported by Sofia University Science Fund and a NSF grant DMS-1101123.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 365–370, 2013.

2 Preliminaries

2.1 Enumeration Reducibility

Definition 1. Given two sets of natural numbers X and Y, say that X is enumeration reducible to Y ($X \leq_e Y$) if for some e, $X = W_e(Y)$, i.e.,

$$(\forall x)(x \in X \iff (\exists v)(\langle x, v\rangle \in W_e \wedge D_v \subseteq Y)).$$

Let $X \equiv_e Y$ if $X \leq_e Y$ and $Y \leq_e X$. The enumeration degree $d_e(X)$ of the set X consists of all sets Y which are enumeration equivalent to X. By $\mathcal{D}_e$ we shall denote the set of all enumeration degrees.

Given a set $X \subseteq \mathbb{N}$, by X^+ we shall denote $X \oplus (\mathbb{N} \setminus X)$. We have for all $X, Y \subseteq \mathbb{N}$ that $X \leq_{\text{c.e.}} Y \iff X \leq_e Y^+$ and $X \leq_T Y \iff X^+ \leq_e Y^+$.

Definition 2. A set X of natural numbers is called *total* if $X \equiv_e X^+$.

The enumeration jump was defined by Cooper in [3]:

Definition 3. Let $X \subseteq \mathbb{N}$. Set $J_e(X) = \{\langle e, x\rangle | x \in W_e(X)\}$. The *enumeration jump* X' *of* X is the set $J_e(X)^+$.

Clearly for all $X \subseteq \mathbb{N}$, X' is a total set. The enumeration jump of X is somewhat weaker than the Turing jump $J_T(X)$ of X. Namely $(\forall X \subseteq \mathbb{N})(J_T(X)^+ \equiv_e (X^+)')$ hence $X' \leq_T (X^+)' \leq_T J_T(X)$. Of course for total sets X, $X' \equiv_T J_T(X)$.

We shall use the following property of the enumeration jump:

Proposition 4. *There exists a computable function j such that for all $e \in \mathbb{N}$ and $X \subseteq \mathbb{N}$, $W_e(X)' = W_{j(e)}(X')$.*

Proof. Consider a computable function λ such that for every a and e and for all X, $W_a(W_e(X)) = W_{\lambda(a,e)}(X)$. Then

$$2\langle a, x\rangle \in W_e(X)' \iff 2\langle \lambda(a, e), x\rangle \in X' \text{ and}$$
$$2\langle a, x\rangle + 1 \in W_e(X)' \iff 2\langle \lambda(a, e), x\rangle + 1 \in X'.$$

Let j be the computable function yielding for every e an index of the c.e. set

$$\{\langle 2\langle a, x\rangle, \{2\langle \lambda(a, e), x\rangle\}\rangle : a, x \in \mathbb{N}\} \cup \{\langle 2\langle a, x\rangle + 1, \{2\langle \lambda(a, e), x\rangle + 1\}\rangle : a, x \in \mathbb{N}\}.$$

Then for all e, $W_e(X)' = W_{j(e)}(X')$.

2.2 Enumeration Reducibility of Sequences of Sets

Definition 5. Let $\mathcal{X} = \{X_n\}_{n<\omega}$ and $\mathcal{Y} = \{Y_n\}_{n<\omega}$ be sequences of sets of natural numbers. Then $\mathcal{X}$ *is enumeration reducible to* $\mathcal{Y}$ ($\mathcal{X} \leq_e \mathcal{Y}$) if for all n, $X_n \leq_e Y_n$ uniformly in n. In other words, if there exists a computable function μ such that for all n, $X_n = W_{\mu(n)}(Y_n)$.

Definition 6. Let $\mathcal{X} = \{X_n\}$ be a sequence of sets of natural numbers. The *jump sequence* $\mathcal{P}(\mathcal{X}) = \{\mathcal{P}_n(\mathcal{X})\}$ of $\mathcal{X}$ is defined by induction:

(i) $\mathcal{P}_0(\mathcal{X}) = X_0$;
(ii) $\mathcal{P}_{n+1}(\mathcal{X}) = \mathcal{P}_n(\mathcal{X})' \oplus X_{n+1}$.

By $\mathcal{P}_\omega(\mathcal{X})$ we shall denote the set $\bigoplus_n \mathcal{P}_n(\mathcal{X})$. Clearly $\mathcal{X} \leq_e \mathcal{P}(\mathcal{X})$ and hence $\bigoplus_n X_n \leq_e \mathcal{P}_\omega(\mathcal{X})$.

Proposition 7. *For all sequences $\mathcal{X}$ of sets of natural numbers the set $\mathcal{P}_\omega(\mathcal{X})$ is total.*

Proof. Fix z_0 so that for all sets X, $W_{z_0}(X) = X$. Then

$$\langle n, x\rangle \notin \mathcal{P}_\omega(\mathcal{X}) \iff x \notin \mathcal{P}_n(\mathcal{X}) \iff x \notin W_{z_0}(\mathcal{P}_n(\mathcal{X})) \iff$$
$$2\langle z_0, x\rangle + 1 \in \mathcal{P}'_n(\mathcal{X}) \iff 2(2\langle z_0, x\rangle + 1) \in \mathcal{P}_{n+1}(\mathcal{X}) = \mathcal{P}'_n \oplus X_{n+1} \iff$$
$$\langle n+1, 2(2\langle z_0, x\rangle + 1)\rangle \in \mathcal{P}_\omega(\mathcal{X}).$$

So, $\mathbb{N} \setminus \mathcal{P}_\omega(\mathcal{X}) \leq_e \mathcal{P}_\omega(\mathcal{X})$.

Proposition 8. *Let $\mathcal{X} = \{X_n\}$ be a sequence of sets of natural numbers, $M \subseteq \mathbb{N}$ and $\mathcal{X} \leq_e \{M^{(n)}\}_{n<\omega}$. Then $\mathcal{P}(\mathcal{X}) \leq_e \{M^{(n)}\}_{n<\omega}$.*

Proof. Let $\lambda(a, b)$ be a computable function such that for all $Y \subseteq \mathbb{N}$, $W_a(Y) \oplus W_b(Y) = W_{\lambda(a,b)}(Y)$ and j be the recursive function defined in proposition 4. Suppose that for all n, $X_n = W_{\mu(n)}(M^{(n)})$.

Now $P_0(\mathcal{X}) = X_0 = W_{\mu(0)}(M^{(0)})$. Suppose that $\mathcal{P}_n(\mathcal{X}) = W_a(M^{(n)})$. Then $\mathcal{P}_{n+1}(\mathcal{X}) = \mathcal{P}_n(\mathcal{X})' \oplus X_{n+1} = W_{j(a)}(M^{(n+1)}) \oplus W_{\mu(n+1)}(M^{(n+1)}) = W_{\lambda(j(a),\mu(n+1))}(M^{(n+1)})$.

2.3 Co-spectra of Structures

We shall identify the Turing degrees and the total enumeration degrees, i.e., the enumeration degrees containing a total set. This assumption is safe because of the standard embedding ι of the Turing degrees into the enumeration degrees defined by $\iota(d_T(A)) = d_e(A^+)$ which is known to preserve also the jump operation.

Definition 9. Let $\mathfrak{M}$ be a countable structure and $\alpha < \omega_1^{CK}$. The α-th co-spectrum of $\mathfrak{M}$ is the set

$$\mathrm{CoSp}_\alpha(\mathfrak{M}) = \{\mathbf{a} | \mathbf{a} \in \mathcal{D}_e \wedge (\forall \mathbf{b} \in \mathrm{Sp}_\alpha(\mathfrak{M}))(\mathbf{a} \leq_e \mathbf{b})\}.$$

Definition 10. Let $\alpha < \omega_1^{CK}$. A subset R of $\mathbb{N}$ is Σ^c_α definable in $\mathfrak{M}$ if there exist a computable function γ taking as values codes of computable Σ_α infinitary formulas $F_{\gamma(x)}$ and finitely many parameters $t_1, \ldots, t_m$ of $|\mathfrak{M}|$ such that

$$x \in R \iff \mathfrak{M} \models F_{\gamma(x)}(t_1, \ldots, t_m).$$

For the definition of the computable Σ_α formulae the reader may consult [4].

Theorem 11. *([5]). Let $\alpha < \omega_1^{CK}$. Then*

1. *If $\alpha < \omega$ then $\mathbf{a} \in \mathrm{CoSp}_\alpha(\mathfrak{M})$ if and only if all elements of $\mathbf{a}$ are $\Sigma^c_{\alpha+1}$ definable in $\mathfrak{M}$.*
2. *If $\omega \leq \alpha$ then $\mathbf{a} \in \mathrm{CoSp}_\alpha(\mathfrak{M})$ if and only if all elements of $\mathbf{a}$ are Σ^c_α definable in $\mathfrak{M}$.*

3 A Property of the ω Co-spectra

Theorem 12. *Let $\mathfrak{M}$ be a countable structure and $\mathbf{a} \in \mathrm{CoSp}_\omega(\mathfrak{M})$. Then there exists a total enumeration degree $\mathbf{b}$ such that $\mathbf{a} \leq_e \mathbf{b}$ and $\mathbf{b} \in \mathrm{CoSp}_\omega(\mathfrak{M})$,*

Proof. Fix an element R of $\mathbf{a}$. The set R is Σ^c_ω definable in $\mathfrak{M}$ and hence there exists a computable function γ and parameters $t_1, \ldots, t_m$ of $|\mathfrak{M}|$ such that

$$x \in R \iff \mathfrak{M} \models F_{\gamma(x)}(t_1, \ldots, t_m).$$

Since each $F_{\gamma(x)}$ is a computable Σ_ω formula, i.e., a c.e. disjunction of computable Σ_{n+1} formulae, $n < \omega$, we may assume that there exists a computable function $\delta(n, x)$ such that for all n and x, $\delta(n, x)$ yields a code of some computable Σ_{n+1} formula $F_{\delta(n,x)}$ and

$$x \in R \iff (\exists n)(\mathfrak{M} \models F_{\delta(n,x)}(t_1, \ldots, t_m)).$$

Let $R_n = \{x | x \in \mathbb{N} \wedge \mathfrak{M} \models F_{\delta(n,x)}(t_1, \ldots, t_n)\}$. It is easy to see that if B is the diagram of some isomorphic copy $\mathfrak{B}$ of $\mathfrak{M}$ on the natural numbers then $\{R_n\} \leq_e \{B^{(n)}\}$. Indeed, let κ be an isomorphism from $\mathfrak{M}$ to $\mathfrak{B}$ and $x_1 = \kappa(t_1), \ldots, x_m = \kappa(t_m)$. Then

$$x \in R_n \iff \mathfrak{B} \models F_{\delta(n,x)}(x_1, \ldots, x_m).$$

Clearly the set of all computable Σ_{n+1} formulae $F_{\delta(n,x)}$ with fixed parameters $x_1, \ldots, x_m$ which are satisfied in $\mathfrak{B}$ is uniformly in n enumeration reducible to $B^{(n)}$.

By proposition 8 we have also that $\mathcal{P}(\{R_n\}) \leq_e \{B^{(n)}\}$. Hence $\mathcal{P}_\omega(\{R_n\}) \leq_e B^{(\omega)}$.

Set $\mathbf{b} = d_e(\mathcal{P}_\omega(\{R_n\}))$. As shown above $\mathbf{b} \in \mathrm{CoSp}_\omega(\mathfrak{M})$. Notice that $\mathbf{b}$ is a total degree. It remains to see that $\mathbf{a} \leq_e \mathbf{b}$. Indeed, since $x \in R \iff (\exists n)(x \in R_n)$, $R \leq_e \bigoplus_n R_n$. On the other hand $\bigoplus_n R_n \leq_e \mathcal{P}_\omega(\{R_n\})$. Therefore $R \leq_e \mathcal{P}_\omega(\{R_n\})$

Now we are ready to define the structure $\mathfrak{A}$ promised in the introduction. Let Y be a set which is quasi-minimal above $\varnothing^{(\omega)}$. This means that $\varnothing^{(\omega)} <_e Y$ and if X is a total set and $X \leq_e Y$ then $X \leq_e \varnothing^{(\omega)}$. The existence of such sets is well known in the theory of the enumeration degrees. For example, one can take $Y = \varnothing^{(\omega)} \oplus G$, where G is one-generic relatively $\varnothing^{(\omega)}$, see [6]. Notice that the enumeration degree of Y does not contain any total set.

Suppose that $\mathfrak{A}$ is a countable structure with $\mathrm{CoSp}(\mathfrak{A}) = \{\mathbf{a}|\mathbf{a} \leq_\mathrm{e} d_\mathrm{e}(Y)\}$. Clearly $\mathrm{Sp}(\mathfrak{A}) \subseteq \{\mathbf{b}|\mathbf{0}^{(\omega)} \leq_T \mathbf{b}\}$. Assume that there exists a countable structure $\mathfrak{M}$ such that $\mathrm{Sp}_\omega(\mathfrak{M}) = \mathrm{Sp}(\mathfrak{A})$. Then $\mathrm{CoSp}_\omega(\mathfrak{M}) = \mathrm{CoSp}(\mathfrak{A})$ and hence there exists a total degree $\mathbf{b}$ in $\mathrm{CoSp}(\mathfrak{A})$ such that $d_\mathrm{e}(Y) \leq \mathbf{b}$. Since $d_\mathrm{e}(Y)$ is the greatest element of $\mathrm{CoSp}(\mathfrak{A})$, $\mathbf{b} = d_\mathrm{e}(Y)$. A contradiction.

It remains to see that there exists a countable structure $\mathfrak{A}$ with co-spectrum equal to the set $\{\mathbf{a}|\mathbf{a} \leq_\mathrm{e} d_\mathrm{e}(Y)\}$. This follows from the fact that every principal ideal of enumeration degrees can be represented as co-spectrum of some subgroup of the additive group Q of the rational numbers, see [7]. To make the presentation self-contained we shall present this short argument here.

Let us fix a non-trivial group $G \subseteq Q$. Let $a \neq 0$ be an element of G. For every prime number p set

$$h_p(a) = \begin{cases} k & \text{if } k \text{ is the greatest number such that } p^k|a \text{ in } G, \\ \infty & \text{if } p^k|a \text{ in } G \text{ for all } k. \end{cases}$$

Let $p_0, p_1, \ldots$ be the standard enumeration of the prime numbers and set

$$S_a(G) = \{\langle i, j\rangle : j \leq h_{p_i}(a)\}.$$

It can be easily seen that if a and b are non-zero elements of G, then $S_a(G) \equiv_\mathrm{e} S_b(G)$. Let $\mathbf{d}_G = d_\mathrm{e}(S_a(G))$, where a is some non-zero element of G. The following proposition follows from results of Coles, Downey and Slaman [8]:

Proposition 13. $\mathrm{Sp}(G) = \{\mathbf{b}|\mathbf{b} \textit{ is total} \wedge \mathbf{d}_G \leq_\mathrm{e} \mathbf{b}\}$.

Corollary 14. $\mathrm{CoSp}(G) = \{\mathbf{a}|\mathbf{a} \leq_\mathrm{e} \mathbf{d}_G\}$.

Proof. Clearly $\mathbf{a} \in \mathrm{CoSp}(G)$ if and only if for all total $\mathbf{b}$, $\mathbf{d}_G \leq_\mathrm{e} \mathbf{b} \Rightarrow \mathbf{a} \leq_\mathrm{e} \mathbf{b}$. According Selman's Theorem [9] the last is equivalent to $\mathbf{a} \leq_\mathrm{e} \mathbf{d}_G$.

Now, let $Y \subseteq \mathbb{N}$. Consider the set $S = \{\langle i, j\rangle : (j = 0) \vee (j = 1 \;\&\; i \in Y)\}$.

Clearly $S \equiv_\mathrm{e} Y$. Let G be the least subgroup of Q containing the set $\{1/p_i^j : \langle i, j\rangle \in S\}$. Then $1 \in G$ and $S_1(G) = S$. So, $\mathbf{d}_G = d_\mathrm{e}(Y)$.

References

1. Goncharov, S., Harizanov, V., Knight, J., McCoy, C., Miller, R., Solomon, R.: Enumerations in computable structure theory. Annals of Pure and Applied Logic 136, 219–246 (2005)
2. Soskova, A., Soskov, I.: A jump inversion theorem for the degree spectra. Journal of Logic and Computation 19, 199–215 (2009)
3. Cooper, S.: 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. Ash, C., Knight, J.: Computable Structures and the Hyperarithmetical Hierarchy. Studies in Logic and the Foundations of Mathematics, vol. 144. North-Holland, Amsterdam (2000)
5. Ash, C., Knight, J., Manasse, M., Slaman, T.: Generic copies of countable structures. Ann. Pure Appl. Logic 42, 195–205 (1989)

6. Copestake, K.: 1-Genericity in the enumeration degrees. J. Symbolic Logic 53, 878–887 (1988)
7. Soskov, I.: Degree spectra and co-spectra of structures. Ann. Univ. Sofia 96, 45–68 (2004)
8. Coles, R., Downey, R., Slaman, T.: Every set has a least jump enumeration. Bulletin London Math. Soc. 62, 641–649 (2000)
9. Selman, A.: Arithmetical reducibilities I. Z. Math. Logik Grundlag. Math. 17, 335–350 (1971)

The Turing Universe in the Context of Enumeration Reducibility

Mariya I. Soskova[1,2⋆]

[1] Department of Mathematical Logic and Applications, Sofia University, Bulgaria
[2] Department of Mathematics, University of California, Berkeley, U.S.A.

A fundamental goal of computability theory is to understand the way that objects relate to each other in terms of their information content. We wish to understand the relative information content between sets of natural numbers, how one subset of the natural numbers Y can be used to specify another one X. This specification can be computational, or arithmetic, or even by the application of a countable sequence of Borel operations. Each notion in the spectrum gives rise to a different model of relative computability. Which of these models best reflects the real world computation is under question.

The most widely used and studied model is the one based on computation: a set of natural numbers A is Turing reducible to set of natural numbers B if there is an effective procedure, by which given a natural number n and using the answers to finitely many membership questions to the oracle B we can correctly decide whether or not n is a member of A. By identifying sets that can be reduced to each other we obtain the partial order of the Turing degrees. Computable sets have least information content and form the least Turing degree $\mathbf{0}_{\mathrm{T}}$. There is a natural way to combine the information content of two sets of natural numbers, A and B are combined into $A \oplus B = \{2n \mid n \in A\} \cup \{2n+1 \mid n \in B\}$, so that every pair of Turing degrees has a least upper bound. Finally, using a relativization of the halting problem, for every Turing degree $\mathbf{d}_{\mathrm{T}}(A)$ we can find a degree which is strictly more complex, namely $\mathbf{d}_{\mathrm{T}}(K_A)$, where K_A is the halting set, relativized to A, which we call the jump of the Turing degree $\mathbf{d}_{\mathrm{T}}(A)$, and denote by $\mathbf{d}_{\mathrm{T}}(A)'$. Thus the structure of the Turing degrees is an upper semi-lattice with least element and jump operation: $\mathcal{D}_{\mathrm{T}} = (D_{\mathrm{T}}, \leq_{\mathrm{T}}, \mathbf{0}_{\mathrm{T}}, \vee, ')$.

In this article we shall examine the structure of the Turing degrees in the context, provided by a slightly weaker form of reducibility, based on the notion of enumeration rather than computation. This reducibility was introduced by Friedberg and Rogers [6] in 1959. A set of natural numbers A is enumeration reducible to a set of natural numbers B, if we can effectively transform any enumeration of the set B into an enumeration of the set A. There is a very close relationship between Turing reducibility and enumeration reducibility. To see this recall that an equivalent way of saying that $A \leq_{\mathrm{T}} B$ is to say that

⋆ Supported by a BNSF grant No. DMU 03/07/12.12.2011, by Sofia University Science Fund and by a Marie Curie international outgoing fellowship STRIDE (298471) within the 7th European Community Framework Programme.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 371–382, 2013.

both A and the complement of A can be enumerated using oracle B, or in other words $A \oplus \overline{A}$ is computably enumerable in B. Going deeper into the definition of *c.e. in*, we can say that there is a c.e. set W, whose members are pairs $\langle n, u\rangle$ of a natural number and a code for a finite set D_u, so that $A \oplus \overline{A}$ can be represented as the set $\left\{n \mid \exists u(\langle n, u\rangle \in W \ \&\ D_u \subseteq B \oplus \overline{B})\right\}$. In this definition the oracle set $B \oplus \overline{B}$ and the reduced set $A \oplus \overline{A}$ have a very specific structure: they both contain all of the positive information about a set in their even part and all the negative information about the same set in their odd part. If we drop this requirement on the structure of the reduced set, we obtain a definition of the relation *c.e. in*. If we consider a more general form of this definition, by dropping the structural requirements on both the oracle and reduced set, we obtain a definition of enumeration reducibility.

Definition 1. *Let A and B be sets of natural numbers. Then $A \leq_{\mathrm{e}} B$ if and only if there is a c.e. set W, such that $A = W(B) = \{x \mid \exists u(\langle x, u\rangle \in W \wedge D_u \subseteq B)\}$. The set W will be called an enumeration operator.*

From this analysis we immediately obtain the following relationship:

Proposition 1. *Let A and B be sets of natural numbers.*

1. *A is c.e. in B if and only if $A \leq_{\mathrm{e}} B \oplus \overline{B}$.*
2. *$A \leq_{\mathrm{T}} B$ if and only if $A \oplus \overline{A} \leq_{\mathrm{e}} B \oplus \overline{B}$.*

Enumeration reducibility carries its own structure. By identifying sets that are enumeration reducible to each other, i.e., enumeration equivalent, we obtain the set of enumeration degrees D_{e}. This is a partial order, where $\mathbf{d}_{\mathrm{e}}(A) \leq_{\mathrm{e}} \mathbf{d}_{\mathrm{e}}(B)$ if and only if $A \leq_{\mathrm{e}} B$. The least element $\mathbf{0}_{\mathrm{e}}$ consists of all computably enumerable sets. The way in which we obtained a least upper bound for Turing degrees, gives a least upper bound in the enumeration degrees, $\mathbf{d}_{\mathrm{e}}(A \oplus B) = \mathbf{d}_{\mathrm{e}}(A) \vee \mathbf{d}_{\mathrm{e}}(B)$. Thus $\mathcal{D}_{\mathrm{e}} = (D_{\mathrm{e}}, \leq_{\mathrm{e}}, \mathbf{0}_{\mathrm{e}}, \vee)$ is as well an upper semi-lattice.

Proposition 1 yields a standard embedding $\iota : D_{\mathrm{T}} \to D_{\mathrm{e}}$ of the Turing degrees into the enumeration degrees, which preserves the order and the least upper bound. This embedding is defined by $\iota(\mathbf{d}_{\mathrm{T}}(A)) = \mathbf{d}_{\mathrm{e}}(A \oplus \overline{A})$. The image of the Turing degrees under this embedding is the set $\mathcal{TOT}$ of total enumeration degrees. The substructure $\mathcal{TOT}$ of the total enumeration degrees plays a central role in the study of the enumeration degrees. Selman [21] showed that enumeration reducibility can be characterized by the following:

Theorem 1. *Let $A, B \subseteq \omega$. Then $A \leq_{\mathrm{e}} B$ if and only if*

$$\{X \mid B \text{ is c.e. in } X\} \subseteq \{X \mid A \text{ is c.e. in } X\}.$$

Translated in terms of enumeration reducibility using the first part of Proposition 1, this means that every enumeration degree is entirely characterized by the set of total degrees above it. In particular the set of total enumeration degrees is an automorphism base for the structure of the enumeration degrees.

We are still missing one ingredient for a complete analogy between the two structures: a jump operation, that agrees with the Turing jump under the standard embedding. The candidate for an analog of the halting set, which first comes to mind is $K^e_A = \{\langle n, e\rangle \mid n \in W_{\mathrm{e}}(A)\}$. A closer look at this set shows that this first choice is not satisfactory, as K^e_A is of the same enumeration degree as A. This is not too surprising, in the Turing case the reason that the halting set is not computable comes from the fact that the complement of the halting set diagonalizes against all possible computations. In contrast to the Turing degrees, the enumeration degrees are not closed under complement. This is why, to get a jump operation, which actually jumps up, we need to use the complement of K^e_A. Thus we define $A' = K^e_A \oplus \overline{K^e_A}$. We shall call sets A, such that $A \equiv_{\mathrm{e}} A \oplus \overline{A}$, *total sets*. Total sets are always members of total degrees. Examples of total sets are graphs of total functions, from where the term *total* originates, and as we just saw - jumps of sets, by definition. In the reverse direction we have the following jump inversion theorem by Soskov [28].

Theorem 2. *For every enumeration degree* $\mathbf{x}$ *there exists a total enumeration degree* $\mathbf{a}$*, such that* $\mathbf{x} \leq_{\mathrm{e}} \mathbf{a}$ *and* $\mathbf{x}' = \mathbf{a}'$*.*

It is not hard to see that this definition of the enumeration jump meets our requirements - it agrees with the Turing jump under the standard embedding ι. Thus the structure of the enumeration degrees provides a richer context in which we can study the structure of the Turing degrees. In this article we shall argue that there are cases in which this approach is useful, shedding light on phenomena that can be observed, but not well explained by viewing the structure of the Turing degrees alone.

1 Computable Model Theory

Our first example comes from a theorem by Coles, Downey and Slaman [3]. For every set of natural numbers A, consider the set $\mathcal{C}(A) = \{X \mid A \text{ is c.e. in } X\}$. It follows from a result by Richter [19] that sets of this form do not always have a member of least Turing degree. Coles, Downey and Slaman show that if you instead look at $\mathcal{C}(A)' = \{X' \mid A \text{ is c.e. in } X\}$, then this set always has a member of least Turing degree.

Theorem 3. *For every sets* A *the set:* $\mathcal{C}(A)' = \{X' \mid A \text{ is c.e. in } X\}$ *has a member of least degree.*

They call this degree *the least jump enumeration of* A. The proof constructs this least jump enumeration using forcing with finite conditions.

The motivation for this result comes from computable model theory, and the notion of degree spectrum, used to characterize different structures. Fix a countable relational structure $\mathcal{A} = (\mathbb{N}, R_1 \dots R_k)$. The *degree spectrum* of $\mathcal{A}$, denoted by $\mathrm{DS_T}(\mathcal{A})$, is the set of Turing degrees of the diagrams of structures $\mathcal{B} \cong \mathcal{A}$. If $\mathrm{DS_T}(\mathcal{A})$ has a least member, it is the (Turing) degree of $\mathcal{A}$. Sometimes

the spectrum of a structure does not behave nicely, and it is useful and more informative to consider the jump spectrum of a structure. The *jump spectrum* of $\mathcal{A}$ is $\mathrm{DS}'_{\mathrm{T}}(\mathcal{A}) = \{\mathbf{d}' \mid \mathbf{d} \in \mathrm{DS}_{\mathrm{T}}(\mathcal{A})\}$. If $\mathrm{DS}'_{\mathrm{T}}(\mathcal{A})$ has a least member, it is the (Turing) jump degree of $\mathcal{A}$.

Now we shall consider one particular instance of this characterization problem. A torsion free abelian group G of rank 1 is (up to isomorphism) a subgroup of the additive group of the rational numbers $(\mathbb{Q}, +, =)$. Fix such a group G. For every prime number p and element $a \in G$ we introduce the notion the p-height of a in G as follows:

$$h_p(a) = \begin{cases} \text{the largest } k, & \text{such that } p^k | a \text{ in } G; \\ \infty, & \text{if } \forall k (p^k | a \text{ in } G) \ . \end{cases}$$

Here $p^k | a$ in G if there exists $b \in G$ such that $p^k.b = a$.

Example: If $G = \mathbb{Q}$ then for all nonzero a and all p, $h_p(a) = \infty$, because for all k, $p^k.\frac{a}{p^k} = a$. If $G = \mathbb{Z}$ then for all nonzero a and all but finitely many p, $h_p(a) = 0$.

In fact it is not difficult to see that in any torsion free abelian group G of rank 1 we have that for nonzero elements a, b and for all but finitely many prime numbers p, $h_p(a) = h_p(b)$. We can therefore assign to every element $a \in G$ an infinite sequence of numbers: $\chi(a) = (h_{p_0}(a), h_{p_1}(a), \dots h_{p_n}(a), \dots)$, which we shall call the *characteristic* of a. So different elements of G have characteristics, which differ at only finitely many places, i.e., they are equivalent with respect to the equivalence relation in ω^ω which places sequences that differ in finitely many positions in the same equivalence class. The type of G, denoted $\chi(G)$ is the equivalence class of $\chi(a)$ for any $a \neq 0$ in G. Baer showed that first of all there are torsion free abelian groups of rank 1 of every possible type. But more importantly he proved that two torsion-free abelian groups of rank 1 are isomorphic if and only if they have the same type. Thus a torsion free abelian group of rank one is completely characterized by its type. And it turns out that this type can also be used to characterize the degree spectrum of every such group.

Let $S(G) = \{\langle i, j\rangle \mid j \leq$ the ith element of $\chi(G)\}$. Here we are actually taking any nonzero element of G and using its characteristic to represent G. Note that $S(G)$ does not depend on the choice of this element, up to many-one equivalence. Downey and Jockusch (see [3]) had shown that the degree spectrum of G is precisely $\{\mathbf{d}_{\mathrm{T}}(Y) \mid S(G)$ is c.e. in $Y\}$. Thus the result by Coles , Downey and Slaman can be restated as follows: Every torsion-free abelian group G of rank 1 has a jump degree. So we know this fact, but just from the world of Turing degrees it is not easy to explain the reason for this fact to be true. We now turn to the wider context of the enumeration degrees and view the same problem there.

Let us fix a countable relational structure $\mathcal{A} = (\mathbb{N}, R_1 \dots R_k)$. Soskov [29] introduced the notion of *enumeration degree spectrum*. The enumeration degree spectrum of $\mathcal{A}$, denoted by $\mathrm{DS_e}(\mathcal{A})$, is the set of e-degrees of the positive diagrams of structures $\mathcal{B} \cong \mathcal{A}$. If $\mathrm{DS_e}(\mathcal{A})$ has a least member, it is the (enumeration) degree

of $\mathcal{A}$. The enumeration jump degree spectrum of $\mathcal{A}$, denoted by $\mathrm{DS}'_{\mathrm{e}}(\mathcal{A})$, is the set of enumeration jumps of elements in $\mathrm{DS}_{\mathrm{e}}(\mathcal{A})$. If $\mathrm{DS}'_{\mathrm{e}}(\mathcal{A})$ has a least member, it is the (enumeration) jump degree of $\mathcal{A}$.

We immediately observe that just like Turing degrees embed into the enumeration degree via the standard embedding ι, we have a connection between the Turing degree spectrum and the enumeration degree spectrum of structures. Let $\mathcal{A}$ be any structure. Consider the structure $\mathcal{A}^+ = (\mathbb{N}, R_1, \overline{R_1} \ldots R_k, \overline{R_k})$. Firstly note that $\mathrm{DS}_{\mathrm{e}}(\mathcal{A}^+)$ consists entirely of total enumeration degrees. Secondly note that the positive diagram of A^+ is enumeration equivalent to the diagram of $\mathcal{A}$. Thus:

$$\mathrm{DS}_{\mathrm{e}}(\mathcal{A}^+) = \{\iota(\mathbf{a}) \mid \mathbf{a} \in \mathrm{DS}_{\mathrm{T}}(\mathcal{A})\} = \iota(\mathrm{DS}_{\mathrm{T}}(\mathcal{A})).$$

$\mathcal{A}$ has Turing degree $\mathbf{a}$ if and only if $\mathcal{A}^+$ has enumeration degree $\iota(\mathbf{a})$. Similarly $\mathrm{DS}'_{\mathrm{e}}(\mathcal{A}^+) = \{\iota(\mathbf{a})' \mid \mathbf{a} \in \mathrm{DS}_{\mathrm{T}}(\mathcal{A})\} = \{\iota(\mathbf{a}') \mid \mathbf{a} \in \mathrm{DS}_{\mathrm{T}}(\mathcal{A})\} = \iota(\mathrm{DS}'_{\mathrm{T}}(\mathcal{A}))$ and $\mathcal{A}$ has Turing jump degree $\mathbf{a}$ if and only if $\mathcal{A}^+$ has enumeration jump degree $\iota(\mathbf{a})$.

