Olaf Owe
Martin Steffen
Jan Arne Telle (Eds.)

LNCS 6914

Fundamentals of Computation Theory

18th International Symposium, FCT 2011
Oslo, Norway, August 2011
Proceedings

Springer

Olaf Owe   Martin Steffen
Jan Arne Telle (Eds.)

Fundamentals of Computation Theory

18th International Symposium, FCT 2011
Oslo, Norway, August 22-25, 2011
Proceedings

Springer

Volume Editors

Olaf Owe
Martin Steffen
University of Oslo
Department of Informatics
Postboks 1080 Blindern, 0316 Oslo, Norway
E-mail: {olaf,msteffen}@ifi.uio.no

Jan Arne Telle
University of Bergen
Department of Informatics
Postboks 7800, 5020 Bergen, Norway
E-mail: jan.arne.telle@ii.uib.no

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-22952-7 e-ISBN 978-3-642-22953-4
DOI 10.1007/978-3-642-22953-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011933807

CR Subject Classification (1998): F.1, F.2, F.4, G.2

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

This volume contains the papers presented at FCT 2011: The 18th International Symposium on Fundamentals of Computer Theory held during August 22–25, 2011 in Oslo.

The Symposium on Fundamentals of Computation Theory was established in 1977 for researchers interested in all aspects of theoretical computer science, in particular in algorithms, complexity, and formal and logical methods. It is a biennial conference, which has previously been held in Poznań (1977), Wendisch-Rietz (1979), Szeged (1981), Borgholm (1983), Cottbus (1985), Kazan (1987), Szeged (1989), Gosen-Berlin (1991), Szeged (1993), Dresden (1995), Kraków (1997), Iaşi (1999), Riga (2001), Malmö (2003), Lübeck (2005), Budapest (2007), and Wrocław (2009).

The suggested topics of FCT 2011 included algorithms (algorithm design and optimization; combinatorics and analysis of algorithms; computational complexity; approximation, randomized, and heuristic methods; parallel and distributed computing; circuits and boolean functions; online algorithms; machine learning and artificial intelligence; computational geometry; and computational algebra), formal methods (algebraic and categorical methods; automata and formal languages; computability and nonstandard computing models; database theory; foundations of concurrency and distributed systems; logics and model checking; models of reactive, hybrid and stochastic systems; principles of programming languages; program analysis and transformation; specification, refinement and verification; security; and type systems) and emerging fields (ad hoc, dynamic, and evolving systems; algorithmic game theory; computational biology; foundations of cloud computing and ubiquitous systems; and quantum computation).

This year there were 78 reviewed submissions, of which 28 were accepted. The program included three invited talks, by Yuri Gurevich (Microsoft Research), Daniel Lokshtanov (UCSD), and José Meseguer (UIUC). This volume contains the accepted papers, abstracts from Yuri Gurevich and Daniel Lokshtanov, and a full paper on "The Rewriting Logic Semantics Project" by José Meseguer et al.

The symposium took place in the university informatics buildings, "Ole-Johan Dahls hus" and "Kristen Nygaards hus", and was one of the first scientific events to be held in the new building "Ole-Johan Dahls hus". The FCT event was part of the official opening of the new building and therefore augmented with a half-day program on Monday morning on the importance and future of object orientation, honoring the work of Ole-Johan Dahl and Kristen Nygaard. An additional invited speaker, Andrew P. Black (Portland State University), was invited for this session.

We would especially like to thank the members of the Program Committee for the evaluation of the submissions and their subreferees for excellent cooperation in this work. We are grateful to the contributors to the conference, in particular to the invited speakers for their willingness to present interesting new developments.

Furthermore we thank the University of Oslo and the Department of Informatics for hosting the event, and we thank the local organization of the PMA group, in particular Johan Dovland, Cristian Prisacariu (Publicity Chair), Volker Stolz (Workshop Chair), Thi Mai Thoung Tran, and Ingrid Chieh Yu. Last but not least, we gratefully thank our sponsors: the Research Council of Norway, Cisco Systems Norway, DNV (Veritas) Norway, and the Department of Informatics.

June 2011

Olaf Owe
Martin Steffen
Jan Arne Telle

Organization

Program Committee

Wolfgang Ahrendt	Chalmers University of Technology, Sweden
David Coudert	INRIA Sophia Antipolis, France
Camil Demetrescu	Sapienza University of Rome, Italy
Johan Dovland	University of Oslo, Norway
Jiří Fiala	Charles University, Prague, Czech Republic
Martin Hofmann	LMU Munich, Germany
Thore Husfeldt	IT University of Copenhagen and Lund University, Sweden
Alexander Kurz	University of Leicester, UK
Andrzej Lingas	Lund University, Sweden
Olaf Owe	University of Oslo, Norway
Miguel Palomino	Universidad Complutense de Madrid, Spain
Yuri Rabinovich	Haifa University, Israel
Saket Saurabh	The Institute of Mathematical Sciences, Chennai, India
Kaisa Sere	Abo Akademi University, Turku, Finland
Martin Steffen	University of Oslo, Norway
Jan Arne Telle	University of Bergen, Norway
Tarmo Uustalu	Tallinn University of Technology, Estonia
Ryan Williams	IBM Almaden Research Center, San José, USA
Gerhard Woeginger	TU Eindhoven, The Netherlands
David R. Wood	The University of Melbourne, Australia
Wang Yi	Uppsala University, Sweden
Erika Ábrahám	RWTH Aachen, Germany
Peter Ölveczky	University of Oslo, Norway

Additional Reviewers

Antoniadis, Antonios
Becchetti, Luca
Bentea, Lucian
Beringer, Lennart
Birgisson, Arnar
Bjorner, Nikolaj
Boronat, Artur
Bro Miltersen, Peter
Bunde, David
Chatterjee, Krishnendu
Chen, Xin
Chlebikova, Janka
Ciancia, Vincenzo
Clarkson, Michael
Cohen, Nathann
Cortesi, Agostino

Corzilius, Florian
Crowston, Robert
D'Angelo, Gianlorenzo
Damaschke, Peter
Damiani, Ferruccio
Degerlund, Fredrik
Del Tedesco, Filippo
Dell, Holger
Delvenne, Jean-Charles
Dijk, Thomas C. Van
Din, Crystal Chang
Dorbec, Paul
Eklund, Tomas
Fernau, Henning
Fernández-Camacho, María Inés
Fleischer, Rudolf
Floderus, Peter
Fábregas, Ignacio
Gasieniec, Leszek
Gaspers, Serge
Gentilini, Raffaella
Giannopoulou, Archontia
Gopalan, Parikshit
Grabowski, Robert
Greenstreet, Mark
Hansen, Helle Hvid
Harrison, William
Hlineny, Petr
Hougardy, Stefan
Hähnle, Reiner
Jansen, Nils
Jervell, Herman Ruge
Jost, Steffen
Kabanets, Valentine
Kaufmann, Michael
Kazemeyni, Fatemeh
Kim, Eun Jung
Klasing, Ralf
Klin, Bartek
Kowaluk, Miroslaw
Kristiansen, Lars
Kulikov, Alexander
Laibinis, Linas
Laneve, Cosimo
Latte, Markus
Laud, Peeter
Leister, Wolfgang
Lepri, Daniela
Levcopoulos, Christos
Licata, Daniel R.
Lluch Lafuente, Alberto
Loup, Ulrich
M.S., Ramanujan
Marti-Oliet, Narciso
Marx, Dániel
Meister, Daniel
Mnich, Matthias
Monaco, Gianpiero
Nakata, Keiko
Nebel, Frank
Neovius, Mats
Niederreiter, Harald
Nilsson, Bengt
Norell, Ulf
Normann, Dag
Olsson, Roland
Palmigiano, Alessandra
Perdrix, Simon
Persson, Mia
Petre, Ion
Pettie, Seth
Piazza, Carla
Pitts, Andrew
Preoteasa, Viorel
Raman, Venkatesh
Ribichini, Andrea
Riedmiller, Martin
Rosa-Velardo, Fernando
Russo, Alejandro
Ryabko, Boris
Saha, Barna
Salomaa, Kai
Sampaio, Leonardo
Santos-Garcia, Gustavo
Sau, Ignasi
Schoepp, Ulrich
Segura, Clara
Shachnai, Hadas
Sledneu, Dzmitry
Soares, Ronan P.

Sorge, Manuel	Verdejo, Alberto
Staton, Sam	Villanger, Yngve
Stojanovski, Toni	Voge, Marie-Emilie
Tamm, Hellis	Watanabe, Osamu
Tarasyuk, Anton	Winterhof, Arne
Thilikos, Dimitrios	Yan, Li
Tulsiani, Madhur	Ytrehus, Øyvind
Uehara, Ryuhei	Yu, Ingrid
Van Rooij, Johan M.M.	Zantema, Hans

Table of Contents

The Rewriting Logic Semantics Project: A Progress Report

José Meseguer and Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign,
{meseguer,grosu}@illinois.edu

Abstract. Rewriting logic is an executable logical framework well suited for the semantic definition of languages. Any such framework has to be judged by its effectiveness to bridge the existing gap between language definitions on the one hand, and language implementations and language analysis tools on the other. We give a progress report on how researchers in the rewriting logic semantics project are narrowing the gap between theory and practice in areas such as: modular semantic definitions of languages; scalability to real languages; support for real time; semantics of software and hardware modeling languages; and semantics-based analysis tools such as static analyzers, model checkers, and program provers.

1 Introduction

The disconnect between theory and practice is one of the worse evils in computer science. Theory disconnected from practice becomes irrelevant; and practice without theory becomes brute-force, costly and ad-hoc engineering. One of the current challenges in formal approaches to language semantics is precisely how to effectively bridge the gap between theory and practice. There are two distinct dimensions to this gap:

(1) Given a language $\mathcal{L}$, there is often a substantial gap between: (i) a formal semantics for $\mathcal{L}$; (ii) an implementation of $\mathcal{L}$; and (iii) analysis tools for $\mathcal{L}$, including static, dynamic, and deductive tools.
(2) Even if a formal semantics exists for a programming language $\mathcal{L}$, there may not be any formal semantics available at the higher level of software designs and models, or at the lower level of hardware.

Regarding (1), a semantics of $\mathcal{L}$ may just be a "paper semantics," such as some SOS rules on a piece of paper; or it may be a "toy semantics," not for $\mathcal{L}$ itself, but for a greatly simplified sublanguage. Furthermore, the way a compiler for $\mathcal{L}$ is written may have no connection whatever with a formal semantics for $\mathcal{L}$, so that different compilers provide different language behaviors. To make things worse, program analysis tools for $\mathcal{L}$, including tools that supposedly provide some formal analysis, may not be systematically based on a formal semantics either, so that the confidence one can place of the answers from such tools is greatly diminished. Regarding (2), one big problem is that software modeling notations often lack a formal semantics. A related problem is that this lack of semantics manifests itself as a lack of *analytic power*, that is, as an incapacity to uncover expensive design errors which could have been caught by formal analysis.

We, together with many other colleagues all over the world, have been working for years on the *rewriting logic semantics project* (see [77,76,112] for some overview papers at different stages of the project). The goal of this project is to substantially narrow the gap between theory and practice in language specifications, implementations and tools, in both of the above dimensions (1)–(2). In this sense, rewriting logic semantics is a *wide-spectrum framework*, where:

1. The formal semantics of a language $\mathcal{L}$ is used as the *basis* on which both language implementations and language analysis tools are built.
2. The same semantics-based approach is used not just for programming languages, but also for software and hardware modeling languages.

Any attempt to bridge theory and practice cannot be judged by theoretical considerations alone. One has to evaluate the practical effectiveness of the approach in answering questions such as the following:

– *Executability.* Is the semantics executable? How efficiently so? Can semantic definitions be tested to validate their agreement with an informal semantics?
– *Range of Applicability.* Can it be applied to programming languages and to software and hardware modeling languages? Can it naturally support nontrivial features such as concurrency and real time?
– *Scalability.* Can it be used in practice to give full definitions of real languages like Java or C? And of real software and hardware modeling languages?
– *Integrability.* How well can the semantics be integrated with language implementations and language analysis tools? Can it really be used as the *basis* on which such implementations and analysis tools are built?

This paper is a progress report on the efforts by various researchers in the rewriting logic semantics project to positively answer these questions. After summarizing some related work below, we give an overview of rewriting logic semantics in Section 2. Subsequent sections then describe in more detail: (i) modularity of definitions and the support for highly modular definitions provided by the $\mathbb{K}$ framework (Section 3); (ii) semantics of programming languages (Section 4); semantics of real-time language (Section 5); (iv) semantics of software modeling languages (Section 6); (v) semantics of hardware description languages (Section 7); (vi) abstract semantics and static analysis (Section 8); (vii) model checking verification (Section 9); and (viii) deductive verification (Section 10). We finish with some concluding remarks in Section 11.

1.1 Related Work

There is much related work on frameworks for defining programming languages. Without trying to be exhaustive, we mention some of them and point out some relationships to rewriting logic semantics (RLS).

Structural Operational Semantics (SOS). Several variants of structural operational semantics have been proposed. We refer to [112] for an in-depth comparison between SOS and RLS. A key point made in [112], and also made in Section 2.5, is that RLS is a framework supporting many different definitional styles. In particular, it can naturally and faithfully express many diffent SOS styles such as: small-step SOS [99], big-step SOS [56], MSOS [87], reduction semantics [129], continuation-based semantics [43], and the CHAM [12].

Algebraic denotational semantics. This approach, (see [125,49,26,85] for early papers and [47,118] for two more recent books), is the special case of RLS where the rewrite theory $\mathcal{R}_{\mathcal{L}}$ defining a language $\mathcal{L}$ is an equational theory. Its main limitation is that it is well suited for giving semantics to *deterministic* languages, but not well suited for concurrent language definitions.

Higher-order approaches. The most classic higher-order approach is *denotational semantics* [109,110,108,86]. Denotational semantics has some similarities with its first-order algebraic cousin mentioned above, since both are based on semantic equations and both are best suited for deterministic languages. Higher-order functional languages or higher-order theorem provers can be used to give an executable semantics to programming languages, including the use of Scheme in [45], the use of ML in [98], and the use of Common LISP within the ACL2 prover in [61]. There is also a body of work on using monads [81,124,65] to implement language interpreters in higher-order functional languages; the monadic approach has better modularity characteristics than standard SOS. Some higher-order approaches are based on the use of higher-order abstract syntax (HOAS) [97,52] and higher-order logical frameworks, such as LF [52] or λ-Prolog [88], to encode programming languages as formal logical systems; for a good example of recent work in this direction see [78] and references there.

Logic-programming-based approaches. Going back to the Centaur project [22,35], logic programming has been used as a framework for SOS language definitions. Note that λ-Prolog [88] belongs both in this category and in the higher-order one. For a recent textbook giving logic-programming-based language definitions, see [113].

Abstract state machines. Abstract State Machine (ASM) [50] can encode any computation and have a rigorous semantics, so any programming language can be defined as an ASM and thus implicitly be given a semantics. Both big- and small-step ASM semantics have been investigated. The semantics of various programming languages, including Java [114], has been given using ASMs.

Other RLS work. RLS is a collective international project. There is by now a substantial body of work demonstrating the usefulness of this approach, e.g., [23,120,117,115,72,119,31,104,122,42,40,55,25,73,77,30,28,41,34,106,1,116, 36,107,58,54,46,39,5], and we describe some even more recent advances in this paper. A first snapshot of the RLS project was given in [77], a second in [76], and a third in [112], with this paper as the fourth snapshot.

2 Rewriting Logic Semantics in a Nutshell

Before describing in more detail the different advances in the rewriting logic semantics project we give here an overview of it. Be begin with a short summary of rewriting logic as a semantic framework for concurrent systems. Then we explain how it can be used to give both an operational and a denotational semantics to a programming language. Thanks to the distinction between equations and rules, this semantics can be given at various levels abstraction. Furthermore, a wide range of definitional styles can be naturally supported. We explain how rewriting logic semantics has been extended to: (i) real-time languages; (ii) software modeling languages; and (iii) hardware description languages. We finally explain how a rewriting logic semantics can be used for static analysis, and for model checking and deductive verification of programs.

2.1 Rewriting Logic

The goal of rewriting logic [69] is to provide a flexible logical framework to specify concurrent systems. A concurrent system is specified as a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$, where (Σ, E) is an equational theory, and R is a set of (possibly conditional) rewrite rules. The equational theory (Σ, E) specifies the concurrent system's set of states as an algebraic data type, namely, as the initial algebra of the equational theory (Σ, E). Concretely, this means that a distributed state is mathematically represented as an E-equivalence class $[t]_E$ of terms built up with the operators declared in Σ, *modulo* provable equality using the equations E, so that two state representations t and t' describe the *same* state if and only if one can prove the equality $t = t'$ using the equations E.

The rules R specify the system's *local concurrent transitions*. Each rewrite rule in R has the form $t \rightarrow t'$, where t and t' are Σ-terms. The lefthand side t describes a *local firing pattern*, and the righthand side t' describes a corresponding *replacement pattern*. That is, any fragment of a distributed state which is an instance of the firing pattern t can perform a local concurrent transition in which it is replaced by the corresponding instance of the replacement pattern t'. Both t and t' are typically *parametric* patterns, describing not single states, but parametric families of states. The parameters appearing in t and t' are precisely the *mathematical variables* that t and t' have, which can be instantiated to different concrete expressions by a *substitution*, that is, a mapping θ sending each variable x to a term $\theta(x)$. The instance of t by θ is then denoted $\theta(t)$.

The most basic *logical deduction steps* in a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ are precisely atomic concurrent transitions, corresponding to applying a rewrite rule $t \rightarrow t'$ in R to a state fragment which is an instance of the firing pattern t by some substitution θ. That is, up to E-equivalence, the state is of the form $C[\theta(t)]$, where C is the rest of the state no affected by this atomic transition. Then, the resulting state is precisely $C[\theta(t')]$, so that the atomic transition has the form $C[\theta(t)] \rightarrow C[\theta(t')]$. Rewriting is *intrinsically concurrent*, because many other atomic rewrites can potentially take place in the rest of the state C (and in the substitution θ), at the same time that the local atomic transition $\theta(t) \rightarrow \theta(t')$

happens. The rules of deduction of rewriting logic [69,27] (which in general allow rules in R to be *conditional*) precisely describe all the possible, complex concurrent transitions that a system can perform, so that concurrent computation and logical deduction *coincide*.

2.2 Defining Programming Languages

The flexibility of rewriting logic to naturally express many different models of concurrency can be exploited to give *formal definitions of concurrent programming languages* by specifying the concurrent model of a language $\mathcal{L}$ as a rewrite theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$, where: (i) the signature $\Sigma_{\mathcal{L}}$ specifies both the syntax of $\mathcal{L}$ and the types and operators needed to specify semantic entities such as the store, the environment, input-output, and so on; (ii) the equations $E_{\mathcal{L}}$ can be used to give semantic definitions for the *deterministic* features of $\mathcal{L}$ (a sequential language typically has only deterministic features and can be specified just equationally as $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}})$); and (iii) the rewrite rules $R_{\mathcal{L}}$ are used to give semantic definitions for the concurrent features of $\mathcal{L}$ such as, for example, the semantics of threads.

By specifying the rewrite theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ in a rewriting logic language like Maude[1] [32], it becomes not just a mathematical definition but an *executable* one, that is, an *interpreter* for $\mathcal{L}$. Furthermore, one can leverage Maude's generic search and LTL model checking features to automatically endow $\mathcal{L}$ with powerful *program analysis capabilities.* For example, Maude's search command can be used in the module $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ to detect any violations of invariants, e.g., a deadlock or some other undesired state, of a program in $\mathcal{L}$. Likewise, for terminating concurrent programs in $\mathcal{L}$ one can model check any desired LTL property. All this can be effectively done not just for toy languages, but for real ones such as Java and the JVM, Scheme, and C (see Section 4 for a discussion of such "real language" applications), and with performance that compares favorably with state-of-the-art model checking tools for real languages.

2.3 Operational vs. Denotational Semantics

A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ has both a *deduction-based operational semantics*, and an *initial model denotational semantics.* Both semantics are defined naturally out of the proof theory of rewriting logic [69,27]. The deduction-based operational semantics of $\mathcal{R}$ is defined as the collection of *proof terms* [69] of the form $\alpha : t \longrightarrow t'$. A proof term α is an algebraic description of a proof tree proving $\mathcal{R} \vdash t \longrightarrow t'$ by means of the inference rules of rewriting logic. What such proof trees describe are the different *finitary concurrent computations* of the concurrent system axiomatized by $\mathcal{R}$.

A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ has also a *model-theoretic semantics*, so that the inference rules of rewriting logic are sound and complete with respect to

[1] Other rewriting logic languages, such as ELAN or CafeOBJ, can likewise be used. Maude has the advantage of efficiencly supporting not only execution, but also LTL model checking verification.

satisfaction in the class of models of $\mathcal{R}$ [69,27]. Such models are *categories* with a (Σ, E)-algebra structure [69]. These are "true concurrency" denotational models of the concurrent system axiomatized by $\mathcal{R}$. That is, this model theory gives a precise mathematical answer to the question: when do two descriptions of two concurrent computations denote *the same* concurrent computation? The class of models of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ has an *initial model* $\mathcal{T}_{\mathcal{R}}$ [69]. The initial model semantics is obtained as a *quotient* of the just-mentioned deduction-based operational semantics, precisely by axiomatizing algebraically when two proof terms $\alpha : t \longrightarrow t'$ and $\beta : u \longrightarrow u'$ denote the same concurrent computation.

In particular, if a rewrite theory $\mathcal{R}_{\mathcal{L}} = (\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$ specifies the semantics of a concurrent programming language $\mathcal{L}$, its denotational semantics is given by the initial model $\mathcal{T}_{\mathcal{R}_{\mathcal{L}}}$, and its operational semantics is given by the proof terms built by the rewriting deduction. As we explain below, many different styles of operational semantics, including various SOS styles, can be naturally obtained as special instances of this general, logic-based operational semantics.

2.4 The Abstraction Dial

Unlike formalisms like SOS, where there is only one type of semantic rule, rewriting logic semantics provides a key distinction between *deterministic rules*, axiomatized by equations, and concurrent and typycally *non-deterministic* rules, axiomatized by non-equational rules. More precisely, for the rewriting logic semantics $\mathcal{R}_{\mathcal{L}}$ of a language $\mathcal{L}$ to have good executabity properties, we require $\mathcal{R}_{\mathcal{L}}$ to be of the form $\mathcal{R}_{\mathcal{L}} = (\Sigma_{\mathcal{L}}, E_{\mathcal{L}} \cup B_{\mathcal{L}}, R_{\mathcal{L}})$, where: (i) $B_{\mathcal{L}}$ is a collection of *structural axioms*, such as associativity and/or commutativity, and/or identity of certain operators in $\Sigma_{\mathcal{L}}$; (ii) the equations $E_{\mathcal{L}}$ are *confluent modulo* the structural axioms $B_{\mathcal{L}}$; and (iii) the rules $R_{\mathcal{L}}$ are *coherent* with the equations $E_{\mathcal{L}}$ modulo the structural axioms $B_{\mathcal{L}}$ [123]. Conditions (i)–(iii) make $\mathcal{R}_{\mathcal{L}}$ *executable*, so that using a rewriting logic language like Maude we automatically get an interpreter for $\mathcal{L}$.

As already mentioned, what the equations $E_{\mathcal{L}}$ axiomatize are the *deterministic features* of $\mathcal{L}$. Instead, the truly concurrent features of $\mathcal{L}$ are axiomatized by the non-equational rules $R_{\mathcal{L}}$. The assumption of determinism is precisely captured by $E_{\mathcal{L}}$ being a set of *confluent equations* (modulo $B_{\mathcal{L}}$), so that their evalution, if terminating, has a *unique* final result.

All this means that rewriting logic comes with a built-in "abstraction dial." The least abstrac possible position for such a dial is to turn the equations $E_{\mathcal{L}}$ into rules, yielding the theory $(\Sigma_{\mathcal{L}}, B_{\mathcal{L}}, E_{\mathcal{L}} \cup R_{\mathcal{L}})$; this is typically the approach taken by SOS definitions. The specification $\mathcal{R}_{\mathcal{L}} = (\Sigma_{\mathcal{L}}, E_{\mathcal{L}} \cup B_{\mathcal{L}}, R_{\mathcal{L}})$ can already achieve an enormous abstraction, which typically makes the difference between tractable and intractable model checking analysis. The point is that the equations $E_{\mathcal{L}}$ now identify all intermedite execution states obtained by deterministic steps, yielding a typically enormous state space reduction. Sometimes we may be able to turn the dial to an *even more abstract position* by further decomposing $R_{\mathcal{L}}$ as a disjoint union $R_{\mathcal{L}} = R'_{\mathcal{L}} \cup G_{\mathcal{L}}$, so that the rewrite theory $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}} \cup G_{\mathcal{L}} \cup B_{\mathcal{L}}, R'_{\mathcal{L}})$ still satisfies conditions (i)–(iii). That is, we may be able to identify rules $G_{\mathcal{L}}$ describing concurrent executions which, by being confluent, can be turned into equations. For example, for $\mathcal{L} = Java$, the JavaFAN rewriting logic semantics of Java developed by the late Feng Chen turns the abstraction dial as far as possible, obtaining a set E_{Java} with hundreds of equations, and a set R_{Java} with just 5 rules. This enormous state space reduction is a key reason why the JavaFAN model checker compares favorably with other state-of-the-art Java model checkers [40].

But the abstraction story does not end here. After all, the semantics $(\Sigma_{\mathcal{L}}, E_{\mathcal{L}} \cup G_{\mathcal{L}} \cup B_{\mathcal{L}}, R'_{\mathcal{L}})$ obtained by turning the abstraction dial as much as possible is still a *concrete* semantics. We might call it “the most abstract concrete semantics possible.” For many different static analysis purposes one wants to take a further abstraction step, which further collapses the set of states by defining a suitable *abstract semantics* for a language $\mathcal{L}$. The point is that, instead of a “concrete semantics” describing the actual execution of programs in $\mathcal{L}$, one can just as easily define an “abstract semantics” $(\Sigma^A_{\mathcal{L}}, E^A_{\mathcal{L}}, R^A_{\mathcal{L}})$ describing any desired abstraction A of $\mathcal{L}$. A good example is type checkig, where the values manipulated by the abstract semantics are the types. All this means that many different forms of program analysis, much more scalable than model checking based on a language’s concrete semantics, become available essentially for free by using a tool like Maude to execute and analyze one’s desired abstract semantics $(\Sigma^A_{\mathcal{L}}, E^A_{\mathcal{L}}, R^A_{\mathcal{L}})$. This is further discussed in Section 8.

2.5 An Ecumenical Movement

For purposes of formally defining the semantics of a programming language, rewriting logic should be viewed not as a competitor to other approaches, but as an “ecumenical movement” providing a framework where many different definitional styles can happily coexist. From its early stages rewriting logic has been recognized as ideally suited for SOS definitions [74, 66], and has been used to give SOS definitions of programming languages in quite different styles, e.g., [119, 25, 121, 122, 40, 42]. What the paper [112] makes explicit is both the wide range of SOS styles supported, and the possibility of defining new styles that may have specific advantages over traditional ones. Indeed, the intrinsic flexibility of rewriting logic means that it does not prescribe a fixed style for giving semantic definitions. Instead, *many different styles* such as, for example, small-step or big-step semantics, reduction semantics, CHAM-style semantics, modular structural operational semantics, or continuation semantics, can all be naturally supported [112]. But not all styles are equally efficient; for example, small-step semantics makes heavy use of conditional rewrite rules, insists on modeling every single computation step as a rule in $R_{\mathcal{L}}$, and is in practice horribly inefficient. Instead, the continuation semantics style described in [112] and used in, e.g., [40] is very efficient. Furthermore, as already mentioned, the distinction between equations and rules provides an “abstraction dial” not available in some definitional styles but enormously useful for state space reduction purposes. Of particular interest are *modular* definitional styles, which are further discussed in Section 3.

2.6 Defining Real-Time Languages

In rewriting logic, real-time systems are specified with *real-time rewrite theories* [93]. These are just ordinary rewrite theories $\mathcal{R} = (\Sigma, E \cup B, R)$ such that: (i) there is a sort *Time* in Σ such that (Σ, E) contains an algebraic axiomatization of a time data type, where time can be either discrete or continuous; (ii) there is also a sort *GlobalState*, where terms of sort *GlobalState* are pairs (t, r), with t an "untimed state" (which may however contain time-related quantities such as timers), and r is a term of sort *Time* (that is, the global state is an untimed state plus a global clock); and (iii) the rules R are either: (a) *instantaneous* rules, which do not change the time and only rewrite the discrete part of the state, or (b) *tick* rules, of the form

$$(t, r) \rightarrow (t', r') \ \ \mathit{if} \ \ C$$

where t and t' are term patterns describing untimed states, r and r' are terms of sort *Time*, and C is the rule's condition. That is, tick rules advance the global clock and also update the untimed state to reflect the passage of time (for example, timers may be decreased, and so on). Real-Time rewrite theories provide a very expressive semantic framework in which many models of real-time systems can be naturally expressed [93]. The Real-Time Maude language [94] is an extension of Maude that supports specification, simulation, and model checking analysis of real-time systems specified as real-time rewrite theories.

How should the formal semantics of a *real-time* programming language be defined? And how can programs in such a language be formally analyzed? The obvious RLS answers are: (i) "with a real-time rewrite theory," and (ii) "by real-time model checking and/or deductive reasoning based on such a theory." Of course, the effectiveness of these answers has to be shown in actual languages. This is done in Section 5.

2.7 Defining Modeling Languages

It is well known that the most expensive errors in system development are not coding errors but design errors. Since design errors affect the overal structure of a system and are often discovered quite late in the development cycle, they can be enormously expensive to fix. All this is uncontroversial: there is widely-held agreement that, to develop systems, designs themselves should be made machine-representable, and that tools are needed to keep such designs consistent and to uncover design errors as early as possible. This has led to the development of many software modeling languages.

There are however two main limitations at present. The first is that some of these modeling notations lack a formal semantics: they can and do mean different things to different people. The second is that this lack of semantics manifests itself at the practical level as a lack of *analytic power*, that is, as an incapacity to uncover expensive design errors which could have been caught by better analysis. It is of course virtually impossible to solve the second problem without solving the first: without a precise mathematical semantics any analytic claims about satisfaction of formal requirements are meaningless.

The practical upshot of all this is that a semantic framework such as rewriting logic can play an important role in: (i) giving a precise semantics to modeling languages; and in (ii) endowing such languages and notations with powerful formal analysis capabilities. Essentially the approach is the same as for programming languages. If, say, $\mathcal{M}$ is a modeling language, then its formal semantics will be a rewrite theory of the form $(\Sigma_{\mathcal{M}}, E_{\mathcal{M}}, R_{\mathcal{M}})$. If the modeling language $\mathcal{M}$ provides enough information about the dynamic behavior of models, the equations $E_{\mathcal{M}}$ and the rules $R_{\mathcal{M}}$ will make $\mathcal{M}$ *executable*, that is, it will be possible to *simulate* models in $\mathcal{M}$ before they are realized by concrete programs, and of course such models thus become amenable to various forms of *formal analysis.* All these ideas are further discussed in Section 6

2.8 Defining Hardware Description Languages

What is hardware? What is software? It depends in part on the level of abstraction chosen, and on specific implementation decisions: a given functionality may sometimes be realized as microcode, other times as code running on an FPGA, and yet other times may be implemented in custom VLSI. All this means that the difference between the semantics of digital hardware in some Hardware Description Language (HDL), and that of a programming language is not an essential one, just one about which level of abstraction is chosen. From the point of view of rewriting logic, both the semantics of an HDL and that of a programming language can be expressed by suitable rewrite theories. We further discuss the rewriting logic semantics of HDLs in Section 7.

2.9 Formal Analysis Methods and Tools

The fact that, under simple conditions, rewriting logic specifications are executable, means that the rewriting logic semantics of a language, whether a programming language, or a modeling language, or an HDL, is *executable* and therefore yields an *interpreter* for the given language when run on a rewriting logic system such as Maude. Since the language in question may not have any other formal semantics, the issue of whether the semantic definitions correctly capture the language's informal semantics is a nontrivial matter; certainly not trivial at all for real languages which may require hundreds of semantic rules. The fact that the semantics is executable is very useful in this regard, since one can *test* the correctness of the definitions by comparing the results from evaluating programs in the interpreter obtained from the rewriting logic semantics and in an actual language implementation. The usefulness of this approach is further discussed for the case of the semantics of C in Section 4.

Once the language specifier is sufficiently convinced that his/her semantic definitions correctly capture the language's informal semantics, various sophisticated forms of program analysis become possible. If some abstract semantics for the language in question has been defined, then the abstract semantic definition can be directly used as an *static analysis tool.* Since various abstract semantics may be defined for diverse analysis purposes, a collection of such tools may be developed. We further discuss this idea in Section 8.

Using a tool like Maude, the concrete rewriting logic semantics of a language becomes not just an interpreter, but also a *model checker* for the language in question. The point is that Maude can model check properties for any user-specified rewrite theory. Specifically, it can perform reachabilty analysis to detect violations of invariants using its breadth-first search feature; and it can also model check temporal logic properties with its LTL model checker. Such features can then be used to model check programs in the language whose rewriting semantics one has defined, or in an abstraction of it, as explained in Section 9.

Static analysis and model checking do not exhaust the formal analysis possibilities. A language's rewriting logic semantics can also be used as the basis for *deductive reasoning* about programs in such a language. The advantage of directly basing deductive reasonign methods on the semantics is that there is no gap between the operational semantics and the "program logic." This approach has been pioneered by *matching logic* [103,102], a program verification logic, with substantial advantages over both Hoare logic and separation logic, which uses a language's rewriting logic semantics, including the possibility of using patterns to symbolically characterize sets of states, to mechanize the formal verification of programs, including programs that manipulate complex data structures. More on matching logic and the MatchC tool in Section 10.

3 Modular Definitions and the $\mathbb{K}$ Framework

One major impediment blocking the broader use of semantic frameworks is the lack of scalability of semantic definitions. Lack of *modularity* is one of the main causes for this lack of scalability. Indeed, in many frameworks one often needs to redefine the semantics of the existing language features in order to include new, unrelated features. For example, in conventional SOS [99] one needs to more than double the number of rules in order to include an abrupt termination construct to a language, because the termination "signal" needs to be propagated through all the language constructs. Mosses' Modular SOS (MSOS) [87] addresses the non-modularity of SOS; it was shown that MSOS can be faithfully represented in rewriting logic, in a way that also preserves its modularity [23,73,25,24,29]. We here report on the $\mathbb{K}$ framework, developed in parallel with the MSOS approach.

$\mathbb{K}$ [105] is a modular executable semantic framework derived from rewriting logic. It works with terms, but its concurrent semantics is best explained in terms of graph rewriting intuitions [111]. $\mathbb{K}$ was first introduced by the second author in the lecture notes of a programming language design course at the University of Illinois at Urbana-Champaign (UIUC) in Fall 2003 [101], as a means to define concurrent languages in rewriting logic using Maude. Programming languages, calculi, as well as type systems or formal analyzers can be defined in $\mathbb{K}$ by making use of special, potentially nested *cell* structures, and *rules*. There are two types of $\mathbb{K}$ rules: *computational rules*, which count as computational steps, and *structural rules* (or "half equations"), which do not count as computational steps. The role of the structural rules is to rearrange the term so that the computational rules

Original language syntax	K Strictness	K Semantics
$AExp ::= Int$		
$\mid\ Id$		$\langle \underset{i}{\underline{x}} \cdots\rangle_{\mathsf{k}}\ \langle \cdots\ x \mapsto i\ \cdots\rangle_{\mathsf{state}}$
$\mid\ AExp$ `+` $AExp$	[*strict*]	i_1 `+` $i_2 \rightarrow i_1 +_{Int} i_2$
$\mid\ AExp$ `/` $AExp$	[*strict*]	i_1 `/` $i_2 \rightarrow i_1 /_{Int} i_2$ where $i_2 \neq 0$
$BExp ::= Bool$		
$\mid\ AExp$ `<=` $AExp$	[*seqstrict*]	i_1 `<=` $i_2 \rightarrow i_1 \leq_{Int} i_2$
$\mid$ `not` $BExp$	[*strict*]	`not` $t \rightarrow \neg_{Bool}\ t$
$\mid\ BExp$ `and` $BExp$	[*strict*(1)]	*true* `and` $b \rightarrow b$ *false* `and` $b \rightarrow$ `false`
$Stmt ::=$ `skip`		`skip` $\rightarrow \cdot$
$\mid\ Id$ `:=` $AExp$	[*strict*(2)]	$\langle \underset{\cdot}{\underline{x \texttt{ := } i}} \cdots\rangle_{\mathsf{k}}\ \langle \cdots\ x \mapsto \underset{i}{\underline{\_}}\ \cdots\rangle_{\mathsf{state}}$
$\mid\ Stmt$ `;` $Stmt$		s_1 `;` $s_2 \rightharpoonup s_1 \curvearrowright s_2$
$\mid$ `if` $BExp$ `then` $Stmt$ `else` $Stmt$	[*strict*(1)]	`if` *true* `then` s_1 `else` $s_2 \rightarrow s_1$ `if` *false* `then` s_1 `else` $s_2 \rightarrow s_2$
$\mid$ `while` $BExp$ `do` $Stmt$		$\langle \underset{\texttt{if } b \texttt{ then } (s \texttt{ ; while } b \texttt{ do } s) \texttt{ else } \cdot}{\underline{\texttt{while } b \texttt{ do } s}} \cdots\rangle_{\mathsf{k}}$
$Pgm ::=$ `var` **List**$\{Id\}$ `;` $Stmt$		$\langle \underset{s}{\underline{\texttt{var } xl \texttt{ ; } s}} \rangle_{\mathsf{k}}\ \langle \underset{xl \mapsto 0}{\underline{\cdot}} \rangle_{\mathsf{state}}$

Fig. 1. $\mathbb{K}$ definition of IMP: syntax (left), annotations (middle) and semantics (right); $x \in Id$, $xl \in$ **List**$\{Id\}$, $i, i_1, i_2 \in Int$, $t \in Bool$, $b \in BExp$, $s, s_1, s_2 \in Stmt$

can apply. $\mathbb{K}$ rules are *unconditional* (they may have side conditions, though), and they are *context-insensitive*.

We introduce $\mathbb{K}$ by means of a simple imperative language, called IMP. In Section 3.2 we extend IMP with dynamic threads into IMP++, and in Section 8.1 we show how one can use $\mathbb{K}$ to define a type checker for IMP++. This language experiment is borrowed from [105], where more details about $\mathbb{K}$ can be found. We also refer the interested reader to `http://k-framework.org` for papers, workshops and an implementation. Our implementation of $\mathbb{K}$, the $\mathbb{K}$-Maude tool, consists of a translator to Maude, which is implemented using Perl scripting (about 6,000 lines) and Maude (about 9,000 lines).