Soskov [29] considers the co-spectrum of a structure, the of enumeration degrees which are lower bounds to the enumeration degree spectrum of a structure. He shows that every countable ideal of enumeration degrees can be realized as the co-spectrum of a structure. The easier case of this theorem is when the ideal is a principal ideal. Soskov shows that every principal ideal $(\mathbf{a})$ is the co-spectrum of a torsion free abelian group of rank one, G. The top element of this ideal $\mathbf{a}$ is precisely the enumeration degree of the type of the group $S(G)$. He further makes the following observation.

Let G be a torsion-free abelian group of rank 1. The only relation in the language (apart from equality) is the graph of the total function $+$ which can also enumerate its negative instances thus $\mathrm{DS}_{\mathrm{e}}(G) = \{\iota(\mathbf{a}) \mid \mathbf{a} \in \mathrm{DS}_{\mathrm{T}}(G)\} = \left\{\mathbf{d}_{\mathrm{e}}(Y \oplus \overline{Y}) \mid S(G) \text{ is c.e. in } Y\right\}$. We can apply the first part of Proposition 1 to simplify the description of the enumeration degree spectrum of G. Denote $\mathbf{d}_{\mathrm{e}}(S(G))$ by $\mathbf{s}_G$—the type degree of G. The enumeration degree spectrum of G is:

$$\mathrm{DS}_{\mathrm{e}}(G) = \{\mathbf{a} \mid \mathbf{a} \in \mathcal{TOT} \ \&\ \mathbf{s}_G \leq_{\mathrm{e}} \mathbf{a}\}.$$

Now it is clear when a group has a degree and when not. The group G has an enumeration degree (and hence G has a Turing degree) if and only if the enumeration degree $\mathbf{s}_G$ is total. This follows from Selman's Theorem 1, as every enumeration degree is completely characterized by the set of total degrees above it. Furthermore, if G has an enumeration degree then it is precisely $\mathbf{s}_G$.

Consider the enumeration jump spectrum of G defined by

$$\mathrm{DS}'_{\mathrm{e}}(G) = \{\mathbf{a}' \mid \mathbf{a} \in \mathcal{TOT} \ \&\ \mathbf{s}_G \leq_{\mathrm{e}} \mathbf{a}\}.$$

By the monotonicity of the enumeration jump all members of $\mathrm{DS}'_{\mathrm{e}}$ are greater than or equal to $\mathbf{s}'_G$. By Soskov's Jump Inversion Theorem 2 there is a total degree $\mathbf{a} \geq \mathbf{s}_G$ such that $\mathbf{a}' = \mathbf{s}'_G$. Thus the enumeration jump spectrum of G always has a least element and it is $\mathbf{s}'_G$. This gives an alternative, more informative proof of the result by Coles, Downey and Slaman.

In recent work Soskov has shown a different application of properties of enumeration degrees to computable model theory. In this case the theme is jump

inversion of spectra of structures. It had been previously shown [12] that for every successor ordinal α if the Turing degree spectrum of a structure $\mathcal{A}$ lies in the cone above $\mathbf{0}^\alpha$ then there is a structure $\mathcal{B}$, whose αth jump spectrum is the spectrum of $\mathcal{A}$. The case for limit ordinals was not known. In [27] Soskov shows that ω-jump inversion is not always possible. The reason for this negative result is seen when one considers the co-set, the set of lower bounds, of the image of the ωth jump spectrum of a structure: it is shown that every member of this co-set is bounded by a total enumeration degree in the same co-set. Thus an example of a structure for which ω-jump inversion fails is given by a torsion free abelian group G of rank one, such that $\mathbf{d}_\mathrm{e}(S(G))$ is not total and above $\mathbf{0}_\mathrm{e}^\omega$.

2 Definability of the Jump Operator

A major theme in degree theory has been the definability theme. Among the most notable definability results in the Turing degrees is Shore and Slaman's [23] proof of the first order definability of the Turing jump operator. The proof of this theorem relies on two main ingredients—the definability of the double jump and a proof of the following structural property: for every Turing degree $\mathbf{a}$ which is not Δ_2^0 there is a Turing degree $\mathbf{g}$ such that $\mathbf{a} \vee \mathbf{g} = \mathbf{g}''$. Since any Δ_2^0 degree $\mathbf{a} \leq \mathbf{0}'_\mathrm{T}$ obviously does not satisfy this property ($\mathbf{a} \vee \mathbf{g} \leq \mathbf{0}'_\mathrm{T} \vee \mathbf{g} \leq \mathbf{g}' < \mathbf{g}''$), this gives a first order definition of $\mathbf{0}'_\mathrm{T}$. Relativizing, one gets the definition of the jump operation for every Turing degree degree modulo the definability of the double jump. The first ingredient is proved with a forcing construction, known as Kumabe-Slaman forcing. The definability of the double jump relies on Slaman and Woodin's [26] analysis of the automorphism group of the Turing degrees. Slaman and Woodin show that every countable relation on the Turing degrees can be coded by finitely many parameters, the Coding Theorem. This shows in particular that one can interpret the theory of second order arithmetic in the Turing degrees, an earlier result due to Simpson [24]. Then using methods from set theory, Slaman and Woodin show that every automorphism of the Turing degrees has an arithmetic presentation, that it is completely determined by its action on one element and that it fixes the cone above $\mathbf{0}''_\mathrm{T}$. Further they show that the biinterpretability conjecture is true modulo finitely many parameters and from this obtain that every relation in the Turing degrees which is induced by a degree invariant relation on 2^ω, which is definable in second order arithmetic, is definable in $\mathcal{D}_\mathrm{T}$ using parameters. If the relation is in addition invariant under automorphisms, then it is definable without parameters. This gives the definability of the double jump. As is stated in [23], this makes the definition of the double jump, and hence the jump operation, in the Turing degrees very far from natural. It cannot be stated in terms of a structural property, similar to the one used for the definition of $\mathbf{0}'_\mathrm{T}$ from the double jump. In the enumeration degrees this is not the case. And the reason for this is the existence of pairs of enumeration degrees with very special properties.

Recall Jockusch's [15] definition of a semi-recursive set. A set of natural numbers A is *semi-recursive* if there is a total computable selector function s_A, such

that $s_A(x,y) \in \{x,y\}$ and if $\{x,y\} \cap A \neq \varnothing$ then $s_A(x,y) \in A$. For example for every set A the set of finite binary strings, which are to the left of the characteristic function of A in the usual lexicographical ordering, $L_A = \{\sigma \in 2^{<\omega} \mid \sigma \leq_L \chi_A\}$, is semi-recursive. Given two finite binary strings the selector function always picks the one which is smaller with respect to $\leq_L$. Jockusch [15] showed that in fact for every non-computable set B there is a semi-recursive set $A \equiv_{\mathrm{T}} B$ such that both A and $\overline{A}$ are not c.e.

Arslanov, Cooper and Kalimullin [2] noticed that the enumeration degrees of semi-recursive sets have very interesting structural properties. If A is a semi-recursive set then $\mathbf{d}_{\mathrm{e}}(A)$ and $\mathbf{d}_{\mathrm{e}}(\overline{A})$ form a minimal pair in a very strong sense:

$$(\forall \mathbf{x} \in \mathcal{D}_{\mathrm{e}})((\mathbf{d}_{\mathrm{e}}(A) \vee \mathbf{x}) \wedge (\mathbf{d}_{\mathrm{e}}(\overline{A}) \vee \mathbf{x}) = \mathbf{x}).$$

It is then natural to wonder if this statement can be transformed into an if and only if statement. Kalimullin showed that it can, but first we need to introduce a generalization of semi-recursive sets.

Definition 2. *A pair of sets A, B is called a $\mathcal{K}$-pair if there is a c.e. set W, such that $A \times B \subseteq W$ and $\overline{A} \times \overline{B} \subseteq \overline{W}$.*

A trivial example of a $\mathcal{K}$-pair is $\{A, U\}$, where A is arbitrary and U is c.e. The c.e. set witnessing this is $\mathbb{N} \times U$. If A is a semi-recursive set, then $\{A, \overline{A}\}$ is a $\mathcal{K}$-pair, witnessed by the set $W = \{\langle x, y\rangle \mid s_A(x,y) = x\}$.

Kalimullin [16] showed that the notion of $\mathcal{K}$-pairs captures precisely the strong minimal pair property that semi-recursive sets have.

Theorem 4. *A pair of sets A, B are a $\mathcal{K}$-pair if and only if their enumeration degrees $\mathbf{a}$ and $\mathbf{b}$ satisfy: $\mathcal{K}(\mathbf{a}, \mathbf{b}) \leftrightharpoons (\forall \mathbf{x} \in \mathcal{D}_{\mathrm{e}})((\mathbf{a} \vee \mathbf{x}) \wedge (\mathbf{b} \vee \mathbf{x}) = \mathbf{x})$.*

$\mathcal{K}$-pairs are unique to the structure of the enumeration degrees. They are always quasi-minimal, i.e., the only total degree below either of them is $\mathbf{0}_{\mathrm{e}}$. A consequence of the existence of nontrivial $\mathcal{K}$-pairs in $\mathcal{D}_{\mathrm{e}}$, which are not below $\mathbf{0}'_{\mathrm{e}}$, is that the Slaman-Shore property used to define the Turing jump fails in the enumeration degrees: there is a degree $\mathbf{a} \not\leq_{\mathrm{e}} \mathbf{0}'_{\mathrm{e}}$, such that for every degree $\mathbf{g}$, $\mathbf{a} \vee \mathbf{g} <_{\mathrm{e}} \mathbf{g}''$. This shows that there are no $\mathcal{K}$-pairs in the structure of the Turing degrees. And furthermore there is no hope that the enumeration jump can be defined using a similar technique to the one used for the definition of the Turing jump.

Nevertheless Kalimullin [16] showed that the enumeration jump is first order definable by a very natural structural property. He showed that $\mathbf{0}'_{\mathrm{e}}$ is the largest degree which can be represented as the least upper bound of a triple $\mathbf{a}, \mathbf{b}, \mathbf{c}$, such that $\mathcal{K}(\mathbf{a}, \mathbf{b})$, $\mathcal{K}(\mathbf{b}, \mathbf{c})$ and $\mathcal{K}(\mathbf{c}, \mathbf{a})$. Using a relativized version of $\mathcal{K}$-pairs, he then obtained the definability of the jump operator in $\mathcal{D}_{\mathrm{e}}$. An alternative definition, which does not even use relativization is given by Ganchev and Soskova [7].

Theorem 5. *For every nonzero enumeration degree $\mathbf{u} \in \mathcal{D}_{\mathrm{e}}$, $\mathbf{u}'$ is the largest among all least upper bounds $\mathbf{a} \vee \mathbf{b}$ of nontrivial $\mathcal{K}$-pairs $\{\mathbf{a}, \mathbf{b}\}$, such that $\mathbf{a} \leq_{\mathrm{e}} \mathbf{u}$.*

As a consequence of this result we obtain that the set of total degrees $\mathbf{a}$ above $\mathbf{0}'_{\mathrm{e}}$ is also first order definable in $\mathcal{D}_{\mathrm{e}}$: a degree above $\mathbf{0}'_{\mathrm{e}}$ is total if and only if it is the jump of some enumeration degree.

3 The Local Structures

Two important substructures of the Turing degrees are the local structure of the Turing degrees $\mathcal{D}_{\mathrm{T}}(\leq \mathbf{0}'_{\mathrm{T}})$, consisting of all Δ^0_2 Turing degrees and its substructure $\mathcal{R}$, consisting of the of the computably enumerable degrees, i.e., Turing degrees which contain c.e. sets. The local structure of the enumeration degrees $\mathcal{D}_{\mathrm{e}}(\leq \mathbf{0}'_{\mathrm{e}})$, consists of all Σ^0_2 enumeration degrees. Recall that $\iota : \mathcal{D}_{\mathrm{T}} \to \mathcal{D}_{\mathrm{e}}$ preserves the jump, hence $\mathcal{D}_{\mathrm{T}}(\leq \mathbf{0}')$ embeds in $\mathcal{D}_{\mathrm{e}}(\leq \mathbf{0}'_{\mathrm{e}})$. It is not difficult to show that $\iota(\mathcal{R})$ is precisely the substructure of the Π^0_1 enumeration degrees. Thus the two local substructures of the Turing degrees live inside the local structure of the enumeration degrees. The three structures are different both in terms of the proper inclusions between their domains and in terms of their theories. Cooper [4] showed that $\mathcal{D}_{\mathrm{e}}(\leq \mathbf{0}'_{\mathrm{e}})$ is dense, hence not elementary equivalent to $\mathcal{D}_{\mathrm{T}}(\leq \mathbf{0}'_{\mathrm{T}})$. Ahmad [1] showed that the diamond can be embedded in $\mathcal{D}_{\mathrm{e}}(\leq \mathbf{0}'_{\mathrm{e}})$, hence the theory of the local structure of the enumeration degrees differs also from the theory of the c.e. Turing degrees.

One possible advantage of the embeddability of the Turing structures into $\mathcal{D}_{\mathrm{e}}(\leq \mathbf{0}'_{\mathrm{e}})$ is that algebraic results proved about the larger structure reveal more information about the smaller structures. We next describe an instance of this idea. A pair of degrees $\mathbf{a}, \mathbf{b}$ form a splitting of $\mathbf{c}$ if $\mathbf{a} < \mathbf{c}$, $\mathbf{b} < \mathbf{c}$ and $\mathbf{a} \vee \mathbf{b} = \mathbf{c}$. The existence of various splittings and non-splittings in a structure is important if one wishes to understand its two quantifier theory, i.e., the set two quantifier sentences true in the structure. The history of splitting results for the c.e. degree goes hand in hand with the evolution of the methods used to prove them, in particular the priority method. Harrington [13], generalizing a result by Lachlan [17], showed that there exists a c.e. Turing degree $\mathbf{a} < \mathbf{0}'_{\mathrm{T}}$, i.e., an incomplete c.e. degree, such that no pair of c.e. degrees above $\mathbf{a}$ are a splitting of $\mathbf{0}'_{\mathrm{T}}$. This came to be known as Harrington's non-splitting theorem and the method used in its proof as the monster priority method. Later on a similar method was used by Harrington and Shelah [14] to show the undecidability of the theory of the c.e. degree. Cooper and Soskova [5] pushed this result further to its limit, by considering this structural property within the local structure of the enumeration degrees. By Sack's splitting theorem, relativized to any Δ^0_2 Turing degree it follows that there is a Δ^0_2 splitting of $\mathbf{0}'_{\mathrm{T}}$ above any incomplete Turing degree. So the only question, which remained unanswered was if one can find a splitting consisting of one c.e. member and one Δ^0_2 member above any incomplete c.e. degree. Cooper and Soskova [5] showed that this is not true:

Theorem 6. *There exists a Π^0_1 enumeration degree $\mathbf{a} <_{\mathrm{e}} \mathbf{0}'_{\mathrm{e}}$, such that no pair of a Π^0_1 and Σ^0_2 e-degrees above $\mathbf{a}$ are a splitting of $\mathbf{0}'_{\mathrm{e}}$.*

So transferring back, via the inverse of the standard embedding ι, there is a c.e. Turing degree $\mathbf{a} <_{\mathrm{T}} \mathbf{0}'_{\mathrm{T}}$, such that no pair of a c.e. degree and a Δ^0_2 degree above $\mathbf{a}$ are a splitting of $\mathbf{0}'_{\mathrm{T}}$. This method was then extended in [31] to show that the full analog of Harrington's non-splitting theorem holds in the local structure of the enumeration degrees.

Theorem 7. *There is an incomplete Σ^0_2 enumeration degree above which $\mathbf{0}'_e$ cannot be split.*

The definability theme for the local structures has also been widely explored. Here one cannot talk about definability of the jump operator, but one can look at a hierarchy of classes of degrees defined in terms of the strength of their jumps.

Definition 3. *Let $n \geq 1$*

1. *The class of low$_n$ degrees is $L_n = \{\mathbf{a} \leq \mathbf{0}' \mid \mathbf{a}^n = \mathbf{0}^n\}$.*
2. *The class of high$_n$ degrees is $H_n = \{\mathbf{a} \leq \mathbf{0}' \mid \mathbf{a}^n = \mathbf{0}^{n+1}\}$*

The definability of the classes L_{n+1} and H_n for all $n \geq 1$ in $\mathcal{R}$ was shown by Nies, Shore and Slaman [18] and in $\mathcal{D}_T(\leq \mathbf{0}')$ by Shore[22]. The proofs of these theorems have the same flavor as the proof of the definability of the jump operation. First it is shown that the theory of first order arithmetic can be interpreted in $\mathcal{R}$ and $\mathcal{D}_T(\leq \mathbf{0}')$. This is then used to show that there is a definable way of mapping a degree $\mathbf{a}$ to a set A in every coded model of arithmetic so that $A'' \in \mathbf{a}''$. Thus every relation which is invariant under double jump and definable in first order arithmetic is definable in the corresponding degree structure. An additional argument, building on top if the first one is then devised to show that H_1 is also first order definable. These methods are however not sufficiently powerful to capture the definability of the low degrees, L_1, in either structure. The definability of L_1 in $\mathcal{D}_T(\leq \mathbf{0}'_T)$ or in $\mathcal{R}$ remains open.

Now lets turn to the local structure of the enumeration degrees $\mathcal{D}_e(\leq \mathbf{0}'_e)$. The first quite important question to settle concerns the complexity of the theory of the local structure, more precisely the question of whether or not one can interpret the theory of true arithmetic in $\mathcal{D}_e(\leq \mathbf{0}'_e)$. Slaman and Woodin [25] prove a coding theorem for the global structure, showing that $Th(\mathcal{D}_e)$ is computably isomorphic to second order arithmetic and a limited effective version of the coding theorem, that is enough to show that the local theory is undecidable, but not sufficient to characterize its complexity fully.

Theorem 8. *Every uniformly low antichain can be coded by parameters in the local structure $\mathcal{D}_e(\leq \mathbf{0}'_e)$.*

Ganchev and Soskova [9] notice that $\mathcal{K}$-pairs can be used to obtain precisely this kind of antichains. In fact every half of a nontrivial $\mathcal{K}$-pair is a uniform low bound to an antichain $\{\mathbf{a}_i\}_{i\in\omega}$ of e-degrees such that if $i \neq j$ then $\{\mathbf{a}_i, \mathbf{a}_j\}$ is a $\mathcal{K}$-pair. Thus if the property "$\mathbf{a}$ and $\mathbf{b}$ form a $\mathcal{K}$-pair" is first order definable in the local structure, then one could use Theorem 8 to show that the theory of first order arithmetic can be interpreted in $\mathcal{D}_e(\leq \mathbf{0}'_e)$.

Unfortunately Kalimullin's global definition of $\mathcal{K}$-pairs given in Theorem 4 starts with a universal quantifier. It is not clear and still open if this formula, interpreted in the local structure, is still a first order definition of the true $\mathcal{K}$-pairs. However, using an additional structural property of the members in $\mathcal{D}_e(\leq \mathbf{0}'_e)$, which is reminiscent of Theorem 7 and proved by a similar technique as the one used there, Ganchev and Soskova [8] prove that $\mathcal{K}$-pairs in the local structure of the enumeration degrees form a definable class.

Theorem 9. *There is a first order formula $\mathcal{LK}(x, y)$, which defines the $\mathcal{K}$-pairs in $\mathcal{D}_e(\leq \mathbf{0}'_e)$.*

This enables Ganchev and Soskova [10] to complete the original idea for interpreting arithmetic in the local structure.

Theorem 10. *The first order theory of $\mathcal{D}_e(\leq \mathbf{0}'_e)$ is computably isomorphic to first order theory of true arithmetic.*

The definability of $\mathcal{K}$-pairs in the local structure, turned out to be a key unlocking the definability of many other classes, including the downwards properly Σ^0_2 enumeration degrees, the upwards properly Σ^0_2 enumeration degrees. Extending a result of Giorgi, Sorbi and Yang [11], Ganchev and Soskova [7] show that in addition the class L_1 is first order definable, by a very natural property.

Theorem 11. *An enumeration degree $\mathbf{a}$ is low_1 if and only if every degree $\mathbf{b} \leq_e \mathbf{a}$ bounds a $\mathcal{K}$-pair.*

So in contrast to the Turing structures $\mathcal{D}_T(\leq \mathbf{0}'_T)$ and $\mathcal{R}$, where only L_1 cannot be shown to be definable, here in the local structure of the enumeration degrees we can show that this class is definable by a very natural property and it is not known whether or not the other jump classes are definable. The next result tips the scale in favor of $\mathcal{D}_e(\leq_e \mathbf{0}'_e)$ at least in terms of richness of definable classes.

Recall that a semi-recursive set A and its complement $\overline{A}$ form a special example of $\mathcal{K}$-pair. In terms of structure this $\mathcal{K}$-pair has one additional property—it is maximal. A $\mathcal{K}$-pair $\{\mathbf{a}, \mathbf{b}\}$ is maximal if there does not exists a $\mathcal{K}$-pair $\{\mathbf{c}, \mathbf{d}\}$, with $\mathbf{a} < \mathbf{c}$ or $\mathbf{b} < \mathbf{d}$. Ganchev and Soskova [7] show that in the local structure of the enumeration degrees maximality is precisely the structural notion which captures $\mathcal{K}$-pairs of a semi-recursive set and its complement. By Jockusch's theorem there is a non c.e. and non co-c.e. semi-recursive set in every Turing degree. In e-degree terms this means that a nonzero enumeration degree is total if and only if it can be represented as the least upper bound of a maximal $\mathcal{K}$-pair.

Theorem 12. *The class of total enumeration degrees is first order definable in $\mathcal{D}_e(\leq \mathbf{0}'_e)$.*

4 An Open Question

From what we have said so far, it follows that the set of total degrees that are comparable with $\mathbf{0}'_e$ is first order definable in $\mathcal{D}_e$. The open question of interest to us, first set by Rogers [20], concerns the definability of the class of all total enumeration degrees.

As noted above it follows from Selman's Theorem 1 that the class of total enumeration degrees is an automorphism base for the $\mathcal{D}_e$. Thus the definability of the total enumeration degrees would link the two major open problems: the existence of nontrivial automorphism for the Turing degrees and the existence

of a nontrivial automorphism of the enumeration degrees. A positive answer to the second question would yield a positive answer to the first question.

One possible solution to the definability of the total degrees would be to extend the characterization that proves the definability of the total degrees in the local structure. Jockusch's result for the existence of semi-recursive sets is valid for every Turing degree, hence one direction is already known to be true: every nonzero total set is enumeration equivalent to the join of the components of a maximal $\mathcal{K}$-pair. The first order definability of the total enumeration degrees would then follow, if it were true that maximality is the additional structural property needed to capture $\mathcal{K}$-pairs of the form $\{A, \overline{A}\}$.

This definition would then relate in a nice way to the definition of the enumeration jump given in Theorem 5. Consider the relation *c.e. in* between Turing degrees defined by: $\mathbf{x}$ is c.e. in $\mathbf{u}$ if there are sets $X \in \mathbf{x}$ and $U \in \mathbf{u}$, such that X is c.e. in U. Ganchev and Soskova [7] show that if $\mathbf{x}$ and $\mathbf{u}$ are Turing degrees such that $\mathbf{u}$ is nonzero then $\mathbf{x}$ is c.e. in $\mathbf{u}$ if and only if there is a $\mathcal{K}$-pair $\{A, \overline{A}\}$ such that $\mathbf{d}_{\mathrm{e}}(A) \leq_{\mathrm{e}} \iota(\mathbf{u})$ and $\iota(\mathbf{x}) = \mathbf{d}_{\mathrm{e}}(A) \vee \mathbf{d}_{\mathrm{e}}(\overline{A})$. Thus if every maximal $\mathcal{K}$-pair is of the form $\{A, \overline{A}\}$ for some A then the total degrees would be definable and the relation *c.e. in* between nonzero total degrees would be definable. The definition of the enumeration jump given in Theorem 5 restricted to the total degrees can then be read as $\mathbf{u}'$ is the largest total enumeration degree which is c.e. in $\mathbf{u}$. The natural definition of the total enumeration degrees proposed above remains currently out of reach.

In recent work Soskova [30] has investigated how much of the techniques for the analysis of the automorphism group of the Turing degrees by Slaman and Woodin [26] can be applied to study the automorphism group of the enumeration degrees. The obtained results bring the definition of the class of total enumeration degrees one step closer—namely a parameter away. The obtained results mirror the originals, it is shown that the automorphism group of the enumeration degrees is at most countable, every member has an arithmetic presentation and that there is an automorphism basis consisting of a single element. It is further shown that every relation in the enumeration degrees which is induced by a degree invariant relation on 2^{ω}, definable in second order arithmetic, is definable in $\mathcal{D}_{\mathrm{e}}$ using parameters. As a consequence we obtain that:

Theorem 13. *The class of total enumeration degrees is definable with parameters in the structure of the enumeration degrees.*

References

1. Ahmad, S.: Embedding the diamond in the Σ_2 enumeration degrees. J. Symbolic Logic 56, 195–212 (1991)
2. Arslanov, M.M., Cooper, S.B., Kalimullin, I.S.: Splitting properties of total enumeration degrees. Algebra and Logic 42, 1–13 (2003)
3. Coles, R., Downey, R., Slaman, T.: Every set has a least jump enumeration. Bulletin London Math. Soc. 62, 641–649 (2000)
4. 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)

5. Cooper, S.B., Soskova, M.I.: How enumeration reducibility yields extended Harrington non-splitting. J. Symbolic Logic 73, 634–655 (2008)
6. Friedberg, R.M., Rogers Jr., H.: Reducibility and completeness for sets of integers. Z. Math. Logik Grundlag. Math. 5, 117–125 (1959)
7. Ganchev, H.A., Soskova, M.I.: Definability via $\mathcal{K}$-pairs (submitted)
8. Ganchev, H.A., Soskova, M.I.: Cupping and definability in the local structure of the enumeration degrees. J. Symbolic Logic 77(1), 133–158 (2012)
9. Ganchev, H.A., Soskova, M.I.: Embedding distributive lattices in the Σ_2^0 enumeration degrees. J. Logic Comput. 22, 779–792 (2012)
10. Ganchev, H.A., Soskova, M.I.: Interpreting true arithmetic in the local structure of the enumeration degrees. To Appear in J. Symbolic Logic (2012)
11. Giorgi, M., Sorbi, A., Yang, Y.: Properly Σ_2^0 enumeration degrees and the high/low hierarchy. J. Symbolic Logic 71, 1125–1144 (2006)
12. Goncharov, S., Harizanov, V., Knight, J., McCoy, C., Miller, R., Solomon, R.: Enumerations in computable structure theory. Ann. Pure Appl. Logic 136, 219–236 (2005)
13. Harrington, L.: Understanding Lachlan's monster paper, Handwritten notes (1980)
14. Harrington, L., Shelah, S.: The undecidability of the recursively enumerable degrees. Bull. Symb. Logic 6(1), 79–80 (1982)
15. Jockusch, C.G.: Semirecursive sets and positive reducibility. Trans. Amer. Math. Soc. 131, 420–436 (1968)
16. Kalimullin, I.S.: Definability of the jump operator in the enumeration degrees. Journal of Mathematical Logic 3, 257–267 (2003)
17. Lachlan, A.H.: A recursively enumerable degree which will not split over all lesser ones. Ann. Math. Logic 9, 307–365 (1975)
18. Nies, A., Shore, R.A., Slaman, T.A.: Interpretability and definability in the recursively enumerable degrees. Proc. London Math. Soc. 77, 241–249 (1998)
19. Richter, L.J.: Degree structures: Local and global investigations. J. Symbolic Logic 46, 723–731 (1981)
20. Rogers Jr., H.: Theory of recursive functions and effective computability. McGraw-Hill Book Company, New York (1967)
21. Selman, A.L.: Arithmetical reducibilities I. Z. Math. Logik Grundlag. Math. 17, 335–350 (1971)
22. Shore, R.A.: Biinterpretability up to double jump in the degrees below 0 (to appear)
23. Shore, R.A., Slaman, T.A.: Defining the Turing jump. Math. Res. Lett. 6, 711–722 (1999)
24. Simpson, S.G.: First order theory of the degrees of recursive unsolvability. Annals of Mathematics 105, 121–139 (1977)
25. Slaman, T.A., Woodin, W.: Definability in the enumeration degrees. Arch. Math. Logic 36, ñ255–ñ267 (1997)
26. Slaman, T.A., Woodin, W.H.: Definability in degree structures (2005), `http://math.berkeley.edu/slaman/talks/sw.pdf`
27. Soskov, I.: A note on ω jump inversion of degree spectra of structures, This proceedings.
28. Soskov, I.N.: A jump inversion theorem for the enumeration jump. Arch. Math. Logic 39, 417–437 (2000)
29. Soskov, I.N.: Degree spectra and co-spectra of structures. Ann. Univ. Sofia 96, 45–68 (2004)
30. Soskova, M.I.: The automorphism group of the enumeration degrees, in preparation
31. Soskova, M.I.: A non-splitting theorem in the enumeration degrees. Ann. Pure Appl. Logic 160, 400–418 (2009)

Computing Game Strategies

Darko Stefanovic[1] and Milan N. Stojanovic[2]

[1] Department of Computer Science & Center for Biomedical Engineering, University of New Mexico, Albuquerque, NM 87131, U.S.A.
[2] Division of Experimental Therapeutics of the Department of Medicine & Department of Biomedical Engineering, Columbia University, New York, NY 10032, U.S.A.

Abstract. We revisit the problem of constructing strategies for simple position games. We derive a general, executable formalism for describing game rules and desired strategy properties. We present the outcomes for several variants of the familiar game of tic-tac-toe.

1 Introduction

Games of strategy, in particular simple board games, have often been used to demonstrate the capabilities of new computing devices. A famous example was Donald Michie's tic-tac-toe (noughts and crosses) with matchboxes [1]. To show off the versatility of solution-phase biochemical computation using deoxyribozyme logic gates beyond demonstrations of binary arithmetic [2, 3], we built two versions of automata that played tic-tac-toe [4, 5]. Each played the game perfectly following its rules, and implemented a strategy that never made mistakes, i.e., it won whenever possible. The strategy was chosen by us and specified as a mapping from the opponent's moves to the automaton's responses. It was then translated by hand into a set of Boolean formulae, and each formula was implemented by a set of deoxyribozyme logic gates. Each such set of gates became a circuit that operated in one of nine test wells, accepting the opponent's moves coded as DNA strands, and signalling the automaton's responses via fluorescence.

As the power of molecular computing increases, larger circuits become practical. For instance, we showed three-input gates with arbitrary assignment of polarity [3]. We have also made progress on cascading gates, allowing greater effective fan-in. Groups working on DNA computing using other basic chemistry, such as strand-displacement reactions, have also built circuits capable of interrogating a large number of inputs [6]. This prompted us to revisit the problem of designing games and games strategies that can be rendered in biochemistry. The desiderata for a demonstration of a biochemical game-playing automaton could be:

- the game is non-trivial, perhaps even interesting, but is simple to describe (has compact rules);

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 383–392, 2013.

- (mandatory) the game possesses a favorable strategy for one of the players, say, a strategy that never loses, and the logic design faithfully implements it;[1]
- (mandatory) the logic design is consistent with an implementation using biochemical logic gates;
- the complexity of (some of) the logic gates used is higher than before, to serve as a showcase for new capabilities, but not excessively high;
- the number of the gates needed is manageable, to contain cost;
- the amount of laboratory work to exhaustively test the automaton is manageable (which means the strategy covers a reasonably small number of game plays).

Of the above, using biochemical logic gates is the most serious constraint, for we consider gates that *cannot be turned off once activated*, as we shall explain shortly. Intuition suggests that this makes querying sequences of inputs difficult, and when, through trial and error, we had found a working strategy for tic-tac-toe [4] we were rather surprised. Here we tackle the two mandatory criteria above by developing a *systematic, constructive procedure* to determine whether, given a game, there exists a favorable strategy computable in biochemical logic. We consider only *position games*—games played on a board with a finite number of fields, in which the two players alternate claiming the fields, these claims are permanent, and the winner is decided depending on whether certain board configurations are reached.

In the following, we carefully define the notion of a game of strategy, its translation to logic, and the idiosyncratic realization of the logic in the wet lab (Section 2); first steps in this direction were taken by Andrews [7]. We now elaborate these definitions (Section 2), and then derive a computational procedure for generating strategies (Section 3). Finally we present the results of the procedure for several games in the tic-tac-toe family (Section 4). Perhaps surprisingly, the difficulty of a game, measured by the size or other complexity metrics applied to the results found, is not easy to predict from its apparent properties. Furthermore, the procedure may alternatively prove that no suitable strategy exists, and we exhibit such examples. The very existence of a solution also appears to be a subtle property and hard to predict. Our approach to computing strategies is effective on small examples; however, it does not assume or exploit any structure in the games and therefore does not scale to games with many fields.

2 Games and Their Biochemical Representation

Deoxyribozyme Logic. We are interested in circuits made out of deoxyribozyme logic gates. Deoxyribozymes are catalytically active single-stranded DNA; the catalytic function we exploit is that of cleaving other single-stranded DNA [8]. By adding a stem-loop control module to a basic deoxyribozyme, we

[1] Whether particular games possess a favorable strategy, and for which player, is a matter of keen study by combinatorial game theorists, and many familiar games, such as Connect Four, have only been solved in the last two decades.

are able to switch the complex from an inactive state to the catalytically active state by providing a control input, itself a single-stranded DNA [9]. We have built gates containing two such control modules, as well as a control module with the opposite sense. Interpreting the function of the complex as a digital logic gate, we were able to implement the Boolean functions id, $\neg$, $\wedge$, $\lambda xy.x \wedge \neg y$, and $\lambda xyz.x \wedge y \wedge \neg z$, i.e., some elementary conjunctions.

The rest of this paper can be understood without reference to biochemistry, except for the following points. A catalytic gate in its active state continuously cleaves substrate molecules into product molecules. It is the buildup of the product that we are able to observe, usually by fluorescence techniques. Thus, viewed as a dynamic system, an active gate is an integrator. In laboratory experiments, gates are run sufficiently long so that enough product builds up for reliable detection, thus the dynamic aspects need not concern us. However, once a product molecules has been created, it never goes away; therefore, even if the gate molecule itself is made inactive again (possible but nontrivial), the fluorescence of the products remains. In effect, the gate once on cannot be switched off, and this means the gate is usable for one-shot computation only. While this may be a disadvantage in some contexts, it is a good match to position games in which moves are permanent.

It is very easy to make a circuit with multiple conjunctions in a disjunction, simply by arranging several gates in the same solution to bind the same substrate and thus yield the same product—the activation of any gate suffices to build up detectable fluorescence. By the same token, *subsequent activation of another gate has no visible effect*, and this we can exploit to hide moves.

From Strategy to Boolean Formulae. We restrict attention to deterministic games of perfect information in which two players alternate, in each move claiming a field of a finite-size board, and these claims are permanent. Many widely played games fall into this category, including tic-tac-toe, Hex, and Connect Four. A strategy for one of the players (first or second) is a self-consistent map from board configurations to that player's moves. The map is only defined for board configurations in which it is that player's turn to move; furthermore by "self-consistent" we mean that the map needs to be defined exactly for those board configurations that are reachable by following the strategy itself.

A Boolean implementation of a strategy is a set of formulae, one for each field, to turn on or off the automaton's response in that field, as a function of the current board configuration. We now place further restrictions on the Boolean implementation in light of its subsequent realization in biochemical logic. Each field is represented as a separate well (test tube) which implements one formula in disjunctive normal form as a circuit consisting of several logic gates working in parallel. Opponent's moves are the inputs to the formulae and are presented as molecules keyed to each field but added to all the wells; this is the only means by which information is shared between the wells. Thus, the circuit in one well can query all the opponent's inputs, but not the outputs of the circuits in the remaining wells: the Boolean implementation of a strategy must be a function of the opponent's inputs alone, not the complete board configuration. We can

now explain the idea of move hiding: we can tolerate the spurious activation of a gate upon the setting of an input (i.e., addition of a molecule), so long as, on that game path, another gate in the same well has already been activated by a previously set input (at an earlier move of the opponent).

A further distinction is possible depending on how we associate opponent's moves with Boolean inputs. A rich encoding [5] maps each ⟨field, turn number⟩ pair to a distinct input, so the formulae can query the order of the opponent's moves. In a compact encoding, each field is keyed to just one input [4], so the formulae can only query the current set of the opponent's moves but not their order. In the following we assume a compact encoding.

The Notion of Isomorphism. In our first tic-tac-toe automaton, we restricted our opponent's first move; the automaton went first and claimed the center field  , upon which the two allowed moves for the opponent were the top left corner  or the left side  . We justified this restriction by an appeal to symmetry—all corners are alike, and all sides are alike. But we could have gone further. Indeed, in treating a game on a square board it is natural to consider board configurations as distinct only up to rotations and perhaps reflections. In general, this notion of board isomorphism is arbitrary, and represents a chosen interpretation of the game. For the 3×3 tic-tac-toe board, the equivalence classes under the natural isomorphism (rotation and reflection) contain at most 8 board configurations each (some, sparsely occupied, have just 4, 2, or 1).

To play a game under a notion of isomorphism, only one representative of each equivalence class under isomorphism is used. Given a board configuration, the appropriate player chooses an unoccupied field and claims it. The resulting board's equivalence class is looked up, and the representative of that class becomes the next board configuration. The new board configuration is a reshuffled version of the old one, which humans can grasp easily for geometrically motivated isomorphisms, but find confusing for arbitrary ones. However, it is not clear how to effect reshuffling in our laboratory protocol. Rotating a well plate is possible but it would be an external non-molecular computation; making a mirror image of a plate would be rather tricky. Reshuffling using molecular computation would require wells to turn off after they were on, and we must avoid that. But what if, somehow, reshuffling didn't actually have to reshuffle at all?

That is the approach we take: if the representative of each equivalence class is chosen deliberately, it may be possible to arrange for the new board configuration always to agree with the old one—for all possible moves from all possible board configuration, as prescribed by a strategy. Whether this "deliberate choice" of representatives actually exists for a given game turns out to be a complex problem in its own right. If it does, then it will *appear* to the human opponent as if the game was played naturally, without reshuffling—the only difference being that from a given board configuration, when it is the human's turn, not all the usual fields are permitted, but rather only at most one for each equivalence class under isomorphism. It is as if the human were supplied with a *rule book* that listed all board configurations and how one may move among them. This rule

book is fair, as it simply projects the adopted notion of isomorphism onto the automaton's strategy; it is a generalization of our old restriction "on 1st move, go only to top left or left side".

3 From Games to Strategies: Problem Graph and Constraints

To analyze a game we first construct a problem graph that captures all possible game plays; its vertices are board configurations (up to isomorphism), and its edges are moves. We then choose a subgraph that represents a feasible strategy; to find one, we translate the problem graph into a constraint satisfaction problem and solve with an external tool.

We formalize the *rules of a game* parametrically, using here the names of these parameters in our Haskell implementation: A numFields value gives the board size. An initialBoard, commonly empty, gives the root for the problem graph. A mkSucc function maps a board to the set of legal successors, claiming one of the unoccupied fields. A hasMotif function computes if the board is final (commonly because one player's claimed fields form a motif, such as a three-in-a-row). A victoryMode value distinguishes normal game play (achieving a motif) from misère play (avoiding a motif). A fieldPerms list enumerates all permutations of the sequence $1 \cdots$ numFields that are to underlie the notion of board isomorphism.

Further parameters describe the *strategy* sought. A playerMode value selects whether the strategy is for the first or the second player. An outcomeMode governs which final outcomes the strategy must guarantee, commonly a win, or either a win or a draw.

To construct the problem graph, we compute all board configurations reachable from initialBoard using mkSucc, collapsing isomorphic ones. Obviously, the need to construct the graph explicitly limits our approach to fairly small games. After labelling final configurations as win, draw, or loss, we can solve the game, i.e., determine whether either player has a winning strategy, by a simple bottom up propagation.

We have not been able to express feasibility of strategies by any such simple computation, so we resort to a constraint-based approach. By inspection of the problem graph, we calculate all the constraints that must be met. Here we present one key example, clauses that express the feasibility of a strategy (deferring to a longer report the full description of all types of clauses used). This constraint captures the idea of avoiding conflicts in the strategy while exploiting move hiding, as first suggested in Ref. [4] and further elucidated in Andrews' thesis [7]:

> Consider vertices at a level of the graph where the automaton moves, i.e., where a choice needs to be made which single outgoing edge to keep for the strategy. For all pairs of these vertices, we compute the potential opponent move sets they have in common. Then for each such opponent move set, we formulate a clause stating that *if both vertices are kept, and if at both vertices the equivalence class representative under isomorphism*

is chosen such as to agree with the opponent move set, then either the outgoing moves from both vertices are labelled with the same field, or the outgoing edges from both fields are directed to the same vertex.

In our current implementation, we express constraints directly in the Boolean domain and invoke an external SAT solver, Minisat [10]. If there is a solution, we interpret the satisfying assignment to recover the chosen vertices and edges to be kept, and the chosen equivalence class representatives. This forms the chosen strategy. We then read out the automaton response formulae and subject them to a step of Boolean minimization. For games with a notion of isomorphism we also read out the rule book for the opponent. We have written an independent validation procedure that follows all paths in the strategy and checks that the formulae evaluate correctly; the generated trace also represents the high-level protocol to be followed by the laboratory technician validating a biochemical realization of the strategy.

4 Results

The procedure described is effective and fast (completes in less than a minute of CPU time) at analyzing games on modest-size boards, such as tic-tac-toe. Table 1 is a summary of our initial findings. We examined eight game variants based on tic-tac-toe. All use the standard 3×3 board and three-in-a-row victory motif, and strategies aim for a win or a draw (since in each case, in game-theoretic terms, the game is a draw). As shown in the leftmost pane of the table, we vary the victory criterion (either achieving or avoiding the motif), the player for whom the strategy is sought (1st or 2nd), and the notion of board isomorphism (we consider only two options, the natural eight-way symmetry (rotations and reflection), or no isomorphism). The next pane gives the size of the problem graph from which the constraints are constructed. The third pane gives the size of the SAT instance generated; we include the ratio of the number of clauses to the number of variables as a tentative indicator of instance hardness. If there is no solution, the remaining columns have a dash. Otherwise, there is a solution for the strategy, and the fourth pane describes the circuit we obtained for it, after Boolean minimization, with the number of logic gates used and the maximum fan-in (number of input literals) needed. Finally, the rightmost column gives the number of play paths the chosen strategy entails.

Of the eight variants, five have solutions. In Figures 1 and 2 we focus on the first two rows of the table, first player normal play, and we show the problem graphs, derived solution strategies, and the Boolean formulae that implement them. The graphs are scaled down, and too small to be read in print, so we intend them primarily to convey a gestalt impression. However, they can be zoomed into to consult the vertex numbers, which match between the problem and the solution. The edges are labelled with the field being played into. The Boolean formulae map inputs i_0–i_8 to outputs o_0–o_8; the fields are numbered according to the schema $\begin{smallmatrix}0&1&2\\3&4&5\\6&7&8\end{smallmatrix}$. As expected, without isomorphism (Figure 2) the problem graph

Table 1. Analysis for eight problem variants based on tic-tac-toe

victory	player	isomorphism	vertices	edges	variables	clauses	ratio	gates	fan-in	paths
normal	1st	natural	539	1212	45878	279798	6.1	23	3	18
normal	1st	none	3878	9331	338345	1623395	4.8	65	4	155
normal	2nd	natural	290	530	18642	119045	6.4	—	—	—
normal	2nd	none	2094	4085	126773	625558	4.9	—	—	—
misère	1st	natural	45	60	1749	11875	6.8	—	—	—
misère	1st	none	294	433	13951	65139	4.7	9	1	216
misère	2nd	natural	546	1395	43239	283159	6.6	39	5	121
misère	2nd	none	3950	10849	305599	1629398	5.3	120	8	503

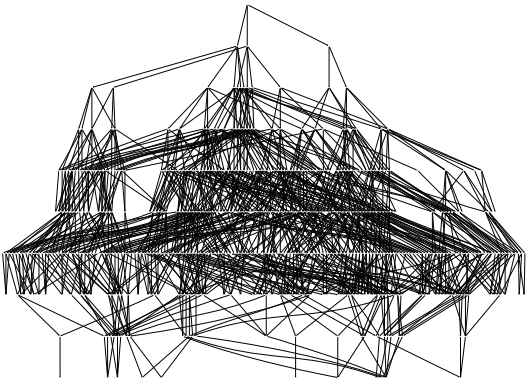
is nearly eight times bigger; the chosen strategy requires three times as many gates and nine times as many paths for validation. This comparison shows the utility of a notion of isomorphism as a disciplined way of reducing the materials and labor cost of the implementation.

On the other hand, it is instructive to compare the two variants for first player misère play (fifth and sixth row of Table 1). Both have small problem graphs, and the variant with no isomorphism has an exceptionally simple solution (which, after some thought, should be anticipated: one constant 1 in the center, and nine YES-gates surrounding it). Yet, the version with isomorphism has no solution. In this case, the additional constraints that result from merging distinct board configurations into single equivalence class representatives are responsible for the lack of a solution.

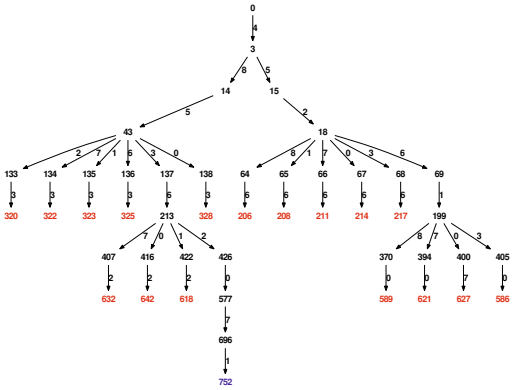
5 Discussion

The space of strategies for position games can be enormous. As shown by Andrews [7], for tic-tac-toe (normal play, first player, no isomorphism) it contains $1.24 \cdot 10^{124}$ strategies, of which $2.6 \cdot 10^{103}$ are favorable. This is in contrast to just a quarter million possible plays. In a custom Monte Carlo approach, Andrews constructed strategies for this game that were feasible and favorable by intelligent design (effectively, by enumerating the local constraint space on the fly as the strategy was constructed level by level) and checked them for implementability within particular technologies (available elementary AND gates). Sampling 50 million strategies, he quickly found five-literal implementations, but could not find a four-literal one; our speculation at the time was that gates with five inputs of arbitrary polarity were needed. We were surprised that the very first strategy generated by our new method was implementable with just up to four-input gates. A possible topic for future work is using an all-solutions constraint solver to systematically sample the solution space.

Our new method improves on the previous approach—it is general and applicable to arbitrary games; it is modular, separating play rules from victory modes from strategy requirements; and it is purely declarative, the constraint-solving



(a) Problem graph

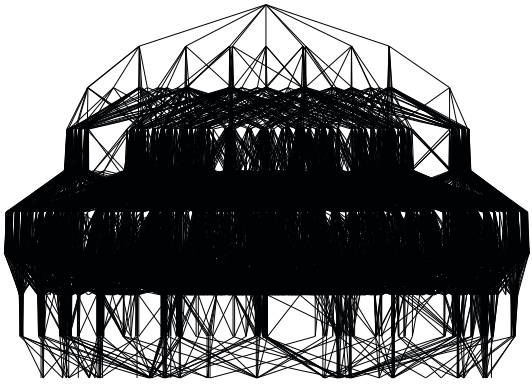


(b) Selected strategy (red leaves: win; blue leaves: draw)

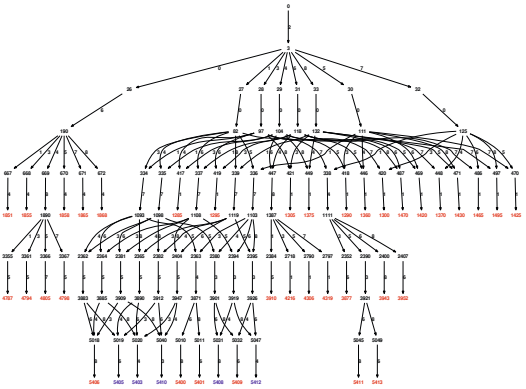
$o_0 = (i_3 \wedge i_6) \vee (i_6 \wedge i_7) \vee (i_2 \wedge i_3) \vee (i_5 \wedge i_6 \wedge i_8)$
$o_1 = (i_6 \wedge \neg i_8) \vee (i_2 \wedge i_7)$
$o_2 = i_5 \vee (i_0 \wedge i_3) \vee (i_1 \wedge i_3) \vee (\neg i_2 \wedge i_3 \wedge i_7)$
$o_3 = (\neg i_5 \wedge i_6) \vee (i_2 \wedge \neg i_3) \vee (\neg i_3 \wedge i_7 \wedge i_8) \vee (i_0 \wedge \neg i_3 \wedge i_8) \vee (i_1 \wedge \neg i_3 \wedge i_8)$
$o_4 = 1$
$o_5 = (\neg i_5 \wedge i_8)$
$o_6 = (i_3 \wedge \neg i_6) \vee (i_1 \wedge \neg i_8) \vee (i_5 \wedge \neg i_6 \wedge i_8) \vee (\neg i_6 \wedge i_7 \wedge \neg i_8) \vee (i_0 \wedge \neg i_6 \wedge \neg i_8)$
$o_7 = (i_0 \wedge i_6)$
$o_8 = 0$

(c) Boolean formulae implementing the strategy

Fig. 1. Tic-tac-toe: Normal play, first player, isomorphism is rotation and reflection



(a) Problem graph



(b) Selected strategy (red leaves: win; blue leaves: draw)

$$
\begin{aligned}
o_0 &= i_6 \vee (\neg i_0 \wedge i_1) \vee (\neg i_0 \wedge i_4) \vee (\neg i_0 \wedge i_3) \vee (\neg i_0 \wedge i_5) \vee (\neg i_0 \wedge i_8) \vee (\neg i_0 \wedge i_7)\\
o_1 &= (\neg i_1 \wedge i_4 \wedge i_8) \vee (\neg i_1 \wedge i_3 \wedge i_5) \vee (\neg i_1 \wedge i_3 \wedge i_8) \vee (\neg i_1 \wedge i_5 \wedge i_8) \vee (\neg i_1 \wedge i_3 \wedge i_6)\\
&\quad \vee (\neg i_1 \wedge i_5 \wedge i_6) \vee (\neg i_1 \wedge i_6 \wedge i_8) \vee (\neg i_1 \wedge i_3 \wedge i_7) \vee (\neg i_1 \wedge i_5 \wedge i_7) \vee (\neg i_1 \wedge i_7 \wedge i_8)\\
&\quad \vee (\neg i_1 \wedge i_6 \wedge i_7) \vee (\neg i_0 \wedge i_4 \wedge i_7) \vee (\neg i_0 \wedge \neg i_1 \wedge i_3 \wedge i_4) \vee (\neg i_0 \wedge \neg i_1 \wedge i_4 \wedge i_5)\\
o_2 &= 1\\
o_3 &= (i_1 \wedge i_4 \wedge i_5) \vee (i_4 \wedge i_6 \wedge i_8) \vee (i_1 \wedge i_5 \wedge i_7) \vee (i_1 \wedge \neg i_3 \wedge i_5 \wedge i_8) \vee (i_1 \wedge \neg i_3 \wedge i_5 \wedge i_6)\\
o_4 &= (i_1 \wedge i_7) \vee (i_0 \wedge i_8) \vee (i_1 \wedge i_3 \wedge i_5) \vee (i_3 \wedge i_6 \wedge i_8) \vee (i_5 \wedge i_6 \wedge i_8) \vee (i_0 \wedge i_1 \wedge \neg i_4)\\
&\quad \vee (i_0 \wedge i_3 \wedge \neg i_4) \vee (i_0 \wedge \neg i_4 \wedge i_5) \vee (i_0 \wedge \neg i_4 \wedge i_7)\\
o_5 &= (i_1 \wedge i_3 \wedge i_4) \vee (i_0 \wedge i_1 \wedge i_4) \vee (i_0 \wedge i_3 \wedge i_4) \vee (i_0 \wedge i_4 \wedge i_7) \vee (i_1 \wedge i_4 \wedge \neg i_5 \wedge i_8)\\
&\quad \vee (i_1 \wedge i_4 \wedge \neg i_5 \wedge i_6) \vee (i_1 \wedge i_3 \wedge \neg i_5 \wedge i_8) \vee (i_1 \wedge i_3 \wedge \neg i_5 \wedge i_6) \vee (i_1 \wedge \neg i_5 \wedge i_6 \wedge i_8)\\
o_6 &= i_0 \vee (i_3 \wedge i_4 \wedge i_8) \vee (i_4 \wedge i_5 \wedge i_8) \vee (i_3 \wedge i_5 \wedge i_8) \vee (i_1 \wedge i_3 \wedge i_7) \vee (i_1 \wedge i_7 \wedge i_8)\\
o_7 &= (i_0 \wedge i_4 \wedge i_5) \vee (i_1 \wedge i_3 \wedge \neg i_7) \vee (i_1 \wedge i_5 \wedge \neg i_7) \vee (i_1 \wedge \neg i_7 \wedge i_8) \vee (\neg i_0 \wedge i_1 \wedge i_4 \wedge \neg i_6)\\
&\quad \vee (i_1 \wedge \neg i_4 \wedge i_6 \wedge \neg i_7)\\
o_8 &= (i_0 \wedge i_4) \vee (i_3 \wedge i_4 \wedge i_6) \vee (i_4 \wedge i_5 \wedge i_6) \vee (i_3 \wedge i_5 \wedge i_6) \vee (i_1 \wedge i_6 \wedge i_7) \vee (\neg i_1 \wedge i_4 \wedge i_6)
\end{aligned}
$$

(c) Boolean formulae implementing the strategy

Fig. 2. Tic-tac-toe: Normal play, first player, no isomorphism

procedural details having been offloaded to an external solver. The principal conceptual advance, however, is the principled and integrated treatment of game symmetries via the notion of board isomorphism.

We are currently investigating how well our approach will scale to larger games. The problem graphs grow quickly with the number of fields, and thus also the SAT instances we generate, surpassing the capacity of off-the-shelf solvers at about 15 fields in the board. It is not clear whether these instances also become intrinsically harder, and this seems to be subtly dependent on the game rules.

Related Work. The literature on combinatorial games is vast and scattered, though the periodical *Games of No Chance* (Mathematical Sciences Research Institute) is a focal point. We have not been able to find prior work on the question of computability of a strategy in (limited) logic. There is a growing body of work in molecular computing; our game analysis approach can be applied to various logic-gate implementations, adapting the constraints to match the particulars of those implementations. Most demonstrations in molecular computing so far have been small to medium-scale. We hope that the development of design tools will facilitate the construction of convincing large circuits.

Acknowledgments. We are grateful to Ben Andrews for many inspirational conversations. We sincerely thank the conference reviewers for their incisive comments, which helped us to clarify the presentation, we hope. This material is based upon work supported by the National Science Foundation under grant 1028238.

References

1. Michie, D.: Trial and error. In: Science Survey, part 2, pp. 129–145. Penguin, Harmondsworth (1961)
2. Stojanovic, M.N., Stefanovic, D.: Deoxyribozyme-based half adder. Journal of the American Chemical Society 125(22), 6673–6676 (2003)
3. Lederman, H., Macdonald, J., Stefanovic, D., Stojanovic, M.N.: Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. Biochemistry 45(4), 1194–1199 (2006)
4. Stojanovic, M.N., Stefanovic, D.: A deoxyribozyme-based molecular automaton. Nature Biotechnology 21(9), 1069–1074 (2003)
5. Macdonald, J., Li, Y., Sutovic, M., Lederman, H., Pendri, K., Lu, W., Andrews, B.L., Stefanovic, D., Stojanovic, M.N.: Medium scale integration of molecular logic gates in an automaton. Nano Letters 6(11), 2598–2603 (2006)
6. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332, 1196–1201 (2011)
7. Andrews, B.: Games, strategies, and boolean formula manipulation. Master's thesis, University of New Mexico (December 2005)
8. Breaker, R.R., Joyce, G.F.: A DNA enzyme that cleaves RNA. Chemistry and Biology 1, 223–229 (1994)
9. Stojanovic, M.N., Mitchell, T.E., Stefanovic, D.: Deoxyribozyme-based logic gates. Journal of the American Chemical Society 124(14), 3555–3561 (2002)
10. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)

On Processes and Structures*

Alexey Stukachev[1,2]

[1] Department of Mechanics and Mathematics, Novosibirsk State University, Russia
[2] Department of Mathematical Logic, Sobolev Institute of Mathematics, Novosibirsk, Russia
aistu@math.nsc.ru

Abstract. We consider an approach to computability in admissible sets based on a general notion of computable process, with Σ-predicates and Σ-operators as special cases, inspired by ideas from the Ershov-Scott theory of approximation spaces. We present some results from different topics in generalized computability, including reducibilities on admissible sets and structures, general notion of a jump, and computable analysis (more exactly, computability over the reals), obtained with the help of this approach, and state some open questions.

1 Introduction

The present paper is a continuation of [15,16,18,19] and especially [17]. It is motivated by three questions in generalized computability which turned out to be closely connected:

1. For a given admissible set $\mathbb{A}$, what is *computability on* $\mathbb{A}$?
2. What is a *jump* of a given computability or structure?
3. How to define a *measure (degree) of complexity* of a given structure or admissible set?

First, a usual understanding of computability theory on admissible sets as the study of Σ-definable objects (predicates, relations, subsets, etc.) is too stringent when various formalizations of computable processes are considered, like in the case of mass problems with reducibilities on them via computable operators in the style of Medvedev, Muchnik and Dyment [15]. Degrees of presentability, being a special case of mass problems on admissible sets in general, provide natural tools for measuring the constructive complexity of structures, so the task of extending this approach is quite actual. Concerning this problem, we present a framework which allows to study both Σ-predicates and Σ-operators from a common viewpoint based on a notion of (constructive) process on a domain of computation. Computability on admissible sets is generated from this point of

* This work was supported by the Ministry of Education and Science of Russian Federation (project 8227), the Russian Foundation for Basic Research (grants 11-01-00688-a, 13-01-91001-ANF-a), and the State Maintenance Program for the Leading Scientific Schools of the Russian Federation (grant N.Sh.-276.2012.1).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 393–402, 2013.

view by classes Σ-processes on Σ-admissible families (in the paper such pairs are called *computability components*). It turns out that these notions in case of HF-computabilities are naturally connected with another motivating problem—the study of jumps of computabilities and jumps of structures. We apply the methods constructed in the first part to formulate a general notion of jump of computability, extending the notions of Turing jump and hyperjump of a set of natural numbers, and the notion of Σ-jump of a structure, considered recently by different authors [7,9,12,18].

The results obtained in the paper explain how new notions and insights can be helpful in quite different areas. We propose, formalize and study the following approaches to the questions stated above:

1. Computability on $\mathbb{A}$ is a family of its components, with each component defined as a pair: a family of objects—subsets of A, and a class of Σ-processes acting on them, with the property that every finite fragment of an output can be obtained using some finite fragments of the arguments and resources.
2. The jump of a component of computability on $\mathbb{A}$ is a structure with the domain consisting of its objects and the diagram is obtained by the *termination* of its processes.
3. The measure of (relative) complexity of a structure is given by its degrees in semilattices of Σ-degrees and degrees of presentability.

There are three main classes of objects considered in this paper: *admissible sets*, *computabilities*, generated by admissible sets, and structures, obtained as *jumps* of computabilities. Note that a structure, in turn, generate admissible sets like HF- or HYP-superstructures, so we can speak about, say, HF-computability over that structure.

The new results in this paper are as follows. First, concerning computabilities on admissible sets in general, we obtain a strengthening of the result of Morozov [8] which states that a certain reducibility between admissible sets implies an embedding of computable objects (i.e., Σ-predicates) on them. Namely, we prove that this reducibility implies much more general fact: there exists an embedding of computabilities on these admissible sets. We show how the theory of admissible sets and , in particular, the notion of Σ-admissible family is connected with the Ershov–Scott theory of approximation spaces.

Second, we present results connecting $0^\diamond$, the jump of the maximal component of HF-computability over 0, with the reals in both algebraical and topological settings. We show that these structures are constructible from, respectively, $(0^\diamond)'$ and $0^\diamond$. These results can be considered as a first step in the study of jump inversions in general, established for the minimal component of HF-computability over arbitrary structure in the strongest possible form [18,19].

The proofs of new results are just outlined, to present the main ideas. In the text, we use definitions and notations from [6,2]. In particular, for an admissible set $\mathbb{A}$, $U^{\mathbb{A}}$ denotes the set of urelements from $\mathbb{A}$, and A^* denotes the set of elements of $\mathbb{A}$ which are sets (i.e., not urelements).

2 Processes and Approximation Spaces

We shall use very basic definitions and facts from the Ershov–Scott theory of approximation spaces.

Definition 1. By a *p-domain* we mean a triple $\mathcal{X} = \langle X, F, \leqslant \rangle$, where X is a topological T_0 space which is a φ-space [3,5], $F \subseteq X$ is the basic subset of *finite elements*, and $\leqslant$ is the specialization order on X.

Every φ-space is an A-space [5]. In particular, we shall use the property of p-domains established in [3] for A-spaces in general: every element $x \in X$ is a limit of its F-approximations:

$$x = \sup\{a \in F \mid a \leqslant x\}.$$

The set F can be viewed as the set of finite approximations for elements from X, and the specialization order $\leqslant$ is usually induced on X by a T_0 topology defined in some natural way. Typical examples we shall consider in this paper are built from an admissible set $\mathbb{A}$, with $F = A$, $\leqslant = \subseteq_{X \setminus U^{\mathbb{A}}} \cup =_{U^{\mathbb{A}}}$, and $X \subseteq A \cup P(A)$ (see Example 1 below). To explain the role of set X we should define the notions of process, constructive process, and computability.

Informally, processes are functions on p-domains which generate an output as the limit of its approximations, using 'finite' fragments of arguments and resources (like space or time). Each process is defined by its *specification* or *presentation*, and constructive processes are defined by specifications which can be 'effectively checked'.

Definition 2. Let $m, n \in \omega$. By a (m, n)-ary *specification* on $\mathcal{X}$ we mean a total function $\alpha_0 : F^{m+n+2} \to \{0, 1\}$ which is monotone with respect to the last 2 arguments in the following sense: for any $\bar{a} \in F^m$, any $\bar{b} \in F^n$, and any $c, c', d, d' \in F$,

$$\text{if } c \leqslant c' \text{ then } \alpha_0(\bar{a}, \bar{b}, c, d) \leqslant \alpha_0(\bar{a}, \bar{b}, c', d);$$

$$\text{if } d \leqslant d' \text{ then } \alpha_0(\bar{a}, \bar{b}, c, d') \leqslant \alpha_0(\bar{a}, \bar{b}, c, d).$$

Informally, $\bar{a}$ are finite arguments, $\bar{b}$ are finite fragments of (possibly infinite) arguments, c and c' are finite fragments of the resources we can use, while d and d' are finite fragments of the result of the process defined by this specification.

Definition 3. Let $\mathcal{X} = \langle X, F, \leqslant, \rangle$ be a p-domain. For $m, n \in \omega$, (m, n)-*ary process* on $\mathcal{X}$ is a partial mapping α from (a subset of) $F^m \times X^n$ to X such that there exists a specification $\alpha_0 : F^{m+n+1} \to \{0, 1\}$ which *defines* α in the following sense: for any $\bar{a} \in F^m$, $\bar{x} \in X^n$ such that $(\bar{a}, \bar{x}) \in \mathrm{dom}(\alpha)$, and any $d \in F$, $d \leqslant \alpha(\bar{a}, \bar{x})$ iff there exist $b_0, \ldots, b_{n-1}, c \in F$ such that $b_0 \leqslant x_0, \ldots, b_{n-1} \leqslant x_{n-1}$, and $\alpha_0(\bar{a}, \bar{b}, c, d) = 1$.