3.1 $\mathbb{K}$ Semantics of IMP

Figure 1 shows the complete $\mathbb{K}$ definition of IMP, except for the configuration (explained below). The left column gives the IMP syntax. The middle column augments it with $\mathbb{K}$ *strictness attributes*, stating the evaluation strategy of some language constructs. Finally, the right column gives the semantic rules.

Language syntax is typically defined in $\mathbb{K}$ using an "algebraic" context-free notation, i.e., one which allows users to make use of list, set, multiset and map structures without defining them. Note, e.g., that we used **List**$\{Id\}$ as a non-terminal in the syntax of IMP in Figure 1. System configurations are defined in the same style. Configurations in $\mathbb{K}$ are organized as potentially nested structures of *cells*, which are typically labeled to distinguish them from each other. We use angle brackets as cell wrappers. The $\mathbb{K}$ configuration of IMP can be defined as:

$$\mathit{Configuration}_{\mathrm{IMP}} \equiv \langle\langle K\rangle_{\mathsf{k}}\ \langle \mathbf{Map}\{\mathit{Id} \mapsto \mathit{Int}\}\rangle_{\mathsf{state}}\rangle_{\top}$$

In words, IMP configurations consist of a top cell $\langle\ldots\rangle_{\top}$ containing two other cells inside: a cell $\langle\ldots\rangle_{\mathsf{k}}$ which holds a term of sort K (the computation) and a cell $\langle\ldots\rangle_{\mathsf{state}}$ which holds a map from variables to integers. As examples of IMP $\mathbb{K}$ configurations, $\langle\langle \texttt{x := 1; y := x+1}\rangle_{\mathsf{k}}\ \langle\cdot\rangle_{\mathsf{state}}\rangle_{\top}$ is a configuration holding program "`x := 1; y := x+1`" and empty state, $\langle\langle \texttt{x := 1; y := x+1}\rangle_{\mathsf{k}}\ \langle \texttt{x} \mapsto 0\ \ \texttt{y} \mapsto 1\rangle_{\mathsf{state}}\rangle_{\top}$ is a configuration holding the same program and a state $\texttt{x} \mapsto 0$ and $\texttt{y} \mapsto 1$.

The sort K, for *computational structures* or simply *computations*, has a special meaning in $\mathbb{K}$. The intuition for terms of sort K is that they have computational contents, such as programs or program fragments have. Technically, computations automatically extend the syntax of the original language (i.e., all syntactic categories are sunk into K) with a list structure with "$\curvearrowright$" (read "followed by") as binary concatenation of computations and with "$\cdot$" as the empty computation. For example, the intuition for a computation of the form $T_1 \curvearrowright T_2 \curvearrowright \cdots \curvearrowright T_n$ is that the enlisted (computational) tasks should be processed sequentially. Computations give a uniform means to define and handle evaluation contexts and/or continuations as special cases: a computation "$v \curvearrowright c$" can be thought of as "$c[v]$, that is, evaluation context c applied to v" or as "passing v to continuation c". In fact, $\mathbb{K}$ allows one to define evaluation contexts over the language syntax both directly, like in [105], or indirectly, by means of *strictness* attributes like in the middle column in Figure 1. However, one should be aware that these are nothing but convenient *notations*, which desugar into rules. For example, the evaluation strategies of sum, comparison and conditional in IMP specified by the strictness attributes in Figure 1 can be defined using the following *structural rules* (for diversity, we assume that the sum `+` evaluates its arguments non-deterministically and the comparison `<=` evaluates its arguments sequentially):

$$\begin{array}{rcl}
a_1 \texttt{ + } a_2 & \rightleftharpoons & a_1 \curvearrowright \square \texttt{ + } a_2 \\
a_1 \texttt{ + } a_2 & \rightleftharpoons & a_2 \curvearrowright a_1 \texttt{ + } \square \\
a_1 \texttt{ <= } a_2 & \rightleftharpoons & a_1 \curvearrowright \square \texttt{ <= } a_2 \\
i_1 \texttt{ <= } a_2 & \rightleftharpoons & a_2 \curvearrowright i_1 \texttt{ <= } \square \\
\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 & \rightleftharpoons & b \curvearrowright \texttt{if } \square \texttt{ then } s_1 \texttt{ else } s_2
\end{array}$$

The symbol $\rightleftharpoons$ stands for two structural rules, one left-to-right and another right-to-left. The right-hand sides of the structural rules above contain, besides the task sequentialization operator $\curvearrowright$, *freezer* operators containing $\square$ in their names, such as $\square \texttt{+} \_$, $\_ \texttt{+} \square$, etc. The first rule above says that in any expression of the form $a_1 \texttt{+} a_2$, a_1 can be scheduled for processing while a_2 is being held for future processing. Since these rules are bi-directional, they can be used at will to structurally re-arrange the computations. Thus, when iteratively applied from left-to-right they fulfill the role of *splitting* syntax into an *evaluation context* (the tail of the resulting sequence of computational tasks) and a *redex* (the head of the resulting sequence), and when applied right-to-left they fulfill the role of *plugging* syntax into context. Our current implementation of $\mathbb{K}$ automatically generates rules like the above, plus heuristics to apply them in one direction or the other, from strictness annotations to syntax like in Figure 1 (middle column).

Structural rules like those above decompose and eventually push the tasks that are ready for processing to the top (or the left) of the computation. Semantic rules then tell how to process the atomic tasks. The right column in Figure 1 shows the $\mathbb{K}$ semantic rules of IMP. To explain them, let us first discuss the important notion of a $\mathbb{K}$ *rule*, which is a strict generalization of the usual notion of a rewrite rule. $\mathbb{K}$ rules explicitly mention the parts of the term that they read, write, or don't care about. The underlined parts are those which are written by the rule; the term underneath the line is the new subterm replacing the one above the line. All writes in a $\mathbb{K}$ rule are applied in *one parallel step*, and, with some reasonable restrictions discussed in [111] that avoid read/write and write/write conflicts, writes in multiple $\mathbb{K}$ rule instances can also apply in parallel. The elipses " $\cdots$ " represent the volatile part of the term, that is, that part that the current rule does not care about and, consequently, can be concurrently modified by other rules. The operations which are not underlined represent the read-only part of the term: they need to stay unchanged during the application of the rule. For example, consider the assignment rule in Figure 1:

$$\langle \underset{\cdot}{\underline{x \mathrel{:=} i}} \cdots\rangle_{\mathsf{k}}\ \langle\cdots\ x \mapsto \underset{i}{\underline{\_}} \cdots\rangle_{\mathsf{state}}$$

It says that once the assignment $x := i$ reaches the top of the computation, the value of x in the store is replaced by i and the assignment dissolves; in $\mathbb{K}$, "_" is a nameless variable of any sort and "$\cdot$" is the unit (or empty) computation ("$\cdot$" is a polymorphic unit of all list, set and multiset structures). The rule for variable declarations in Figure 1 (last one) expects an empty state and allocates and initializes with 0 all the declared variables; the dotted or dashed lines signify that the rule is structural, which is discussed next.

$\mathbb{K}$ rules are split in two categories: *computational* and *structural*. Computational rules capture the intuition of computational steps in the execution of the defined system or language, while structural rules capture the intuition of structural rearrangement, rather than computational evolution, of the system. We use dashed or dotted lines in the structural rules. Ordinary rewrite rules are particular $\mathbb{K}$ rules, when the entire term is replaced; in this case, we prefer to use the standard notation $l \rightarrow r$ as syntactic sugar for computational rules and the notation $l \rightharpoondown r$ or $l \rightharpoonup r$ as syntactic sugar for structural rules. Figure 1 shows three explicit structural rules (as already discussed, the strictness attributes correspond to implicit ones): $s_1 \mathrel{;} s_2$ is rearranged as $s_1 \curvearrowright s_2$, loops are unrolled when they reach the top of the computation (unconstrained unrolling leads to non-termination), and declared variables are allocated in the state.

$\mathbb{K}$ rewriting is a hybrid between term rewriting and graph rewriting, aimed at keeping the syntactic simplicity of the former and achieving the concurrency semantics benefits of the latter. While rewriting logic can theoretically capture the intended concurrent semantics of graph rewriting [70], the representation in [70] is impractical. The concurrent semantics of $\mathbb{K}$ is given in terms of graph

Original language syntax	K Strictness	K Semantics
$AExp ::= \ldots \mid \texttt{++}\,Id$		$\langle \frac{\texttt{++}\,x}{i +_{Int} 1} \cdots\rangle_{\mathsf{k}}\ \langle\cdots\ x \mapsto \frac{i}{i +_{Int} 1} \cdots\rangle_{\mathsf{state}}$
$Stmt ::= \ldots$		
$\mid\ \texttt{print}\ AExp$	[*strict*]	$\langle \frac{\texttt{print}\, i}{\cdot} \cdots\rangle_{\mathsf{k}}\ \langle\cdots\ \frac{\cdot}{i}\rangle_{\mathsf{output}}$
$\mid\ \texttt{halt}$		$\langle \frac{\texttt{halt} \curvearrowright \_}{\cdot}\rangle_{\mathsf{k}}$
$\mid\ \texttt{spawn}\ Stmt$		$\langle \frac{\texttt{spawn}\, s}{\cdot} \cdots\rangle_{\mathsf{k}}\ \frac{\cdot}{\langle s \curvearrowright die\rangle_{\mathsf{k}}}$
$K ::= \ldots \mid die$		$\langle die\rangle_{\mathsf{k}} \rightharpoonup \cdot$

Fig. 2. K definition of IMP++ (extends that in Figure 1, *without changing anything*)

rewriting, taking into account the explicit sharing and variable volatility in $\mathbb{K}$ rules, but avoiding the notational complexity of graph rewriting. However, our current implementation straightforwardly translates $\mathbb{K}$ rules into rewrite rules and then uses Maude for execution and formal analysis. For example, the rule for assignment above gets translated into a rewrite rule of the form:

$$\langle x \texttt{ := } i \curvearrowright rest\rangle_{\mathsf{k}}\ \langle before\ x \mapsto j\ after\rangle_{\mathsf{state}} \rightarrow \langle rest\rangle_{\mathsf{k}}\ \langle before\ x \mapsto i\ after\rangle_{\mathsf{state}}$$

Even though our current translation to Maude loses concurrency, a serializability result in [111] connecting $\mathbb{K}$ rewriting and rewriting logic reasoning guarantees the soundness of execution and formal analysis of $\mathbb{K}$ using Maude.

3.2 Extending IMP

In this section we highlight the modularity of $\mathbb{K}$ by extending the IMP language in Section 3.1 with variable increment and dynamic threads. Figure 2 shows how the $\mathbb{K}$ semantics of IMP is seamlessly extended into a semantics for IMP++. To accommodate the output, a new cell needs to be added to the configuration:

$$Configuration_{\text{IMP++}} \equiv \langle\langle K\rangle_{\mathsf{k}}\ \langle \textbf{Map}\{Id \mapsto Int\}\rangle_{\mathsf{state}}\ \langle \textbf{List}\{Int\}\rangle_{\mathsf{output}}\ \rangle_{\top}$$

However, note that none of the existing IMP rules needs to change, because each of them only matches what it needs from the configuration. The construct `print` is strict and its rule adds the value of its argument to the end of the output buffer (matches and replaces the unit "·" at the end of the buffer). The rule for `halt` dissolves the entire computation, and the rule for `spawn` creates a new $\langle\ldots\rangle_{\mathsf{k}}$ cell wrapping the spawned statement. The code in this new cell will be processed concurrently with the other threads. The last rule cools down a terminated thread by simply dissolving it; it is a structural rule since, again, we do not want it to count as a computational step.

4 Programming Language Semantics

Having formal semantics for real programming languages, regardless of the formalism that is being used, is undoubtedly a very important step, useful not only to help us understand those languages better but also to serve as a solid foundation for implementations and for program analysis and verification techniques and tools. Using rewriting logic as a formalism for such semantics has the additional benefit that such techniques and tools can be *directly derived* from the language semantics with minimal effort, as shown throughout this paper.

The rewriting logic semantics technique described in Section 3 has been used to define several programming languages or large fragments of them. Some of these languages serve as models for teaching various language paradigms, which we do not mention here but can be found on webpages for programming language courses at UIUC and can be reached from `http://k-framework.org`, while others are real programming languages, such as C [37], Scheme [67], or Java 1.4 [40, 42]. In this section we only briefly discuss the rewrite logic semantics of C [37], more precisely of the ISO/IEC 9899:1999 (C99) standard, as formalized by Chucky Ellison using the $\mathbb{K}$ framework. This semantics is currently being used by several researchers and research groups, both directly in their tools and indirectly as a basis for understanding (and sometimes criticizing) the C language. This has led to the "C Semantics" Google code project repository at `http://c-semantics.googlecode.com/`.

The C semantics defined by Chucky Ellison defines approximately 120 C syntactic operators and 200 intermediate or auxiliary semantic operators. The definitions of these operators are given by 400 semantic rules and 172 helper rules spread over 2333 lines of code (LOC). However, it takes only 37 of those rules (201 LOC) to cover the behavior of statements, and another 119 for expressions (417 LOC). There are 353 rules for dealing with types, memory, and other necessary mechanisms. Finally, there are about 63 rules for the core of the standard library.

This is the most comprehensive formal semantics of C to date. It is executable and thoroughly tested. All aspects related to the features mentioned below are given a direct semantics. *Expressions*: referencing and dereferencing, casts, array indexing, structure members, arithmetic, bitwise, and logical operators, sizeof, increment and decrement, assignments, sequencing, ternary conditional; *Statements*: for, do-while, while, if, if/else, switch, goto, break, continue, return; *Types and Declarations*: enums, structs, unions, bitfields, initializers, static storage, typedefs; *Values*: regular scalar values (signed/unsigned arithmetic and pointer types), structs, unions; *Standard Library*: malloc/free, set/longjmp, basic I/O; *Environment*: command line arguments; *Conversions*: (implicit) argument and parameter promotions and arithmetic conversion, and (explicit) casts.

No matter what the intended use is for a formal semantics, such a use is limited if one cannot achieve confidence in its correctness. To achieve this aim, executable semantics has an immense practical advantage over non-executable semantics, because one can simply test it. The C semantics in [37] has been encapsulated inside a drop-in replacement for Gnu's C Compiler (GCC), called "KCC". This allows one to test the semantics as one would test a compiler. Indeed, the C semantics has been successfully run against all the examples in the Kernigham and Ritchie manual that supposedly cover all the features of ANSI C. Moreover, a series of challenging C programs collected from the Internet, such as programs from the Obfuscated C programming competition, totaling more than 10,000 LOC are included in the regression tests of the C semantics, so these are all executed each time the semantics is changed. In addition to the above, the GCC C-torture-test (which contains 715 C programs conforming to the standard semantics of C99) has been executed in the C semantics and its behavior compared to that of GCC itself, as well as to Intel's C Compiler (ICC).

C is so complex that even dedicated and broadly used compilers like GCC or ICC cannot compile and execute all the programs in the GCC torture-test. All in all, considering all the tests that the C semantics has been tested on, the GCC and ICC compilers successfully passed 99% of them, while the C semantics (compiled into Maude using the $\mathbb{K}$ framework tool) passed 96% of them. The C semantics ran over 90% of these programs in under 5 seconds (each). An additional 6% completed in 10 minutes, 1% in 40 minutes, and 2% further in under 2 days. The remaining programs either did not finish because they were computationally very intensive (such as FFTs), or they made use of features which were not yet defined (such as, e.g., unicode characters in strings). While this is not terribly fast performance, especially when compared to compiled C, the reader should keep in mind that this is an interpreter obtained *for free* from a formal semantics and that other existing semantics of C are either "paper" definitions (e.g., [51]), or not executable (e.g., [91]), or very slow (e.g., we were not able to execute factorial of 6 or the 4th Fibonacci's number using the Haskell-based definition in [95,96]), or covering only a C fragment (e.g., [14]). Moreover, our semantics of C can be used *directly* and *unchanged* for other purposes, such as for model checking (Section 9) and for deductive verification (Section 10).

5 Real-Time Language Semantics

Three real-time programming languages have been given formal semantics as real-time rewrite theories [93] in Real-Time Maude [94]. Using the model checking features of Real-Time Maude it then becomes possible to formally analyze programs in such languages.

In [4], AlTurki et al. present a language for real-time concurrent programming for industrial use in DOCOMO Labs called L. The goal of L is to serve as a programming model for higher-level software specifications in SDL or UML. A related goal is to support formal analysis of L programs by both real-time model checking and static analysis, so that software design errors can be caught at design time. The way all this is accomplished is by giving a formal semantics to L in Real-Time Maude, which automatically provides an interpreter and a real-time model checker for L. Static analysis capabilities are added to L by using Maude to define an *abstract semantics* for L in rewriting logic, which is then used as the static analyzer.

The Orc model of real-time concurrent computation [79,80,126] has been given semantics in rewriting logic using real-time rewrite theories [5,6]. Although Orc is a very simple and elegant language, its real-time semantics is quite subtle for two reasons. First, in the evaluation of any Orc expression, internal computation always has higher priority than the handling of external events; this means that, even without modeling time, a vanilla-flavored SOS semantics is not expressive enough to capture these different priorities: two SOS relations are needed [80]. Second, Orc is by design a real-time language, where time is a crucial feature. Using real-time rewrite theories, this double subtlety of the Orc semantics was faithfully captured in [5]; furthermore, this semantics yielded of course an Orc interpreter and a real-time model checker. But Orc is not just a model of computation: it is also a concurrent programming language. This suggested the following challenge question: can a correct-by-construction distributed Orc implementation be derived from its rewriting logic semantics? This question was answered in two stages. Since, as discussed in Section 2.5, a small-step SOS semantics is typically horribly inefficient and it was certainly so in the case of Orc, a much more efficient *reduction semantics* was first defined in [6], and was proved to be bisimilar to the small-step SOS semantics. This semantics provided a much more efficient interpreter and model checker. Furthermore, to explicitly model different Orc clients and various web sites, and their message passing communication, the Orc semantics was seamlessly extended in [6] to a distributed object-based Orc semantics, which modeled what a distributed implementation should look like. The only remaining step was to pass from this model of a distributed implementation to an actual Maude-based distributed real-time implementation. This was accomplished in [7] using three main ideas: (i) the use of sockets in Maude to actually deploy a distributed implementation; (ii) the systematic replacement of logical time by physical time, supported by Ticker objects external to Maude, while retaining the rewriting semantics throughout; and (iii) the experimental estimation of the physical time required for "zero-time" Maude subcomputations, to ensure that the granularity of time ticks is such that all "instantaneos transitions" have already happened before the next tick.

Creol is an object-oriented language supporting concurrent objects which communicate through asynchronous method calls. Its rewriting-logic-based operational semantics was defined in [55] without real-time features. However, to support applications such as sensor systems with wireless communication, where messages expire and may collide with each other, Creol's design and operational semantics have been extended in [13] to Timed Creol using rewriting logic. The notion of time used by Timed Creol is described as a "lightweight" one in [13]. Time is discrete and is represented by a time object. This approach does not require a full use of the features in Real-Time Maude (Maude itself is sufficient to define the real-time semantics). The effectiveness of Timed Creol in the modeling and analysis of applications such as sensor networks is illustrasted in [13] through a case study.