Let $n \in \omega$. We use the following terminology to denote the fact that a given process is either defined only on 'finite' arguments, or on arbitrary arguments approached only via approximations. Namely, for $n \in \omega$

– n-ary *functional* is a $(n, 0)$-process from (a subset of) F^n to X;
– n-ary *operator* is a $(0, n)$-process from (a subset of) X^n to X.

We denote classes of n-ary functionals and operators on $\mathcal{X}$ by $\mathcal{F}_n(\mathcal{X})$ and $\mathcal{O}_n(\mathcal{X})$, correspondingly, and the class of all processes on $\mathcal{X}$ is denoted by $\mathcal{P}(\mathcal{X})$.

Definition 4. 1) *Termination* of a (partial) functional $\alpha : F^n \to X$ is a total function $\alpha_t : F^{n+1} \to \{0, 1\}$ defined as follows: for any $\bar{a} \in F^n$, $b \in F$,

$$\alpha_t(\bar{a}, b) = 1 \text{ iff } b \leqslant \alpha(\bar{a}).$$

2) *Termination* of a (partial) operator $\beta : X^n \to X$ is a total function $\beta_t : X^{n+1} \to \{0, 1\}$ defined as follows: for any $\bar{a} \in X^n$, $b \in X$,

$$\beta_t(\bar{a}, b) = 1 \text{ iff } b = \beta(\bar{a}).$$

3) *Termination* of a (partial) process $\gamma : F^m \times X^n \to X$, $n > 0$, is a total function $\beta_t : F^m \times X^{n+1} \to \{0, 1\}$ defined as follows: for any $\bar{a} \in F^m$, $\bar{b} \in X^n$, $c \in X$,

$$\gamma_t(\bar{a}, \bar{b}, c) = 1 \text{ iff } c = \gamma(\bar{a}, \bar{b}).$$

3 Computabilities and Reducibilities on Admissible Sets

Example 1. Let $\mathbb{A}$ be an admissible set. A p-domain $\mathcal{X}$ on $\mathbb{A}$ can be constructed in the following way. Let $X_0 \subseteq P(A)$ be an arbitrary Σ-admissible family in sense of [6], $X = A \cup X_0$, $F = A$, and $\leqslant = \subseteq_{X \setminus U^{\mathbb{A}}} \cup =_{U^{\mathbb{A}}}$. So, the topology on $X \cap P(A)$ is a strong topology from [6], and the topology on $U^{\mathbb{A}}$ is trivial. A class of processes can be taken as a suitable subclass of $\mathcal{P}(\mathcal{X})$—the set of operators on $\mathbb{A}$ which are strongly continuous [6]. The set of all Σ-predicates and Σ-operators on $\mathbb{A}$ is, in fact, a subset of $\mathcal{P}(\mathcal{X})$ (and form a subclass of 'computable processes'), by the axiom of Δ_0-Collection and the definition of Σ-admissible family, respectively. Note that in case when $\mathbb{A} = \mathbb{HF}(\mathfrak{M})$ it is possible to take $X_0 = P(\mathrm{HF(M)})$ [6].

On the other hand, we can first fix a subclass of computable processes $\mathcal{C} \subseteq \mathcal{P}(\mathcal{X})$ and then take $X_0 \subseteq P(A)$ to be a Σ-admissible family relative to $\mathcal{C}$. We shall use the following technical modifications of some basic notions from [2,6]. Let $\mathbb{A}$ be an admissible set.

1) Mapping $\alpha : A^n \to P(A)$ is called *Σ-predicate on* $\mathbb{A}$ if there is a Σ-formula $\varphi_\alpha(x_1, \ldots, x_n, y)$ of signature $\sigma_{\mathbb{A}}$ (with no parameters from A) such that, for all $a_1, \ldots, a_n, b \in A$, $b \in \alpha(a_1, \ldots, a_n)$ iff $\mathbb{A} \models \varphi_\alpha(a_1, \ldots, a_n, b)$ (φ_α is called *Σ-specification*, or *Σ-presentation*, of α).
2) Mapping $\beta : P(A)^n \to P(A)$ is called *Σ-operator on* $\mathbb{A}$ if there is a Σ-formula $\varphi_\beta(x_1, \ldots, x_n, y)$ of signature $\sigma_{\mathbb{A}}$ (with no parameters from A) such that, for all $S_1, \ldots, S_n \in P(A)$, $b \in A$

$$b \in \beta(S_1, \ldots, S_n) \text{ iff } \exists a_1 \subseteq S_1, \ldots, \exists a_n \subseteq S_n \text{ s.t. } \mathbb{A} \models \varphi_\beta(a_1, \ldots, a_n, b)$$

(here it is assumed that $a_1, \ldots, a_n \in A^*$). Again, φ_β is called *Σ-specification*, or *Σ-presentation*, of β.

We assume that if $\mathbb{A}$ is fixed, $\leqslant$ denotes $\subseteq_{A^*} \cup =_{U^{\mathbb{A}}}$.

Definition 5. Let $\mathbb{A}$ be an admissible set, and let $m, n \in \omega$. Mapping γ from $A^m \times (A \cup P(A))^n$ to $P(A)$ is called a *Σ-process on* $\mathbb{A}$ if there is a Σ-formula $\varphi_\gamma(x_1, \ldots, x_m, y_1, \ldots, y_n, z)$ of signature $\sigma_{\mathbb{A}}$ (with no parameters from A) such that, for all $a_1, \ldots, a_m \in A$, $x_1, \ldots, x_m \in A \cup P(A)$, $c \in A$,

$$c \in \gamma(\bar{a}, \bar{x}) \text{ iff } \exists b_1 \leqslant x_1, \ldots, \exists b_n \leqslant x_n \text{ s.t. } \mathbb{A} \models \varphi_\gamma(\bar{a}, \bar{b}, c).$$

Formula φ_γ is called *Σ-specification*, or *Σ-presentation*, of γ. The set of all Σ-presentations of a given process γ is denoted by $Pres_\Sigma(\gamma)$.

Any Σ-presentation $\varphi_\gamma(\bar{x}, \bar{y}, c)$ of a process γ can be transformed into the Δ_0-formula $\theta_\gamma(\bar{x}, \bar{y}, c, d)$ which is a specification of γ in the sense of Definitions 2, 3: take

$$\theta_\gamma(\bar{x}, \bar{y}, c, d) \leftrightharpoons (\forall c' \in c)\varphi_\gamma(\bar{x}, \bar{y}, c')^{(d)},$$

where $\varphi_\gamma(\bar{x}, \bar{y}, c')^{(d)}$ is the relativization of φ_γ to d [2].

We denote by $\mathcal{F}_\Sigma(\mathbb{A})$ the class of all Σ-predicates on $\mathbb{A}$, by $\mathcal{O}_\Sigma(\mathbb{A})$ the class of all Σ-operators on $\mathbb{A}$, and by $\mathcal{P}_\Sigma(\mathbb{A})$ the class of all Σ-processes on $\mathbb{A}$ (hence, $\mathcal{P}_\Sigma(\mathbb{A}) \supseteq \mathcal{F}_\Sigma(\mathbb{A}) \cup \mathcal{O}_\Sigma(\mathbb{A})$).

Definition 6. Let $\mathbb{A}$ be an admissible set and let $\mathcal{C} \subseteq \mathcal{P}_\Sigma(\mathbb{A})$ be a class of Σ-processes on $\mathbb{A}$. A family $\mathcal{S} \subseteq A \cup P(A)$ is called *Σ-admissible relative to $\mathcal{C}$* if

1) $\mathcal{S}$ is closed relative to processes from $\mathcal{C}$:

$$\forall \alpha^k \in \mathcal{C} \forall x_1, \ldots, x_k \in \mathcal{S}\ \alpha(x_1, \ldots, x_k) \in \mathcal{S};$$

2) processes from $\mathcal{C}$ are strongly continuous on elements from $\mathcal{S}$: for any (m, n)-process $\alpha \in \mathcal{S}$,

$$\forall a_1, \ldots, a_m \in A \forall x_1, \ldots, x_n \in \mathcal{S} \forall c \in A(c \leqslant \alpha(\bar{a}, \bar{x}) \rightarrow$$

$$\rightarrow \exists b_1 \in A \ldots \exists b_n \in A(b_1 \leqslant x_1 \wedge \ldots \wedge b_n \leqslant x_n \wedge c \leqslant \alpha(\bar{a}, \bar{b}))).$$

Definition 7. Let $\mathbb{A}$ be an admissible set. By a *computability component on* $\mathbb{A}$ we mean a pair $(\mathcal{S}, \mathcal{C})$, where

1) $A \subseteq \mathcal{S} \subseteq P(A)$ is a Σ-admissible family relative to $\mathcal{C}$, and
2) $\mathcal{F}_\Sigma(\mathbb{A}) \subseteq \mathcal{C} \subseteq \mathcal{P}_\Sigma(\mathbb{A})$ is a class of Σ-processes on $\mathbb{A}$ which is closed under superposition.

For an admissible set $\mathbb{A}$, by *computability on* $\mathbb{A}$ we mean the family $\mathrm{Com}(\mathbb{A})$ of all computability components on $\mathbb{A}$:

$$\mathrm{Com}(\mathbb{A}) = \{(\mathcal{S}, \mathcal{C}) \mid (\mathcal{S}, \mathcal{C}) \text{ is a computability component on } \mathbb{A}\}.$$

To demonstrate the usefulness of these new notions, we prove a strengthening of the result of Morozov [8] which states that a certain reducibility on admissible sets implies an embedding of computable objects (i.e., Σ-predicates) on them.

The reducibility on admissible sets was defined by Morozov [8] as a modification of the notion of Σ-*definability* of a structure in an admissible set, introduced by Ershov [4,6]. (A structure Σ-definable in $\mathbb{A}$ is called $\mathbb{A}$-*constructivizable.*) In analogous way, the relation of Σ-*reducibility* $\leqslant_\Sigma$ on (types of isomorphism of) structures was also defined using this notion. Namely, for structures $\mathfrak{M}$ and $\mathfrak{N}$, we denote by $\mathfrak{M} \leqslant_\Sigma \mathfrak{N}$ the fact that $\mathfrak{M}$ is Σ-definable in $\mathbb{HF}(\mathfrak{N})$. This relation is reflexive and transitive, and the corresponding notion of Σ-degree gives a natural measure of complexity for structures of arbitrary cardinalities, see [16,17,18,19].

We shall also need a 'positive' version of Σ-definability: for a structure $\mathfrak{M}$ and an admissible set $\mathbb{A}$, $\mathfrak{M}$ is Σ^+-*definable in* $\mathbb{A}$ if there exist a computable sequence of Σ-formulas $\Phi(x_0, y), \Phi_0(x_0, \ldots, x_{n_0-1}, y), \Phi_1(x_0, \ldots, x_{n_1-1}, y), \ldots$ such that, for some parameter $a \in A$ and an onto mapping $\nu : \Phi^{\mathbb{A}}(x_0, a) \twoheadrightarrow M$, for every $i \in \omega$ and every $a_0, \ldots, a_{n_i-1} \in \Phi^{\mathbb{A}}(x_0, a)$,

$$\mathbb{A} \models \Phi_i(a_0, \ldots, a_{n_i-1}, a) \iff \mathfrak{M} \models P_i(\nu(a_0), \ldots, \nu(a_{n_i-1})).$$

Again, for structures $\mathfrak{M}$ and $\mathfrak{N}$, we denote by $\mathfrak{M} \leqslant^+_\Sigma \mathfrak{N}$ the fact that $\mathfrak{M}$ is Σ^+-definable in $\mathbb{HF}(\mathfrak{N})$. It should be noted, however, that $\leqslant^+_\Sigma$ is transitive only in case when all structures are treated 'positively' in the sense that their atomic diagrams are not necessarily closed under negations.

For a structure with an infinite computable signature, we assume that some Gödel numbering of formulas of this signature is fixed. We assume that the signature of $\mathbb{HF}(\mathfrak{B})$ contains a predicate symbol Sat^2 interpreted by the satisfiability relation for atomic formulas in $\mathfrak{B}$, with respect to a fixed Gödel numbering. In the case of structures with a finite signature this assumption is not essential.

The next definition is a technical modification of the original one (it was used in this form in [9]).

Definition 8 (Morozov [8]). Let $\mathbb{A}$ and $\mathbb{B}$ be admissible sets. $\mathbb{A}$ *is* Σ-*reducible to* $\mathbb{B}$ (denoted $\mathbb{A} \sqsubseteq_\Sigma \mathbb{B}$) if there is an onto mapping $\nu : B \twoheadrightarrow A$ such that

1) ν is a $\mathbb{B}$-constructivization of $\mathbb{A}$ as a structure in sense of [4,6];
2) there is a binary Σ-predicate E on $\mathbb{B}$ s.t. $pr_1(E) = B$ and, for all $b, c \in B$,

$$\langle b, c\rangle \in E \text{ implies } \nu(b) = \{\nu(z) | z \in c\}.$$

Definition 9. If, for admissible sets $\mathbb{A}, \mathbb{B}$, there exist mappings $\nu : B \twoheadrightarrow A$ and $\mu : Pres_\Sigma(\mathcal{P}_\Sigma(\mathbb{A})) \to Pres_\Sigma(\mathcal{P}_\Sigma(\mathbb{B}))$ such that μ is computable and, for every $(\mathcal{S}, \mathcal{C}) \in \mathrm{Com}(\mathbb{A})$, there exists $(\mathcal{S}', \mathcal{C}') \in \mathrm{Com}(\mathbb{B})$ such that

$$(\nu^{-1}(\mathcal{S}), \mu(Pres(\mathcal{C}))) \text{ is isomorphic to } (\mathcal{S}', \mathcal{C}'),$$

we say that $\mathrm{Com}(\mathbb{A})$ *is* Σ-*embeddable into* $\mathrm{Com}(\mathbb{B})$.

Theorem 10. Let $\mathbb{A}, \mathbb{B}$ be admissible sets. If $\mathbb{A} \sqsubseteq_\Sigma \mathbb{B}$ then $\mathrm{Com}(\mathbb{A})$ is Σ-embeddable into $\mathrm{Com}(\mathbb{B})$.

Proof. We prove that if $\mathbb{A}$ is Σ-reducible to $\mathbb{B}$ then Σ-processes on $\mathbb{A}$ are represented, in an effective and uniform way, by Σ-processes on $\mathbb{B}$ working with

the names of elements from A. So, the result is somethat similar to one on relationship between Σ-degrees and degrees of presentability of structures.

We present a uniform effective procedure which transform any Σ-specification of a Σ-process on $\mathbb{A}$ into some Σ-specification of Σ-process on $\mathbb{B}$ representing the first one. In a standard way, we define an effective uniform transformation $\Phi(\bar{x}, \bar{a}) \mapsto \Phi^*(\bar{x}, \bar{b})$ of Σ-formulas of signature $\sigma_{\mathbb{A}}$ with parameters $\bar{a}$ from A, to Σ-formulas of signature $\sigma_{\mathbb{B}}$ with parameters $\bar{b}$ from B, by induction on the complexity. Now, any Σ-specification Φ_α of (m, n)-ary Σ-process α on $\mathbb{A}$ is mapped to a Σ-formula $(\Phi_\alpha)^*$ which is a Σ-specification of (m, n)-ary Σ-process α^* on $\mathbb{B}$ such that, if $\langle S_0, \ldots, S_{n-1}\rangle \in \delta_c(\alpha)$ (δ_c denotes the domain of strong continuity), then $\langle \nu^{-1}(S_0), \ldots, \nu^{-1}(S_{n-1})\rangle \in \delta_c(\alpha^*)$ and

$$\nu^{-1}(\alpha(S_0, \ldots, S_{n-1})) = \alpha^*(\nu^{-1}(S_0), \ldots, \nu^{-1}(S_{n-1})).$$

Hence, defining mapping μ on Σ-processes as $\mu(p) = (p)^*$, the pair (ν^{-1}, μ) establish the desired isomorphism.

4 Jumps of Computabilities: Σ-Jump of a Structure as the Jump of the Minimal Component of HF-Computability

Definition 11. Let $\mathbb{A}$ be an admissible set, and let $(\mathcal{S}, \mathcal{C})$ be a computability component on $\mathbb{A}$. *Jump of* $(\mathcal{S}, \mathcal{C})$ is a structure $J_{\mathbb{A}}(\mathcal{S}, \mathcal{C})$ with domain $\mathcal{S}$ and atomic diagram consisting of a unary predicate distinguishing the set A of finite objects and the termination $t(\mathcal{C})$ of processes from $\mathcal{C}$.

This extends in a natural way all existing definitions of jump operations defined on subsets of natural numbers or on structures. Indeed, in the last case, we use the fact that every structure generates the least admissible set containing it—HF-superstructure. If we take the least computability component on that HF-superstructure and terminate all its processes (i.e., all Σ-predicates), we get the structure which is called *Σ-jump* of the original one. The formal definition is as follows:

Definition 12. Let $\mathfrak{A}$ be a structure. By *Σ-jump*, or *minimal Σ-jump*, of $\mathfrak{A}$, we mean the structure

$$\mathfrak{A}' = (X; F, \mathcal{T}),$$

with the domain $X = HF(A)$, and relations $F = HF(A)$ (domain consists of finite objects only, so the unary relation F is trivial in this case and usually skipped), and $\mathcal{T} = t(\mathcal{F}_\Sigma(\mathbb{HF}(\mathfrak{A})))$ as the termination of all Σ-predicates on $\mathbb{HF}(\mathfrak{A})$ (denoted here, as in [18,19], by $\Sigma\text{-Sat}_{\mathbb{HF}(\mathfrak{A})}$).

In a similar way the jump operation was introduced in [1] for the semilattice of s-degrees of countable structures. Also, in the same way a notion of the jump of an admissible set with respect to various effective reducibilities was introduced in [8,9].

The operation of Σ-jump agrees with the jump operations for Turing and enumeration degrees w.r.t. the natural embeddings i and j: if a structure $\mathfrak{A}$ has a (e-)degree $\mathbf{a}$, then the structure $\mathfrak{A}'$ has (e-)degree $\mathbf{a}'$. In fact, it is true that the mappings $i : \mathcal{D} \to \mathcal{S}_\Sigma$ and $j : \mathcal{D}_e \to \mathcal{S}_\Sigma$ are embeddings preserving 0, $\vee$ and the jump operation (see [16,19] for details). Hence, the operation of Σ-jump is a natural extension of Turing and enumeration jumps. One of the important facts about the minimal HF-computability is that the jump inversion theorem from classical computability theory is still true in this general setting.

Theorem 13 ([18,19]). Let $\mathfrak{A}$ be a structure such that $\mathbf{0}' \leqslant_\Sigma \mathfrak{A}$. Then there exists a structure $\mathfrak{B}$ such that

$$\mathfrak{B}' \equiv_\Sigma \mathfrak{A}.$$

5 Jumps of Maximal Components of HF-Computabilities: $P\Sigma$-Jump of 0 and the Reals

Definition 14. Let $\mathfrak{A}$ be a structure. By *$P\Sigma$-jump*, or *maximal Σ-jump*, of $\mathfrak{A}$, we mean the structure

$$\mathfrak{A}^\diamond = (X; F, \mathcal{T}),$$

with the domain $X = HF(A) \cup P(HF(A))$, and the atomic diagram consisting of relations $F = HF(A)$ distinguishing finite objects, and $\mathcal{T}$ as the termination of all Σ-processes on $\mathbb{HF}(\mathfrak{A})$.

It follows from the definition that relations $\in$ and $\subseteq$ between elements of the sets F and X are obtained as terminations of Σ-processes which act on X and depend on a in an effective and uniform way. Also, it is easy to note that $P\Sigma$-jump is indeed a jump with respect to $\leqslant_\Sigma$, because immediately from cardinality reasons we get that, for any structure $\mathfrak{A}$,

$$\mathfrak{A} <_\Sigma \mathfrak{A}^\diamond.$$

A natural question is an analogue of Jump Inversion Theorem for $P\Sigma$-jump. We start from investigating the Σ-degree of $0^\diamond$.

Definition 15. Let $\mathbb{R}$ denote the set of real numbers. We consider the following structures:

1) algebraical field of reals $\mathcal{R} = (\mathbb{R}, +, \times, 0, 1, =)$;
2) topological field of reals

$$\mathcal{R}_o = (\mathbb{R}, \Gamma_+^A, \Gamma_+^B, \Gamma_\times^A, \Gamma_\times^B, 0, 1, <),$$

where $\Gamma_+^A = \{\langle x, y, z\rangle \in \mathbb{R}^3 | x + y < z\}$, $\Gamma_+^B = \{\langle x, y, z\rangle \in \mathbb{R}^3 | z < x + y\}$ (similar definitions for $\Gamma_\times^A$, $\Gamma_\times^B$).

Theorem 16. $\mathcal{R} \leqslant_\Sigma (0^\diamond)'$.

Proof. We define a Σ-presentation of $\mathcal{R}$ in $(0^\diamond)' = (\mathbb{HF}(0^\diamond), \Sigma\text{-Sat}_{\mathbb{HF}(0^\diamond)})$ as follows. Take as the domain the set

$$R = \{\langle k, m, \alpha\rangle | k \in \{-1, 0, 1\}, m \in \omega, \alpha \in \mathrm{Fun}(\omega, 2)\},$$

where $\mathrm{Fun}(\omega, 2)$ is the set of total functions from ω to $2 = \{0, 1\}$, and each triple $x = \langle k, m, \alpha\rangle$ represents the real number

$$r_x = k(m + \sum_{n\in\omega} \frac{\alpha(n)}{2^{n+1}}).$$

Lemma 17. R is Σ-definable in $(\mathbb{HF}(0^\diamond), \Sigma\text{-Sat}_{\mathbb{HF}(0^\diamond)})$.

Proof. For arbitrary $S \in HF(0^\diamond)$, $S \in \mathrm{Fun}(\omega, 2)$ if and only if $\exists X(S = F(X)) \wedge \Phi(S)$, where Σ-operator F on $\mathbb{HF}(\varnothing)$ is defined as follows: for any $X \subseteq HF(\varnothing)$,

$$F(X) = \{y | \exists a \subseteq X \exists n \exists k[(a = \{y\}) \wedge (y = \langle n, k\rangle) \wedge \mathrm{Nat}(n) \wedge (k \in 2)]\},$$

and

$$\Phi(S) = \forall n(Nat(n) \to ((((\langle n, 0\rangle \in S) \wedge (\langle n, 1\rangle \notin S)) \vee ((\langle n, 1\rangle \in S) \wedge (\langle n, 0\rangle \notin S))).$$

Since $\mathrm{Fn}(S)$ is a Π-formula in $\mathbb{HF}(0^\diamond)$, R is Σ-definable in $(\mathbb{HF}(0^\diamond), \Sigma\text{-Sat}_{\mathbb{HF}(0^\diamond)})$.

Theorem 18. $\mathcal{R}_o \leqslant^+_\Sigma 0^\diamond$.

Proof. We take the set

$$R_o = \{\langle k, m, S\rangle | k \in \{-1, 0, 1\}, m \in \omega, S \subseteq \mathbb{HF}(\varnothing)\},$$

as the domain of the presentation. It is easy to note that the cardinality of the presentation of $\mathcal{R}_0$ is the same as the cardinality of $\mathcal{R}_0$, which is not necessary for structures with no equality. The proof follows from the following lemmas.

Lemma 19. For any set $S \subseteq \mathbb{HF}(\varnothing)$, the following sets could be obtained as the results of some Σ-operators acting on S:

1) $S \cap \omega$;
2) $S \cap n\, (= S \cap \{0, \ldots, n-1\})$.

In particular, in case 2 the corresponding Σ-operator depends on n in uniform and effective way.

Lemma 20. The strict order relation $\{\langle x_1, x_2\rangle \in R_o^2 \,|\, r_{x_1} < r_{x_2}\}$ is Σ-definable in $\mathbb{HF}(0^\diamond)$.

In the same way, it can be proved that relations $\Gamma^A_+, \Gamma^B_+, \Gamma^A_\times, \Gamma^B_\times$ are all Σ-definable in $\mathbb{HF}(0^\diamond)$.

6 Open Questions

1. What is an analogue of Jump Inversion for a given computability component of HF-computability over 0 or any given structure?
2. What is an analogue of Jump Inversion for a given computability component of $\mathbb{A}$-computability? This question is especially interesting for the least computability component of $\mathbb{H}\mathrm{YP}(\mathfrak{M})$-computability.
3. Is $0^{\diamond} \leqslant^{+}_{\Sigma} \mathcal{R}_0$? This would mean that in the maximal component of HF-computability over 0 holds an analogue of the Matijasevich Theorem. Also, is it natural to ask, whether or not $(0^{\diamond})' \leqslant_{\Sigma} \mathcal{R}$.

References

1. Baleva, V.: The jump operation for structure degrees. Arch. Math. Logic. 45, 249–265 (2006)
2. Barwise, J.: Admissible Sets and Strucrures. Springer, Berlin (1975)
3. Ershov, Y.L.: The theory of A-spaces. Algebra and Logic 12, 209–232 (1973)
4. Ershov, Y.L.: Σ-definability in admissible sets. Sov. Math. Dokl. 32, 767–770 (1985)
5. Ershov, Y.L.: Theory of domains and nearby. In: Pottosin, I.V., Bjorner, D., Broy, M. (eds.) FMP&TA 1993. LNCS, vol. 735, pp. 1–7. Springer, Heidelberg (1993)
6. Ershov, Y.L.: Definability and Computability. Plenum, New York (1996)
7. Montalbán, A.: Notes on the jump of a structure. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 372–378. Springer, Heidelberg (2009)
8. Morozov, A.S.: On the relation of Σ-reducibility between admissible sets. Sib. Math. J. 45, 522–535 (2004)
9. Puzarenko, V.G.: About a certain reducibility on admissible sets. Sib. Math. J. 50, 330–340 (2009)
10. Scott, D.S.: Outline of a mathematical theory of computation. In: Proceedings of 4th Annual Princeton Conference on Information Science and Systems, pp. 165–176 (1970)
11. Soskova, A.A.: A jump inversion theorem for the degree spectra. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 716–726. Springer, Heidelberg (2007)
12. Soskov, I.N., Soskova, A.A.: A jump inversion theorem for the degree spectra. J. Log. Comput. 19, 199–215 (2009)
13. Stukachev, A.I.: Σ-admissible families over linear orders. Algebra Logic 41, 127–139 (2002)
14. Stukachev, A.I.: Presentations of structures in admissible sets. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 470–478. Springer, Heidelberg (2005)
15. Stukachev, A.: On mass problems of presentability. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 772–782. Springer, Heidelberg (2006)
16. Stukachev, A.I.: Degrees of presentability of structures, I. Algebra Logic 46, 419–432 (2007)
17. Stukachev, A.I.: Degrees of presentability of structures, II. Algebra Logic 47, 65–74 (2008)
18. Stukachev, A.I.: A Jump Inversion Theorem for the semilattices of Σ-degrees. Sib. Elektron. Mat. Izv. 6, 182–190 (2009) (Russian)
19. Stukachev, A.I.: A Jump Inversion Theorem for the semilattices of Σ-degrees. Sib. Adv. Math. 20, 68–74 (2010)

Various Regularity Lemmas in Graphs and Hypergraphs

Endre Szemerédi

Computer Science Department, Rutgers University, New Brunswick, NJ 08901, U.S.A.
szemered@cs.rutgers.edu

We are going to formulate and discuss the philosophy of these lemmas and their applications in property testing and at the designing polynomial time approximation schemes for certain graph problems.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, p. 403, 2013.

Three Debates about Computing

Matti Tedre

Department of Computer and Systems Sciences, Stockholm University,
Stockholm, Sweden
firstname.lastname@acm.org

Abstract. Computing as a discipline has a short but vivid history. Computing started to develop a disciplinary identity only after the birth of the stored-program paradigm in the 1940s, and its academic image has seen dramatic changes in the past six decades. This article presents three debates that are central in understanding how computing as a discipline developed to what it is now: the formal verification debate, the software engineering debate, and the experimental computer science debate.

1 Introduction

The history of computing reveals a rich combination of technical and theoretical breakthroughs, and often it is hard to draw a line between the two. Many forefathers of automatic computing, take Pascal and Babbage, for example, were mathematicians versed in theory and practice of building the machines. In the 1930s and 1940s advances on multiple fronts started a new era of automatic computing. The Newton–Maxwell gap from electromechanical devices to fully electronic devices was crossed, and the central ideas of the stored-program paradigm were articulated. That decade was also a time of constant exchange between technology and theory. The fathers of the first digital electronic computer, Atanasoff and Berry, were electrical engineers. The designers of the first Turing-complete, fully electronic, digital computer ENIAC, were electrical engineers, mathematicians, and scientists. Turing was enthusiastic with actual machinery, not only with the abstractions for which he is best known [28]. All in all, modern computing was born at a conjunction of new technical and theoretical insights.

After the birth of the stored-program paradigm, the emerging discipline of computing started to develop a unique research agenda [32]. The first serious debates about computing as an academic discipline were concerned with the field's independence from other disciplines, especially mathematics. Disciplinary identity was necessary for a large number of reasons, and considerable effort was invested in separating computing from mathematics. But while mathematics was something that computing pioneers wanted to distinguish the field from, science is a different story. Starting from the late 1960s there was a strong movement to liken the discipline of computing with empirical sciences. Finally, although engineering was central to the birth of modern computing—also in the academic world—for decades engineering, with its practical aims, was seriously undervalued in academic computing. It took nearly four decades to establish

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 404–413, 2013.

an understanding of computing as an inseparable combination of theoretical, technical, and scientific elements.

This article presents three debates about the soul of computing as a discipline. The first debate is the formal verification debate, which has roots in the identity-forming years of the field, and which characterized the 1970s and 1980s discussions about computing as a discipline. The second debate is the engineering debate, which intensified in the end of the 1960s when software engineering promised to fix the problems that crippled software industry, and which lasted until very recently. The third debate is the experimental computer science debate, which started in the 1980s and which is still very much alive. The role of those debates in discipline-building efforts is not well studied. Perhaps those debates shaped computing; perhaps they reflected already ongoing changes in computing. This article is not a historical study—great references in the history of computing can be found in, e.g., [25]—but it presents a number of debates that were considered to be so significant that practically all prominent computer scientists had something to say about at least some of them.

2 The Formal Verification Debate

After computing began to form up as a discipline, its ties to mathematics started to weaken. On the other hand, theoretical computer science advanced on several fronts. On the other hand, numerical analysis lost its central place in the field, computing started to gain an identity separate from mathematics, and the ranks of mathematicians in computing departments were gradually replaced by newly graduated computer scientists [39]. Many abstractly oriented mathematicians lacked respect for the pragmatic orientation of computing, too [39].

The separation of computing and mathematics was not just a process of growing apart, but computer scientists were actively divorcing from mathematics. A large number of articles were written about the differences between computer science and mathematics. A distinct disciplinary identity was important for various reasons. It gave computing departments research fellow and student quotas, leverage in university politics, representation in policy-making committees and boards, professional identity, access to directed grants and funding, and increased societal influence [22,23]. Finally, in 1974 the National Science Foundation of the U.S. granted computer science a category distinct from other science and engineering disciplines [23].

While there was a movement to establish computing as a discipline distinct from mathematics, at the same time many members of that movement defended the mathematical nature of computing [10,40]. The role of mathematical proof was extended from theoretical computer science to program construction, and the advocates of *formal verification* of software were vocal and resolute. A sort of an extreme position—call it, for instance, 'strong formal verificationism' or 'mathematical reductionism'—was articulated by Hoare [26], who argued that all of computing can be reduced to mathematics: Computers are mathematical machines, computer programs are mathematical expressions, programming languages are mathematical theories, and programming is a mathematical activity.

Although the formal verification movement was, from its start in the early 1960s, light years away from the reality of actual programming practice, many believed in its intellectual superiority. For two decades the critics of formal verificationism had no credible counterargument, but by the end of the 1980s the formal verification movement was seriously running out of steam. Software engineering had been established already in the end of the 1960s, and by the late 1980s it had developed into a serious program of investigation and practice. Meanwhile, the scientific–modeling–empirical branches of computing were intertwining with other disciplines, creating new and exciting fields as they moved on—take biocomputing, cognitive science, and quantum computing, for example.

The coup de grâce to strong formal verificationism was finally delivered around the 1980s by three arguments. Firstly, it was noted that the nature of proofs of program correctness are very different from proofs in mathematics [7]. Secondly, it was noted that there are fundamental gaps between programs, specifications, and the physical world where computers work [44]. Third, it was noted that unlike theoretical constructions, the physical world is uncertain [20]. Although one may be able to prove—in some cases and on the abstract level—correspondence between the inputs and outputs of a program text, executable programs do not evade the physical world: computers are physical machines and programs are swarms of electrons. Gradually the formal verification movement abandoned its extreme position and moved towards the mainstream by acknowledging the limits of verification and by consolidating formal methods with software engineering methods [13,27].

The formal verification debate characterized the 1970s and 1980s disciplinary debates in computing. It had great impact on the discipline at many levels—theoretical as well as practical. However, its implications should not be exaggerated. The arguments that debunked the most extreme versions of formal verification were not even meant to threaten the foundational status and importance of theoretical computer science. The issue at stake was rather the acceptance of empirical and engineering methods in the discipline. Today formal *methods* are used more than ever, and their power is broadly acknowledged.

3 Fall and Rise of Engineering

Although engineering and engineers were central to the birth of modern computing, the technical and engineering aspects of computing were downplayed right from the start [16]. Although work on the technical aspects of computing quickly developed the structures of a profession, that professional status was low [9,16]. There seemed to be no room for technology in the academic discipline of computing.

Early university programs in computing were explicitly distanced from any technical aspects and actual equipment [17]. Ignoring technology in computing was made to look like a virtue. That spirit lingered in academic computing for decades to come—perhaps best expressed by oft-quoted phrases like "*computer science is not about machines, in the same way that astronomy is not about*

telescopes" [19] and "*the computing scientist could not care less about the specific technology that might be used to realize machines, be it electronics, optics, pneumatics, or magic*" [11].

In fields other than computing, systems engineering emerged in the early 1900s when the complexity of new tools and machines grew to a point where single individuals could no longer design them. Similar, as software systems grew in size and complexity, increasingly large teams were needed for designing, implementing, and maintaining them. By the 1960s computer and software systems had grown in size to the point where they were unmanageable without new approaches [5,9]. The resulting problems manifested as delayed, over-budget, and low-quality software products, and the situation was dubbed 'software crisis'. Blame was thrown around: some accused academic education for failing to meet the needs of industry [15,46] while others blamed the industry for its shoddy standards, poor methods, and inadequate tools [9].

A specific term *software engineering* was officially introduced in 1968 at a conference held to discuss the software crisis [35]. But starting in the middle of the acute software crisis, software engineering got a lousy start. A large number of 'silver bullet' solutions were proposed [4], yet they were unable to prevent the situation from developing into 'software's chronic crisis' [24] (see also [14]). By the 1980s, as computing was largely recognized as a mature discipline [40], also the legitimization pressures increasingly piled on new branches of computing, such as software engineering.

The struggle of software engineering for recognition was met with fiery resistance. One author called software engineering "the doomed discipline" [12], another argued that it is based on anecdotal evidence and human authority [29], and third reported that in their study, one third of software engineering articles failed to validate their results experimentally [51]. Mainstream software engineering adopted a practical, opportunistic, and business-oriented mode of working and took some distance from academic research (e.g., [45, pp.2–3]). One of the early textbooks stated that the software engineer "*cannot afford to experiment with each and every new technique put forward by research scientists*" [45, p.3]. That being the case, the rejection of software engineering in many academic computing circles seems unsurprising. The practical bent of computing had earlier undermined the field's struggle for academic status [1], and acknowledging an inherently practical, hands-on endeavor as an integral part of the discipline might risk whatever status computing had achieved.

The formal verification debates and software engineering debates had a very different end. A theoretical claim can be struck down by a forceful counterargument, but practical arguments demonstrate their value over the course of time. Similar to the formal verification debate, the critique of the engineering tradition was quite narrowly focused. The critics rarely, if ever, argued that technical implementations would be unnecessary. The computer had already shown its ability to revolutionize science and society. The production of useful and reliable computer systems was agreed to be an important aim. Instead of criticizing engineering aims and its value to society, critics of software engineering attacked

the lack of rigor and the academic status of software engineering. However, as software engineering has gradually evolved into a legitimate branch of computing that develops at a rapid pace, many objections have lost their edge.

4 Good Science, Bad Science, No Science

The third central bone of contention in the disciplinary history of computing was concerned with science. There were various debates that centered around the scientific aspects of computing. Firstly, especially in the early days of the discipline, many debates about the name of the field centered about whether the discipline should be called a 'science'. The "What's in a name?" question was repeated in a large number of opinion pieces over the decades—but its usefulness was also questioned from time to time [22,31].

Secondly, another string of debates was centered around the subject matter of computing: Different from natural sciences, which study naturally occurring things, the subject matter of computing is artificial, human-made. It was asked whether sciences of the artificial can be sciences in the strict sense [36,43]. Thirdly, many recent debates are concerned with methodological rigor in computing, and those discussions also revolve around science terminology. Debates about 'experimental computer science' reveal great concern about the quality of work in computing, but those debates also reveal a great lack of common understanding between the discussants. This section outlines those three threads of debates, starting from the oldest.

The first engineering society for computing professionals was founded in 1946—the same year the first fully electronic, digital, Turing-complete computer ENIAC was unveiled [50]. That was followed by founding of the predecessor to *Association for Computing Machinery* (ACM) in 1947. From the beginning, ACM was focused on theoretical computer science and its applications [49]. Although the early professional societies were clearly focused on computing, it is difficult to say if the members of those societies considered themselves primarily as computing professionals or perhaps electrical engineers or mathematicians who focus on computing. In the midst of the engineering and theoretical associations, a view of computing as a science started to develop. The term 'computer science' was introduced already before discussions about the scientific merits of the discipline emerged. It is difficult to pinpoint the exact origins: One pioneer of computing dated his first thought of using the umbrella term 'computer sciences' to 1956, while he first used the term in a printed report in 1957 and it first appeared to the broad public in 1959 [17,33].

The 1960s clearly saw the emergence of computing as a distinct academic discipline. In 1962 Purdue University established a department that bore the name 'computer sciences' [42]. Doctoral degrees in computer science were awarded as early as 1965 [41, p.59]. The ACM released its first draft of computer science curriculum in 1965 [6]. In 1967 one of the landmark texts in the disciplinary history of computing described 'computer science' in the magazine *Science* as "the study of the phenomena surrounding computers" [36]. But although the term

'science' was frequently tossed around in debates that concerned naming of the field, there was relatively little discussion about what exactly in the disciplinary nature of computing makes it scientific.

Already in the 1960s the 'theory and practice' division [21,22] started to evolve into a tripartite description of computing where the theoretical, technical, and empirical aspects of computing each got due credit (e.g., [2]). In the 1970s, as the discipline matured further, the scientific aspects got more attention in analyses of computing as a discipline. Computing was divided into 'scientific, mathematical, and technological' aspects, of which the scientific parts were "concerned with the empirical study of a class of phenomena" [48]. In the late 1980s one of the most famous accounts of computing as a discipline named the theoretical, scientific, and engineering aspects 'theory, abstraction (modeling), and design' [8], and that division has been frequently quoted ever since.

Many famous defenses of the scientific nature of computing focused on the subject matter of computing (e.g., [36]). But one could argue that science is not defined by its subject matter but by its method of inquiry. That line of defense of the *science* in computing was popularized in public discussions by the emergence of discussions on *experimental computer science*. Experimental computer science was brought to limelight by a strong campaign for 'rejuvenating experimental computer science' at the turn of the 1980s [18].

However, it was never clear what exactly was meant by 'experimental computer science'. In one sense of the word, 'experimental' can refer to exploratory work on novel and untested ideas or techniques. In another sense of the word, 'experimental' can refer to the use of controlled experiments for testing hypotheses. The 'rejuvenating' report [18] teetered between the two meanings of the word but never made it clear what exactly was meant by 'experimental' computer science. What followed was several decades of polemics where discussants talked past each other, referring to experimental computer science, but meaning different things.

The terms 'experiment' and 'experimental' take various meanings in computing literature. The first meaning refers to exploratory work on novel and untested ideas or techniques; in that type of work a demonstration of experimental technology shows that something indeed can be done—"more an existence proof than experiment" [3]. The second meaning refers to testing how well a system meets its technical specifications [34]. The third meaning refers to evaluating how well a system works in its intended context of use; in information systems that work is often called "field experiment" [37]. The fourth meaning refers to a comparison of two designs or implementations in order to "provide evidence of the superiority of your algorithmic ideas" [30]. The fifth meaning refers to the traditional experiment-based research, using "controlled experiments to generate measurable, empirical data" [38].

The experimental computer science debates are well alive today. However, discussions about the scientific merits of computing as a discipline are often fruitless because the discussants do not share a common vocabulary on the topic [47]. Firstly, discussants tend to make overarching statements about all of computing,

based on their personal ideas or their own favorite branch of computing [13]. Secondly, discussants hold very different views of what science, strictly speaking, is—yet they still may talk as if there were a monolithic Science, against which other things can be measured [47]. Terms 'computing' and 'science' hold very different meanings for different people. With two such grab bags of concepts—computing and science—it is no wonder that discussants are unable to find a common ground.

5 Conclusions

Computing as a discipline was born from a seamless combination of theoretical and practical aspects. The forefathers of modern computing had no issues working with technical and theoretical issues at the same time. There again, none of the early pioneers would have considered themselves computer scientists. Although the field of computing has roots in things like office machinery and mathematical logic, automatic computing was neither a profession nor an academic discipline before the second half of the twentieth century. Only after computers started to become common, did the discipline start to form up.

The first wave of debates about the disciplinary nature of computing were aimed at establishing the discipline's independence from the fields that gave birth to it. As computing was often seen as a branch of numerical mathematics, it was important to articulate the differences between computing and mathematics. A large number of pioneers of computer science wrote analyses on what makes computing unique—although usually those analyses also pointed out the similarities of the two fields. Debates on the relationship between mathematics and computing culminated in the formal verification debate led by a number of hard-line verificationists. After those debates reached a culmination point in the late 1980s, the focus moved to more modest discussions about how to utilize formal methods in computing practice.

The second wave of debates were concerned with reinstating engineering in the core of computing as a discipline. When computing started to form up as an academic discipline, the theoretical branches of computing took a prominent role in defining the field. As the academic image of computing had earlier suffered from the discipline's practical orientation, technical subjects were often excluded from academic programs. The introduction of software engineering as a solution to the software crisis was met with furious resistance, but through decades of painful development it gradually became a core part of computing as a discipline.

The third wave of debates were concerned with the discipline's place among the empirical sciences. At the beginning those debates were concerned with the field's self-selected title 'computer science', and countless arguments were made about the (mis)naming of computing as a science. In the anglophone countries the name stuck, and the debates moved on to the subject matter of computing. The subject matter of computing is very different from the subject matters in natural sciences, which led to discussions about what kind of a science computing is—natural, 'unnatural', artificial, or something else. From the subject matter,

the science debates soon shifted focus to methodology. The 'experimental computer science' debate has been a defining feature of computing's disciplinary self-image since the 1980s, and that debate shows no signs of petering out.

The three debates presented in this article paint a picture of the discipline's growth pains. The process starts from the field's necessary detachment from its roots in mathematics and electrical engineering, yet those strings showed to be painful to cut. When computers became commonplace in universities, offices, and finally in homes, the need for massive numbers of new applications required completely new management and development approaches, which, when new and immature, were first met with considerable resistance. When the markets (in business and intellectual terms) were becoming satisfied, discussions about quality re-emerged both in the academia and in the industry. It was no longer enough to just get it done, but scientific rigor was increasingly often required.

Acknowledgments. This research was funded by the Academy of Finland grant #132572.

References

1. Aspray, W.: Was early entry a competitive advantage? US universities that entered computing in the 1940s. IEEE Annals of the History of Computing 22(3), 42–87 (2000)
2. Atchison, W.F., Conte, S.D., Hamblen, J.W., Hull, T.E., Keenan, T.A., Kehl, W.B., McCluskey, E.J., Navarro, S.O., Rheinboldt, W.C., Schweppe, E.J., Viavant, W., David, J., Young, M.: Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science. Communications of the ACM 11(3), 151–197 (1968)
3. Basili, V.R., Zelkowitz, M.V.: Empirical studies to build a science of computer science. Communications of the ACM 50(11), 33–37 (2007)
4. Brooks Jr., F.P.: No silver bullet: Essence and accidents of software engineering. IEEE Computer 20(4), 10–19 (1987)
5. Campbell-Kelly, M., Aspray, W.: Computer: A History of the Information Machine, 2nd edn. Westview Press, Oxford (2004)
6. Conte, S.D., Hamblen, J.W., Kehl, W.B., Navarro, S.O., Rheinboldt, W.C., David, J., Young, M., Atchinson, W.F.: An undergraduate program in computer science–preliminary recommendations. Communications of the ACM 8(9), 543–552 (1965)
7. De Millo, R.A., Lipton, R.J., Perlis, A.J.: Social processes and proofs of theorems and programs. Communications of the ACM 22(5), 271–280 (1979)
8. Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A., Turner, A.J., Young, P.R.: Computing as a discipline. Communications of the ACM 32(1), 9–23 (1989)
9. Dijkstra, E.W.: The humble programmer. Communications of the ACM 15(10), 859–866 (1972)
10. Dijkstra, E.W.: Programming as a discipline of mathematical nature. American Mathematical Monthly 81(6), 608–612 (1974)
11. Dijkstra, E.W.: On a cultural gap. The Mathematical Intelligencer 8(1), 48–52 (1986)

12. Dijkstra, E.W.: On the cruelty of really teaching computer science. Communications of the ACM 32(12), 1398–1404 (1989)
13. Dijkstra, E.W.: The tide, not the waves. In: Denning, P.J., Metcalfe, R.M. (eds.) Beyond Calculation: The Next Fifty Years of Computing, pp. 59–64. Springer, New York (1997)
14. Eden, A.H.: Three paradigms of computer science. Minds & Machines 17(2), 135–167 (2007)
15. Egan, L.G.: Closing the "gap" between the university and industry in computer science. SIGCSE Bulletin 8(4), 19–25 (1976)
16. Ensmenger, N.L.: The 'question of professionalism' in the computer fields. IEEE Annals of the History of Computing 23(4), 56–74 (2001)
17. Fein, L.: The role of the university in computers, data processing, and related fields. Communications of the ACM 2(9), 7–14 (1959)
18. Feldman, J.A., Sutherland, W.R.: Rejuvenating experimental computer science: A report to the National Science Foundation and others. Communications of the ACM 22(9), 497–502 (1979)
19. Fellows, M.R.: Computer science and mathematics in the elementary schools. In: Fisher, N.D., Keynes, H.B., Wagreich, P.D. (eds.) Mathematicians and Education Reform. Issues in Mathematics Education, vol. 3, pp. 1990–1991. American Mathematical Society, Providence (1993)
20. Fetzer, J.H.: Program verification: the very idea. Communications of the ACM 31(9), 1048–1063 (1988)
21. Forsythe, G.E.: A university's educational program in computer science. Communications of the ACM 10(1), 3–11 (1967)
22. Forsythe, G.E.: What to do till the computer scientist comes. American Mathematical Monthly 75, 454–461 (1968)
23. Galler, B.A.: Letter from a past president: Distinction of computer science. Communications of the ACM 17(6), 300 (1974)
24. Gibbs, W.W.: Software's chronic crisis. Scientific American 271(3), 86–95 (1994)
25. Haigh, T.: The history of information technology. Annual Review of Information Science and Technology 45(1), 431–487 (2011)
26. Hoare, C.A.R.: The mathematics of programming. In: Maheshwari, S.N. (ed.) FSTTCS 1985. LNCS, vol. 206, pp. 1–18. Springer, Heidelberg (1985)
27. Hoare, C.A.R.: Retrospective: An axiomatic basis for computer programming. Communications of the ACM 52(10), 30–32 (2009)
28. Hodges, A.: Alan Turing: The Enigma. Vintage Books, London (1983)
29. Holloway, C.M.: Software engineering and epistemology. SIGSOFT Software Engineering Notes 20(2), 20–21 (1995)
30. Johnson, D.S.: A theoretician's guide to the experimental analysis of algorithms. In: Goldwasser, M.H., Johnson, D.S., McGeoch, C.C. (eds.) Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 59, pp. 215–250. American Mathematical Society, Providence (2002)
31. Knuth, D.E.: Algorithmic thinking and mathematical thinking. American Mathematical Monthly 92, 170–181 (1985)
32. Mahoney, M.S.: Histories of Computing. Harvard University Press, Cambridge (2011)
33. McCorduck, P.: An interview with Louis Fein. Charles Babbage Institute, The Center for the History of Information Processing (May 9, 1979)

34. McCracken, D.D., Denning, P.J., Brandin, D.H.: An ACM executive committee position on the crisis in experimental computer science. Communications of the ACM 22(9), 503–504 (1979)
35. Naur, P., Randell, B. (eds.): Software Engineering: Report on a Conference Sponsored by the Nato Science Committee, Garmisch, Germany, October 7-11. NATO Scientific Affairs Division, Brussels (1968, 1969)
36. Newell, A., Perlis, A.J., Simon, H.A.: Computer science. Science 157(3795), 1373–1374 (1967)
37. Palvia, P., Mao, E., Salam, A.F., Soliman, K.S.: Management information systems research: What's there in a methodology? Communications of the Association for Information Systems 11(16), 1–32 (2003)
38. Peisert, S., Bishop, M.: I am a scientist, not a philosopher! IEEE Security and Privacy 5(4), 48–51 (2007)
39. Ralston, A.: Computer science, mathematics, and the undergraduate curricula in both. The American Mathematical Monthly 88(7), 472–485 (1981)
40. Ralston, A., Shaw, M.: Curriculum '78–is computer science really that unmathematical? Communications of the ACM 23(2), 67–70 (1980)
41. Reilly, E.D.: Milestones in Computer Science and Information Technology. Greenwood Press, Westport (2003)
42. Rice, J.R., Rosen, S.: Computer sciences at Purdue University–1962 to 2000. IEEE Annals of the History of Computing 26(2), 48–61 (2004)
43. Simon, H.A.: The Sciences of the Artificial, 1st edn. MIT Press, Cambridge (1969)
44. Smith, B.C.: Limits of correctness in computers. Technical Report CSLI-85-36, Center for the Study of Language and Information, Stanford University, Stanford, CA, USA (1985)
45. Sommerville, I.: Software Engineering. Addison-Wesley, Bedford Square (1982)
46. Spier, M.J.: A critical look at the state of our science. SIGOPS Operating Systems Review 8(2), 9–15 (1974)
47. Tedre, M.: Computing as a science: A survey of competing viewpoints. Minds & Machines 21(3), 361–387 (2011)
48. Wegner, P.: Research paradigms in computer science. In: ICSE 1976: Proceedings of the 2nd International Conference on Software Engineering, pp. 322–330. IEEE Computer Society Press, Los Alamitos (1976)
49. Williams, S.B.: The association for computing machinery. Journal of the ACM 1(1), 1–3 (1954)
50. Wood, H.M.: Computer society celebrates 50 years. IEEE Annals of the History of Computing 17(4), 6 (1995)
51. Zelkowitz, M.V., Wallace, D.R.: Experimental validation in software engineering. Information and Software Technology 39(11), 735–743 (1997)

Another Jump Inversion Theorem for Structures

Stefan Vatev*

Faculty of Mathematics and Informatics, Sofia University, 5 James Bourchier blvd., 1164 Sofia, Bulgaria
stefanv@fmi.uni-sofia.bg

Abstract. In this paper we investigate the question of existence of a jump inversion structure for a given structure $\mathcal{A}$ in the context of their respective degree spectra and the sets definable in them by computable infinitary formulae. More specifically, for a countable structure $\mathcal{A}$ and a computable successor ordinal α, we show that we can apply the construction from [4] to build a structure $\mathcal{N}_\alpha$ such that the sets definable in $\mathcal{A}$ by $\Sigma_1^{c,\Delta_\alpha^0}$ formulae are exactly the sets definable in $\mathcal{N}_\alpha$ by Σ_α^c formulae.

1 Introduction

We shall work with abstract structures of the form $\mathcal{A} = (A; R_0, \ldots, R_{s-1})$, where A is countable and infinite, $R_i \subseteq A^{n_i}$. We use the letters $\mathcal{A}$, $\mathcal{B}$ to denote structures and the letters A, B to denote their respective universes.

We call f an *enumeration* of the set A if f is a partial one-to-one mapping of $\mathbb{N}$ onto A. We say that f is an enumeration of the structure $\mathcal{A}$ if f is an enumeration of its universe A.

If f is an enumeration of A and $R \subseteq A^n$, we denote $f^{-1}(R) = \{\langle x_1, \ldots, x_n\rangle \mid x_1, \ldots, x_n \in \mathrm{Dom}(f)\ \&\ (f(x_1), \ldots, f(x_n)) \in R\}$. For $\mathcal{A} = (A; R_0, \ldots, R_{s-1})$ we define the total function $f^{-1}(\mathcal{A})$ in the following way:

- if $u = \langle k, v\rangle$ and $k < s$, then $f^{-1}(\mathcal{A})(u) = i$ iff $f^{-1}(R_k)(v) = i$, for $i \in \{0, 1\}$;
- if $u = \langle k, v\rangle$ and $k \geq s$, then $f^{-1}(\mathcal{A})(u) = 0$.

We call $f^{-1}(\mathcal{A})$ a copy of $\mathcal{A}$.

Richter [5] initiates the study of the notion of the degree spectrum of a countable structure.

Definition 1. *The degree spectrum of the structure $\mathcal{A}$ is the set of Turing degrees*

$$\mathrm{DS}(\mathcal{A}) = \{\mathbf{a} \mid \mathbf{a} \text{ computes a copy of } \mathcal{A}\}.$$

For a computable ordinal α, we define the α-th jump degree spectrum of $\mathcal{A}$ to be

$$\mathrm{DS}_\alpha(\mathcal{A}) = \{\mathbf{a}^{(\alpha)} \mid \mathbf{a} \in \mathrm{DS}(\mathcal{A})\}.$$

* The research leading to these results has received funding from the [European Community's] Seventh Framework Programme [FP7/2007-2013] under grant agreement 238381 and by Sofia University Science Fund, contract 131/09.05.2012.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 414–423, 2013.

The notion of degree spectra gives us one way to compare structures. That is, for structures $\mathcal{A}$ and $\mathcal{B}$ and computable ordinals α, β, we ask whether $\mathrm{DS}_\alpha(\mathcal{A}) = \mathrm{DS}_\beta(\mathcal{B})$.

Now we give an informal definition of the set of *infinitary* Σ_α formulae in the language of $\mathcal{A}$, denoted by Σ_α. The Σ_0 and Π_0 formulae are the finitary quantifier free formulae. For a computable ordinal $\alpha > 0$, a Σ_α formula $\varphi(\overline{x})$ is an infinitary disjunction of a set of formulae of the form $\exists \overline{y} \psi$, where ψ is a Π_β formula, for $\beta < \alpha$, and $\overline{y}$ includes the variables of ψ which are not in $\overline{x}$. The Π_α formulae are the negations of the Σ_α formulae. The *computably infinitary* Σ_α formulae, denoted Σ^c_α, are those Σ_α formulae whose infinitary disjunctions are over c.e. sets. By $\Sigma^{c,X}_\alpha$ we mean the computable relative to the set X infinitary formulae. We refer the reader to [1, chap. 7] for more background information.

A set $X \subseteq A$ is Σ^c_α *definable* in the structure $\mathcal{A}$ if there is a Σ^c_α formula $\psi(x, \overline{y})$ and a finite number of parameters $\overline{a}$ in A such that $b \in X \leftrightarrow \mathcal{A} \models \psi(b, \overline{a})$. We denote by $\Sigma^c_\alpha(\mathcal{A})$ the family of all sets Σ^c_α definable in $\mathcal{A}$.

The notion of definability gives us another way to compare structures. That is, for structures $\mathcal{A}$, $\mathcal{B}$ such that $A \subseteq B$ and computable ordinals α, β, we ask whether $(\forall X \subseteq A)[X \in \Sigma^c_\alpha(\mathcal{A}) \leftrightarrow X \in \Sigma^c_\beta(\mathcal{B})]$.

For simplicity, in most of the constructions that follow we shall consider only structures of the form $\mathcal{A} = (A; R)$. In the end it should be clear that these constructions can be generalised to structures in any finite or effectively listed relational language. The next definition gives us the scheme that we follow to define our jump inversion structures.

Definition 2 ([4]). *Given a structure $\mathcal{A} = (A; R)$, $R \subseteq A^n$, and a pair of structures $\mathcal{B}_0$, $\mathcal{B}_1$ for the same relational language, let $\mathcal{N} = (A \cup U; A, U, Q, \dots)$, where*

1) *$A \cap U = \emptyset$;*
2) *Q is an $(n+1)$-ary relation which assigns to each n-tuple $\bar{a}$ in A an infinite set $U_{\bar{a}}$, where $x \in U_{\bar{a}}$ iff $\mathcal{N} \models Q(\bar{a}, x)$. We also want $\bar{a} \neq \bar{b} \leftrightarrow U_{\bar{a}} \cap U_{\bar{b}} = \emptyset$;*
3) *The sets $U_{\bar{a}}$ form a partition of U;*
4) *Each of the other relations of $\mathcal{N}$ (in $\dots$) corresponds to some symbol in the language of $\mathcal{B}_0$, $\mathcal{B}_1$, and is the union of its restrictions to the sets $U_{\bar{a}}$;*
5) *For each n-tuple $\bar{a}$ in A, if $\mathcal{U}_{\bar{a}} = (U_{\bar{a}}, \dots)$, then*

$$\mathcal{U}_{\bar{a}} \cong \begin{cases} \mathcal{B}_1, & \text{if } \mathcal{A} \models R(\bar{a}) \\ \mathcal{B}_0, & \text{if } \mathcal{A} \models \neg R(\bar{a}) \end{cases}$$

For a set of natural numbers X and a computable ordinal α, we denote by $X^{(\alpha)}$ the α-th Turing jump of X. Moreover, we define

$$\begin{aligned} &\Delta^0_{\alpha+1}(X) = X^{(\alpha)}, \text{ if } \alpha < \omega, \\ &\Delta^0_{\alpha+1}(X) = X^{(\alpha+1)}, \text{ if } \alpha \geq \omega, \\ &\Delta^0_\alpha(X) = \bigcup_p \{\langle y, p\rangle \mid y \in \Delta^0_{\alpha(p)}(X)\}, \text{ if } \alpha = \lim \alpha(p). \end{aligned}$$

We write Δ^0_a for $\Delta^0_a(\emptyset)$.

Although not explicitly stated as a theorem by Goncharov, Harizanov, Knight, McCoy, Miller and Solomon [4], the following result is a form of a jump inversion theorem for structures in the context of their respective degree spectra.

Theorem 1 ([4]). *Let $\mathcal{A} = (A; R)$ be a structure and for $\alpha > 1$ a computable* successor *ordinal, let $\mathcal{B}_0, \mathcal{B}_1$ be structures that satisfy the properties:*

a) $\mathcal{B}_0$ and $\mathcal{B}_1$ are computable structures whose universes are the natural numbers and defined in the same relational language $\mathscr{L}$,
b) $\{\mathcal{B}_0, \mathcal{B}_1\}$ is α-friendly,
c) $\mathcal{B}_0, \mathcal{B}_1$ satisfy the same Σ_β sentences (of $\mathscr{L}_{\omega_1\omega}$, i.e., not only computable) for all $\beta < \alpha$,
d) each $\mathcal{B}_i$ satisfies some Σ^c_α sentence that is not true in the other.

Let $\mathcal{N}$ be the structure built as in Definition 2 for $\mathcal{A}$, $\mathcal{B}_0$ and $\mathcal{B}_1$. Then for any $X \subseteq \mathbb{N}$, $\mathcal{A}$ has a $\Delta^0_\alpha(X)$-computable copy iff $\mathcal{N}$ has an X-computable copy. It follows that

$$\mathrm{DS}(\mathcal{A}) \subseteq \{\mathbf{a} \mid \mathbf{0}^{(\beta)} \leq \mathbf{a}\} \text{ implies } \mathrm{DS}(\mathcal{A}) = \mathrm{DS}_\beta(\mathcal{N}),$$

where $\beta = \alpha - 1$, if $\alpha < \omega$ and $\beta = \alpha$, if $\alpha \geq \omega$.

The proof of Theorem 1 relies on Ash's α-systems, which is a framework for priority constructions. The requirement that $\{\mathcal{B}_0, \mathcal{B}_1\}$ is α-friendly is essential for their proof.

For a set $X \subseteq \mathbb{N}$, let us denote the structure $\mathfrak{A}_X = (\mathbb{N}; X, G_S)$, where G_S is the graph of the successor function on $\mathbb{N}$. For a set $X \subseteq \mathbb{N}$ and a structure $\mathcal{A} = (A; R_0, \ldots, R_{s-1})$ with $A \cap \mathbb{N} = \emptyset$, let us denote by $A \oplus X$ the cardinal sum of the structures $\mathcal{A}$ and $\mathfrak{A}_X$, i.e., $\mathcal{A} \oplus X = (A \cup \mathbb{N}; A, \mathbb{N}, R_0, \ldots, R_{s-1}, X, G_S)$.

Our goal in this paper is to prove the following theorem, which is similar to Theorem 1, but without the requirement that $\{\mathcal{B}_0, \mathcal{B}_1\}$ is α-friendly.

Theorem 2. *Let $\mathcal{A} = (A; R)$ be a structure. Moreover, for $\alpha > 1$ a computable* successor *ordinal, let $\mathcal{B}_0, \mathcal{B}_1$ be structures that satisfy the following:*

a) $\mathcal{B}_0$ and $\mathcal{B}_1$ are computable $\mathscr{L}$-structures whose universes are the natural numbers, where $\mathscr{L}$ is a relational language, which includes equality,
b) $\mathcal{B}_0, \mathcal{B}_1$ satisfy the same Σ^c_β sentences, for all $\beta < \alpha$,
c) each $\mathcal{B}_i$ satisfies some Σ^c_α sentence that is not true in the other.

Then for $\mathcal{N}$, built as in Definition 2 for $\mathcal{A}$, $\mathcal{B}_0$ and $\mathcal{B}_1$, we have the following:

1) $\mathrm{DS}_\beta(\mathcal{N}) = \mathrm{DS}(\mathcal{A} \oplus \Delta^0_\alpha)$, where $\beta = \alpha - 1$, if $\alpha < \omega$ and $\beta = \alpha$, if $\alpha \geq \omega$, and
2) $(\forall X \subseteq A)[X \in \Sigma^c_\alpha(\mathcal{N}) \leftrightarrow X \in \Sigma^c_1(\mathcal{A} \oplus \Delta^0_\alpha) \leftrightarrow X \in \Sigma^{c,\Delta^0_\alpha}_1(\mathcal{A})]$,

It is important to remark that the proof of Theorem 2 will not imply that if $\mathcal{A}$ has a $\Delta^0_\alpha(X)$-computable copy, then $\mathcal{N}$ has an X-computable copy. Our proof is based on the notion of forcing and building a generic copy of the structure $\mathcal{N}$.

For finite ordinals, our result can be obtained by applying a different construction, the so-called Marker's extension. It is used by A. Soskova, I. Soskov [6] and by Stukachev [7] to prove a jump inversion theorem in the context of Turing degree spectra and in the context of Σ-reducibility, respectively.