6 Modeling Language Semantics

Modeling languages are quite useful, but they can be made even more useful by substantially increasing their analytic power through formal analysis, since this can make it possible to catch expensive design errors very early. Formal analysis is impossible or fraudulent without a formal semantics. Early work in developing rewriting-logic-based formal semantics focused on object-oriented design notations and languages [127,90,89], and stimulated subsequent work on UML and UML-like notations, e.g., [44,62,63,128,8,33,83,82,84].

A more ambitious question is: can we give semantics not just to a single modeling language, but to an entire *modeling framework* where different modeling languages can be defined? This question has been answered positively in [16,15,17,19,20]. This line of research has led to MOMENT2, an algebraic model management framework and tool written in Maude and developed by Artur Boronat [15]. It permits manipulating software models in the Eclipse Modeling Framework (EMF). It uses OMG standards, such as Meta-Object Facility (MOF), Object Constraint Language (OCL) and Query/View/Transformation (QVT), as a clean interface between rewriting-logic-based formal methods and model-based industrial tools. Specifically, it supports formal analyses based on rewriting logic and graph transformations to endow model-driven software engineering with strong analytic capabilties. MOMENT2 supports not just one fixed modeling language, but any modeling language whose *meta-model* is specified in MOF. In more detail, a modeling language is specified as a pair $(\mathcal{M}, \mathcal{C})$, where $\mathcal{M}$ is its MOF-based metamodel, and $\mathcal{C}$ are the OCL constraints that $\mathcal{M}$ should satisfy. Using rewriting-logic-based reflection and its efficient support in Maude, MOMENT2 provides an *executable algebraic semantics* for such metamodel specifications $(\mathcal{M}, \mathcal{C})$ in the form of a theory $\mathbb{A}(\mathcal{M}, \mathcal{C})$ in membership equational logic (MEL) [71], so that a model M conformant with the metamodel $(\mathcal{M}, \mathcal{C})$ is exactly a term of sort *Model* in $\mathbb{A}(\mathcal{M}, \mathcal{C})$, and so that satisfaction of OCL constraints is also decidable using the algebraic semantics [18,20].

Due to the executability of MEL specifications in Maude, the realization of MOF metamodels as MEL theories enhances the formalization and prototyping of model-driven development processes, such as: (i) model transformations; (ii) model-driven roundtrip engineering; (iii) model traceability; and (iv) model management. These processes permit, for example, merging models, generating mappings between models, and computing differences between models; they can be used to solve complex scenarios such as the roundtrip problem. In MOMENT2 the formal semantics of *model transformations* is given by rewrite theories specified in a user-friendly QVT-based syntax [17]. Such model transformations can describe the dynamic evolution of systems at the level of their models. Using the search and *LTL* model checking features of Maude, properties about the dynamic evolution of a model M conformant with a metamodel specification $(\mathcal{M}, \mathcal{C})$ can then be formally analyzed by model checking [17]. Real-time modeling languages can likewise be supported and analyzed [21]; this is further discussed below.

6.1 Semantics of Real-Time Modeling Languages

There is strong interest in modeling languages for real-time and embedded systems. The rewriting logic semantics for such modeling languages can be naturally based on real-time rewrite theories. Using a tool like Real-Time Maude, what this means in practice is that such models can then be simulated; and that their formal properties, in particular their safety requirements, can be model checked. Furthermore, the simulations and formal analysis capabilities added to the given modeling language can be offered as "plugins" to already existing modeling tools, so that much of the formal analysis happens "under the hood," and somebody already familiar with the given modeling notation can perform such formal analysis without having an in-depth understanding of the underlying formalism.

The Ptolemy II modeling language (`http://ptolemy.eecs.berkeley.edu`) supports design and simulation of concurrent, real-time, embedded systems expressed in several models of computation (MoCs), such as state machines, data flow, and discrete-event models, that govern the interaction between concurrent components. A user can visually design and simulate hierarchical models, which may combine different MoCs. Furthermore, Ptolemy II has code generation capabilities to translate models into other modeling or programming languages such as C or Java. Discrete-Event (DE) Models are among the most central in Ptolemy II. Their semantics is defined by the *tagged signal model* [64]. The work by Bae et al. in [11] endows DE models in Ptolemy II with formal analysis capabilities by: (i) defining a semantics for them as real-time rewrite theories; (ii) automating such a formal semantics as a model transformation using Ptolemy II's code generation features; (iii) providing a Real-Time Maude plugin, so that Ptolemy II users can use an extended GUI to define temporal logic properties of their models in an intutitive syntax and can invoke Real-Time Maude from the GUI to model check their models. This work has been further advanced in [9] to support not just flat DE models, but *hierarchical* ones. That is, above tasks (i)–(iii) have been extended to hierarchical DE models; this extension is nontrivial, because it requires combining synchronous fixpoint computations with hierarchical structure.

AADL (`http://www.aadl.info/`) is a standard for modeling embedded systems that is widely used in avionics and other safety-critical applications. However, AADL lacks a formal semantics, which severely limits both unambiguous communication among model developers and the formal analysis of AADL models. In [92] Ölveczky et al. define a formal object-based real-time concurrent semantics for a behavioral subset of AADL in rewriting logic, which includes the essential aspects of AADL's behavior annex. Such a semantics is directly executable in Real-Time Maude and provides an AADL simulator and LTL model checking tool called *AADL2Maude*. *AADL2Maude* is integrated with OSATE, so that OSATE's code generation facility is used to automatically transform AADL models into their corresponding Real-Time Maude specifications. Such transformed models can then be executed and model checked by Real-Time Maude. One difficulty with AADL models is that, by being made up of various hierarchical components that communicate asynchronously with each other, their model checking formal analysis can easily experience a combinatorial explosion. However, many such models express designs of distributed embedded systems which, while being asynchronous, should behave in a virtually synchronous way. This suggest the possibility of using the PALS pattern [75], which reduces distributed real-time systems with virtual synchrony to synchronous ones, to pass from simple synchronous systems, which have much smaller state spaces and are much easier to model check, to semantically equivalent asynchronous systems, which often cannot be directly model checked but can be verified indirectly through their synchronous counterparts. This has led to the design of the Synchronous AADL sublanguage in [10], where the user can specify synchronous AADL models by using a sublanguage of AADL with some special keywords. A synchronous rewriting semantics for such models has also been defined in [10]. Using OSATE's code generation facility, synchronous AADL models can be transformed into their corresponding Real-Time Maude specifications in the *SynchAADL2Maude* tool, which is provided as a plugin to OSATE. Likewise, the user can define temporal logic properties of synchronous AADL models based on their features, without requiring knowledge of the underlying formalism, and can model check such models in Real-Time Maude.

A more ambitious goal is to provide a *framework*, where a wide range of real-time Domain-Specific Visual Languages (DSVLs), as well as their dynamic real-time behavior, can be specified with a rigorous semantics. This is precisely the goal of two frameworks and associated tools: (i) the *e-Motions* framework [100]; and (ii) *MOMENT2*'s support for real-time DSVLs [21].

- In *e-Motions*, DSVLs are specified by their corresponding metamodels, and dynamic behavior is specified by rules that define in-place model transformations. But the goals of *e-Motions* do not remain at the syntax/visual level: they also include giving a precise rewriting logic semantics in Real-Time Maude to the different real-time DSVLs that can be defined in *e-Motions*, and to automatically support simulation and formal analysis of models by using the underlying Real-Time Maude engine. The formal semantics translates the metamodel of a DSVL as an object class, the corresponding models as object configurations of that class, and the *e-Motions* rules as rewrite rules. Since all these translations are automatic and define a DSVL's formal semantics, a modeling language designer using *e-Motions* does not have to explicitly define the DSVL's formal semantics: it comes for free, together with the simulation and model checking features, once the DSVL's metamodel and the dynamic behavior rules are specified.
- In [21], the *MOMENT2* framework has been extended to support the formal specification and analysis of real-time model-based systems. This is achieved by means of a collection of built-in timed constructs for defining the timed behavior of such systems. Timed behavior is specified using in-place model transformations. Furthermore, the formal semantics of a *timed behavioral specification* in *MOMENT2* is given by a corresponding real-time rewrite theory. In this way, models can be simulated and model checked using MOMENT2's Maude-based analysis tools. In addition, by using in-place multi-domain model transformations in *MOMENT2*, an existing model-based system can be extended with timed features in a non-intrusive way, in the sense that no modification is needed for the class diagram.

7 Hardware Description Language Semantics

The rewriting logic semantics project has been naturally extended from the level of programming languages to that of *hardware description languages* (HDLs). In this way, hardware designs written in an HDL can be both simulated and analyzed using the executable rewriting semantics of the HDL and tools like ELAN, CafeOBJ, or Maude. The first HDL to be given a rewriting logic semantics in Maude was ABEL [58]; this semantics was used not only for hardware designs, but also for hardware/software co-designs. An important new development has been the use of the rewriting logic semantics of an HDL for *generating sophisticated test inputs for hardware designs.* The point is that random testing can catch a good number of design errors, but uncovering deeper errors after random testing is hard and costly and requires a good understanding of the design to exercise complex computation sequences. The key insight, due to Michael Katelman, is that the rewriting semantics can be used *symbolically* to generate desired test inputs, not on a device's concrete states, but on states that are partly symbolic (contain logical variables) and partly concrete. This symbolic approach, first outlined in [60] and more fully developed in [59], has a number of unique features including: (i) the use of SAT solvers to symbolically solve Boolean constraints; (ii) support for user-guided random generation of partial instantiations; and (iii) a flexible *strategy language*, in which a hardware designer can specify in a declarative, high-level way the kind of test that needs to be generated. The effectiveness of this approach for generating sophisticated tests on real hardware designs has already been demonstrated for medium-sized Verilog designs [59]. The `vlogsl` tool is currently undergoing further enhancements to efficiently handle large designs.

But the value of the rewriting semantics of an HDL is not restricted to testing. For example, the recent Maude-based rewriting logic semantics of Verilog in [68] is arguably the most complete formal semantics to date, both in the sense of covering the largest subset of the language and in its faithful modeling of nondeteministic features. Besides being executable and supporting formal analysis, this semantics has uncovered several nontrivial bugs in various mature Verilog tools, and can serve as a practical and rigorous standard to ascertain what the correct behavior of such tools should be in complex cases.

A more exotic application of rewriting logic semantics, for which it is ideally suited due to its intrinsically concurrent nature, is that of *asynchronous hardware designs.* These are digital designs which do not have a global clock, so that different gates in a device can fire at different times. Such devices can behave correctly in much harsher environments (e.g., a satellite in outer space) and with much wider ranges of physical operating conditions than clocked devices. Asynchronous designs can be specified with the notation of *production rules*, which roughly speaking describe how each gate behaves when inputs to its wires are available. In [57] a rewriting logic semantics of asynchronous digital devices specified as sets of production rules is given and is realized in Maude. This is the first executable formal semantics of such devices we are aware of. It can be used both for simulation purposes and for model checking verification of small-sized devices (about 100 gates). An interesting challenge is how to scale up model checking for larger devices; this is nontrivial due to the large combinatorial explosion caused by their asynchronous behavior.

8 Abstract vs. Concrete Semantics and Static Analysis

In addition to helping with understanding and experimenting with language designs, a rewriting logic semantics can have several direct uses without having to change the semantics at all. Two such uses of unchanged semantics in the context of program verification are discussed in Sections 9 and 10. Nevertheless, there are program analysis needs where the desired information is not necessarily available in the code itself, or where the desired domain of analysis is not included in, and cannot be obtained from, the concrete domain in which the language semantics operates. In such cases, one can modify the concrete language semantics to operate within a target *abstract domain*. We next first show an overly simplified example, where the concrete semantics of IMP and IMP++ in Sections 3.1 and 3.2 are abstracted into type systems for the defined languages, which yield type checkers when executed. Then we discuss uses of similar but larger scale and more practical abstractions of rewrite logic semantics.

8.1 $\mathbb{K}$ Definition of a Type System for IMP++

The $\mathbb{K}$ semantics of IMP/IMP++ in Sections 3.1 and 3.2 can be used to execute even ill-typed IMP/IMP++ programs, which may be considered undesirable by some language designers. In this section we show how to define a type system for IMP/IMP++ using the very same $\mathbb{K}$ framework. The type system is defined like an (executable) semantics of the language, but one in the more abstract domain of types rather than in the concrete domain of integer and Boolean values.

The typing policy that we want to enforce on IMP/IMP++ programs is easy: all variables in a program have by default integer type and must be declared, arithmetic/Boolean operations are applied only on expressions of corresponding types, etc. Since programs and program fragments are now going to be rewritten into their types, we need to add to computations some basic types. Also, in addition to the computation to be typed, configurations must also hold the declared variables. Thus, we define the following (the "…" in the definition of K includes all the default syntax of computations, such as the original language syntax, $\curvearrowright$, freezers, etc.):

$$\begin{aligned} K &::= \ldots \mid \mathit{int} \mid \mathit{bool} \mid \mathit{stmt} \mid \mathit{pgm} \\ \mathit{Configuration}^{\mathit{Type}}_{\mathrm{IMP++}} &\equiv \langle \langle K \rangle_{\mathsf{k}}\ \langle \mathbf{List}\{\mathit{Id}\} \rangle_{\mathsf{vars}} \rangle_{\top} \end{aligned}$$

Figure 3 shows the IMP/IMP++ type system as a $\mathbb{K}$ system over such configurations. Constants reduce to their types, and types are straightforwardly propagated through each language construct. Note that almost each language construct is strict now, because we want to type all its arguments in almost all cases in order to apply the typing policy of the construct. Two constructs are

Original language syntax	K Strictness	K Semantics
$\mathit{AExp} ::= \mathit{Int}$		$i \rightarrow \mathit{int}$
$\mid \mathit{Id}$		$\langle \underset{\mathit{int}}{\underline{x}} \cdots\rangle_{\mathsf{k}}\ \langle\cdots x \cdots\rangle_{\mathsf{var}}$
$\mid \mathit{AExp}$ `+` AExp	$[\mathit{strict}]$	int `+` $\mathit{int} \rightarrow \mathit{int}$
$\mid \mathit{AExp}$ `/` AExp	$[\mathit{strict}]$	int `/` $\mathit{int} \rightarrow \mathit{int}$
$\mid$ `++` Id		$\langle \underset{\mathit{int}}{\underline{\texttt{++}\,x}} \cdots\rangle_{\mathsf{k}}\ \langle\cdots x \cdots\rangle_{\mathsf{var}}$
$\mathit{BExp} ::= \mathit{AExp}$ `<=` AExp	$[\mathit{strict}]$	int `<=` $\mathit{int} \rightarrow \mathit{bool}$
$\mid$ `not` BExp	$[\mathit{strict}]$	`not` $\mathit{bool} \rightarrow \mathit{bool}$
$\mid \mathit{BExp}$ `and` BExp	$[\mathit{strict}]$	bool `and` $\mathit{bool} \rightarrow \mathit{bool}$
$\mathit{Stmt} ::=$ `skip`		`skip` $\rightarrow \mathit{stmt}$
$\mid \mathit{Id}$ `:=` AExp	$[\mathit{strict}(2)]$	$\langle \underset{\mathit{stmt}}{\underline{x \texttt{ := } \mathit{int}}} \cdots\rangle_{\mathsf{k}}\ \langle\cdots x \cdots\rangle_{\mathsf{var}}$
$\mid \mathit{Stmt}\,;\,\mathit{Stmt}$	$[\mathit{strict}]$	$\mathit{stmt}\,;\,\mathit{stmt} \rightarrow \mathit{stmt}$
$\mid$ `if` BExp		
`then` Stmt `else` Stmt	$[\mathit{strict}]$	`if` bool `then` stmt `else` $\mathit{stmt} \rightarrow \mathit{stmt}$
$\mid$ `while` BExp `do` Stmt	$[\mathit{strict}]$	`while` bool `do` $\mathit{stmt} \rightarrow \mathit{stmt}$
$\mid$ `print` AExp	$[\mathit{strict}]$	`print` $\mathit{int} \rightarrow \mathit{stmt}$
$\mid$ `halt`		`halt` $\rightarrow \mathit{stmt}$
$\mid$ `spawn` Stmt	$[\mathit{strict}]$	`spawn` $\mathit{stmt} \rightarrow \mathit{stmt}$
$\mathit{Pgm} ::=$ `var` $\mathbf{List}\{\mathit{Id}\}$; Stmt		$\langle \underset{s \curvearrowright \mathit{pgm}}{\underline{\texttt{var}\ \mathit{xl}\ ;\ s}}\rangle_{\mathsf{k}}\ \langle \underset{\mathit{xl}}{\underline{\cdot}} \rangle_{\mathsf{vars}}$
		$\mathit{stmt} \curvearrowright \mathit{pgm} \rightarrow \mathit{pgm}$

Fig. 3. K type system for IMP++ (and IMP)

exceptional, namely, increment and assignment. The typing policy of these constructs is that they take precisely a variable and not something that types to an integer. If we defined, e.g., the assignment strict and with rule int `:=` $\mathit{int} \rightarrow \mathit{stmt}$, then our type system would allow ill-formed programs like `x+y := 0`. Note how we defined the typing policy of programs `var` xl ; s: the declared variables xl are stored into the $\langle\ldots\rangle_{\mathsf{vars}}$ cell (which is expected to initially be empty) and the statement is scheduled for typing (using a structural rule), placing a "reminder" in the computation that the *pgm* type is expected; once/if the statement is correctly typed, the type *pgm* is generated.

8.2 Examples of Abstract Rewriting Logic Semantics

We briefly discuss three practical uses of abstract rewriting logic semantics.

C Pluggable Policies. Many programs make implicit assumptions about data. Common examples include assumptions about whether variables have been initialized or can only contain non-null references. Domain-specific examples are also common; a compelling example is units of measurement, used in many scientific computing applications, where different variables and values are assumed to have specific units at specific times/along specific execution paths. These implicit assumptions give rise to implicit domain *policies*, such as requiring assignments to non-null pointers to also be non-null, or requiring two operands in an addition operation to have compatible units of measurement.

Mark Hills et al. [53] propose a framework for *pluggable policies* for C which allows these implicit policies to be made explicit and checked. The core of the framework is a shared annotation engine and parser, allowing annotations in multiple policies to be inserted by developers as comments in C programs, and a shared abstract rewriting logic semantics of C designed as a number of reusable modules that allow for new policies to be quickly developed and plugged in. For instance, a case study for checking non-null references was developed in under two days; another case study for checking units of measurement reuses the shared abstract semantics and only adds domain knowledge [53].

Polymorphic Type Inference. The technique in Section 8.1 for defining type systems using $\mathbb{K}$ is very general and has been used to define more complex type systems, such as higher-order polymorphic ones by Ellison et al. [38]. The $\mathbb{K}$ definition of the type system in [38] is more declarative and thus cleaner and easier to understand than alternative algorithmic definitions. Moreover, the $\mathbb{K}$ definition is formal, so it is amenable for formal reasoning. Interestingly, as shown in [38], the resulting $\mathbb{K}$ definition, when compiled to and executed using Maude, was faster than algorithmic implementations of the same type system found on the internet as teaching material. In fact, experiments in [38] show that it was comparable to state of the art implementations of type inferencers in conventional functional languages! For example, it was only about twice as slow on average than that of OCaml, and had average times comparable, or even better than those of Haskell ghci and SML/NJ.