2 The Notion of Forcing

We define the finite parts into the set B as those finite mappings from $\mathbb{N}$ into B, which are also one-to-one. Given a finite part τ and a relation $R \subseteq B^n$, we define the finite function $\tau^{-1}(R)$ as follows:

$$\tau^{-1}(R)(u) \downarrow= 1 \leftrightarrow (\exists x_1, \ldots, x_n \in \mathrm{Dom}(\tau))[u = \langle x_1, \ldots, x_n \rangle \ \& \ (\tau(x_1), \ldots, \tau(x_n)) \in R],$$
$$\tau^{-1}(R)(u) \downarrow= 0 \leftrightarrow (\exists x_1, \ldots, x_n \in \mathrm{Dom}(\tau))[u = \langle x_1, \ldots, x_n \rangle \ \& \ (\tau(x_1), \ldots, \tau(x_n)) \notin R].$$

For a structure $\mathcal{A} = (A; R_0, \ldots, R_{s-1})$, we define the finite function $\tau^{-1}(\mathcal{A})$ in the following way:

1) if $u = \langle k, v \rangle$ and $k < s$, then $\tau^{-1}(\mathcal{A})(u) \downarrow= i$ iff $\tau^{-1}(R_k)(v) \downarrow= i$, for $i \in \{0, 1\}$.
2) if $u = \langle k, v \rangle$, $k \geq s$, but $u < \max\{x \mid x \in \mathrm{Dom}(\tau)\}$, then $\tau^{-1}(\mathcal{A})(u) \downarrow= 0$.

We remark that we need condition 2) so that we have the equality

$$f^{-1}(\mathcal{A}) = \bigcup_{\tau \subseteq f} \tau^{-1}(\mathcal{A}).$$

Partial Conditions

Let us fix two structures $\mathcal{B}_0$ and $\mathcal{B}_1$ with the same universe B and in the same language $\mathscr{L}$. *Partial conditions* are finite sequences of the form

$$\mathscr{C} = (\tau_0^{\mathscr{C}}, \tau_1^{\mathscr{C}}, \ldots, \tau_{k-1}^{\mathscr{C}}),$$

where every $\tau_i^{\mathscr{C}}$ is a finite part. We denote the partial conditions by the letters $\mathscr{C}$, $\mathscr{D}$ and $\mathscr{E}$. Let us denote the length of $\mathscr{C}$ by $|\mathscr{C}|$. For $n < |\mathscr{C}|$, we denote

$$\mathscr{C} \restriction n = (\tau_0^{\mathscr{C}}, \ldots, \tau_{n-1}^{\mathscr{C}}).$$

We say that $\mathscr{D}$ *extends* $\mathscr{C}$, denoted $\mathscr{C} \subseteq \mathscr{D}$, if

$$|\mathscr{C}| \leq |\mathscr{D}| \ \& \ (\forall i)[i < |\mathscr{C}| \ \rightarrow \ \tau_i^{\mathscr{C}} \subseteq \tau_i^{\mathscr{D}}].$$

We say that $\mathscr{D}$ *partially extends* $\mathscr{C}$, denoted $\mathscr{C} \subseteq_p \mathscr{D}$, if

$$|\mathscr{C}| \leq |\mathscr{D}| \ \& \ (\forall i)[i < |\mathscr{C}| \ \rightarrow \ \tau_i^{\mathscr{C}} = \tau_i^{\mathscr{D}}].$$

For a sequence of sets of natural numbers $\{B_i\}_{i<\kappa}$, with $\kappa \leq \omega$, we denote $\bigoplus_{i<\kappa} B_i = \{\langle i, x \rangle \mid i < \kappa \ \& \ x \in B_i\}$. We define the *diagram* of the partial condition $\mathscr{C}$ *with respect to* $X \in 2^\omega$ as

$$D_X(\mathscr{C}) = \bigoplus_{j<|\mathscr{C}|} (\tau_j^{\mathscr{C}})^{-1}(\mathcal{B}_{X(j)}).$$

The Forcing Relation

If φ is a partial function and $e \in \omega$, then by W_e^{φ} we denote the set of all x such that the computation $\{e\}^{\varphi}(x)$ halts successfully. We assume that if during a computation the oracle φ is called with an argument outside of its domain, then the computation halts unsuccessfully. Let Fin_2 be the set of all finite functions from the natural numbers taking values into $\{0, 1\}$.

The definition of the forcing relation will follow the definition of the α-th Turing jump. For all natural numbers e, x, computable ordinal $\alpha \geq 1$ and partial condition $\mathscr{C}$, we define the forcing relations $\Vdash_{\alpha}^{X}$ in the following way:

(i) $\mathscr{C} \Vdash_1^X F_e(x) \leftrightarrow x \in W_e^{D_X(\mathscr{C})}$.
(ii) Let $\alpha = \beta + 1$. Then

$$\begin{aligned}\mathscr{C} \Vdash_{\beta+1}^X F_e(x) \leftrightarrow (\exists \delta \in Fin_2)[x \in W_e^{\delta} \;\&\; (\forall z \in \mathrm{Dom}(\delta))[\\ (\delta(z) = 1 \;\&\; \mathscr{C} \Vdash_{\beta}^X F_z(z)) \vee \\ (\delta(z) = 0 \;\&\; \mathscr{C} \Vdash_{\beta}^X \neg F_z(z))]].\end{aligned}$$

(iii) Let $\alpha = \lim \alpha(p)$. Then

$$\begin{aligned}\mathscr{C} \Vdash_{\alpha}^X F_e(x) \leftrightarrow (\exists \delta \in Fin_2)[x \in W_e^{\delta} \;\&\; (\forall z \in \mathrm{Dom}(\delta))[z = \langle x_z, p_z\rangle \;\& \\ ((\delta(z) = 1 \;\&\; \mathscr{C} \Vdash_{\alpha(p_z)}^X F_{x_z}(x_z)) \vee \\ (\delta(z) = 0 \;\&\; \mathscr{C} \Vdash_{\alpha(p_z)}^X \neg F_{x_z}(x_z)))]].\end{aligned}$$

(iv) $\mathscr{C} \Vdash_{\alpha}^X \neg F_e(x) \leftrightarrow (\forall \mathscr{D})[\mathscr{C} \subseteq \mathscr{D} \rightarrow \mathscr{D} \nVdash_{\alpha}^X F_e(x)]$.

Lemma 1. *For computable ordinals $\alpha \geq 1$ we have the following:*

1) If $\mathscr{C} \Vdash_{\alpha}^X F_e(x)$ and $\mathscr{C} \subseteq \mathscr{D}$, then $\mathscr{D} \Vdash_{\alpha}^X F_e(x)$.
2) If $\mathscr{C} \Vdash_{\alpha}^X \neg F_e(x)$ and $\mathscr{C} \subseteq \mathscr{D}$, then $\mathscr{D} \Vdash_{\alpha}^X \neg F_e(x)$.

Let δ be a finite part and $\mathrm{Dom}(\delta) = \{d_0 < d_1 < \cdots < d_k\}$. We write $\bar{\delta}$ for the tuple $(\delta(d_0), \delta(d_1), \ldots, \delta(d_k))$. Furthermore, let us denote

$$\mathscr{C} \approx_l \mathscr{D} \leftrightarrow \bigwedge_{i \neq l} (\tau_i^{\mathscr{C}} = \tau_i^{\mathscr{D}}),$$

i.e., the partial conditions $\mathscr{C}$ and $\mathscr{D}$ are allowed to differ only in their l-th coordinates.

Note that when we say that $X \in 2^{\omega}$ is finite, we mean that there is i_0 such that $X(i) = 0$ for all $i > i_0$. Also, for a condition $\mathscr{C}$, we let $X_{\mathscr{C}} \in 2^{\omega}$ be such that $X_{\mathscr{C}}(i) = X(i)$ for $i < |\mathscr{C}|$ and $X_{\mathscr{C}}(i) = 0$ for $i \geq |\mathscr{C}|$.

Lemma 2. *Let $\mathcal{B}_0$ and $\mathcal{B}_1$ be computable structures in the language $\mathscr{L} = \{P_0, \ldots, P_{k-1}\}$, which includes equality. Let X be finite, $\mathscr{C}$ be a partial condition, l be a number such that $l < |\mathscr{C}|$, and let $D = \{x_0 < \cdots < x_d\}$.*

Then for all natural numbers e, x, and a computable ordinal $\alpha \geq 1$, there is a Σ^c_α formula $\Phi^\alpha_{\mathscr{C},D,e,x}$ in $\mathscr{L}$ with free variables $X_0, \ldots, X_d$ such that for every finite part ρ with $\mathrm{Dom}(\rho) = D$, we have

$$\mathscr{D} \approx_l \mathscr{C} \ \&\ \tau_l^{\mathscr{D}} = \rho \ \&\ \mathscr{D} \Vdash^X_\alpha F_e(x) \ \leftrightarrow\ \mathcal{B}_{X(l)} \models \Phi^\alpha_{\mathscr{C},D,e,x}(\bar{\rho}).$$

We remark that if X is not computable, then $\Phi^\alpha_{\mathscr{C},D,e,x}$ will be a $\Sigma^{c,X}_\alpha$ formula.

Corollary 1. *Under the conditions of Lemma 2, for a computable ordinal $\alpha \geq 1$, there is a Σ^c_α sentence $\Phi^\alpha_{\mathscr{C},e,x}$ in the language $\mathscr{L}$ such that*

$$(\exists \mathscr{D})[\mathscr{D} \approx_l \mathscr{C} \ \&\ \mathscr{D} \Vdash^X_\alpha F_e(x)] \ \leftrightarrow\ \mathcal{B}_{X(l)} \models \Phi^\alpha_{\mathscr{C},e,x}.$$

Lemma 3. *Let us fix a computable ordinal $\alpha \geq 1$. Let $\mathcal{B}_0$ and $\mathcal{B}_1$ be computable structures in the same language $\mathscr{L}$ with equality and both structures satisfy the same Σ^c_α sentences in $\mathscr{L}$. Moreover, let us fix a condition $\mathscr{C}$ and finite X, Y such that $X_{\mathscr{C}} = Y_{\mathscr{C}}$, $X \neq Y$ and they differ only at points $< m$. Then we have the equivalence:*

$$(\exists \mathscr{D} \supseteq_p \mathscr{C})[\mathscr{D} \Vdash^X_\alpha F_e(x) \ \&\ |\mathscr{D}| = m] \ \leftrightarrow\ (\exists \mathscr{D} \supseteq_p \mathscr{C})[\mathscr{D} \Vdash^Y_\alpha F_e(x) \ \&\ |\mathscr{D}| = m].$$

Proof. For $(\rightarrow)$, let us fix $\mathscr{D} \supseteq_p \mathscr{C}$ such that $\mathscr{D} \Vdash^X_\alpha F_e(x)$, $|\mathscr{D}| = m$ and let $l = |\mathscr{C}|$. For $i = l, l+1, \ldots, m$, let the finite $X_i \in 2^\omega$ be such that $X_i(j) = X(j)$ for $j \notin [l, i)$ and $X_i(j) = Y(j)$ for $j \in [l, i)$. We remark that $X_l = X$ and $X_m = Y$. We shall define by induction on i the partial conditions $\mathscr{D}_i$ such that $\mathscr{D}_i \supseteq_p \mathscr{C}$, $|\mathscr{D}_i| = m$ and $\mathscr{D}_i \Vdash^{X_i}_\alpha F_e(x)$. For $i = l$, let $\mathscr{D}_i = \mathscr{D}$, which satisfies our requirements. Now suppose we have defined $\mathscr{D}_i$. Then $\mathscr{D}_i \Vdash^{X_i}_\alpha F_e(x)$ trivially implies $(\exists \mathscr{D}')[\mathscr{D}' \approx_i \mathscr{D}_i \ \&\ \mathscr{D}' \Vdash^{X_i}_\alpha F_e(x)]$. By Corollary 1, there is a Σ^c_α sentence $\Phi^\alpha_{\mathscr{D}_i,e,x}$ such that $(\exists \mathscr{D}')[\mathscr{D}' \approx_i \mathscr{D}_i \ \&\ \mathscr{D}' \Vdash^{X_i}_\alpha F_e(x)] \leftrightarrow \mathcal{B}_{X_i(i)} \models \Phi^\alpha_{\mathscr{D}_i,e,x}$. We have that $\mathcal{B}_0$ and $\mathcal{B}_1$ satisfy the same Σ^c_α sentences. Thus, $\mathcal{B}_{X_i(i)} \models \Phi^\alpha_{\mathscr{D}_i,e,x}$ iff $\mathcal{B}_{Y(i)} \models \Phi^\alpha_{\mathscr{D}_i,e,x}$. Since $X_{i+1}(i) = Y(i)$ and $X_i(j) = X_{i+1}(j)$ for $j \neq i$, by Corollary 1, $(\exists \mathscr{D}')[\mathscr{D}' \approx_i \mathscr{D}_i \ \&\ \mathscr{D}' \Vdash^{X_{i+1}}_\alpha F_e(x)] \leftrightarrow \mathcal{B}_{Y(i)} \models \Phi^\alpha_{\mathscr{D}_i,e,x}$. By combining the above equivalences, we obtain

$$(\exists \mathscr{D}')[\mathscr{D}' \approx_i \mathscr{D}_i \ \&\ \mathscr{D}' \Vdash^{X_i}_\alpha F_e(x)] \ \leftrightarrow\ (\exists \mathscr{D}')[\mathscr{D}' \approx_i \mathscr{D}_i \ \&\ \mathscr{D}' \Vdash^{X_{i+1}}_\alpha F_e(x)].$$

We set $\mathscr{D}_{i+1}$ to be this $\mathscr{D}' \approx_i \mathscr{D}_i$ such that $\mathscr{D}' \Vdash^{X_{i+1}}_\alpha F_e(x)$. Since $i \geq |\mathscr{C}| = l$ and $\mathscr{D}_i \supseteq_p \mathscr{C}$, we have $\mathscr{D}_{i+1} \supseteq_p \mathscr{C}$. Eventually, we obtain $\mathscr{D}_m$ such that $|\mathscr{D}_m| = m$, $\mathscr{D}_m \supseteq_p \mathscr{C}$ and $\mathscr{D}_m \Vdash^Y_\alpha F_e(x)$. The direction $(\leftarrow)$ is symmetric. □

Lemma 4. *Let us fix a computable ordinal $\alpha \geq 1$. Let $\mathcal{B}_0$ and $\mathcal{B}_1$ be computable structures in the language $\mathscr{L}$ with equality and both structures satisfy the same Σ^c_α sentences in $\mathscr{L}$. Then for every partial condition $\mathscr{C}$, $X \in 2^\omega$ and natural numbers e, x:*

1) $\mathscr{C} \Vdash^X_\alpha F_e(x) \ \leftrightarrow\ \mathscr{C} \Vdash^{X_{\mathscr{C}}}_\alpha F_e(x)$,
2) $\mathscr{C} \Vdash^X_\alpha \neg F_e(x) \ \leftrightarrow\ \mathscr{C} \Vdash^{X_{\mathscr{C}}}_\alpha \neg F_e(x)$.

Proof. We prove 1) and 2) simultaneously by transfinite induction on α.

Let $\alpha = 1$. For 1), it is clear, by the definition of $\Vdash_1^X$, that for every e and x,

$$\mathscr{C} \Vdash_1^X F_e(x) \leftrightarrow \mathscr{C} \Vdash_1^{X_{\mathscr{C}}} F_e(x).$$

For 2), we have two cases to consider.

i) Let $\mathscr{C} \Vdash_1^X \neg F_e(x)$ and assume $\mathscr{C} \nVdash_1^{X_{\mathscr{C}}} \neg F_e(x)$. Fix $\mathscr{D}_0 \supseteq \mathscr{C}$ such that $\mathscr{D}_0 \Vdash_1^{X_{\mathscr{C}}} F_e(x)$ and let $m = |\mathscr{D}_0|$, $\mathscr{D}' = \mathscr{D}_0 \restriction |\mathscr{C}|$. Since $X_{\mathscr{C}} = X_{\mathscr{D}'}$, we have $(\exists \mathscr{D} \supseteq_p \mathscr{D}')[\mathscr{D} \Vdash_1^{X_{\mathscr{D}'}} F_e(x) \ \&\ |\mathscr{D}| = m]$. Since the finite $X_{\mathscr{D}_0}, X_{\mathscr{D}'}$ differ only at positions $< m$ and $(X_{\mathscr{D}_0})_{\mathscr{D}'} = X_{\mathscr{D}'}$, by Lemma 3, $(\exists \mathscr{D} \supseteq_p \mathscr{D}')[\mathscr{D} \Vdash_1^{X_{\mathscr{D}_0}} F_e(x) \ \&\ |\mathscr{D}| = m]$. We conclude that there is $\mathscr{D} \supseteq_p \mathscr{D}' \supseteq \mathscr{C}$ such that $\mathscr{D} \Vdash_1^{X_{\mathscr{D}}} F_e(x)$ and by 1), $\mathscr{D} \Vdash_1^X F_e(x)$. We reach a contradiction with $\mathscr{C} \Vdash_1^X \neg F_e(x)$.

ii) Let $\mathscr{C} \Vdash_1^{X_{\mathscr{C}}} \neg F_e(x)$ and assume $\mathscr{C} \nVdash_1^X \neg F_e(x)$. In a similar way as in i) we show that we can apply Lemma 3 to reach a contradiction with $\mathscr{C} \Vdash_1^{X_{\mathscr{C}}} \neg F_e(x)$.

For $\alpha > 1$, case 1) follows easily by the definition of the forcing relation $\Vdash_\alpha^X$ and the induction hypothesis for cases 1) and 2). Since we can apply Lemma 3 for every $\beta \leq \alpha$, the proof of 2) for $\alpha > 1$ is essentially the same as for $\alpha = 1$. □

Total Conditions

Let us again fix structures $\mathcal{B}_0$ and $\mathcal{B}_1$ with the same universe B. The *total conditions* are infinite sequences $\mathbf{C} = (f_0, f_1, f_2, \ldots, f_i, \ldots)$, where for all i, f_i is an enumeration of the set B. We denote the total conditions by the letters $\mathbf{C}$ and $\mathbf{G}$. We define the *diagram* of $\mathbf{C}$ *with respect* to $X \in 2^\omega$ to be

$$D_X(\mathbf{C}) = \bigoplus_{j<\omega} f_j^{-1}(\mathcal{B}_{X(j)}).$$

For total conditions, we define the modelling relation $\models_\alpha^X$ for every computable ordinal $\alpha \geq 1$ in a way that mirrors the definition of the forcing relation:

(i) $\mathbf{C} \models_1^X F_e(x) \leftrightarrow x \in W_e^{D_X(\mathbf{C})}$

(ii) Let $\alpha = \beta + 1$. Then

$$\begin{aligned}\mathbf{C} \models_{\beta+1}^X F_e(x) \leftrightarrow (\exists \delta \in Fin_2)[x \in W_e^\delta \ \&\ (\forall z \in \mathrm{Dom}(\delta))[\\ (\delta(z) = 1 \ \&\ \mathbf{C} \models_\beta^X F_z(z)) \vee \\ (\delta(z) = 0 \ \&\ \mathbf{C} \models_\beta^X \neg F_z(z))]].\end{aligned}$$

(iii) Let $\alpha = \lim \alpha(p)$. Then

$$\begin{aligned}\mathbf{C} \models_\alpha^X F_e(x) \leftrightarrow (\exists \delta \in Fin_2)[x \in W_e^\delta \ \&\ (\forall z \in \mathrm{Dom}(\delta))[z = \langle x_z, p_z \rangle \ \& \\ ((\delta(z) = 1 \ \&\ \mathbf{C} \models_{\alpha(p_z)}^X F_{x_z}(x_z)) \vee \\ (\delta(z) = 0 \ \&\ \mathbf{C} \models_{\alpha(p_z)}^X \neg F_{x_z}(x_z)))]].\end{aligned}$$

(iv) $\mathbf{C} \models^X_\alpha \neg F_e(x) \leftrightarrow \mathbf{C} \not\models^X_\alpha F_e(x)$.

Lemma 5. *Let* $\mathbf{C}$ *be a total condition and* $\alpha \geq 1$ *be a computable ordinal. Then*

$$x \in W_e^{\Delta^0_\alpha(D_X(\mathbf{C}))} \leftrightarrow \mathbf{C} \models^X_\alpha F_e(x).$$

For a computable ordinal $\alpha \geq 1$, we say that $\mathbf{C}$ is *α-generic* with respect to X if for every e, x and $1 \leq \beta < \alpha$, $(\exists \mathscr{C} \subset \mathbf{C})[\mathscr{C} \Vdash^X_\beta F_e(x) \vee \mathscr{C} \Vdash^X_\beta \neg F_e(x)]$.

Lemma 6. *For every computable ordinal* $\alpha \geq 1$ *we have the following:*

1) Let $\mathbf{C}$ *be* α*-generic with respect to* X*. Then*

$$\mathbf{C} \models^X_\alpha F_e(x) \leftrightarrow (\exists \mathscr{C} \subset \mathbf{C})[\mathscr{C} \Vdash^X_\alpha F_e(x)].$$

2) Let $\mathbf{C}$ *be* $(\alpha+1)$*-generic with respect to* X*. Then*

$$\mathbf{C} \models^X_\alpha \neg F_e(x) \leftrightarrow (\exists \mathscr{C} \subset \mathbf{C})[\mathscr{C} \Vdash^X_\alpha \neg F_e(x)].$$

3 Construction of a Generic Copy of $\mathcal{N}$

For two functions f and h, let us denote $E(f,h) = \{\langle x, y\rangle \mid f(x) = h(y)\}$.

Proposition 1. *Let* $\mathcal{A} = (A; R)$*,* $R \subseteq A^n$*, and* $\mathcal{N}$ *be defined as in Definition 2. For every total condition* $\mathbf{C} = (q_0, q_1, \dots)$ *and total enumeration* f *of* $\mathcal{A}$*, there is an enumeration* $h_{\mathbf{C}}$ *of* $\mathcal{N}$ *such that* $h_{\mathbf{C}}^{-1}(\mathcal{N}) \leq_T D_{f^{-1}(R)}(\mathbf{C})$ *and* $E(h_{\mathbf{C}}, f)$ *is computable.*

Proposition 2. *For every enumeration* f *of* $\mathcal{A} \oplus X$*, there is a total enumeration* h *of* $\mathcal{A}$ *such that*

1) $E(f,h) \leq_T f^{-1}(\mathcal{A} \oplus X)$*, and*
2) $h^{-1}(\mathcal{A}) \oplus X \leq_T f^{-1}(\mathcal{A} \oplus X)$*.*

Lemma 7. *Let* $\mathcal{A} = (A; R)$*,* α *be a computable* successor *ordinal, and* $\mathcal{B}_0$ *and* $\mathcal{B}_1$ *be computable structures such that:*

a) $\mathcal{B}_0, \mathcal{B}_1$ *are defined in the same language* $\mathscr{L}$*, which includes equality,*
b) $\mathcal{B}_0, \mathcal{B}_1$ *satisfy the same* Σ^c_β *sentences in* $\mathscr{L}$ *for all* $\beta < \alpha$*.*

Then for every enumeration f *of* $\mathcal{A} \oplus \Delta^0_\alpha$*, there is an enumeration* g *of* $\mathcal{N}$ *such that*

1) $E(f,g) \leq_T f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$*,*
2) $\Delta^0_\alpha(g^{-1}(\mathcal{N})) \leq_T f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$*.*

Proof. Let $\alpha = \beta + 1$. By Proposition 2, let us fix, for the given enumeration f of $\mathcal{A} \oplus \Delta^0_\alpha$, a total enumeration h of $\mathcal{A}$ such that $h^{-1}(\mathcal{A}) \oplus \Delta^0_\alpha \leq f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$ and $E(f,h)$ is computable. Our goal is to build a total α-generic condition $\mathbf{G}$ in stages, such that $\mathbf{G} = \bigcup \mathscr{C}_k$. The desired enumeration g will be $h_{\mathbf{G}}$, defined as in Proposition 1. At each stage k, we define a partial condition $\mathscr{C}_{k+1}$ and a finite $X_{k+1} \in 2^\omega$ such that $X_{k+1} = h^{-1}(R)\restriction|\mathscr{C}_{k+1}|$. Let $\mathscr{C}_0 = \emptyset$ and $X_0 = \emptyset$. At step $k = \langle e, x\rangle + 1$, we ask whether $(\exists \mathscr{D} \supseteq \mathscr{C}_k)[\mathscr{D} \Vdash_\beta^{X_k} F_e(x)]$. Since X_k is finite and $\mathcal{B}_0, \mathcal{B}_1$ are computable, this question can be expressed by a Σ^c_β sentence and thus we can decide whether such $\mathscr{D}$ exists effectively relative to Δ^0_α.

If such $\mathscr{D}$ does not exists, then, by definition, $\mathscr{C}_k \Vdash_\beta^{X_k} \neg F_e(x)$. We set $\mathscr{C}_{k+1} = \mathscr{C}_k$, $X_{k+1} = X_k$ and go to the next step.

If such $\mathscr{D}$ exists, let $\mathscr{E} = \mathscr{D}\restriction|\mathscr{C}_k|$ and $X' = h^{-1}(R)\restriction|\mathscr{D}|$. Since $X_k = h^{-1}(R)\restriction|\mathscr{C}_k|$, we have $(X')_{\mathscr{E}} = X_k$. Then according to Lemma 3, $(\exists \mathscr{D}' \supseteq_p \mathscr{E})[\mathscr{D}' \Vdash_\alpha^{X'} F_e(x)\ \&\ |\mathscr{D}'| = |\mathscr{D}|]$. We can find the pair $(\mathscr{D}', X')$ such that $\mathscr{D}' \Vdash_\alpha^{X'} F_e(x)$ effectively relative to $h^{-1}(\mathcal{A}) \oplus \Delta^0_\alpha$. Then, if necessary, we enlarge $\mathscr{D}'$ so that for every $i < |\mathscr{C}_k|$, $\tau_i^{\mathscr{D}'}$ is defined on an initial segment of $\mathbb{N}$ and $\tau_i^{\mathscr{D}'} \supsetneq \tau_i^{\mathscr{C}_k}$. By the monotonicity property of the forcing relation, that is Lemma 1, we know that we can do this safely. We set $\mathscr{C}_{k+1}$ to be this enlarged $\mathscr{D}'$ and set $X_{k+1} = X'$. Then we go to the next step.

In the end, we set $\mathbf{G} = \bigcup_i \mathscr{C}_i$, where $g_k = \bigcup_i \tau_k^{\mathscr{C}_i}$ and $\mathbf{G} = (g_0, g_1, \dots)$. By Proposition 1, for $\mathbf{G}$ we define the enumeration $h_{\mathbf{G}}$ of $\mathcal{N}$. Then

$$\begin{aligned} x \in \Delta^0_\alpha(h_{\mathbf{G}}^{-1}(\mathcal{N})) &\leftrightarrow \mathbf{G} \models_\beta^{h^{-1}(R)} F_{\mu(x,\beta)}(x) \\ &\leftrightarrow (\exists k)[\mathscr{C}_k \subseteq \mathbf{G}\ \&\ \mathscr{C}_k \Vdash_\beta^{h^{-1}(R)} F_{\mu(x,\beta)}(x)] \\ &\leftrightarrow (\exists k)[\mathscr{C}_k \subseteq \mathbf{G}\ \&\ \mathscr{C}_k \Vdash_\beta^{X_k} F_{\mu(x,\beta)}(x)]. \end{aligned}$$

By the construction above, we know that at step $k = \langle \mu(x,\beta), x\rangle + 1$ we have answered the question whether $\mathscr{C}_k \Vdash_\beta^{X_k} F_{\mu(x,\beta)}(x)$ or $\mathscr{C}_k \Vdash_\beta^{X_k} \neg F_{\mu(x,\beta)}(x)$. Since the sequence $\{\langle \mathscr{C}_k, X_k\rangle\}_{k\in\omega}$ is computable in $h^{-1}(\mathcal{A}) \oplus \Delta^0_\alpha$, we conclude that $\Delta^0_\alpha(h_{\mathbf{G}}^{-1}(\mathcal{N})) \leq_T h^{-1}(\mathcal{A}) \oplus \Delta^0_\alpha \leq_T f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$. Moreover, by Proposition 1, $E(h_{\mathbf{G}}, h)$ is computable and since $E(h,f) \leq_T f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$ it follows that $E(h_{\mathbf{G}}, f) \leq_T f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$. □

Corollary 2. *Under the conditions of Lemma 7, we have the following:*

1) $\mathrm{DS}(\mathcal{A} \oplus \Delta^0_\alpha) \subseteq \mathrm{DS}_\beta(\mathcal{N})$*, where* $\beta = \alpha - 1$*, if* $\alpha < \omega$ *and* $\beta = \alpha$*, if* $\alpha \geq \omega$*;*
2) $(\forall X \subseteq A)[X \in \Sigma^c_\alpha(\mathcal{N}) \rightarrow X \in \Sigma^c_1(\mathcal{A} \oplus \Delta^0_\alpha)]$.

Proof. We proved in Lemma 7 that for every enumeration f of the structure $\mathcal{A} \oplus \Delta^0_\alpha$, there is an enumeration h of $\mathcal{N}$ such that $\Delta^0_\alpha(h^{-1}(\mathcal{N})) \leq_T f^{-1}(\mathcal{A} \oplus \Delta^0_\alpha)$. Then Property 1) follows from the fact that the degree spectra of $\mathcal{A} \oplus \Delta^0_\alpha$ and $\mathcal{N}$ are closed upwards.

Property 2) follows easily from the theorem by Ash-Knight-Manasse-Slaman [2] and Chisholm [3] that the relatively intrinsically Σ^0_α relations in a structure $\mathcal{A}$ are exactly the Σ^c_α definable relations in $\mathcal{A}$. □

Lemma 8. *Let $\mathcal{A} = (A; R)$, α be a computable* successor *ordinal and $\mathcal{B}_0, \mathcal{B}_1$ be computable structures such that:*

a) $\mathcal{B}_0, \mathcal{B}_1$ are defined in the same language $\mathscr{L}$, which includes equality,
b) each $\mathcal{B}_i$ satisfies some Σ^c_α sentence in $\mathscr{L}$ that is not true in the other.

Then for every enumeration f of $\mathcal{N}$, there is an enumeration h of $\mathcal{A} \oplus \Delta^0_\alpha$ such that:

1) $E(f, h) \leq_T f^{-1}(\mathcal{N})$, and
2) $h^{-1}(\mathcal{A} \oplus \Delta^0_\alpha) \leq_T \Delta^0_\alpha(f^{-1}(\mathcal{N}))$.

Proof. Let f be the given enumeration of $\mathcal{N}$. We define h, an enumeration of $A \cup \mathbb{N}$, as $h(2n) = f(n)$ for all $n \in f^{-1}(A)$ and $h(2n+1) = n$, for all $n \in \mathbb{N}$. It is clear that $E(f, h)$ is computable in $f^{-1}(\mathcal{N})$.

For any $x_1, \ldots, x_n$, let $i = \langle x_1, \ldots, x_n \rangle$ and $\bar{a}_i = (f(x_1), \ldots, f(x_n))$. To check if $2i \in h^{-1}(R)$, we need to determine k in $\mathcal{U}_{\bar{a}_i} \cong \mathcal{B}_k$. Since we have Σ^c_α sentences Φ and Ψ such that $\mathcal{B}_0 \models (\Phi \ \& \ \neg\Psi)$ and $\mathcal{B}_1 \models (\neg\Phi \ \& \ \Psi)$, we can do that effectively relative to $\Delta^0_\alpha(f^{-1}(\mathcal{N}))$. Thus, $h^{-1}(R) \leq_T \Delta^0_\alpha(f^{-1}(\mathcal{N}))$.

The sets $h^{-1}(G_S)$ and $h^{-1}(\mathbb{N})$ are computable and since $h^{-1}(\Delta^0_\alpha) \equiv_T \Delta^0_\alpha$, we conclude that $h^{-1}(\mathcal{A} \oplus \Delta^0_\alpha) \leq_T \Delta^0_\alpha(f^{-1}(\mathcal{N}))$. □

We conclude by stating the following corollary, which is symmetric to Corollary 2.

Corollary 3. *Under the conditions of Lemma 8, we have the following:*

1) $\mathrm{DS}_\beta(\mathcal{N}) \subseteq \mathrm{DS}(\mathcal{A} \oplus \Delta^0_\alpha)$, where $\beta = \alpha - 1$, if $\alpha < \omega$ and $\beta = \alpha$, if $\alpha \geq \omega$;
2) $(\forall X \subseteq A)[X \in \Sigma^c_1(\mathcal{A} \oplus \Delta^0_\alpha) \ \rightarrow \ X \in \Sigma^c_\alpha(\mathcal{N})]$.

Now Corollary 2 and Corollary 3 give us exactly Theorem 2.

References

1. Ash, C., Knight, J.: Computable Structures and the Hyperarithmetical Hierarchy. Elsevier Science (2000)
2. Ash, C., Knight, J., Manasse, M., Slaman, T.: Generic Copies of Countable Structures. Annals of Pure and Applied Logic 42, 195–205 (1989)
3. Chisholm, J.: Effective Model Theory vs. Recursive Model Theory. The Journal of Symbolic Logic 55(3), 1168–1191 (1990)
4. Goncharov, S., Harizanov, V., Knight, J., McCoy, C., Miller, R., Solomon, R.: Enumerations in computable structure theory. Annals of Pure and Applied Logic 136, 219–246 (2005)
5. Richter, L.: Degrees of Structures. The Journal of Symbolic Logic 46(4), 723–731 (1981)
6. Soskova, A., Soskov, I.: A Jump Inversion Theorem for the Degree Spectra. Journal of Logic and Computation 19, 199–215 (2009)
7. Stukachev, A.I.: A Jump Inversion Theorem for the semilttices of Σ-degrees. Siberian Advances in Mathematics 20(1), 68–74 (2009)

On Algorithmic Strong Sufficient Statistics

Nikolay Vereshchagin*

Department of Mathematical Logic and the Theory of Algorithms,
Faculty of Mechanics and Mathematics, Lomonosov Moscow State University,
Leninskie gory 1, Moscow 119991, Russia
ver@mccme.ru

Abstract. The notion of a strong sufficient statistic was introduced in [8]. In this paper, we give a survey of nice properties of strong sufficient statistics and show that there are strings for which complexity of every strong sufficient statistic is much larger than complexity of its minimal sufficient statistic.

1 Introduction

Sufficient Statistics. Let x be a binary string. A finite set $A \subset \{0,1\}^*$ is called an *(algorithmic) sufficient statistic for* x if $x \in A$ and the sum of the Kolmogorov complexity[1] of A and the binary logarithm of the cardinality of A is close to the Kolmogorov complexity of x:

$$C(A) + \log |A| \approx C(x).$$

More specifically, we call A an *ε-sufficient* statistic for x if the left hand side exceeds the right hand side by at most ε. We do not require the inverse inequality, as it holds with precision $O(\log C(x))$ anyway.

For every x the singleton $\{x\}$ is an $O(1)$-sufficient statistic for x. The complexity of this statistic is about $C(x)$. If x is a random string of length n (that is, $C(x) \approx n$) then there is a $O(\log n)$-sufficient statistic for x of much lower complexity: the set of all strings of length n, whose complexity is about $\log n$, is a $O(\log n)$-statistic for x. We shall think further of ε as having the order $O(\log n)$ and call such values *negligible*.

Sufficient Statistics and Useful Information. Sufficient statistics for x are usually thought to capture all the "useful" information from x. The explanation is the following. Let A be a sufficient statistic for x. One can show that in this case both the *randomness deficiency* $\log |A| - C(x|A)$ of x in A and $C(A|x)$ are negligible.[2] Let z be the binary notation of the ordinal number of x in A

* The work was in part supported by the RFBR grant 12-01-00864 and the ANR grant ProjetANR-08-EMER-008.

[1] Kolmogorov complexity of finite subsets of $\{0,1\}^*$ is defined as follows. We fix any computable bijection $B \mapsto [B]$ from the family of all finite subsets of $\{0,1\}^*$ to the set of binary strings, called an *encoding*. Then we define $C(A)$ as the complexity $C([A])$ of the code $[A]$ of A.

[2] $C(x|A)$ and $C(A|x)$ are defined as $C(x|[A])$ and $C([A]|x)$, respectively, where $A \mapsto [A]$ is a fixed computable encoding of sets by strings (see the previous footnote).

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 424–433, 2013.

(with respect to the lexicographical order on A). As $C(A|x)$ is negligible, both conditional complexities $C(x|A,z)$ and $C(A,z|x)$ are also negligible.[3] Speaking informally, the two part code (A,z) of x has the same information as x itself, and its second part z is a string of length $\log|A|$ that is random conditional to its first part A. (Indeed, $C(z|A)$ is up to an additive constant equal to $C(x|A)$, which is close to $\log|A|$.) This encourages us to qualify z as an accidental information (noise) in the pair (A,z), and hence in x. In other words, all useful information from x is captured by the set A.

Minimal Sufficient Statistics. If x has a sufficient statistic A of complexity i and log-cardinality j (so that $i+j \approx C(x)$) then for every $k \leqslant j$ it has a sufficient statistic B of complexity $i+k$ and log-cardinality $j-k$ (this statement is true with logarithmic precision; the complexity of B is actually $i+k+O(\log j)$). This was observed in [3,2,5]: the set B is obtained by partitioning A into subsets of size at most 2^{j-k} and considering the part containing x. Thus the most valuable sufficient statistic is the one that has smallest complexity and largest cardinality. Such statistics are informally called *minimal sufficient statistics, MSS,* for x. MSS for x are often considered as the models extracting all useful information from x and having no noise.

When trying to define the notion of an MSS formally, we face the following problem: for certain strings x a negligible increase in ε may cause a large decrease of the minimal complexity of ε-sufficient statistics for x. For such x it is not clear which value of ε to choose in the definition of ε-sufficient statistic and the notion of MSS cannot be defined in a meaningful way. In this paper we shall focus on strings for which this is not the case. To define more carefully what it means, consider for a given string x its *structure set* P_x. It consists of all pairs (i,j) of natural numbers for which x has an (i,j)-description, where an (i,j)-*description* is any set $A \ni x$ with $C(A) \leqslant i$ and $\log|A| \leqslant j$. The boundary of P_x is the graph of the function $h_x(i) = \min\{j \mid (i,j) \in P_x\}$, called the *structure function* of x. For every x the boundary of P_x lies above the *sufficiency line* (with logarithmic precision), which by definition consists of all pairs (i,j) with $i+j = C(x)$ (the dash line on Fig. 1). Sufficient statistics correspond to those pairs (i,j) from P_x that are close to the sufficiency line. We shall say (quite informally) that *a string x has an MSS*, if there is a natural i with $h_x(i) \approx C(x)-i$ and $h_x(i') \gg C(x)-i'$ for all i' which are "significantly less" than i. Notice that by observation from [3,2,5] mentioned above, in this case we also have $h_x(i') \approx C(x)-i'$ for all $i \leqslant i' \leqslant C(x)$ (with logarithmic precision).

Example 1. Let y be a string whose structure function h_y leaves the sufficiency line at the point $(C(y),0)$ (so that $\{y\}$ is essentially the only sufficient statistic

[3] $C(x|A,z)$ is defined as $C(x|[[A],z])$, where $(x,y) \mapsto [x,y]$ is a computable bijection between pairs of strings and strings; the notation $C(A,z|x)$ is understood in a similar way.

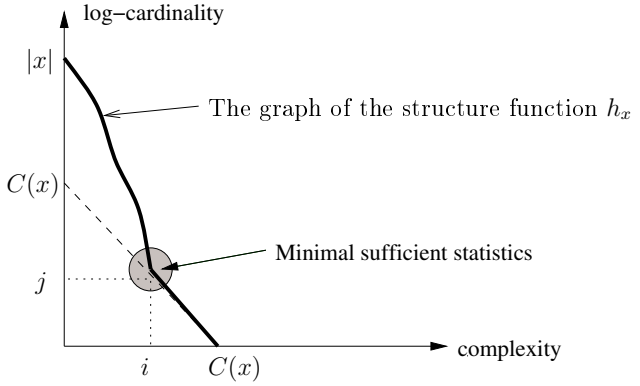


Fig. 1. The structure function h_x. The complexity and log-cardinality of minimal sufficient statistics for x are i and j, respectively.

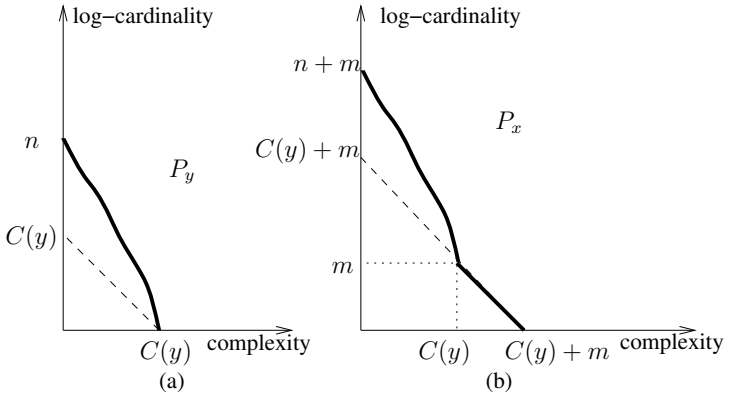


Fig. 2. The sets P_y and P_x

for y), see Fig. 2(a).[4] Let $x = [y, z]$, where z is a string of length m that is random conditional to y (that is, $C(z|y) \approx m$). Intuitively, x is obtained from y by adding m bits of noise and y captures all useful information from x. One can show ([8]) that the set P_x looks as drawn on Fig. 2(b).[5] Consider the set $A = \{[y, z'] \mid |z'| = m\}$ as a model for x. This model is a $(C(y) + O(\log m), m)$-description of x and hence an MSS for x. The information in A is almost the

[4] One can show ([7]) that for every decreasing function $h : \{0, 1, \ldots, k\} \to \mathbb{N}$ with $h(0) \leqslant n$ and $h(k) = 0$ there is a string y of length n for which the boundary of the set P_y is at the distance at most $O(\log n)$ from the graph of h.

[5] More specifically, the set P_x is $O(\varepsilon + \log(C(x) + m + j))$-close to the set

$$\{(i, j) \mid (j \leqslant m \Rightarrow i + j \geqslant C(x)) \ \wedge \ (j \geqslant m \Rightarrow (i, j - m) \in P_x)\}.$$

same as in y, which supports the viewpoint that an MSS for x captures all useful information in x.

Universal Sufficient Statistics. However, as discovered in [2,7], for every string x that has an MSS there is an MSS that can hardly be considered as a denoised version of x. To find such an MSS, fix an algorithm $\mathcal{A}$ that for input k enumerates (in some order) all strings of complexity at most k. Let N_k stand for the number of such strings and let $N_k = 2^{j_1} + 2^{j_2} + \cdots + 2^{j_s}$ be its binary expansion, where $j_1 > j_2 > \cdots > j_s$. Partition the list of strings enumerated by $\mathcal{A}(k)$ into 2^{j_1} first enumerated strings, 2^{j_2} strings enumerated after them and so on. Let S_{k,j_1}, S_{k,j_2}, …, S_{k,j_s} denote the obtained parts.

By definition $|S_{k,j}| = 2^j$ and it is not hard to show that $C(S_{k,j}) \leqslant k - j + O(\log k)$. Let $k = C(x)$. Consider the part where x goes, i.e., find j such that x belongs to $S_{k,j}$. In this case $S_{k,j}$ is a sufficient statistic for x, as $C(S_{k,j}) + \log |S_{k,j}| \leqslant (k - j + O\log k)) + j \approx C(x)$. One can show ([7]) that for every x which has an MSS, for some k close to $C(x)$ and for some j the set $S_{k,j}$ is an MSS for x. This fact is discouraging, because the family $S_{k,j}$ has only two parameters k, j. It implies that for all strings x from Example 1 there are k, j such that the set $S_{k,j}$ is also an MSS for x (where $k \approx C(x) \approx C(y) + m$ and $j \approx m$). Intuitively $S_{k,j}$ has no information about x, and on the other hand one can show that both conditional complexities $C(S_{k,j}|y)$ and $C(y|S_{k,j})$ are negligible. (See [7,8] for more details.)

Total Conditional Complexity. Thus we have to explain why it happens that the good model A from Example 1 has the same information as the bad model $S_{k,j}$. Also we would like to identify a property of MSS allowing to distinguish between good and bad models, such as the model A from Example 1 and the model $S_{k,j}$.

The first question is easy to answer: our definition of "having the same information" is too broad, we implicitly assumed that u and v have the same information, if both $C(u|v)$ and $C(v|u)$ are negligible. Under this assumption every string x has the same information as its shortest description x^*. In the context of separating the information into a useful one and an accidental one, such an assumption is certainly misleading. Indeed, the entire information in x^* (which is a random string) is noise, while x may have useful information. In algorithmic statistics, it is more helpful to think that u and v have the same information only if *total* conditional complexities $\mathrm{CT}(u|v)$ and $\mathrm{CT}(v|u)$ are negligible. The total conditional complexity $\mathrm{CT}(u|v)$ is defined as the minimal length of a total program p for u conditional to v: $\mathrm{CT}(u|v) = \min\{|p| \mid U(p, v) = u$ and $U(p, z)$ halts for all $z\}$ (here U is the universal Turing machine). The total conditional complexity can be much greater than the ordinary one [6].[6] If both $\mathrm{CT}(u|v)$ are

[6] In particular, in the full version of the paper we shall show that for all n there is string x of length n with $\mathrm{CT}(x|p) \geqslant n/3 - O(1)$ for every shortest description p of x. Moreover, this inequality holds for every description p of x of length at most $C(x) + n/3$. On the other hand, by a result of [1], for every x of length n there is a description p of x with $\mathrm{CT}(p|x) = O(\log n)$ and $|p| \leqslant C(x) + O(1)$.

$\mathrm{CT}(v|u)$ are negligible, then their structure sets P_u and P_v are close to each other and u, v have similar algorithmic-statistical properties. We shall call such strings *equivalent* in the sequel.

Strong Sufficient Statistics and Their Nice Properties. To distinguish between good and bad models, the paper [8] introduced a notion of a *strong* sufficient statistic. We call $A \ni x$ a *strong* statistic (or model) for x if $\mathrm{CT}(A|x)$ is negligible. (We do not assume that A is a sufficient statistic.) As we mentioned, the sufficiency requirement implies only that ordinary (not total) conditional complexity $C(A|x)$ is negligible. That is, not all sufficient statistics are strong (later we shall prove that). More specifically, we call A an *ε-strong* model for x if $\mathrm{CT}(A|x) \leqslant \varepsilon$ and we call A an *ε-good* model for x if A is ε-strong and ε-sufficient for x. We call (quite informally) a set A a *strong MSS for x* if A is an MSS for x and A is a strong model for x.

It easy to see that A is a strong model for x iff both total complexities $\mathrm{CT}(x|A, z)$, $\mathrm{CT}(A, z|x)$ are negligible, where z is the ordinal number of x in A. Indeed, given the pair (A, z) we can find x by means of a short total program (even if A is not strong). Conversely, if A is a strong statistic for x, then from x we can compute A by means of a short total program and then compute the ordinal number of x in A.

Strong sufficient statistics have the following nice properties.

(a) The model A from Example 1 is a strong MSS for x Indeed, given x we can find A by a constant length total program that maps $[y, z]$ to the set $\{[y, z'] \mid |z'| = |z|\}$. That is, x has a strong MSS if and only if x is equivalent to a string of the form specified in Example 1.

(b) Strong MSS are unique in the following sense: if both A, B are strong MSS for x, then $\mathrm{CT}(A|B) \approx \mathrm{CT}(B|A) \approx 0$ [8, Theorem 6]. We state here the result in a highly informal way, for the precise statement see [8].

(c) Good statistics satisfy the observation from [3,2,5]: If x has a good statistic A of complexity i and log-cardinality j, then for every $k \leqslant j$ it has a good statistic B of complexity $i + k$ and log-cardinality $j - k$ (with logarithmic precision): again, the set B is obtained by partitioning A into subsets of size at most 2^{j-k} and considering the part containing x.

Our Result. Recall that one of the goals of introducing the notion of a strong MSS is to separate MSS from Example 1 from MSS of the form $S_{k,j}$. We conjecture that this is true: there are strings x that have ε-strong MSS but have no ε-strong MSS of the form $S_{k,j}$ for some $\varepsilon = \Omega(|x|)$. In this paper we answer another question left open in [8]: is it true that every string that has an MSS has also a strong MSS? We show that this is not the case: there are strings that have MSS but all their strong sufficient statistics have much larger complexity than that of their MSS.

2 Results

Our results establish the existence of strings x that have an MSS but all their strong sufficient statistics have much larger complexity than that of their MSS.

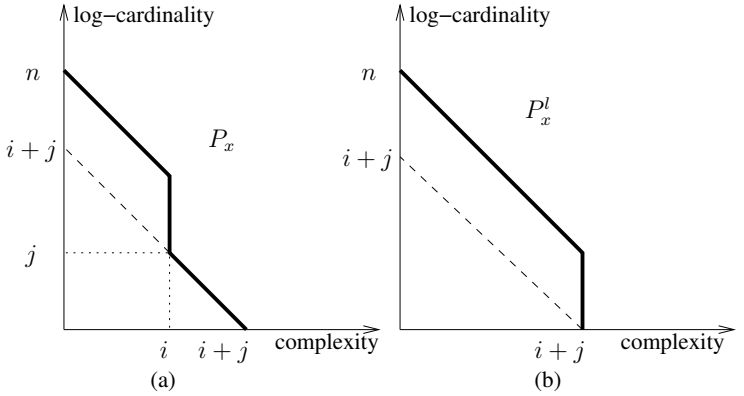


Fig. 3. The sets P_x and P_x^l

Recall that P_x denotes the set of all pairs (i, j) such that x has an (i, j)-description, that is, x belongs to a set A with $C(A) \leqslant i$ and $\log |A| \leqslant j$. Define a similar set P_x^l for l-strong models. By definition P_x^l consists of all pairs (i, j) such that x has an l-strong i, j-description:

$$P_x^l = \{(i, j) \mid (\exists A \ni x)\ C(A) \leqslant i,\ \log |A| \leqslant j,\ \mathrm{CT}(A|x) \leqslant l\}.$$

Our main result, Theorem 1, gives an example of a string x when the set of all explanations and good explanations differ in a maximal possible way: the sets P_x and P_x^l are shown on Fig. 3. The complexity of every l-strong sufficient statistic for x (for some non-negligible l) is at least (about) j bits more than that of its MSS, which is about i. This is the best separation possible, as the singleton $\{x\}$ is an $O(1)$-strong sufficient statistic of complexity about $C(x)$ for every x, and in our case $C(x)$ is about j bits more than i.

Theorem 1. *Assume that integer numbers i, j satisfy the inequality $i+j \leqslant n-4$. Then there is a string x of length n and complexity $i+j+O(\log n)$ such that (a) $(i + O(\log n), j) \in P_x$, (b) $(i, n-i-4) \notin P_x$ and (c) $(i+j, n-i-j-4) \notin P_x^i$.*

Item (a) of Theorem 1 is responsible for the right slanted segment of the boundary of P_x and item (b) is responsible for the left slanted segment of the boundary of P_x. Item (c) is responsible for the graph of P_x^l for any $O(\log n) \leqslant l \leqslant i$.

Let (say) in Theorem 1 $i = j = n/3$. Then the string x existing by the theorem has an MSS of complexity $n/3$ while all $n/3$-strong $n/3$-sufficient statistics for x have complexity at least $2n/3$.

Theorem 1 does not say anything about how rare are such strings x. Such strings are rare, as for majority of strings x of length n the set $\{0, 1\}^n$ is a strong MSS for x. A more meaningful question is whether such strings might appear with high probability in a statistical experiment. More specifically, assume that we sample a string x in a given set $A \subset \{0, 1\}^n$, where all elements

are equiprobable. Might it happen that with high probability (say with probability 99%) x has an MSS but has no strong MSS? An affirmative answer to this question is given in the following

Theorem 2. *Assume that integer i, j, k satisfy the inequalities*

$$i + j \leqslant n - 4, \quad k \leqslant j.$$

Then there is set $A \subset \{0,1\}^n$ of cardinality 2^j and complexity at most $i + O(k + \log n)$ such that all but 2^{j-k} its elements x have complexity $i + j + O(k + \log n)$ and have neither $i, (n - i - 4)$-descriptions nor i-strong $(i + j), (n - i - j - 4)$-descriptions.

If $k = \log n$, say, then the set A is an MSS for a majority of its elements. Indeed, the structure set of all but $|A|/n$ elements from A has the shape shown on Fig. 3(a). On the other hand, for those elements the set P_x^l (for any $O(\log n) \leqslant l \leqslant i$) has the shape shown on Fig. 3(b).

Remark 1. Theorem 2 brings up the following questions. Imagine that somebody suggests a set A as a "statistical explanation" for the data x (that belongs to A). What properties of A are required to make this explanation reasonable? For example, do we want that A is a sufficient statistics for most of its elements? Do we want A to be simple (in the sense of normal conditional complexity or the total one) conditional to *every* its element? We think that defining the notion of a "reasonable explanation" one should be most restrictive, as far as every model as in Example 1 satisfies the restrictions. More specifically, we would call an MSS A a "reasonable explanation" for x if there is a short total program p that maps *every* $x' \in A$ to $[A]$. That is, the total conditional complexity of $[A]$ given any element of A is low in a uniform way. (This implies that A is a sufficient statistics for most of its elements.) This requirement is not that strong as one could think. Indeed, assume that A is a strong MSS for x and p a short total program with $U(p, x) = [A]$. Then the model $A' = \{x' \in A \mid U(p, x') = [A]\}$ is an MSS for x that is a "reasonable explanation" in this sense. Indeed, here is a total program of length about $|p|$ that transforms any $x' \in A'$ to $[A']$: given x' apply p to x' to find A and return the code of the set consisting of all $x'' \in A$ with $U(p, x'') = [A]$.

Proofs of Theorems 1 and 2. We start with the following observation.

Lemma 1. *Assume that A is an i-strong statistic for a string x of length n. Let $y = [A]$ be the code of A. Then y has an $(i + O(\log n), n)$-description.*

Proof. Let p be a string of length at most i such that $U(p, x)$ is defined for all strings x of length n. Consider the set $\{U(p, x) \mid x \in \{0,1\}^n\}$. Its cardinality is at most 2^n and complexity at most $i + O(\log n)$. If $\mathrm{CT}(y|x) \leqslant i$ for some $x \in \{0,1\}^n$ then there is p such that y belongs to such a set and hence y has a $(i + O(\log n), n)$-description.

By Lemma 1, to prove Theorem 1 it suffices to find a set $A \subset \{0,1\}^n$ with
(a) $C(A) \leqslant i + O(\log n)$, $\log |A| \leqslant j$
which is not covered by sets from the following three families:
(b) the family $\mathcal{B}$ consisting of all sets $B \subset \{0,1\}^*$ with with $C(B) \leqslant i$, $\log |B| \leqslant n - i - 4$,
(c) the family $\mathcal{C}$ consisting of all sets M with $C(M) \leqslant i+j$, $\log |M| \leqslant n-i-j-4$ whose code $[M]$ has a $(i + O(\log n), n)$-description, and
(d) the family $\mathcal{D}$ consisting of all singletons sets $\{x\}$ where $C(x) < i + j$.
As x we can take any non-covered string in A. Notice that item (a) implies that the complexity of x is at most $i + j + O(\log n)$, and item (d) implies that it is at least $i + j$.

A direct counting reveals that the family $\mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ covers at most

$$2^{i+1}2^{n-i-4} + 2^{i+j+1}2^{n-i-j-4} + 2^{i+j} \leqslant 2^{n-3} + 2^{n-3} + 2^{n-4} < 2^{n-1}$$

strings and hence at least half of all n-bitstrings are non-covered. However we cannot let A be any 2^j-element non-covered set of n-bitstrings, as in that case $C(A)$ could be large.

We first show how to find A, as in (a), that is not covered by $\mathcal{B} \cup \mathcal{D}$ (but may be covered by $\mathcal{C}$). This is done using the method of [7]. To construct A notice that both the families $\mathcal{B}$ and $\mathcal{D}$ can be enumerated given i, j, n by running the universal machine U in parallel on all inputs. We start such an enumeration and construct A "in several attempts". During the construction we maintain the list of all strings covered by sets from $\mathcal{B} \cup \mathcal{D}$ enumerated so far. Such strings are called *marked*. Initially, no strings are marked and A contains the lexicographic first 2^j strings of length n. Each time a new set $B \in \mathcal{B}$ appears, all its elements receive a b-mark and we replace A by any set consisting of 2^j yet non-marked n-bitstrings. Each time a new set $\{x\}$ in D appears, the string x receives a d-mark, but we do not immediately replace A. We do that only when all strings in A receive a d-mark, replacing it by any set consisting of 2^j yet non-marked n-bitstrings. The above counting shows that such replacements are always possible.

The last version of A (i.e., the version obtained after the last set in $\mathcal{B} \cup \mathcal{D}$ have appeared) is the sought set. Indeed, by construction $|A| = 2^j$ and A is not covered by sets in $\mathcal{B} \cup \mathcal{D}$. It remains to verify that $C(A) \leqslant i + O(\log n)$. This follows from the fact that A is replaced at most $O(2^i)$ times, and hence can be identified by the number of its replacements and i, j, n (we run the above construction of A and wait until the given number of replacements are made).

Why is A replaced at most $O(2^i)$ times? The number of replacements caused by appearance of a new set $B \in \mathcal{B}$ is at most 2^{i+1}. The number of strings with a d-mark is at most 2^{i+j} and hence A can be replaced at most $2^{i+j}/2^j = 2^i$ times due to receiving d-marks.

Now we have to take into account strings covered by sets from the family $\mathcal{C}$. We cannot modify the above arguments just by putting a c-mark on all strings from each set C enumerated into $\mathcal{C}$. Indeed, up to 2^{n-4} strings may receive a c-mark, and hence A might be replaced up to 2^{n-j-4} times due to c-marks.

We change the construction of A as follows. First we represent $\mathcal{C}$ as an intersection of two families, $\mathcal{C}'$ and $\mathcal{C}''$. The first family $\mathcal{C}'$ consists of all sets M with

$C(M) \leqslant i+j$ and the second family $\mathcal{C}''$ of all sets C with $\log|C| \leqslant n-i-j-4$ whose code $[C]$ has a $(i+O(\log n), n)$-description. The first family is small (less than 2^{i+j+1} sets) and the second family has only small sets (at most $2^{n-i-j-4}$-element sets) and is not very large ($|\mathcal{C}''| = 2^{O(n)}$). Both families can be enumerated given i, j, n and, moreover, the sets from $\mathcal{C}''$ appear in the enumeration in at most $2^{i+O(\log n)}$ portions. Due to this property of $\mathcal{C}''$ we can update A each time a new portion of sets in $\mathcal{C}''$ appears—this will increase the number of replacements of A by $2^{i+O(\log n)}$, which is OK.

The crucial change in construction is the following: each time A is replaced, its new version is not just any set of 2^j non-marked n-bitstrings but a carefully chosen such set: we choose any such set that has at most $O(n)$ common strings with *every* set from the part of $\mathcal{C}''$ enumerated so far. (We shall show later that such a set always exists.)

Why does this solve the problem? There are two types of replacements of A: those after enumerating a new set in $\mathcal{B}$ or a new bunch of sets in $\mathcal{C}''$ and those after all elements in A have received c- or d-marks. The number of replacement of the first type is at most $2^{i+O(\log n)}$. Replacements of the second type are caused by enumerating new singleton sets in $\mathcal{D}$ and enumerating new sets C in $\mathcal{C}'$ which were enumerated into $\mathcal{C}''$ on earlier steps. Due to the careful choice of A, when each such set C appears in the enumeration of $\mathcal{C}'$ it can mark only $O(n)$ strings in the current version of A. The total number of sets in $\mathcal{C}'$ is at most 2^{i+j+1}. Therefore the total number of events "a string in the current version of A receives a c-mark" is at most $O(n2^{i+j})$. The total number of d-marks is at most 2^{i+j}. Hence the number of replacements of the second type is at most

$$(O(n2^{i+j}) + 2^{i+j})/2^j = O(n2^i).$$

Thus it remains to show that we indeed can always choose A, as described above. This will follow from a lemma that says that in a large universe one can always choose a large set that has a small intersection with every set from a given small family of small sets.

Lemma 2. *Assume that a finite family $\mathcal{C}$ of subsets of a finite universe U is given and each set in $\mathcal{C}$ has at most s elements. If*

$$|\mathcal{C}|\binom{N}{t+1}\left(\frac{s}{|U|-t}\right)^{t+1} < 1$$

then there is an N-element set $A \subset U$ that has at most t common elements with each set in $\mathcal{C}$.

Proof. To prove the lemma we use probabilistic method. The first element a_1 of A is chosen at random among all elements in U with uniform distribution, the second element a_2 is chosen with uniform distribution among the remaining elements and so forth.

We have to show that the statement of the theorem holds with positive probability. To this end note that for every fixed C in $\mathcal{C}$ and for every fixed set of

indexes $\{i_1, \ldots, i_{t+1}\} \subset \{1, 2, \ldots, N\}$ the probability that *all* $a_{i_1}, \ldots, a_{i_{t+1}}$ fall in C is at most $\left(\frac{s}{|U|-t}\right)^{t+1}$. The number of sets of indexes as above is $\binom{N}{t+1}$. By union bound the probability that a random set A does not satisfy the lemma is upper bounded by the left hand side of the displayed inequality.

We apply the lemma for U consisting of all non-marked n-bitstrings, $N = 2^j$ and $\mathcal{C}$ consisting of all sets in $\mathcal{C}''$ appeared so far. Thus we need to show that for some $t = O(n)$ it holds

$$2^{O(n)} \binom{2^j}{t+1} \left(\frac{2^{n-i-j-4}}{2^{n-1}-t}\right)^{t+1} < 1,$$

which easily follows from the inequality $\binom{2^j}{t+1} \leqslant 2^{j(t+1)}$. Theorem 1 is proved.

Theorem 2 is proved similarly to Theorem 1. The only difference that we change A each time when at least 2^{j-k} strings in A receive c- or d-marks. As the result, the number of changes of A will increase 2^k times and the complexity of A will increase by k.

Acknowledgments. The author is sincerely grateful to anonymous referees for helpful comments, especially for the questions discussed in Remark 1.

References

1. Bauwens, B., Makhlin, A., Vereshchagin, N., Zimand, M.: Short lists with short programs in short time. ECCC report TR13-007, `http://eccc.hpi-web.de/report/2013/007/`
2. Gács, P., Tromp, J., Vitányi, P.M.B.: Algorithmic statistics. IEEE Trans. Inform. Th. 47(6), 2443–2463 (2001)
3. Kolmogorov, A.N.: Talk at the Information Theory Symposium in Tallinn, Estonia (1974)
4. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and its Applications, 2nd edn. Springer, New York (1997)
5. Kh, A.: Shen, Discussion on Kolmogorov complexity and statistical analysis. The Computer Journal 42(4), 340–342 (1999)
6. Shen, A.: Game Arguments in Computability Theory and Algorithmic Information Theory. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 655–666. Springer, Heidelberg (2012)
7. Vereshchagin, N.K., Vitányi, P.M.B.: Kolmogorov's structure functions and model selection. IEEE Trans. Information Theory 50(12), 3265–3290 (2004)
8. Vereshchagin, N.: Algorithmic Minimal Sufficient Statistic Revisited. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 478–487. Springer, Heidelberg (2009)

Analytic Root Clustering: A Complete Algorithm Using Soft Zero Tests*

Chee Yap[1,**], Michael Sagraloff[2], and Vikram Sharma[3]

[1] Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, U.S.A.
yap@cs.nyu.edu
[2] Max-Planck Institute of Computer Science, Saarbrücken, Saarland, Germany
msagralo@mpi-inf.mpg.de
[3] Institute of Mathematical Sciences, Chennai, India
vikram@imsc.res.in

Abstract. A challenge to current theories of computing in the continua is the proper treatment of the **zero test**. Such tests are critical for extracting geometric information. Zero tests are expensive and may be uncomputable. So we seek geometric algorithms based on a weak form of such tests, called **soft zero tests**. Typically, algorithms with such tests can only determine the geometry for "nice" (e.g., non-degenerate, non-singular, smooth, Morse, etc) inputs. Algorithms that avoid such niceness assumptions are said to be **complete**. Can we design complete algorithms with soft zero tests? We address the basic problem of determining the geometry of the roots of a complex analytic function f. We assume effective box functions for f and its higher derivatives are provided. The problem is formalized as the **root clustering problem**, and we provide a complete (δ, ϵ)**-exact** algorithm based on soft zero tests.