Security Policy Checking. An elegant application of a programming language's *abstract* rewriting logic semantics to Java code security is presented by Alba-Castro et al. in [2, 3] as part of their rewriting-logic-semantics-based approach to proof carrying code. The key idea is to use an abstract rewriting logic semantics of Java that correctly approximates security properties such as *noninterference* (that is, the specification of what objects should not have any effects on other objects according to a stated security policy [48]), and *erasure* (a security policy that mandates that secret data should be removed after its intended use). Since the abstract rewriting semantics is finite-state, it supports the automatic creation of certificates for noninterference and erasure properties of Java programs that are independently checkable and small enough to be practical.

9 Model Checking Verification

Once a programming language or system is defined as a rewrite theory, one can use any general-purpose tools and techniques for rewriting logic to obtain tools and techniques specialized for the defined programming language or system. We have reported in the past on the use of Maude's general purpose LTL model checking capabilities to obtain model checkers specialized for various concurrent programming languages, including Java and the JVM (see, e.g., [76, 40, 42]).

In this paper we report on some new model checking experiments performed in the context of the C definition discussed in Section 4. We thank Chucky Ellison for extending his C semantics with concurrency primitives and for conducting these experiments. A more detailed presentation of these can be found in [37].

The C semantics in Section 4 can be extended to include semantics for concurrency primitives like "spawn", "sync", "lock", and "unlock". The former is used to dynamically spawn a new execution thread, "sync" waits for all of the other threads to die before continuing, and "lock" and "unlock" synchronize threads on memory locations (similar to Java locking on references). When formalizing the semantics of C, we did not plan to introduce concurrency. Despite that, as hoped for, the existing rules were left unchanged upon adding configuration support and the semantics of threads.

Dekker's Algorithm. We now take a look at the classical Dekker's algorithm, in order to explore thread interleavings.

```
void dekker1(void) {
  flag1 = 1;  turn = 2;
  while((flag2 == 1) && (turn == 2)) ;
  critical1();
  flag1 = 0;
}
```

```
void dekker2(void) {
  flag2 = 1;  turn = 1;
  while((flag1 == 1) && (turn == 1)) ;
  critical2();
  flag2 = 0;
}
```

These two functions get called by the two threads respectively to ensure mutual exclusion of the calls to `criticaln()`. In the program we used for testing, these threads each contain infinite loops while the function `main()` waits on a `sync()`. Thus, the program never terminates.

To test the mutual exclusion property, we model check the following LTL formula: $\Box\neg(\text{enabled}(\mathit{critical1})\wedge\text{enabled}(\mathit{critical2}))$, stating that the two critical sections can never be called at the same time. Applying this formula to our program yields "`result Bool: true`", in 400ms. If we break the algorithm by changing a `while` to an `if`, the tool instead returns a list of rules, together with the resulting states, that represent a counterexample.

Dining Philosophers. Another classic example is the dining philosophers problem.

```
void philosopher( int n ) {
  while(1) {
    // Hungry: obtain chopsticks
    if ( n % 2 == 0 ) {  // Even number: Left, then right
      lock(&chopstick[(n+1) % NUM_PHILOSOPHERS]);
      lock(&chopstick[n]);
    } else {  // Odd number: Right, then left
      lock(&chopstick[n]);
      lock(&chopstick[(n+1) % NUM_PHILOSOPHERS]);
    }
    // Eating
    // Finished Eating: release chopsticks
    unlock(&chopstick[n]);
    unlock(&chopstick[(n+1) % NUM_PHILOSOPHERS]);
    // Thinking
  }
}
```

The above code shows a solution to the dining philosophers that has even-numbered philosophers picking up their left chopstick first, while odd-numbered philosophers pick up their right chopstick first. This strategy ensures that there is no deadlock. We can use Maude's search command to verify that there is no deadlock simply by searching for final states. Here are the results:

	No Deadlock		With Deadlock	
n	number of states	time (s)	number of states	time (s)
1	19	0.1	–	–
2	92	0.8	63	0.6
3	987	14.0	490	7.2
4	14610	293.5	5690	119.8
5	288511	8360.3	84369	2376.5

In the "No Deadlock" column we see the results for the code above. We were able to verify that with this algorithm, there were no deadlocks for up to five philosophers. In the "With Deadlock" column, we altered the code so that all philosophers would try to pick up their left chopstick first. For this algorithm, we were able to find counterexamples showing that the program has deadlocks.

While the classic programs above are toy examples, which are far from the complexity of real-life software, we believe that they are sufficient to show that a programming language semantics can be more than a "useless academic intellectual exercise". The well-known state-space explosion of model checking cannot be avoided, no matter whether one uses a formal semantics of the language or not, but one should note that this is a problem of model checking and not of using a formal semantics for model checking. Also, there are well-known techniques to address the state explosion problem, like partial-order reduction, which can and have also been applied in the context of rewriting logic semantics [41]. And one can use an *abstract semantics* (Section 8) as the basis of the model checker to make it more scalable. The next section shows another use of rewriting logic semantics of programming languages, for deductive program verification.

10 Deductive Verification and Matching Logic

As discussed above, one of the major advantages of giving a rewriting logic semantics to a language is that one can use it not only to obtain a reference implementation of the language, but also to formally analyze programs in the defined language using general-purpose tools developed for rewriting logic, such as Maude's model checker. Moreover, the original rewriting logic semantics of the language is used unchanged for model checking or other similar analyses, which is not only immensely convenient but also offers a high confidence in the results of the analysis (because it excludes the problem of implementing a wrong language semantics in the analyzer). One question, however, still remains unanswered: can we use the language semantics, also unchanged, in a program logic fashion, that is, for deductive verification of programs?

Early work in this direction includes two Hoare logic provers that use directly the rewriting logic semantics of a Pascal like-language and of a fragment of Java and the Maude ITP [34,107]. Furthermore, the rewriting logic semantics of Java was used in [1] to automatically validate the inference rules of a Java verification tool. In the remainder of this section we report on an alternative approach.

Matching logic [102, 103] is a new program verification logic, which builds upon rewriting logic semantics. Matching logic specifications are constrained symbolic program configurations, called *patterns*, which can be *matched* by concrete configurations. By building upon an executable semantics of the language and allowing specifications to directly refer to the structure of the configuration, matching logic has at least three benefits: (1) one's familiarity with the formalism reduces to one's familiarity with the formal semantics of the language, that is, with the language itself; (2) the verification process proceeds the same way as the program execution, making debugging failed proof attempts manageable because one can always see the "current configuration" and "what went wrong", almost like in a debugger; and (3) nothing is lost in translation, that is, there is no gap between the language definition and its verifier. Moreover, direct access to the structure of the configuration facilitates defining sub-patterns that one may reason about, such as disjoint lists or trees in the heap, as well as supporting framing in various components of the configuration at no additional cost.

To use matching logic for program verification, one must know the structure of the configurations that are used in the executable language semantics. For example, the configuration of some language may contain, besides the code itself, an environment, a heap, stacks, synchronization resources, etc. The configuration of C (see Section 4 and [37]), for example, consists of 75 cells, each containing either other cells or some piece of semantic information. Matching logic specifications, or patterns, allow one to refer directly to the configuration of the program. Moreover, we can use logical variables and thus combine the desired configuration structure with first-order constraints. For example, the pattern

$$\begin{array}{l}\langle\quad \langle \beta,\ I\rangle_{\text{in}}\quad \langle \mathsf{x} \mapsto x,\ \mathsf{i} \mapsto i,\ \mathsf{n} \mapsto n,\ E\rangle_{\text{env}}\quad \langle \mathsf{list}(x,\alpha),\ H\rangle_{\text{heap}}\quad \mathsf{C}\quad \rangle_{\text{config}}\\ \quad \wedge\ \ i \leq n \quad \wedge \quad |\beta| = n - i \quad \wedge \quad A = \mathit{rev}(\alpha)@\beta\end{array}$$

specifies the set of configurations where program variables x, i and n are bound in the environment to some values x, i, and respectively n, such that $i \leq n$, the input buffer contains a sequence β of size $n - i$, and the heap contains a linked list starting with pointer x comprising the sequence of elements α such that the sequence A is the reverse of the sequence α concatenated with β. Here A is a free variable of type sequence of elements. The other variables play the role of cell frames: I is a variable matching the rest of the input cell, E matches the rest of the environment, H the rest of the heap, and C the rest of the configuration. Note that nothing special needs to be done for framing in matching logic (that is, framing is a special case of the more general principle of matching).

A major benefit of matching logic is that is can be used to turn an executable semantics into a program logic without any change to the original semantics. The idea is that the executable semantics can be regarded as a set of rewrite rules between matching logic patterns, and one can use first-order reasoning over patterns to turn the pattern resulting from the application of some rule into a pattern that the next rule expects to match. This way, one can derive rewrite

```
#include <stdlib.h>
#include <stdio.h>

struct listNode { int val; struct listNode *next; };

void readWriteBuffer(int n)
/*@ rule <k> $ => return; </k>  <in> A => epsilon <_/in>  <out_> epsilon => rev(A) </out>
    if n = len(A) */
{
  int i;  struct listNode *x;
  i = 0;  x = 0;
  /*@ inv <in> ?B <_/in> <heap_> list(x)(?A) <_/heap>
          /\ i <= n /\ len(?B) = n - i /\ A = rev(?A) @ ?B */
  while (i < n) {
    struct listNode *y;
    y = x;
    x = (struct listNode*) malloc(sizeof(struct listNode));
    scanf("%d", &(x->val));
    x->next = y;
    i += 1;
  }

  //@ inv <out_> ?A </out> <heap_> list(x)(?B) <_/heap> /\ A = rev(?A @ ?B)
  while (x) {
    struct listNode *y;
    y = x->next;
    printf("%d ",x->val);
    free(x);
    x = y;
  }
}

void main() {
  int n;
  //@ assume <in> [5, 1, 2, 3, 4, 5] </in>  <out> epsilon </out>
  scanf("%d", &n);
  readWriteBuffer(n);
  //@ assert <in> epsilon </in> <out> [5, 4, 3, 2, 1] </out>
}
```

Fig. 4. C program making use of the I/O and the heap, verified using MatchC

rules from other rewrite rules, using matching logic reasoning as a mechanism to rearrange configurations so that rewrite rules can match and apply.

With the help of Andrei Ştefănescu, we implemented a proof-of-concept matching logic verifier for a fragment of C, called MatchC, which can be downloaded and executed online at `http://fsl.cs.uiuc.edu/ml`. MatchC builds upon an executable rewrite-based semantics of this fragment of C, extending it (unchanged) with semantics for pattern specifications. Both the executable semantics and the verifier are implemented using the $\mathbb{K}$ framework (see Section 3).

Figure 4 shows a C program verified using MathC. The `main()` function reads `n` from the standard input and then calls `readWriteBuffer(n)`. Then `readWriteBuffer(n)` reads from the standard input `n` elements and allocates a linked list putting each element at the top of the list, followed by traversing the linked list and printing each element while deallocating the list nodes. This way, we end up with the reversed sequence of elements printed to the the standard output and with the heap unchanged. There are four types of annotations in this program: (1) *assumptions*, which allow one to assume a certain pattern for the remaining program; (2) *assertions*, which generate matching logic proof obligations, namely, that the current pattern implies the asserted pattern; (3) *rules*, which give the claimed 𝕂 semantics of the subsequent piece of code; and (4) *invariants*, which are patterns that should hold at each loop iteration.

Some explanations regarding MatchC's notation are necessary. MatchC annotations are introduced like C comments starting with @, so they are ignored by C compilers. We use an XML-like notation to specify when cells start and when they end. We use the usual rewriting relation "`=>`" for the in-place rewriting within 𝕂 rules. The "`$`" symbol that appears in the computation cell of a rule stands for the subsequent statement (the function body, in our case here). Fourth, to avoid writing quantifiers, variables starting with a question mark are existentially quantified over the pattern. Fifth, we use an underscore in the XML tag to state that the corresponding cell is open in that direction, which can be regarded as an abbreviation for using a fresh variable; for example, "`<in> ?B <_/in>`" in the invariant of the first loop abbreviates "`<in> ?B, ?E </in>`". Finally, to avoid writing the environment cell all the time, MatchC allows users to refer directly to program variables in patterns; this avoids having to add a binding of the program variable to a logical variable in the environment cell and then using the logical variable throughout the pattern.

The rule giving the semantics of `readWriteBuffer(n)` states that this function returns nothing ("`$ => return;`", that is, its body behaves as if it returns) and takes a sequence A of length `n` (see the condition "`n = len(A)`") from the beginning of the input cell ("`<in> A => epsilon <_/in>`") and places it reversed at the end of the output cell ("`<out_> epsilon => A </out>`"). Since we have a rewrite-based semantics, the fact that no other cells are mentioned implicitly means that *nothing else is modified by this function*, including the heap. The invariant of the first loop is exactly the pattern that we discussed at the beginning of this section. The invariant of the second loop is similar, but dual. We do not show the axiom (matching logic formula) governing the list pattern in the heap cell; the interested reader can check [102,103]. Nevertheless, since `x` is null at the end of the second loop, it follows that the list it points to is empty, so the heap changes by the first loop will be cleaned by the end of the second.

MatchC verifies the program in Figure 4 in about 100 milliseconds:

```
Compiling program ... DONE! [0.311s]
Loading Maude ....... DONE! [0.209s]
Verifying program ... DONE! [0.099s]
Verification succeeded! [82348 rewrites, 4 feasible and 2 infeasible paths]
Output: 5 4 3 2 1
```

We encourage the reader to run MatchC online at `http://fsl.cs.uiuc.edu/ml`.

11 Conclusions and Future Work

We have given a progress report on the rewriting logic semantics project. Our main goal has been to show how research in this area is closing the gap between theory and practice by supporting executable semantic definitions that scale up to real languages at the three levels of software modeling languages, programming languages, and HDLs, and with features such as concurrencyy and real-time semantics. We have also shown how such semantic definitions can be *directly* used as a basis for interpreters and for sophisticated program analysis tools, including static analyzers, model checkers, and program proving tools.

Although reasonably efficient *interpreters* can be currently generated from rewriting logic specifications, one important future challenge is the automatic generation from language definitions of high-performance language implementations that are correct by construction. Another area that should be further developed is that of *meta-reasoning* methods, to prove formal properties not about programs, but about entire language definitions. A third promising future research direction is exploring the systematic interplay between abstract semantics and model checking, as well as the systematic application of state space reduction techniques in the model checking of programs from their rewriting logic language definitions; the overall goal is achieving a high degree of *scalability* in model checking analyses, with a wide spectrum of analysis choices ranging from model checking of programs according to their concrete semantics to various forms of static analysis based on different kinds of abstract semantics.

Acknowledgments. We thank the organizers of FCT 2011 for giving us the opportunity of presenting these ideas, and for their helpful suggestions for improving the exposition. We also thank all the researchers involved in the rewriting logic semantics project for their many contributions, which we have tried to summarize in this paper without any claims of completeness. This research has been partially supported by NSF Grants CNS 08-34709, CCF 09-05584, and CCF-0916893, by NSA contract H98230-10-C-0294, by (Romanian) SMIS-CSNR 602-12516 contract no. 161/15.06.2010, by the Boeing Grant C8088, and by the Samsung SAIT grant 2010-02664.

References

1. Ahrendt, W., Roth, A., Sasse, R.: Automatic Validation of Transformation Rules for Java Verification Against a Rewriting Semantics. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 412–426. Springer, Heidelberg (2005)
2. Alba-Castro, M., Alpuente, M., Escobar, S.: Abstract certification of global non-interference in rewriting logic. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) FMCO 2009. LNCS, vol. 6286, pp. 105–124. Springer, Heidelberg (2010)
3. Alba-Castro, M., Alpuente, M., Escobar, S.: Approximating non-interference and erasure in rewriting logic. In: Proc. SYNASC, pp. 124–132. IEEE, Los Alamitos (2010)
4. AlTurki, M., Dhurjati, D., Yu, D., Chander, A., Inamura, H.: Formal specification and analysis of timing properties in software systems. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 262–277. Springer, Heidelberg (2009)
5. AlTurki, M., Meseguer, J.: Real-time rewriting semantics of Orc. In: Proc. PPDP, Poland, pp. 131–142. ACM Press, New York (2007)
6. AlTurki, M., Meseguer, J.: Reduction semantics and formal analysis of Orc programs. In: Proc. Workshop on Automated Specification and Verification of Web

Systems (WWV 2007). ENTCS, vol. 200(3), pp. 25–41. Elsevier, Amsterdam (2008)

7. AlTurki, M., Meseguer, J.: Dist-Orc: A rewriting-based distributed implementation of Orc with formal analysis. In: Proc. RTRTS 2010. Electronic Proceedings in Theoretical Computer Science, vol. 36, pp. 26–45. CoRR (2010)
8. Aoumeur, N.: Integrating and rapid-prototyping UML structural and behavioural diagrams using rewriting logic. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 296–310. Springer, Heidelberg (2002)
9. Bae, K., Ölveczky, P.C.: Extending the Real-Time Maude semantics of Ptolemy to hierarchical DE models. In: Proc. RTRTS 2010. Electronic Proceedings in Theoretical Computer Science, vol. 36, pp. 46–66. CoRR (2010)
10. Bae, K., Ölveczky, P.C., Al-Nayeem, A., Meseguer, J.: Synchronous AADL and its formal analysis in Real-Time Maude. Technical report, University of Illinois at Urbana-Champaign (2005), `http://hdl.handle.net/2142/25091`
11. Bae, K., Ölveczky, P.C., Feng, T.H., Tripakis, S.: Verifying Ptolemy II Discrete-Event Models Using Real-Time Maude. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 717–736. Springer, Heidelberg (2009)
12. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Science 96(1), 217–248 (1992)
13. Bjørk, J., Johnsen, E.B., Owe, O., Schlatte, R.: Lightweight time modeling in timed Creol. In: Proc. RTRTS 2010. Electronic Proceedings in Theoretical Computer Science, vol. 36, pp. 67–81. CoRR (2010)
14. Blazy, S., Leroy, X.: Mechanized semantics for the Clight subset of the C language. Journal of Automated Reasoning 43(3), 263–288 (2009)
15. Boronat, A.: MOMENT: A Formal Framework for MOdel ManageMENT. PhD thesis, Universitat Politècnica de València, Spain (2007)
16. Boronat, A., Carsí, J.A., Ramos, I.: Automatic reengineering in MDA using rewriting logic as transformation engine. In: Proc. CSMR 2005, pp. 228–231. IEEE, Los Alamitos (2005)
17. Boronat, A., Heckel, R., Meseguer, J.: Rewriting logic semantics and verification of model transformations. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 18–33. Springer, Heidelberg (2009)
18. Boronat, A., Meseguer, J.: Algebraic semantics of OCL-constrained metamodel specifications. In: Oriol, M., Meyer, B. (eds.) TOOLS EUROPE 2009. LNBIP, vol. 33, pp. 96–115. Springer, Heidelberg (2009)
19. Boronat, A., Meseguer, J.: MOMENT2: EMF model transformations in Maude. In: Vallecillo, A., Sagardui, G. (eds.) Actas de las XIV Jornadas de Ingeniería del Software y Bases de Datos, JISBD 2009, San Sebastián, España, September 8-11, pp. 178–179 (2009)
20. Boronat, A., Meseguer, J.: An algebraic semantics for MOF. Formal Aspects of Computing 22(3-4), 269–296 (2010)
21. Boronat, A., Ölveczky, P.C.: Formal real-time model transformations in MOMENT2. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 29–43. Springer, Heidelberg (2010)
22. Borras, P., Clément, D., Despeyroux, T., Incerpi, J., Kahn, G., Lang, B., Pascual, V.: CENTAUR: The system. In: Software Development Environments (SDE), pp. 14–24 (1988)
23. Braga, C.: Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brazil (2001)

24. Braga, C., Haeusler, E.H., Meseguer, J., Mosses, P.D.: Mapping modular SOS to rewriting logic. In: Leuschel, M. (ed.) LOPSTR 2002. LNCS, vol. 2664, pp. 262–277. Springer, Heidelberg (2003)
25. Braga, C., Meseguer, J.: Modular rewriting semantics in practice. In: Proc. WRLA 2004. ENTCS, vol. 117, pp. 393–416. Elsevier, Amsterdam (2004)
26. Broy, M., Wirsing, M., Pepper, P.: On the algebraic definition of programming languages. ACM TOPLAS 9(1), 54–99 (1987)
27. Bruni, R., Meseguer, J.: Semantic foundations for generalized rewrite theories. Theor. Comput. Sci. 360(1-3), 386–414 (2006)
28. Chalub, F.: An implementation of Modular SOS in Maude. Master's thesis, Universidade Federal Fluminense, Niterói, RJ, Brazil (May 2005)
29. Chalub, F., Braga, C.: Maude MSOS tool. Universidade Federal Fluminense, `www.ic.uff.br/~frosario/2o-workshop-vas-novembro-2004.pdf`
30. Chalub, F., Braga, C.: A Modular Rewriting Semantics for CML. Journal of Universal Computer Science 10(7), 789–807 (2004)
31. Chen, F., Roşu, G., Venkatesan, R.P.: Rule-based analysis of dimensional safety. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 197–207. Springer, Heidelberg (2003)
32. Clavel, M., Durán, F., Eker, S., Meseguer, J., Lincoln, P., Martí-Oliet, N., Talcott, C.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
33. Clavel, M., Egea, M.: ITP/OCL: A rewriting-based validation tool for UML+OCL static class diagrams. In: Johnson, M., Vene, V. (eds.) AMAST 2006. LNCS, vol. 4019, pp. 368–373. Springer, Heidelberg (2006)
34. Clavel, M., Santa-Cruz, J.: ASIP + ITP: A verification tool based on algebraic semantics. In: López-Fraguas, F.J. (ed.) Actas de las V Jornadas sobre Programación y Lenguajes, PROLE 2005, Granada, España, Septiembre 14-16, pp. 149–158. Thomson (2005)
35. Clément, D., Despeyroux, J., Hascoet, L., Kahn, G.: Natural semantics on the computer. In: Fuchi, K., Nivat, M. (eds.) Proceedings, France-Japan AI and CS Symposium, ICOT 1986, pp. 49–89 (1986); Also, Information Processing Society of Japan, Technical Memorandum PL-86-6
36. d'Amorim, M., Roşu, G.: An Equational Specification for the Scheme Language. Journal of Universal Computer Science 11(7), 1327–1348 (2005); Also Technical Report No. UIUCDCS-R-2005-2567 (April 2005)
37. Ellison, C., Roşu, G.: A formal semantics of C with applications. Technical Report University of Illinois (November 2010), `http://hdl.handle.net/2142/17414`
38. Ellison, C., Şerbănuţă, T.F., Roşu, G.: A rewriting logic approach to type inference. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 135–151. Springer, Heidelberg (2009)
39. Farzan, A.: Static and dynamic formal analysis of concurrent systems and languages: a semantics-based approach. PhD thesis, University of Illinois at Urbana-Champaign (2007)
40. Farzan, A., Chen, F., Bevilacqua, V., Roşu, G.: Formal Analysis of Java Programs in JavaFAN. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 501–505. Springer, Heidelberg (2004)
41. Farzan, A., Meseguer, J.: Partial order reduction for rewriting semantics of programming languages. In: Proc. WRLA 2006. ENTCS, vol. 176(4), pp. 61–78 (2007)

42. Farzan, A., Bevilacqua, V., Roşu, G.: Formal JVM code analysis in javaFAN. In: Rattray, C., Maharaj, S., Shankland, C. (eds.) AMAST 2004. LNCS, vol. 3116, pp. 132–147. Springer, Heidelberg (2004)
43. Felleisen, M., Friedman, D.P.: Control operators, the SECD-machine, and the λ-calculus. In: 3rd Working Conference on the Formal Description of Programming Concepts, Denmark, pp. 193–219 (August 1986)
44. Fernández Alemán, J.L., Toval Álvarez, J.A.: Can intuition become rigorous? Foundations for UML model verification tools. In: Proc. ISSRE 2000, pp. 344–355. IEEE, Los Alamitos (2000)
45. Friedman, D.P., Wand, M., Haynes, C.T.: Essentials of Programming Languages, 2nd edn. MIT Press, Cambridge (2001)
46. Garrido, A., Meseguer, J., Johnson, R.: Algebraic semantics of the C preprocessor and correctness of its refactorings. Technical Report UIUCDCS-R-2006-2688, CS Dept., University of Illinois at Urbana-Champaign (February 2006)
47. Goguen, J., Malcolm, G.: Algebraic Semantics of Imperative Programs. MIT Press, Cambridge (1996)
48. Goguen, J., Meseguer, J.: Security policies and security models. In: Proceedings of the 1982 Symposium on Security and Privacy, pp. 11–20. IEEE, Los Alamitos (1982)
49. Goguen, J.A., Parsaye-Ghomi, K.: Algebraic denotational semantics using parameterized abstract modules. In: Díaz, J., Ramos, I. (eds.) Formalization of Programming Concepts. LNCS, vol. 107, pp. 292–309. Springer, Heidelberg (1981)
50. Gurevich, Y.: Evolving algebras 1993: Lipari Guide. In: Börger, E. (ed.) Specification and Validation Methods, pp. 9–37. Oxford University Press, Oxford (1994)
51. Gurevich, Y., Huggins, J.K.: The semantics of the C programming language. In: Martini, S., Börger, E., Kleine Büning, H., Jäger, G., Richter, M.M. (eds.) CSL 1992. LNCS, vol. 702, pp. 274–308. Springer, Heidelberg (1993)
52. Harper, R., Honsell, F., Plotkin, G.D.: A framework for defining logics. Journal of the ACM 40(1), 143–184 (1993)
53. Hills, M., Chen, F., Roşu, G.: Pluggable Policies for C. Technical Report UIUCDCS-R-2008-2931, University of Illinois at Urbana-Champaign (2008)
54. Hills, M., Şerbănuţă, T.F., Roşu, G.: A rewrite framework for language definitions and for generation of efficient interpreters. In: Proc. of WRLA 2006. ENTCS, vol. 176(4), pp. 215–231. Elsevier, Amsterdam (2007)
55. Johnsen, E.B., Owe, O., Axelsen, E.W.: A runtime environment for concurrent objects with asynchronous method calls. In: Proc. WRLA 2004. ENTCS, vol. 117, Elsevier, Amsterdam (2004)
56. Kahn, G.: Natural semantics. In: Brandenburg, F.J., Wirsing, M., Vidal-Naquet, G. (eds.) STACS 1987. LNCS, vol. 247, pp. 22–39. Springer, Heidelberg (1987)
57. Katelman, M., Keller, S., Meseguer, J.: Concurrent rewriting semantics and analysis of asynchronous digital circuits. In: Ölveczky, P.C. (ed.) WRLA 2010. LNCS, vol. 6381, pp. 140–156. Springer, Heidelberg (2010)
58. Katelman, M., Meseguer, J.: A rewriting semantics for ABEL with applications to hardware/software co-design and analysis. In: Proc. WRLA 2006. ENTCS, vol. 176(4), pp. 47–60. Elsevier, Amsterdam (2007)
59. Katelman, M., Meseguer, J.: vlogmt: A strategy language for simulation-based verification of hardware. In: Raz, O. (ed.) HVC 2010. LNCS, vol. 6504, pp. 129–145. Springer, Heidelberg (2010)
60. Katelman, M., Meseguer, J., Escobar, S.: Directed-logical testing for functional verification of microprocessors. In: MEMOCODE 2008, pp. 89–100. IEEE, Los Alamitos (2008)

61. Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: ACL2 Case Studies. Kluwer Academic Press, Dordrecht (2000)
62. Knapp, A.: Generating rewrite theories from UML collaborations. In: Futatsugi, K., Nakagawa, A.T., Tamai, T. (eds.) Cafe: An Industrial-Strength Algebraic Formal Method, pp. 97–120. Elsevier, Amsterdam (2000)
63. Knapp, A.: A Formal Approach to Object-Oriented Software Engineering. Shaker Verlag, Aachen, Germany, 2001. PhD thesis, Institut für Informatik, Universität München (2000)
64. Lee, E.A.: Modeling concurrent real-time processes using discrete events. Ann. Software Eng. 7, 25–45 (1999)
65. Liang, S., Hudak, P., Jones, M.: Monad transformers and modular interpreters. In: Proc. POPL 1995, pp. 333–343. ACM Press, New York (1995)
66. Martí-Oliet, N., Meseguer, J.: Rewriting logic as a logical and semantic framework. In: Gabbay, D.M., Guenthner, F. (eds.) Handbook of Philosophical Logic, 2nd edn., vol. 9, pp. 1–87. Kluwer, Dordrecht (2002)
67. Meredith, P., Hills, M., Roşu, G.: A K definition of Scheme. Technical Report UIUCDCS-R-2007-2907, Department of Computer Science, University of Illinois at Urbana-Champaign (2007)
68. Meredith, P., Katelman, M., Meseguer, J., Roşu, G.: A formal executable semantics of Verilog. In: Proc. MEMOCODE 2010, pp. 179–188. IEEE, Los Alamitos (2010)
69. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96(1), 73–155 (1992)
70. Meseguer, J.: Rewriting logic as a semantic framework for concurrency: a progress report. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 331–372. Springer, Heidelberg (1996)
71. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998)
72. Meseguer, J.: Software specification and verification in rewriting logic. In: Broy, M., Pizka, M. (eds.) Models, Algebras, and Logic of Engineering Software, NATO Advanced Study Institute, Marktoberdorf, Germany, July 30 – August 11, pp. 133–193. IOS Press, Amsterdam (2002)
73. Meseguer, J., Braga, C.: Modular rewriting semantics of programming languages. In: Rattray, C., Maharaj, S., Shankland, C. (eds.) AMAST 2004. LNCS, vol. 3116, pp. 364–378. Springer, Heidelberg (2004)
74. Meseguer, J., Futatsugi, K., Winkler, T.: Using rewriting logic to specify, program, integrate, and reuse open concurrent systems of cooperating agents. In: Proceedings of the 1992 International Symposium on New Models for Software Architecture, Tokyo, Japan, pp. 61–106. Research Institute of Software Engineering (November 1992)
75. Meseguer, J., Ölveczky, P.C.: Formalization and correctness of the PALS architectural pattern for distributed real-time systems. In: Dong, J.S., Zhu, H. (eds.) ICFEM 2010. LNCS, vol. 6447, pp. 303–320. Springer, Heidelberg (2010)
76. Meseguer, J., Roşu, G.: The rewriting logic semantics project. Theoretical Computer Science 373, 213–237 (2007)
77. Bevilacqua, V., Roşu, G.: Rewriting logic semantics: From language specifications to formal analysis tools. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 1–44. Springer, Heidelberg (2004)

78. Miller, D.: Representing and reasoning with operational semantics. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 4–20. Springer, Heidelberg (2006)
79. Misra, J.: Computation orchestration: A basis for wide-area computing. In: Broy, M. (ed.) Proc. of the NATO Advanced Study Institute, Engineering Theories of Software Intensive Systems Marktoberdorf, Germany. NATO ASI Series (2004)
80. Misra, J., Cook, W.R.: Computation orchestration. Software and System Modeling 6(1), 83–110 (2007)
81. Moggi, E.: An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Edinburgh University, Dept. of Computer Science (June 1989)
82. Mokhati, F., Badri, M.: Generating Maude specifications from UML use case diagrams. Journal of Object Technology 8(2), 319–136 (2009)
83. Mokhati, F., Gagnon, P., Badri, M.: Verifying UML diagrams with model checking: A rewriting logic based approach. In: Proc. QSIC 2007, pp. 356–362. IEEE, Los Alamitos (2007)
84. Mokhati, F., Sahraoui, B., Bouzaher, S., Kimour, M.T.: A tool for specifying and validating agents' interaction protocols: From Agent UML to Maude. Journal of Object Technology 9(3), 59–77 (2010)
85. Mosses, P.D.: Unified algebras and action semantics. In: Cori, R., Monien, B. (eds.) STACS 1989. LNCS, vol. 349, pp. 17–35. Springer, Heidelberg (1989)
86. Mosses, P.D.: Denotational semantics. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, ch. 11, North-Holland, Amsterdam (1990)
87. Mosses, P.D.: Modular structural operational semantics. J. Log. Algebr. Program. 60-61, 195–228 (2004)
88. Nadathur, G., Miller, D.: An overview of λProlog. In: Bowen, K., Kowalski, R. (eds.) Fifth Int. Joint Conf. and Symp. on Logic Programming, pp. 810–827. The MIT Press, Cambridge (1988)
89. Nakajima, S.: Using algebraic specification techniques in development of object-oriented frameworks. In: Woodcock, J.C.P., Davies, J. (eds.) FM 1999. LNCS, vol. 1709, pp. 1664–1683. Springer, Heidelberg (1999)
90. Nakajima, S., Futatsugi, K.: An object-oriented modeling method for algebraic specifications in CafeOBJ. In: Proc. ICSE 1997. ACM, New York (1997)
91. Norrish, M.: C formalised in HOL. Technical Report UCAM-CL-TR-453, University of Cambridge (December 1998)
92. Ölveczky, P.C., Boronat, A., Meseguer, J.: Formal Semantics and Analysis of Behavioral AADL Models in Real-Time Maude. In: Hatcliff, J., Zucca, E. (eds.) FMOODS 2010. LNCS, vol. 6117, pp. 47–62. Springer, Heidelberg (2010)
93. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theoretical Computer Science 285(2), 359–405 (2002)
94. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
95. Papaspyrou, N.S.: A Formal Semantics for the C Programming Language. PhD thesis, National Technical University of Athens (February 1998)
96. Papaspyrou, N.S.: Denotational semantics of ANSI C. Computer Standards and Interfaces 23(3), 169–185 (2001)
97. Pfenning, F., Elliott, C.: Higher-order abstract syntax. In: Proc. PLDI 1988, pp. 199–208. ACM Press, New York (1988)
98. Pierce, B.: Types and Programming Languages. MIT Press, Cambridge (2002)
99. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60-61, 17–139 (2004); Previously published as technical report DAIMI FN-19, Aarhus University (1981)

100. Rivera, J.E., Durán, F., Vallecillo, A.: On the behavioral semantics of real-time domain specific visual languages. In: Ölveczky, P.C. (ed.) WRLA 2010. LNCS, vol. 6381, pp. 174–190. Springer, Heidelberg (2010)
101. Roşu, G.: CS322, Fall, - Programming Language Design: Lecture Notes. Technical Report UIUCDCS-R-2003-2897, University of Illinois at Urbana-Champaign, Dept. of Computer Science, Notes of a course taught at UIUC (2003)
102. Roşu, G., Ştefănescu, A.: Matching logic: A new program verification approach (nier track). In: Proc. ICSE 2011(2011)
103. Roşu, G., Ellison, C., Schulte, W.: Matching logic: An alternative to hoare/Floyd logic. In: Johnson, M., Pavlovic, D. (eds.) AMAST 2010. LNCS, vol. 6486, pp. 142–162. Springer, Heidelberg (2011)
104. Roşu, G., Venkatesan, R.P., Whittle, J., Leuştean, L.: Certifying optimality of state estimation programs. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 301–314. Springer, Heidelberg (2003)
105. Roşu, G., Şerbănuţă, T.F.: An overview of the K semantic framework. Journal of Logic and Algebraic Programming 79(6), 397–434 (2010)
106. Sasse, R.: Taclets vs. rewriting logic – relating semantics of Java. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, Technical Report in Computing Science No. 2005-16 (May 2005)
107. Sasse, R., Meseguer, J.: Java+ITP: A verification tool based on hoare logic and algebraic semantics. In: Proc. WRLA 2006. ENTCS, vol. 176(4), pp. 29–46 (2007)
108. Schmidt, D.A.: Denotational Semantics – A Methodology for Language Development. Allyn and Bacon, Boston (1986)
109. Scott, D.: Outline of a mathematical theory of computation. In: Proceedings, Fourth Annual Princeton Conference on Information Sciences and Systems, pp. 169–176. Princeton University, Princeton (1970); Also appeared as Technical Monograph PRG 2, Oxford University, Programming Research Group
110. Scott, D., Strachey, C.: Toward a mathematical semantics for computer languages. In: Proc. Symp. on Computers and Automata. Microwave Research Institute Symposia Series, vol. 21, Polytechnical Institute of Brooklyn (1971)
111. T. F. Şerbănuţă. A Rewriting Approach to Concurrent Programming Language Design and Semantics. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign (2010)
112. Şerbănuţă, T.F., Roşu, G., Meseguer, J.: A rewriting logic approach to operational semantics. Information and Computation 207(2), 305–340 (2009)
113. Slonneger, K., Kurtz, B.L.: Formal Syntax and Semantics of Programming Languages. Addison-Wesley, Reading (1995)
114. Stärk, R.F., Schmid, J., Börger, E.: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer, Heidelberg (2001)
115. Stehr, M.-O., Talcott, C.: PLAN in Maude: Specifying an active network programming language. In: Proc. WRLA 2002. ENTCS, vol. 117, Elsevier, Amsterdam (2002)
116. Stehr, M.-O., Talcott, C.L.: Practical techniques for language design and prototyping. In: Dagstuhl Seminar 05081 on Foundations of Global Computing, February 20 – 25, Schloss Dagstuhl, Wadern (2005)
117. Thati, P., Sen, K., Martí-Oliet, N.: An executable specification of asynchronous Pi-Calculus semantics and may testing in Maude 2.0. In: Proc. WRLA 2002. ENTCS, Elsevier, Amsterdam (2002)
118. van Deursen, A., Heering, J., Klint, P.: Language Prototyping: An Algebraic Specification Approach. World Scientific, Singapore (1996)

119. Verdejo, A.: Maude como marco semántico ejecutable. PhD thesis, Facultad de Informática, Universidad Complutense, Madrid, Spain (2003)
120. Verdejo, A., Martí-Oliet, N.: Implementing CCS in Maude 2. In: Proc. WRLA 2002. ENTCS, Elsevier, Amsterdam (2002)
121. Verdejo, A., Martí-Oliet, N.: Two case studies of semantics execution in Maude: CCS and LOTOS. Formal Methods in System Design 27(1-2), 113–172 (2005)
122. Verdejo, A., Martí-Oliet, N.: Executable structural operational semantics in Maude. Journal of Logic and Algebraic Programming 67(1-2), 226–293 (2006)
123. Viry, P.: Equational rules for rewriting logic. Theoretical Computer Science 285, 487–517 (2002)
124. Wadler, P.: The essence of functional programming. In: Proc. POPL 1992, pp. 1–14. ACM Press, New York (1992)
125. Wand, M.: First-order identities as a defining language. Acta Informatica 14, 337–357 (1980)
126. Wehrman, I., Kitchin, D., Cook, W.R., Misra, J.: A timed semantics of Orc. Theor. Comput. Sci. 402(2-3), 234–248 (2008)
127. Wirsing, M., Knapp, A.: A formal approach to object-oriented software engineering. In: Proc. WRLA 1996. ENTCS, vol. 4, pp. 322–360 (1996)
128. Wirsing, M., Knapp, A.: A formal approach to object-oriented software engineering. Theoretical Computer Science 285(2), 519–560 (2002)
129. Wright, A.K., Felleisen, M.: A syntactic approach to type soundness. Information and Computation 115(1), 38–94 (1994)

Impugning Randomness, Convincingly

Yuri Gurevich

Microsoft Research

John organized a state lottery, and his wife won the main prize. One may feel that the event of her winning isn't particularly random, but is it possible to convincingly impugn the alleged randomness, in cases like this, in a fair court of law? We develop an approach to do just that.