1 Introduction: Soft Zero Tests

Almost a century ago, mathematicians and logicians began to develop a theory of computation. It led to the highly successful theory of recursive functions and its higher analogues [16]. Subsequently, in the hands of computer scientists, the lower analogues (at the subrecursive levels) were developed. This is Complexity Theory as we know it today [9]. The lower analogues turn out to have a richer and harder theory: thus, the P versus NP is easily resolved at the higher level. The main line of this development, especially in computer science, is largely about computing over a discrete universe like strings or natural numbers. The issues of computing in the continua, or its surrogate, the real line ($\mathbb{R}$) is sidestepped by this development. One approach to the continua is to use abstract computational models that have operations on continua data, given as primitives. Examples include Theoretical Computer Science under the Real RAM Model,

* This paper was presented at an invited Special Session on "Computational Complexity in the Continuous World" at **Computability in Europe** (CiE2013), July 1-5, Milan, Italy.

** This work is supported by an National Science Foundation Grant #CCF-0917093.

P. Bonizzoni, V. Brattka, and B. Löwe (Eds.): CiE 2013, LNCS 7921, pp. 434–444, 2013.

the Algebraic School [2], and also Information-Based Complexity (IBC) [19]. But a more foundational approach is to consider computational models which (at least in principle) truly operate at the bit level, like Turing machines. The analytic school of real computation [20,11] is the main representative.

Computing over a discrete universe is vastly different than computing over the continua; e.g.,, "brute force search" is not an option in the continua. From the perspective of Exact Geometric Computation (EGC), the touchstone is the **Zero Problem**: deciding if a real constant is zero. According to current models of continua computing either (A) the problem is undecidable, or (B) the problem is trivial by fiat (zero test is a primitive in the model). Our approach in [23] allows the zero problems to have a range of complexity, consistent with what is observed in practice.

The EGC viewpoint is motivated by practical and correct implementation of continua algorithms. It is the most successful approach in computational geometry, and implemented in libraries such as `LEDA`, `CGAL`, `Core Library` (see references in [23]). Nevertheless, there are barriers when we address non-linear and/or non-algebraic problems. We are therefore motivated to study weaker notions of exactness in geometric computation. In particular, we explore models of real computation in which only *the non-zero sign* of real constants can be decided: given a numerical constant x (represented implicitly in some way) we can only ask whether $x > 0$ or $x < 0$, but not $x = 0$. See [3], and [22, Section VI]. In terms of programming constructs, we allow guarded statements of the form "if $x > 0$ then do ..." (but there is no immediate else-clause because the failure of "$x > 0$" does not allow us to conclude that $x \leq 0$). The test $x > 0$ is implemented by iterative approximation of x, a paradigm which is nicely captured in the subdivision framework (e.g., [22]). We call these **soft zero tests** (see Sect. 5), and they embody the well-known dictum in numerical computation: *never compare a quantity to zero.* A realistic theoretical model for such computation is the numerical pointer machine [23] based on Schönhage's pointer machines.

What kind of geometric information can we compute using "soft algorithms", i.e., with soft zero tests? Clearly, in practice most computational scientists use such algorithms. But we are interested in **exact algorithms** that guarantee the correct geometry. A striking example is Plantinga and Vegter's soft algorithm [14] for computing isotopic approximations of curves and surfaces. We recently [22] gave a soft algorithm for the Voronoi diagram of polygonal objects. Both these examples had to assume "nice" inputs: the curves and surfaces must be non-singular [14], the Voronoi diagram must be non-degenerate [22]. Algorithms that avoid niceness assumptions on inputs are said to be **complete**. So the main challenge of this paper is to design soft algorithms that are also complete. One way to obtain soft-and-complete algorithms is to exploit algebraic zero bounds. For analytic problems, such bounds are not readily available and we must weaken the exact geometry criteria using the backwards error idea from numerical analysis. Informally, we propose to compute "an ϵ-correct output for some δ-perturbation of the input". The precise usage of these δ, ϵ parameters

will depend on the problem, but generally they lead to the concept of (δ, ϵ)-**exactness**. In summary, our specific goal is to construct (δ, ϵ)-exact algorithms that uses only soft zero tests, and are complete.

In this paper, we achieve this goal for one of the simplest geometric problems in the continua: determining the geometry of zeros of a complex analytic function f [12]. One formulation of this classical problem is called **root isolation**, defined as follows: given an input function f and a region of interest $B_0 \subseteq \mathbb{C}$, to compute a maximal set $\mathcal{D} = \{D_i : i = 1, \ldots, n\}$ of pairwise disjoint disks, each containing exactly one distinct root of f in B_0. For algebraic polynomials, various techniques are available [6]. With soft zero tests, our analytic techniques cannot distinguish between a root of multiplicity k and a cluster of k roots. Hence it is usually assumed that f is "nice", namely, it has simple roots only. With our completeness goal, however, we must allow multiple roots. So we now associate a multiplicity $\mu_i \geq 1$ with each output disk D_i, meaning that D_i contains a "cluster" of μ_i roots (counted with multiplicity). Thus the exact root isolation problem is transformed into the **root clustering problem**. Ours appear to be the first exact algorithm for analytic root clusters.

Related Work

A classic reference for the geometry of roots is Marden [12]; a comprehensive modern account is given in [15]. There is a large literature on exact root isolation for polynomials and its complexity (see [6]). For analytic functions, Giusti et al. [7] noted that "in contrast to polynomials, few algorithms are known for locating and approximating clusters of zeros of analytic functions". Their paper [7] contains a review of what is known, and contains an analysis of Newton iteration (generalized to multiple roots with Schröder's iteration) using a generalization of Smale's α-theory. Like Rump [17], many papers (e.g., [13]) focus on predicates for confirming analytic root clusters; they do not necessarily synthesize these predicates into a global method for locating root clusters. Yakoubsohn [21] uses only exclusion methods (but without root confirmation) and ϵ cut-offs for analytic zeros; he further provided complexity analysis. Another approach to analytic zeros is to use subdivision combined with the argument principle [10,4]. But they are suboptimal because of unnecessary exact root determination in *each* subdivision box.

2 Conditions for Root Clustering

We address two basic questions. First, when does the set of roots in a disk D form a meaningful cluster? Second, what computational properties of the input function f allow us to construct effective and exact root clustering algorithms?

2.1 What Is a Root Cluster?

For a disk $D \subseteq \mathbb{C}$, let $r(D)$ and $m(D)$ denote its radius and center, and for $\alpha > 0$, let αD denote the disk centered at $m(D)$ with radius $\alpha r(D)$. Suppose

$f : \mathbb{C} \to \mathbb{C}$ is an analytic function. Define $\tau(\mu) := \min\{1 + \mu, 3\}$. A disk $D \subseteq \mathbb{C}$ is **isolating** for $f(z)$ if there is a $\mu \geq 0$ such that both D and $\tau(\mu)D$ contain exactly μ roots of f (counted with multiplicity). If $\mu = 0$, then D is called an **exclusion disk**. If $\mu \geq 1$, the non-empty set of roots in D is called a (root) **cluster**. The following shows that our clusters are natural, and are determined only by the "geometry of the roots".

Lemma 1. *Let C_0 be a root cluster of f. Then there is a unique unordered tree $T(C_0)$ rooted at C_0 whose set of nodes are the root clusters contained in C_0. Parent child relation in $T(C_0)$ is defined using the relation: $C \subseteq C' \subseteq C_0$ iff C is a descendent of C'.*

A collection $\mathcal{D} = \{D_1, \ldots, D_n\}$ of pairwise disjoint isolating disks is called an **isolating system** for f in B_0 if (1) each D_i has at least one root and $m(D_i) \in B_0$, and (2) each root of f in B_0 is in some D_i. Call $\mathcal{D}$ an **ϵ-isolating system** in case each $D_i \in \mathcal{D}$ has radius at most ϵ. Note that roots outside B_0 but within distance ϵ from the boundary of B_0 are allowed to appear in $\mathcal{D}$.

We now formalize the **root clustering problem**: given an analytic function $f : \mathbb{C} \to \mathbb{C}$, a closed square box $B_0 \subseteq \mathbb{C}$ and $\epsilon > 0$, to compute an ϵ-isolating system for f in B_0. We may omit the ϵ parameter if $\epsilon = \infty$.

2.2 On Box Functions and (δ, ϵ)-Approximations

Unlike algebraic polynomials, it is a non-trivial issue to specify an input analytic function f. In practice, functions are parametrized by numerical parameters. E.g., polynomials are parametrized by coefficients, and hypergeometric functions by their hypergeometric parameters. Such functions may be composed using standard operations. These parameters may be arbitrarily approximated (e.g., the coefficients are algebraic numbers). *Based on these parameters, we assume that f and all its higher derivatives are effectively approximated by box functions*, as explained next.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a function. Write $|x|$ for the ∞-norm of $x \in \mathbb{R}^d$. Following [23], real numbers are approximated by elements of the set $\mathbb{F} = \{m2^n : m, n \in \mathbb{Z}\}$ of dyadic numbers; also, let $\square\mathbb{F}$ denote the set of closed intervals with endpoints in $\mathbb{F}$. A **box function** for f, usually denoted $\square f$, is $\square f : \square\mathbb{F}^d \to \square\mathbb{F}$ such that, for any sequence of boxes $B_i \subseteq \square\mathbb{F}^d$ $(i = 1, 2, \ldots)$ that strictly converges to a point $\alpha \in \mathbb{R}^d$ as $i \to \infty$, then $\square f(B_i) \to f(\alpha)$. Box functions are easy to construct using interval arithmetic. We also need the following: A **(δ, ϵ)-approximation** of f is

$$\widehat{f} : \mathbb{F}^{d+1} \to \mathbb{F}^2 \tag{1}$$

such that,[1] for all $x \in \mathbb{F}^d$ and $p \in \mathbb{F}$, if $x' = x \pm 2^{-\widehat{f}_0(x;p)}$, then

$$f(x') = \widehat{f}_1(x; p) \pm 2^{-p}.$$

[1] We write "$a = b \pm \epsilon$" to mean that $|b - a| \leq \epsilon$, and write "$[a \pm \epsilon]$" for the interval $[a - \epsilon, a + \epsilon]$.

Here, $\widehat{f}(x;p)$ is written as the pair $(\widehat{f_0}(x;p), \widehat{f_1}(x;p)) \in \mathbb{F}^2$. We can view $\delta := 2^{-\widehat{f_0}(x;p)}$ and $\epsilon := 2^{-p}$ as the input and output perturbation bounds. The function f is clearly continuous if it has a (δ, ϵ)-approximation, corresponding to the standard definition of continuity: for all $\epsilon > 0$, there exists $\delta > 0$, such that if $x' = x \pm \delta$ then $f(x') = f(x) \pm \epsilon$. Note that given $\square f$, we can construct $\widehat{f}$; the converse is less clear.

These definitions extend to a complex function $f : \mathbb{C} \to \mathbb{C}$ provided we view it as the function $f = (f_x, f_y) : \mathbb{R}^2 \to \mathbb{R}^2$. Then a (δ, ϵ)-approximation of f is just a pair $(\widehat{f_x}, \widehat{f_y})$ where each $\widehat{f_i}$ $(i = x, y)$ is a (δ, ϵ)-approximation of f_i. But we can combine the δ and ϵ parameters of the individual f_i's and obtain $\widehat{f} = (\widehat{f_x}, \widehat{f_y}) : \mathbb{F}^3 \to \mathbb{F}^3$.

An example of parametrized family of functions are hypergeometric functions. For this class, (δ, ϵ)-algorithms are known ([5]); moreover, the derivatives of a hypergeometric function is effectively derived from its parameters.

3 Predicates for Root Clusters

To provide a complete method for localizing roots, we need a predicate $C_k(D)$ to confirm that a given disk $D \subseteq \mathbb{C}$ contains k roots of f, counted with multiplicity. Rump [17] reviewed this problem, giving 10 different predicates. We shall be focusing on one of these predicates, from Pellet [12,15].

Fix an analytic function $f : \mathbb{C} \to \mathbb{C}$. For integer $k \geq 0$ and reals $r, K \geq 1$, define the predicate

$$C_k(m, r, K) : \quad |f_k(m)|r^k > K \sum_{i \neq k} |f_i(m)|\, r^i \tag{2}$$

where $f_i(m) := \frac{f^{(i)}(m)}{i!}$ (coefficients of z^i in the Taylor expansion of $f(z)$ at m). The constant K will be important later when discussing soft versions of these tests. When $K = 1$, just write "$C_k(m, r)$" for $C_k(m, r, K)$. Note that $C_0(m, r)$ (i.e., $k = 0$) is exclusion predicate of [18].

Lemma 2. *If $C_k(m, r)$ holds then the complex disk $D_m(r) \subseteq \mathbb{C}$ contains exactly k roots of f.*

When f is a polynomial, we obtain Pellet's theorem [12].

Theorem 1 (Pellet (1881)). *If $Q(z) = \sum_{i=0}^{n} q_i z^i$ with $q_n q_0 \neq 0$ and $|q_k|z^k - \sum_{i \neq k} |q_i|z^i$ has two real positive roots $r < R$, then Q has exactly k roots in $D_0(r)$ and there are no roots in the annulus $D_0(R) \setminus D_0(r)$.*

Rump observed that Pellet's method is among the best of his 10 methods; the main limitation is that the size of its coefficients tend to overflow machine precision (his experimental setup is limited to machine precision).

Effective Analytic Version C_k Test

For an analytic function f, the C_k test is not effective. For this, we need the complex form of Taylor's Theorem with Remainder. This seems to be a little known result[2] due to Darboux (1876). A more general statement with proof is conveniently provided by Batra [1, Appendix].

Theorem 2 (Darboux). *Let $f : D_0 \to \mathbb{C}$, analytic in an open disk D_0 be given, and let $a, b \in D_0$. Then there exists $0 \leq \Theta \leq 1$ and $\omega \in \mathbb{C}, |\omega| \leq 1$ such that for $h := b - a$ and $\xi := a + \Theta(b - a)$ it holds true that*

$$f(b) = \sum_{\nu=0}^{k} f_\nu(a)h^\nu + \omega h^{k+1} f_{k+1}(\xi).$$

Now we introduce the interval version of the C_k test of (2) above:

$$\square C_k(m, r, K): \quad |f_k(m)|r^k > K\left(\sum_{i=0}^{k-1} |f_i(m)|\, r^i + |\square f_{k+1}(D_m(r))|\, r^{k+1}\right). \quad (3)$$

Again, "$\square C_k(m, r)$" refers to $\square C_k(m, r, 1)$. Here, $\square f_{k+1}(D)$ is some box function for $f_{k+1}(z)$. Note: we use the absolute value $|\square f_{k+1}(\cdots)|$ of the output box. The analogue of Lemma 2 can be shown using Darboux's theorem:

Lemma 3. *If $\square C_k(m, r)$ holds, then $D_m(r)$ contains exactly k roots counted with multiplicities.*

4 Exact Algorithm for Root Clustering

We give a simple version of our root clustering algorithm, assuming the exact evaluation of the predicates C_k and $\square C_k$ and ignoring fine tuning that may be important in practice. Our algorithm uses the classic subdivision paradigm (e.g., [22]). This may be viewed as the repeated subdivision of an initial box $B_0 \subseteq \mathbb{C}$, each box being subdivided ("split") into four congruent subboxes, until all the boxes satisfy some predicate. If X is a box or disk with center m_X and radius r_X, then we write "$C_k(X)$" instead of $C_k(m_X, r_X)$.

Define the **function firstC**(B, N) to return the smallest $k = 0, \ldots, N$ such that $D(2k \cdot B)$ is isolating and contains k roots; otherwise, firstC(B, N) returns -1. To verify that $D(2k \cdot B)$ is isolating, we can check the predicates $\square C_k(2k \cdot B)$ and $\square C_k(\tau(k)2k \cdot B)$. Alternatively, in case f is a polynomial, we can check that $C_k(2k \cdot B)$ and $C_k(\tau(k)2k \cdot B)$ holds.

Our algorithm's input has the form (f, B_0, N) where f is analytic and B_0 is a closed square box such that $D(B_0)$ has at most N roots. For instance, if f is a polynomial, we can choose N to be its degree. For general analytic functions, this N may be first estimated by numerical integration. Our algorithm has three

[2] Thanks to Prashant Batra for bringing this to our attention.

queues Q_0, Q_1 and $\mathcal{D}$. Queue Q_0 contains boxes in arbitrary order, Q_1 is a max-priority queue containing box-integer pairs (B, k), with k as the priority. Queue $\mathcal{D}$ is the output, and contains (B, k) pairs in arbitrary order. Each (B, k) represents an isolating disk $2k \cdot D(B)$ containing k roots. A pair (B, k) and (B', k') is said to be **in conflict** if their isolating disks intersect.

EXACT ROOT CLUSTERING ALGORITHM

Input: $f : \mathbb{C} \to \mathbb{C}$, $B_0 \subseteq \mathbb{C}$, $N \geq 1$, as described.
Output: An isolating system $\mathcal{D}$ for f in B_0.

$Q_0 \leftarrow \{B_0\}, Q_1 \leftarrow \emptyset, \mathcal{D} \leftarrow \emptyset$ ◁ *Initialize Queues*
0. while (Q_0 is non-empty)
  $B \leftarrow Q_0.pop()$
  $k \leftarrow \text{firstC}(B, N)$
1. If $k < 0$, split B and push its 4 children into Q_0.
2. elif $1 \leq k \leq N$, $Q_1.push(B, k)$
3. while (Q_1 is non-empty)
  $(B, k) \leftarrow Q_1.pop()$
4. If (B, k) does not conflict with any pair in $\mathcal{D}$,
5. $\mathcal{D}.push(B, k)$
Return $\mathcal{D}$

Theorem 3. *The Exact Root Clustering Algorithm halts, and produces an isolating system for the roots of f in B_0.*

We easily modify this algorithm to compute an ϵ-isolating systems: Let the precision $p \in \mathbb{F}$ is given as input, $p := \lg(1/\epsilon)$. Replace $\text{firstC}(B, N)$ by $\text{firstC}(B, N, p)$ which returns -1 if $r(B) > 2^{-p} = \epsilon$. Otherwise, it returns $\text{firstC}(B, N)$ as before. For ϵ small enough, we isolate only the roots in B_0.

5 Applications of Soft Zero Tests

The preceding algorithm is exact but not effective as it assumes the exact evaluation of C_k or $\square C_k$ in firstC. Such algorithms are often deemed sufficient (cf. the papers in the section on related work). It is assumed that a numerical implementation of the algorithm can invoke error analysis to tell us the circumstances under which the output is correct. Unfortunately, this falls short of the usual standard for algorithms in theoretical computer science. The solution we shall now provide is to replace the above predicates by their soft versions, denoted $\widetilde{C}_k$ and $\square\widetilde{C}_k$, respectively.

5.1 Soft Zero Test

First consider the following **soft zero test**: given two numerical expressions A and B, both non-negative and at least one positive, determine either the

non-zero sign of $A - B$, or that A, B are **relatively equal** in the sense that $\frac{1}{2}A < B < 2A$. Observe that if A, B are relative equal but $A \neq B$, then the output is non-deterministic: both the (correct) non-zero sign of $A-B$ or relative equality are possible outputs. Write $(A)_p$ to mean any p-bit approximation of A, i.e., $(A)_p = A \pm 2^{-p}$. We are allowed to compute any p-bit approximation of A and B for this problem. Here is our Soft Zero Test procedure: start with $p = 1$. We halt if one of the following two conditions hold:

(I) $|(A)_p - (B)_p| > 2^{1-p}$.
(II) $|(A)_p - (B)_p| \leq 2^{1-p}$ and $\max\{(A)_p, (B)_p\} \geq 7 \cdot 2^{-p}$.

If (I) holds, output the sign of $(A)_p - (B)_p$, and if (II) holds, output "RELATIVE EQUALITY". Otherwise, we double p and repeat.

Theorem 4. *The Soft Zero Test procedure halts and is correct.*

We apply the soft zero test to implement soft predicate $\square\widetilde{C}_k(m, r)$ (the case of $\widetilde{C}_k(m, r)$ is similar when we consider polynomials). Recall that we know (δ, ϵ)-approximations $\widehat{f}_i : \mathbb{F}^3 \to \mathbb{F}^3$ of each Taylor coefficient function $f_i(z)$, $i \geq 0$, (see (1)). To decide $\square\widetilde{C}_k(m, r)$, let us write the predicate (3) as the inequality $A > B$ where $A := |f_k(m)|r^k$, and $B = E + F$, with $E := \sum_{i=0}^{k-1} |f_i(m)|r^i$ and $F := |\square f_{k+1}(D_m(r))|\, r^{k+1}$. It is easy to compute $(A)_p, (E)_{p+1}$ and $(F)_{p+1}$ using the $\square f_i$'s. Note that F is an interval, say $[a, b]$, and our approximation amounts to widening the output interval by at most 2^{-p}, $(F)_{1+p} \subseteq [a - 2^{-1-p}, b + 2^{-1-p}]$. So $(B)_p = (E)_{p+1} + (F)_{p+1}$. Therefore, we could apply our soft zero test to determine the non-zero sign $A-B$, or determine the "RELATIVE EQUALITY" of A, B. If $A - B$ is positive, we output success for our soft predicate $\square\widetilde{C}_k(m, r)$, and otherwise failure.

Lemma 4
(a) If the soft $\square\widetilde{C}_k(m, r)$ *succeeds, then exact* $\square C_k(m, r)$ *succeeds.*
(b) If exact $\square C_k(m, r, 2)$ *succeeds, then soft* $\square\widetilde{C}_k(m, r)$ *succeeds.*

We now describe our **Soft Root Clustering Algorithm**. Basically, we use the soft $\square\widetilde{C}_k$ instead of the exact $\square C_k$ in the Exact Root Clustering Algorithm. These predicates are used within the function firstC(B, N). But there is an important twist: we must now test if the disks $D(4k \cdot B)$, not $D(2k \cdot B)$, are isolating for $k = 0, 1, \ldots, N$. With this modification, Thm. 6 (below) and Lemma 4(b) implies halting. Finally, by exploiting our (δ, ϵ)-approximations $\widehat{f}_i : \mathbb{F}^3 \to \mathbb{F}^3$ of the Taylor coefficients, we can turn this into a (δ, ϵ)-algorithm in the sense that we also compute a $\delta^* > 0$ such that for all δ^*-perturbations of the input, our ϵ-output remains correct. Recall that the ϵ input parameter is not explicitly described, but it is easy to take this into account. This yields:

Theorem 5. *The Soft Root Clustering Algorithm is a complete* (δ, ϵ)*-algorithm for the root clustering problem that is based on soft zero tests.*

6 Analysis of C_k Test

Suppose the analytic function $f(z)$ has a root α of multiplicity $k \geq 0$. So $f^{(k)}(\alpha) \neq 0$ and $f^{(j)}(\alpha) = 0$ for $j = 0, \ldots, k-1$. Then

$$f(z) = \sum_{i \geq 0} f_i(\alpha)(z-\alpha)^i = \sum_{i \geq k} f_i(\alpha)(z-\alpha)^i.$$

Notation: Let m denote a point near α, and $r := |m - \alpha|$ (the "radius"). If E, F are numerical expressions that depend on r, we shall write "$E \simeq F$" to mean that, as $r \to 0$, we have $E = F(1 \pm o(1))$. Also "$E \lesssim F$" means $E < F$ as $r \to 0$. Likewise, "$E = O(F)$" means there is a constant $K > 0$ such that $E \leq K \cdot F$ for all r small enough. These notations are illustrated in the next lemma.

Lemma 5. *For $j \geq 0$:*

$$|f_j(m)|r^j \begin{cases} \simeq |f_k(\alpha)|r^k \binom{k}{j} & \text{if } \ j \leq k \\ = O(r^j) & \text{if } \ j > k \end{cases}$$

In our application, instead of using radius $r = |m - \alpha|$, we need to consider cr for some constant $c > 0$:

Lemma 6

$$\sum_{j=0}^{k} \left| \frac{f_j(m)(cr)^j}{f_k(m)(cr)^k} \right| \simeq \left(1 + \frac{1}{c}\right)^k.$$

This follows from the previous lemma by summation. By separating out the f_k term in the previous lemma, we get:

Lemma 7. *If $c \geq k$, then*

$$\sum_{j=0}^{k-1} |f_j(m)|\,(cr)^j \lesssim |f_k(m)|r^k \left(kc^{k-1}(e-1)\right).$$

Theorem 6. *Let $D_i = D_{m_i}(r_i)$ $(i \geq 0)$ be a sequence of disks, $D_{i+1} \subseteq D_i$, that converges to a point α. Let α have multiplicity $k \geq 0$, and c be any constant greater than $(e-1)kK$.*
(1) The test $\square C_k(m_i, cr_i, K)$ succeeds for i large enough.
(2) If f is a polynomial, the test $C_k(m_i, cr_i, K)$ succeeds for i large enough.

7 Conclusion

There is increasing interest in numerical, evaluation-based approaches to exact geometric algorithms: from root isolation to topology of curve and surfaces. Such algorithms are realistic, practical, and have adaptive complexity. It is part of the trend towards symbolic-numeric computation. Until now, the evaluation

algorithms for isolating the roots of a function f have two limitations: (1) they require f to have simple roots, and (2) they assume that f is a polynomial. In this paper, we have produced an evaluation-based algorithm for the general problem of root clustering. Our algorithm (1') allows f to have multiple roots and (2') applies to analytic functions. In the future, we plan to produce complexity analysis as well as implementation of our algorithms. A general challenge is to produce similar soft-but-complete algorithms for other geometric problems.

References

1. Batra, P.: Globally convergent, iterative path-following for algebraic equations. Math. in Computer Sci. 4(4), 507–537 (2010); Special Issue
2. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, New York (1998)
3. Brattka, V., Hertling, P.: Feasible real random access machines. J. of Complexity 14(4), 490–526 (1998)
4. Dellnitz, M., Schütze, O., Zheng, Q.: Locating all the zeros of an analytic function in one complex variable. J. Comput. Appl. Math. 138(2), 325–333 (2002)
5. Du, Z., Yap, C.: Absolute approximation of the general hypergeometric functions. In: In Proc. 7th Asian Symp. on Computer Math (ASCM), December 8-10, pp. 246–249. KIAS, Seoul (2005)
6. Emiris, I.Z., Pan, V.Y., Tsigaridas, E.P.: Algebraic and numerical algorithms. In: Atallah, M.J., Blanton, M. (eds.) Algorithms and Theory of Computation Handbook, 3rd edn., vol. 1, ch. 17. CRC Press Inc., Boca Raton (2012)
7. Giusti, M., Lecerf, G., Salvy, B., Yakoubsohn, J.-C.: On location and approximation of clusters of zeros of analytic functions. Found. Comp. Math. 5(3), 257–311 (2005)
8. Halperin, D., Fogel, E., Wein, R.: CGAL Arrangements and Their Applications. Springer, Berlin (2012)
9. Hemaspaandra, L.A., Ogihara, M.: The Complexity Theory Companion. Springer (2002)
10. Johnson, T., Tucker, W.: Enclosing all zeros of an analytic function - a rigorous approach. J. Comput. Appl. Math. 228(1), 418–423 (2009)
11. Ko, K.-I.: Complexity Theory of Real Functions. Progress in Theoretical Computer Science. Birkhäuser, Basel (1991)
12. Marden, M.: The Geometry of Zeros of a Polynomial in a Complex Variable. Math. Surveys. American Math. Soc., New York (1949)
13. Niu, X.-M., Sakurai, T., Sugiura, H.: A verified method for bounding clusters of zeros of analytic functions. J. Comput. Appl. Math. 199(2), 263–270 (2007)
14. Plantinga, S., Vegter, G.: Isotopic approximation of implicit curves and surfaces. In: Proc. Symp. on Geometry Processing, pp. 245–254. ACM Press, New York (2004)
15. Rahman, Q.I., Schmeisser, G.: Analytic Theory of Polynomials. OUP (2002)
16. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)
17. Rump, S.M.: Ten methods to bound multiple roots of polynomials. J. Computational and Applied Mathematics 156, 403–432 (2003)
18. Sagraloff, M., Yap, C.K.: A simple but exact and efficient algorithm for complex root isolation. In: 36th ISSAC, June 8-11, pp. 353–360. San Jose, California (2011)

19. Traub, J., Wasilkowski, G., Woźniakowski, H.: Information-Based Complexity. Academic Press, Inc. (1988)
20. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)
21. Yakoubsohn, J.-C.: Numerical analysis of a bisection-exclusion method to find zeros of univariate analytic functions. J. of Complexity 21(5), 652–690 (2005)
22. Yap, C., Sharma, V., Lien, J.-M.: Towards Exact Numerical Voronoi diagrams. In: IEEE 9th Int. Symp. of Voronoi Diagrams in Sci. and Eng (ISVD), June 27-29, pp. 2–16. Rutgers U, NJ (2012); Invited Talk
23. Yap, C.K.: Theory of real computation according to EGC. In: Hertling, P., Hoffmann, C.M., Luther, W., Revol, N. (eds.) Real Number Algorithms. LNCS, vol. 5045, pp. 193–237. Springer, Heidelberg (2008)

Author Index

Ambos-Spies, Klaus 1

Barbuti, Roberto 330
Baumbach, Jan 33
Böcker, Sebastian 33
Bournez, Olivier 12
Braga, Marília D.V. 22
Brandt, Ulrike 1
Braverman, Mark 32

Case, John 45
Cervelle, Julien 55
Cicalese, Ferdinando 65
Cordasco, Gennaro 65
Cormode, Graham 78

de Rugy-Altherre, Nicolas 87
Dondi, Riccardo 97
Dragomir, Ciprian 284
Durand-Lose, Jérôme 108

Ehrenfeucht, Andrzej 120
El-Mabrouk, Nadia 97
Endriss, Ulle 123

Finocchi, Irene 124
Fischbach, Tim 135
Fortnow, Lance 147
Franco, Giuditta 149
Franklin, Johanna N.Y. 161
Fujiwara, Makoto 171

Gao, Ziyuan 181
Gargano, Luisa 65
Gasperin, Anthony 191
Gavruskin, Alexander 200
Gawrychowski, Paweł 210
Genova, Daniela 220
Gheorghe, Marian 284
Graça, Daniel S. 12
Greenberg, Noam 230

Habič, Miha E. 231
Haigh, Thomas 241

Hansen, Thomas Dueholm 252
Hashagen, Ulf 263

Ibsen-Jensen, Rasmus 252

Jain, Sanjay 181

Kach, Asher M. 161
Kari, Lila 271
Kartzow, Alexander 273
Khoussainov, Bakhadyr 200
Krishna, Shankara Narayanan 284

Le Roux, Stéphane 294
Lubarsky, Robert S. 306

Maggiolo-Schettini, Andrea 330
Manea, Florin 210
McGregor, Andrew 316
Melnikov, Alexander G. 320
Milanese, Alessio 149
Milanič, Martin 65
Milazzo, Paolo 330
Miller, Russell 161

Nies, André 320
Nowotka, Dirk 210

Pagnani, Andrea 329
Pardini, Giovanni 330
Parida, Laxmi 340
Pauly, Arno 294
Peretyat'kin, Mikhail G. 342
Pouly, Amaury 12

Ralston, Michael 45
Rathjen, Michael 306
Rosone, Giovanna 353
Rozenberg, Grzegorz 120

Sagraloff, Michael 434
Schlicht, Philipp 273
Sciortino, Marinella 353
Seyfferth, Benjamin 135
Sharma, Vikram 434

Solomon, Reed 161
Soskov, Ivan N. 365
Soskova, Mariya I. 371
Stefanovic, Darko 383
Stephan, Frank 181
Stojanovic, Milan N. 383
Stukachev, Alexey 393
Szemerédi, Endre 403

Tedre, Matti 404
Tini, Simone 330

Vaccaro, Ugo 65
Vatev, Stefan 414
Vereshchagin, Nikolay 424

Yap, Chee 434
Yokoyama, Keita 171

Zhong, Ning 12
Ziegler, Martin 1