We start with a principle that bridges between probability theory and the real world. The principle is known under different names in the history of probability theory. It is often formulated thus: events of sufficiently small probability do not happen. This formulation of the principle is too liberal, however. Events of small probability happen all the time. The common way to tighten the principle is to restrict attention to predicted events of sufficiently small probability. Further, we better restrict attention to a single probabilistic experiment. If you repeat the experiment sufficiently many times then there is a good chance that the predicted event of small probability will happen.

The tightened principle plays an important role in statistics. It justifies Fisher's test of statistical significance. Here, the "null hypothesis" is the hypothesis that the given probability distribution is the actual distribution governing the real-world experiment in question, and the "critical region" is induced by the predicted event of small probability.

Our Bridge Principle is a liberalization of the tightened principle above in two directions. One is that the specification of the small-probability event does not have to be given ahead of time; it suffices that it is given independently from the observed outcome of the experiment. The other and bolder liberalization is that the specification may be implicit. It is the Bridge Principle that we use to impugn the alleged randomness in cases like the lottery.

Note that traditional probability theory does not even have the notion of random events. The bolder liberalization above is informed by information complexity theory (also known as Kolmogorov complexity theory and as algorithmic information theory). Unfortunately, traditional information complexity theory is not applicable to real-world scenarios like the lottery one. We use logical definability to generalize information complexity theory (in a way that makes much of the traditional machinery irrelevant).

The talk touches upon issues that have been hotly discussed in several scientific communities, and we finish with an expansive survey of the ocean of related work.

The talk reflects joint work with Grant Passmore of Cambridge University.

Kernelization: An Overview

Daniel Lokshtanov

University of California, San Diego, La Jolla, CA 92093-0404, USA
daniello@ii.uib.no

Abstract. Kernelization is a mathematical framework for the study of polynomial time pre-processing. Over the last few years Kernelization has received considerable attention. In this talk I will survey the recent developments in the field, and highlight some of the interesting research directions.

1 Introduction

The use of computers to solve optimization problems has revolutionized society over the last fifty years. Unfortunately, most computational problems turn out to be NP-complete, and thus we do not hope for provably efficient programs that find the optimal solution for all instances of these problems. Determining whether all NP-complete problems have efficient solutions is known as the P versus NP problem, perhaps the most important open problem in contemporary mathematics and theoretical computer science. In practice, however, instances of NP-complete problems are solved efficiently every day. The reason for this is that instances arising in applications often exhibit some additional structure.

Parameterized algorithms and complexity (PC) is a subfield of theoretical computer science devoted to exploiting additional structure of the input data in order to solve computational problems efficiently. In PC every problem instance comes with a relevant secondary measurement, an integer k called the *parameter*. The parameter k could measure the size or quality of the solution searched for, or reflect how structured the input data is. We seek to contain the "combinatorial explosion" in the running time to the parameter k, instead of the entire instance size. Formally, we say that a problem is *fixed parameter tractable* (FPT) if instances of size n with parameter k can be solved in time $f(k) \cdot n^c$ for a function f depending only on k and a constant c independent of k. The idea is that the parameter k reflects how hard the instance is, with low parameter value meaning that the instance is somehow "easy" or structured. For an introduction to PC, see the textbooks [2,3,5].

One way of obtaining efficient FPT algorithms is through polynomial time preprocessing. When an instance is unreasonably large compared to its parameter, it indicates that the "difficult part" of the instance is very small. For many problems it is possible to deal with, and remove, the easier parts of the instance in polynomial time, leaving only a small, hard core. Formally, a problem has an $f(k)$-kernel if there is a polynomial time algorithm that given an instance with parameter k in polynomial time outputs an equivalent instance with size and parameter value bounded by $f(k)$. Thus kernelization is preprocessing with a performance guarantee, and the definition of kernels opens up for many interesting questions. Which problems admit $f(k)$-kernels for any function f? Which admit kernels where f is polynomial, or even linear? Over the last two decades a number of polynomial and linear kernels for basic problems has been obtained. However, after the recent development of a complexity-theoretic framework for ruling out polynomial kernels [4,1], the field has started to gain considerable attention.

While early kernelization algorithms relied mostly on simple combinatorics, the contemporary "kernelization toolbox" is much more diverse. It includes classical combinatorial min-max theorems, techniques from approximation algorithms, algebraic methods and ideas from logic and automata theory. These tools have been used to show sufficient conditions for large classes of problems to have polynomial, or even linear kernels. In some cases, by using the framework for proving kernelization lower bounds [4,1], the sufficient conditions can even be showed to also be necessary, thus yielding kernelization dichotomies. The presentation aims to give a panoramic overview of the field, a tasty sample from the toolbox and a hazy glimpse of the future.

References

1. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
2. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, New York (1999)
3. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
4. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct pcps for np. In: STOC, pp. 133–142 (2008)
5. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford University Press, Oxford (2006)

Almost Transparent Short Proofs for $\mathrm{NP}_{\mathbb{R}}$[⋆]

Klaus Meer

Computer Science Institute, BTU Cottbus
Konrad-Wachsmann-Allee 1
D-03046 Cottbus, Germany
meer@informatik.tu-cottbus.de

Abstract. We study probabilistically checkable proofs (PCPs) in the real number model of computation as introduced by Blum, Shub, and Smale. Our main result is $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), \mathit{polylog}(n))$, i.e., each decision problem in $\mathrm{NP}_{\mathbb{R}}$ is accepted by a verifier that generates $O(\log n)$ many random bits and reads $\mathit{polylog}(n)$ many proof components. This is the first non-trivial characterization of $\mathrm{NP}_{\mathbb{R}}$ by real $\mathrm{PCP}_{\mathbb{R}}$-classes. As a byproduct this result implies as well a characterization of real nondeterministic exponential time via $\mathrm{NEXP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(\mathrm{poly(n)}, \mathrm{poly(n)})$.

1 Introduction

In the last two decades probabilistically checkable proofs PCPs have turned out to be of major importance in theoretical computer science. They provide a surprising characterization of the complexity class NP and have had significant impact in the area of approximation algorithms. The famous PCP theorem [2,1] states that all problems in NP can be accepted by verifiers which generate a logarithmic number of random bits and probe a constant number of proof bits only, i.e., $\mathrm{NP} = \mathrm{PCP}(O(\log n), O(1))$. A different proof of the theorem has been given in [7].

In the present paper we study probabilistically checkable proofs in the framework of real number complexity theory as introduced by Blum, Shub, and Smale [6]. Real number PCPs in the Blum-Shub-Smale model, henceforth BSS model for short, were first analyzed in [10]. There the existence of long transparent proofs for problems in $\mathrm{NP}_{\mathbb{R}}$, i.e., the inclusion $\mathrm{NP}_{\mathbb{R}} \subset \mathrm{PCP}_{\mathbb{R}}(\mathit{poly}(n), O(1))$, was shown. For precise definitions of the $\mathrm{PCP}_{\mathbb{R}}$-classes see below.

Here, we are interested in reducing the number of random bits generated by verifiers for $\mathrm{NP}_{\mathbb{R}}$ problems. Our main theorem below states that $\mathrm{NP}_{\mathbb{R}}$ has verifiers that use a logarithmic number of random bits and inspect a polylogarithmic number of proof components. More formally we show the characterization $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), \mathit{polylog}(n))$. This provides the first non-trivial characterization of $\mathrm{NP}_{\mathbb{R}}$ by a $\mathrm{PCP}_{\mathbb{R}}$-class. A straightforward corollary of this result is a characterization of $\mathrm{NEXP}_{\mathbb{R}}$ as $\mathrm{PCP}_{\mathbb{R}}(\mathit{poly}(n), \mathit{poly}(n))$. The proof of the main result uses two ingredients which are present in classical PCP results as well,

[⋆] Partially supported by DFG project ME 1424/7-1.

namely low-degree testing and the sum-checking procedure introduced in [9]. However, whereas these two procedures in the classical framework are applied on domains which are finite fields, here we need them to work on appropriately defined finite subsets of the reals without additional algebraic structure. The necessity for this is due to the $\mathrm{NP}_{\mathbb{R}}$-complete Quadratic Polynomial Systems QPS problem for which we want to construct a corresponding verifier. The problem asks for the existence of a real zero of such a system. The verifier expects a proof to contain such a zero coded as a real valued polynomial. Since the components of a zero cannot be guaranteed to be located in a specially structured subset of $\mathbb{R}$ the domains on which our algorithms have to work cannot be assumed to be further restricted any longer.

The paper is organized as follows: In the next section we introduce the necessary notions and define the central real decision problem QPS. It is shown that verifying solvability of an instance polynomial system in a point $a \in \mathbb{R}^n$ can be reduced to evaluate a finite number of canonically derived functions over a not too large finite subset of some $\mathbb{R}^{3k}$, where $k = O(\frac{\log n}{\log \log n})$. We are then in the framework of performing a variant of the LFKN sum-check procedure [9] and of testing whether a given real valued function is a low-degree polynomial. Its variants in our setting are discussed in sections 3.2 and 4. Therein, also the main theorem is proven.

Due to space limitations full details of all the proofs are given in a full version of the paper.

2 Basic Notions

We assume the reader to be familiar with real number complexity theory, see [5]. Very briefly, a BSS machine is a uniform Random Access Machine that computes with real numbers as basic entities. An input $x \in \mathbb{R}^n$ is given the algebraic size $size_{\mathbb{R}}(x) := n$, and each operation $\{+, -, *, :, \geq 0?\}$ among real numbers can be performed with (algebraic) costs 1. The complexity class $\mathrm{NP}_{\mathbb{R}}$ consists of all decision problems L for which there exists a polynomial time verification procedure that satisifes the following requirements. Given $x \in L$ there is a proof y of polynomial length in the (algebraic) size of x such that the procedure accepts (x, y). And for every $x \notin L$ the procedure rejects all tuples (x, y), no matter how y looks like. The Quadratic Polynomial Systems problem introduced below is a typical example of a problem in $\mathrm{NP}_{\mathbb{R}}$.

Usually, the natural verification procedures for $\mathrm{NP}_{\mathbb{R}}$ problems have to inspect all components of y before the decision is made. The question one studies in relation with PCPs is whether this has to be the case. It is formalized using special randomized algorithms.

Definition 1. *(Verifiers) Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be two functions. An $(r(n), q(n))$-restricted verifier V in the BSS model is a particular randomized real number algorithm working in three phases. For an input $x \in \mathbb{R}^* := \bigcup_{i \geq 1} \mathbb{R}^i$ of algebraic size n and another vector $y \in \mathbb{R}^*$ representing a potential membership proof of x in a certain set $L \subseteq \mathbb{R}^*$, the verifier in a first phase produces non-adaptively* *a sequence of $O(r(n))$ many random bits (under the uniform distribution on $\{0,1\}^{O(r(n))}$). Given x and these $O(r(n))$ many random bits V in the next phase computes deterministically the indices of $O(q(n))$ many components of y. Finally, in the decision phase V uses the input x together with the values of the chosen components of y in order to perform a deterministic polynomial time algorithm in the BSS model. At the end of this algorithm V either accepts or rejects x. For an input x, a guess y and a sequence of random bits ρ we denote by $V(x,y,\rho) \in \{0,1\}$ the result of V in case the random sequence generated for (x,y) was ρ.*

The time used by the verifier in the decision phase 3 is also called its decision-time.

Though being a real number algorithm the verifier generates discrete random bits in phase 1. The use of these bits is for addressing registers of the machine in which the basic units of a proof y, i.e., real numbers are stored. Therefore it is appropriate to work with this discrete kind of randomness.

We can define the real language accepted by a verifier together with complexity classes $\mathrm{PCP}_{\mathbb{R}}(r(n), q(n))$ as follows.

Definition 2. *(*$\mathrm{PCP}_{\mathbb{R}}$*-classes) Let* $r, q : \mathbb{N} \mapsto \mathbb{N}$; *a real number decision problem $L \subseteq \mathbb{R}^*$ is in class* $\mathrm{PCP}_{\mathbb{R}}(r(n), q(n))$ *iff there exists an $(r(n), q(n))$-restricted verifier V such that conditions a) and b) below hold:*

a) For all $x \in L$ there exists a $y \in \mathbb{R}^*$ such that for all randomly generated strings $\rho \in \{0,1\}^{O(r(size_{\mathbb{R}}(x)))}$ the verifier accepts.
b) For any $x \notin L$ and for each $y \in \mathbb{R}^*$ the verifier rejects with probability at least $\frac{3}{4}$.

In both cases the probability is chosen uniformly over all random strings ρ.

Let *poly* denote the class of univariate polynomials and *polylog* be the class of all univariate functions of type $n \rightarrow c \cdot log^k(n)$ for some constants $c > 0, k \in \mathbb{N}$.

2.1 The QPS Problem

In this section we develop the scenario for what a verifier has to check in order to show the main result. Note that the definition of $\mathrm{PCP}_{\mathbb{R}}$-classes is closed under polynomial time many-one reductions; such a reduction can be included in the third phase of a verifier's computation. Starting from a suitable $\mathrm{NP}_{\mathbb{R}}$-complete problem we have to take into account the way in which the real number framework influences the task. It turns out that as in the discrete setting the problem is to probabilistically check whether the sum of each of finitely many multivariate polynomials over a particular domain evaluates to zero.

The central $\mathrm{NP}_{\mathbb{R}}$-complete problem we consider here is a version of the real Hilbert-Nullstellensatz decision problem. Below it is defined in a restricted yet sufficiently general form. This form is needed for elaborating the task the verifier has to solve.

Definition 3. *An instance of the QPS decision problem is given as follows. Let $m, n \in \mathbb{N}$, where m is polynomially bounded in n; let $\mathcal{P} = (P_1, \ldots, P_m)$ be a system of real polynomials over n variables $x_1, \ldots, x_n$ such that the following holds:*

a) each P_i has degree at most 2;
b) each P_i depends on at most three variables and is of one of the following forms
   i) $P_i := x_j - c$ for some $j \in \{1, \ldots, n\}$ and a real $c \in \mathbb{R}$; the finitely many different c in such polynomials are part of the input;
   ii) $P_i := x_j - x_k \circ x_\ell$, where $j, k, \ell \in \{1, \ldots, n\}$ and $\circ \in \{+, -, \cdot\}$. Here, $k = \ell$ is allowed.

The QPS decision problem asks for the existence of a real zero $a \in \mathbb{R}^n$ of $\mathcal{P}$, i.e., $P_i(a) = 0$ for all $1 \le i \le m$.

The QPS problem is $\mathrm{NP}_{\mathbb{R}}$-complete [6]. The usual $\mathrm{NP}_{\mathbb{R}}$-verification procedure for showing QPS $\in \mathrm{NP}_{\mathbb{R}}$ guesses an $a \in \mathbb{R}^n$ and plugs it into each P_i in order to see whether it is a zero of the system. This in general requires reading all components of a and thus is useless for our purposes. In order to construct a better verifier we follow the classical approach of coding a via a low degree polynomial on a suitable domain H as well as on an extension F of H. We shall see next that checking whether such a polynomial codes a zero of the system results in verifying that a finite family of certain polynomial functions arising canonically from the given QPS instance and a potential zero vanishes identically. Whereas this idea in principle is the same as in the classical PCP proof for constructing verifiers for 3-SAT using low degree polynomials some differences occur. As a minor one the polynomial functions that have to be considered on some H^k are different ones due to their origin in real polynomial systems. As a more serious one the domains H and F now are subsets of $\mathbb{R}$ in contrast to the classical setting where F is a finite field. The reason for this extension is that a has real components. This will make it necessary to use low-degree testing on more general domains. We shall mainly rely on a version of such a test given in [8].

3 Low Degree Polynomials On Suitable Real Subsets

We first set up the framework for the testing. We use the notation in [3] adapted to our situation.

3.1 What the Verifier Should Check

For a QPS instance of polynomials in n variables let $H \subset \mathbb{R}$ be a finite set of cardinality at least $\lceil \log n \rceil$, F a superset of H of cardinality $\Omega(\log^4 n)$ and $k := \lceil \frac{\log n}{\log \log n} \rceil$. Since $|H|^k \ge n$ a point $a \in \mathbb{R}^n$ can be coded as a function $f_a : H^k \to \mathbb{R}$. Our verifiers below frequently pick arguments from H or F by random and wish to know certain function values in the arguments picked.

Therefore, though the results of such an oracle call might be real the arguments picked have to be addressed by bitstrings. The following definition takes this issue into account. It collects the sets and parameters we are going to use.

Definition 4. *a) For $n \in \mathbb{N}$ let $H := \{0, 1, \ldots, \lceil \log n \rceil\}, F := \{0, 1, \ldots, q\}$, where $q := 100 \cdot \lceil \log^4 n \rceil, d := O(\log n)$ and $k := \lceil \frac{\log n}{\log \log n} \rceil$.*
b) For $s, t \in \mathbb{N}$ let $F_{s,t}$ denote the set of all real polynomials P in t variables of total degree s, i.e., $P : F^t \to \mathbb{R}$ is the restriction of a degree s polynomial over $\mathbb{R}^t$ to F^t.

The following is standard.

Lemma 1. *For $n \in \mathbb{N}$ let F be as above and let $s, t \in \mathbb{N}$ be such that $s \leq |F|$.*
a) Two distinct polynomials in $F_{s,t}$ agree on at most $s \cdot |F|^{t-1}$ elements of F^t.
b) Let $f : H^t \to \mathbb{R}$. There is a unique polynomial $p_f \in F_{t(|H|-1),t}$ of degree at most $|H| - 1$ in each variable and thus of total degree $\leq t \cdot |H|$ which extends f to F^t.

If $s = O(\log^2 n)$, then $|F| = \Omega(\log^4 n)$ together with a) imply in particular that two different polynomials in $F_{s,t}$ agree on less than a fraction of $O(\log^{-2} n)$ many points in F^t.

We code a potential zero $a \in \mathbb{R}^n$ of a polynomial system via a function $f_a : H^k \to \mathbb{R}$. For f_a we consider its low degree extension to $F^k \to \mathbb{R}$ as given by part b) of the lemma above. Note that the resulting total degree will be at most $O(\log^2 n)$.

The verification procedure will check whether certain polynomials defined on H^{3k} identically vanish. We next introduce the corresponding polynomials that have to be inspected. The polynomials occuring in an instance of QPS can be divided into the following different types:

- type 1 are polynomials of form $x_j - c$,
- type 2 are polynomials of form $x_j - x_k \cdot x_\ell$,
- type 3 and type 4 are the same as type 2 but with $+$ and $-$ instead of $\cdot$ as operation combining x_k and x_ℓ, respectively.

Let us first consider all polynomials of type 2 in $\mathcal{P}$. Denote by $\chi^{(2)} : H^{3k} \to \mathbb{R}$ the Lagrange polynomial which for $i_1, i_2, i_3 \in H^k$ (recall that $|H|^k \geq n$) gives result 1 if a polynomial of type 2 in variables $x_{i_1}, x_{i_2}, x_{i_3}$ occurs in $\mathcal{P}$, i.e., $x_{i_1} - x_{i_2} \cdot x_{i_3} = 0$ is an equation. Otherwise $\chi^{(2)}$ has value 0. Note that the order of i_1, i_2, i_3 has to be respected in the argument.

If $a \in \mathbb{R}^n$ is a zero of $\mathcal{P}$ and $\chi^{(2)}(i_1, i_2, i_3) = 1$, then

$$p_a^{(2)}(i_1, i_2, i_3) := a_{i_1} - a_{i_2} \cdot a_{i_3} = f_a(i_1) - f_a(i_2) \cdot f_a(i_3)$$

has to vanish. Consequently, $a \in \mathbb{R}^n$ is a zero of all type 2 polynomials in $\mathcal{P}$ if and only if

$$\sum_{(i_1,i_2,i_3) \in H^{3k}} \left[\chi^{(2)}(i_1, i_2, i_3) \cdot p_a^{(2)}(i_1, i_2, i_3)\right]^2 = 0 \qquad (1)$$

For later purposes we consider the low degree extensions of $\chi^{(2)}$ to F^{3k} and of f_a to F^k, respectively, and denote them again by the same symbols. The following observations are crucial for the further ongoing. First, by Lemma 1 the polynomial $g^{(2)} := \left(\chi^{(2)} \cdot p_a^{(2)}\right)^2 : F^{3k} \rightarrow \mathbb{R}$ is of total degree at most $10k|H| = O(\frac{\log^2 n}{\log \log n})$ and of degree $6|H| = O(\log n)$ in each variable. Next, for any $r := (i_1, i_2, i_3) \in F^{3k}$ the value $\chi^{(2)}(r)$ is deterministically computable in polynomial time with respect to the size of the QPS system $\mathcal{P}$. The same is true for $p_a^{(2)}(r)$ once the three values $f_a(i_1), f_a(i_2), f_a(i_3)$ are known. It follows that for all $r \in F^{3k}$ the value $g^{(2)}(r)$ can be computed efficiently once the three values of f_a are available.

For polynomials of type 3 and 4 the same arguments result in the existence of polynomials $g^{(3)}, g^{(4)}$ having analogue properties. For polynomials of type 1 we argue as follows. Suppose the instance involves constants $c_1, \ldots, c_s$ in polynomials of type 1. For $a \in \mathbb{R}^n$ fixed and each $1 \leq \ell \leq s$ let $\chi_\ell^{(1)}, p_{a,\ell}^{(1)} : H^k \rightarrow \mathbb{R}, \chi_\ell^{(1)}(i_1) = \begin{cases} 1 \text{ if } x_{i_1} - c_\ell \text{ occurs in } \mathcal{P} \\ 0 \text{ else} \end{cases}$ and $p_{a,\ell}^{(1)}(i_1) := a_{i_1} - c_\ell = f_a(i_1) - c_\ell$.

Let $\chi_\ell^{(1)}, f_a$ once again denote as well the low degree extensions. Define $g^{(1)} : F^k \rightarrow \mathbb{R}$ as $g^{(1)}(i_1) := \sum_{\ell=1}^{s} \left[\chi_\ell^{(1)}(i_1) \cdot p_{a,\ell}^{(1)}(i_1)\right]^2$. Then $a \in \mathbb{R}^n$ satisfies all equations $P_j(a) = 0$ for P_j of type 1 iff $\sum_{i_1 \in H^k} g^{(1)}(i_1) = 0$. The total degree of $g^{(1)}$ is upper bounded by $4k(|H| - 1) = O(\frac{\log^2 n}{\log \log n})$.

We have thus shown

Theorem 1. *Given an instance $\mathcal{P}$ of QPS a point $a \in \mathbb{R}^n$ is a zero of all the polynomials in $\mathcal{P}$ iff the polynomials $g^{(j)}, 1 \leq j \leq 4$ defined above satisfy*

$$\sum_{r \in H^k} g^{(1)}(r) = 0 \text{ and } \sum_{r \in H^{3k}} g^{(j)}(r) = 0 \text{ for } j = 2, 3, 4 \tag{2}$$

The $g^{(j)}$ are all of degree $O(\frac{\log^2 n}{\log \log n})$. Moreover, evaluating $g^{(j)}$ in a single point r can be done deterministically in polynomial time in size$_{\mathbb{R}}(\mathcal{P})$ in case at most three values of f can be obtained from an oracle at unit cost.

3.2 The Sum-Check Procedure

Let us fix one of the functions $g^{(j)}$, say $g := g^{(2)}$ to explain how condition (2) is checked probabilistically by the verifier. This is done by applying the well-known LKFN-procedure [9] and is standard in the area. For sake of self-containment we thus only briefly explain what is needed to see how the verifier works and why the necessary probability estimates hold as well for the real domains used here.

The verifier probabilistically evaluates $\sum_{r \in H^{3k}} g(r)$ by expecting a proof for solvability of a QPS instance to contain as additional information certain univariate polynomials. For $1 \leq i \leq 3k$ define $g_i : F^i \rightarrow F$ as

$$g_i(x_1,\ldots,x_i) := \sum_{y_{i+1}\in H}\sum_{y_{i+2}\in H}\cdots\sum_{y_{3k}\in H} g(x_1,\ldots,x_i,y_{i+1},\ldots,y_{3k})$$

called the partial-sum polynomials of g.

Note that $\sum\limits_{r\in H^{3k}} g(r) = \sum\limits_{x_1\in H} g_1(x_1)$ and $g_i(x_1,\ldots,x_i) = \sum\limits_{y\in H} g_{i+1}(x_1,\ldots,x_i,y)$. The verifier expects a proof to contain for each $1\le i\le 3k, (r_1,\ldots,r_{i-1})\in F^{i-1}$ a univariate polynomial $x\to g_i'(r_1,\ldots,r_{i-1},x)$ of degree at most d. The proof is required to represent such a polynomial by specifying its $d+1$ many real coefficients. An ideal proof is supposed to use the corresponding restriction $x\to g_i(r_1,\ldots,r_{i-1},x)$ of the partial-sum polynomial g_i as $g_i'(r_1,\ldots,r_{i-1},x)$. The verifier then performs the following test for checking whether (2) holds:

Test Sum-Check

INPUT: A function value table for $g: F^{3k}\to\mathbb{R}$; for each $1\le i\le 3k, b\in F^{i-1}$ a univariate polynomial $x\to g_i'(b,x)$ of degree $d=O(\log n)$ represented by $d+1$ many reals (its coeffcients);

1. read the $d+1$ real coefficients of $x\to g_1'(x)$; evaluate $\sum\limits_{x\in H} g_1'(x)$; reject if result is $\neq 0$;
2. choose random elements $r_1,\ldots,r_{3k}$ from F according to the uniform distribution;
3. for $i=2,\ldots,3k$ read the $d+1$ real coefficients of $x\to g_i'(r_1,\ldots,r_{i-1},x)$; evaluate $\sum\limits_{x\in H} g_i'(r_1,\ldots,r_{i-1},x)$; reject if result is $\neq g_{i-1}'(r_1,\ldots,r_{i-1})$;
4. evaluate $g_{3k}'(r_1,\ldots,r_{3k})$; reject if result is $\neq g(r_1,\ldots,r_{3k})$.

If none of the checks results in rejection, then accept.

In the theorem below we suppose that g can be evaluated efficiently in a random point. In the next section we deal with the low degree test for f_a in order to show that this assumption is correct with high probability.

Theorem 2. *Let $\beta>0$ be sufficiently small and fixed. Let $g: F^{3k}\to\mathbb{R}$ be a polynomial of total degree $\tilde{d} := O(\frac{\log^2 n}{\log\log n})$ and degree $d=O(\log n)$ in each single variable. Assume $g(r)$ can be obtained from one call to an oracle for any randomly chosen $r\in F^{3k}$. Then a verifier V which performs $\lceil\log\frac{1}{\beta}\rceil$ many rounds of Test Sum-Check has the following properties:*

a) If $\sum\limits_{r\in F^{3k}} g(r)=0$ there is a proof such that V accepts with probability 1;
b) if $\sum\limits_{r\in F^{3k}} g(r)\neq 0$, then for all proofs V rejects with probability $\ge 1-\beta$ in at least one of its rounds;
c) V uses $O(\log n)$ random bits and reads polylog(n) many components of the proof;
d) the decision time of V is polylog(n).

Proof. The proof follows the one in the discrete setting. We thus only briefly indicate how it works and where the stated amount of resources comes in. Part a) is obvious. For b) assume $\sum\limits_{r \in F^{3k}} g(r) \neq 0$. If $g_1(x) \neq g_1'(x)$ an error is detected in Step 1. Otherwise, for all $1 \leq i \leq 3k$ and the randomly chosen $r_1, \ldots, r_{3k}$ an error is detected if the partial sum polynomial $x \rightarrow g_i(r_1, \ldots, r_{i-1}, x)$ and the polynomial $x \rightarrow g_i'(r_1, \ldots, r_{i-1}, x)$ given in the proof are not identical on F and this is witnessed by plugging in r_i. According to Lemma 1 two different univariate real polynomials of degree $\tilde{d}$ agree on at most $\tilde{d}$ many points from F (recall that $\tilde{d} < |F|$). Thus in each round of Step 3 the difference of g_i and g_i' is detected with probability $\geq 1 - \frac{\tilde{d}}{|F|}$ by choosing a random r_i. An error in a single round therefore is detected with probability $\geq (1 - \frac{\tilde{d}}{|F|})^{3k} \geq 1 - \frac{3k\tilde{d}}{|F|} > \frac{3}{4}$. In $\lceil \log \frac{1}{\beta} \rceil$ many rounds an error is then detected with probability at least $1 - \beta$.

The relevant complexity parameters used by the test are as follows. Note that since β is a fixed constant it does not have to be listed in the estimates. At the moment we also disregard what is needed to represent f_a in order to evaluate g in the random point; this analysis is postponed to the low degree test. Each univariate polynomial of degree d is specified by $d + 1$ real numbers. For each $1 \leq i \leq 3k$ there are $|F|^i$ many random choices for $r_1, \ldots, r_i$. Thus the proof has to contain $\left(\sum\limits_{i=1}^{3k} |F|^i \right) \cdot (d+1) = O(\log n) \cdot |F|^{3k+1} = \mathit{poly}(n)$ many reals for coding the univariate polynomials g_i'. The amount of randomness needed is bounded by $3k \log |F| = O(\log n)$ many bits for choosing $r_1, \ldots, r_{3k} \in F$. The verifier inspects $3k(d+1) = \mathit{polylog}(n)$ components of the proof plus three values of f_a for evaluating $g(r)$. The decision time of V is basically determined by evaluating $O(k)$ many times the sum of the values of a univariate degree d polynomial in all points $x \in H$. Since $d = O(\log n)$ each single evaluation can be done with $O(\log n)$ many real number operations, summing up to $O(k \cdot |H| \cdot \log n) = \mathit{polylog}(n)$ many operations in total. $\square$

4 Low Degree Test over Unstructured Domains

In order to complete the description of the verifier a test has to be provided which checks whether a function value table for an $f : F^k \rightarrow \mathbb{R}$ indeed corresponds to a low degree polynomial f_a. The verifier has to respect the corresponding probability requirements. More precisely, if for a given $\delta > 0$ it holds $\Pr\limits_{r \in F^k}(f(r) \neq f_a(r)) < \delta$, then the final evaluation in the sum check test of $g(r)$ in a randomly chosen $r \in F^{3k}$ with probability $\geq 1 - 3\delta$ gives the correct value since it relies on three oracle calls to a table which claims to represent f_a. Thus the task is to design a verifier performing a low degree test which has the following properties: Given a proof in form of a function value table for $f : F^k \rightarrow \mathbb{R}$ and may be some additional information the verifier should accept the proof with probability 1 if f is a polynomial of degree d in each variable. It should reject the proof if f is not δ-close to the set of such polynomials for a suitable choice of δ. Closeness is defined as follows:

Definition 5. *Let $\delta > 0$. For a set $F \subseteq \mathbb{R}, d, k \in \mathbb{N}$ two functions $f, g : F^k \to \mathbb{R}$ are δ-close on domain F^k iff the fraction of points $x \in F^k$ such that f and g disagree is at most δ,i.e., with respect to the uniform distribution on F^k it is $\Pr_{x \in F^k}(f(x) \neq g(x)) \leq \delta$.*

Such tests come in a huge variety of versions depending on the situation for which they are developed. In relation to the classical PCP theorems they are usually designed for finite fields as underlying domain F. This turns out to be crucial in order to optimize parameters like randomness and the number and structure of queries to the proof, see [2,1].

For proving our main theorem we need such a low degree test to work for non-structured finite subdomains of $\mathbb{R}$. Here we follow a variant studied and analyzed in [8].

4.1 Low Degree Test and Proof of Main Theorem

The test in [8] is a max-degree test, i.e., it checks whether a given function is close to a polynomial of some given degree in each of its variables. Note that the low degree extension f_a we are dealing with is a polynomial of max-degree d, where $d = O(\log n)$. Let F be an arbitrary finite subset of $\mathbb{R}, k, d \in \mathbb{N}$. The following test is applied to a function $f : F^k \to \mathbb{R}$.

Low-degree Test

INPUT: A function value table for $f : F^k \to \mathbb{R}$.

1. Fix arbitrary elements $a_1, \ldots, a_{d+1} \in F$;
2. choose a random $i \in \{1, \ldots, k\}$ and random elements $r_1, \ldots, r_k$ from F according to the uniform distribution;
3. check if the values of f in the $d+2$ many points $(r_1, \ldots, r_{i-1}, x, r_{i+1}, \ldots, r_k)$ where $x \in \{r_i, a_1, \ldots, a_{d+1}\}$ lie on a univariate polynomial of degree d with respect to x. If not reject, otherwise accept.

Remark 1. Certain parameters in tests like the one above depend on the model of computation used. For the applications below we choose F as a subset of $\mathbb{Z}$; addressing a random element of F requires $\log |F|$ many random bits. Reading the real value of f in such an argument however in the BSS model corresponds to reading one proof component instead of $\log |\tilde{F}|$ many in the Turing model, where $\tilde{F}$ denotes the image of f on F^k.

The following theorem, adapted to our situation, is proven in [8]:

Theorem 3 ([8]). *Let $\delta > 0$ be sufficiently small, $k, d \in \mathbb{N}$. Consider a finite set $F \subset \mathbb{R}$ that is sufficiently large, i.e., both satisfies $|F| > 18 \cdot k \cdot d^3$ and $|F| > 32 \cdot k^2 \cdot d \cdot \frac{1}{\delta^2}$. Let $f : F^k \to \mathbb{R}$ be a function given by a table of its values. Then a verifier performing $O(\frac{k}{\delta})$ many independent repetitions of Low-degree Test applied to f has the following properties:*

i) *If f is a polynomial of degree d in each of its variables the verifier accepts the table with probability* 1*;*

ii) *if f is not δ-close to such a polynomial the verifier rejects with probability* $\geq \frac{7}{8}$*;*

iii) *V generates $O(\frac{1}{\delta} \cdot k \cdot \log |F|)$ random bits and inspects $O(\frac{1}{\delta} \cdot k \cdot d)$ many values in the table for f.*

The low degree test together with the sum-check procedure of the previous section now allow us to design a verifier for QPS obeying the claimed resource bounds. For the proof of the main theorem when given a QPS-instance over n variables we choose $F := \{0, 1, \ldots, c \cdot \log^4(n)\}$ for an appropriate $c > 0$; thus $|F| = \Theta(\log^4(n))$. Let $H := \{0, 1, \ldots, \lceil \log n \rceil\} \subset F, d := \lceil \log n \rceil, k := \lceil \frac{\log n}{\log \log n} \rceil$ and $\delta > 0$ sufficiently small to make all tests working with the necessary probability estimates.

Theorem 4. *$NP_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), polylog(n))$. Moreover, for every problem in $NP_{\mathbb{R}}$ there is a corresponding verifier with decision time $polylog(n)$.*

Proof. The inclusion $\supseteq$ is trivial: An $\mathrm{NP}_{\mathbb{R}}$-verification procedure for a problem in $\mathrm{PCP}_{\mathbb{R}}(O(\log n), polylog(n))$ simulates the given verifier on the polynomially many random strings it can generate. It then makes its decision according to the verifier's result for all its computations.

For the converse let $\mathcal{P}$ be a QPS-instance over n variables and with polynomial size in n. Choose $\beta, \delta > 0$ sufficiently small such that in addition $4\beta + 12 \log \frac{1}{\beta} \cdot \delta < \frac{1}{8}$. This is achievable by first choosing β sufficiently small and then δ according to the above requirement.

The verifier V to be constructed expects a proof of $\mathcal{P}$'s satisfiability to contain a table of function values of an $f : F^k \to \mathbb{R}$ and for each $1 \leq j \leq 4$ a list of univariate polynomials $x \to g_i^{(j)}(r_1, \ldots, r_{i-1}, x)$ of degree d for all $1 \leq i \leq 3k, (r_1, \ldots, r_{i-1}) \in F^{i-1}$ and each given by $d + 1$ many real coefficients.

In a first step V performs $O(\frac{k}{\delta})$ many rounds of Low-degree Test, where $\{a_1, \ldots, a_{d+1}\} := \{0, 1, \ldots, d\}$ are chosen as the required fixed elements from F. If no rejection occurs V continues with four sum-check tests, one for each $g^{(j)}, 1 \leq j \leq 4$. Towards this aim V interprets the $g_i^{(j)}$ as the restrictions of the partial-sum polynomials for $g^{(j)}$ as introduced in Section 3.2. The proof is accepted if none of the four sum-check tests rejects. Given Theorems 1,2 and 3 V has the following properties: If $\mathcal{P}$ has a zero $a \in \mathbb{R}^n$ and V gets a proof which correctly codes the low-degree extension $f_a \in F_{dk,k}$ and the partial-sum polynomials of the corresponding $g^{(j)}, 1 \leq j \leq 4$, then V accepts with probability 1.

Let us then assume that $\mathcal{P}$ has no zero. If the f given in the proof is not δ-close to an $\widehat{f} \in F_{dk,k}$ with degree d in each variable this is detected by V with probability $\geq \frac{7}{8}$. So assume f to be δ-close to such an $\widehat{f} \in F_{dk,k}$ and that $\widehat{a} \in \mathbb{R}^n$ is the point coded by $\widehat{f}$'s restriction to H^k. Since $\mathcal{P}$ has no zero by Theorem 1 $\sum\limits_{r \in H^{3k}} g^{(j)}(r) \neq 0$ for at least one j, where the $g^{(j)}$'s are defined according to $\widehat{f}$. By Theorem 2 this is detected in $O(\log \frac{1}{\beta})$ many rounds of Test Sum-Check (performed for each j) with probability at least $1-4\beta$ if $\widehat{f}$ is computed correctly in the $12 \cdot \log \frac{1}{\beta}$ points necessary for evaluating each $g^{(j)}$ in the chosen random points of F^{3k}. Since $f, \widehat{f}$ are δ-close the latter is satisfied with probability $\geq (1 - 12 \cdot \log \frac{1}{\beta}\delta)$ if V performs the evaluations using the values of f in the required random arguments. Thus, in this case a fault is detected with probability $\geq 1 - 4\beta - 12 \log \frac{1}{\beta}\delta$. By the choices of β and δ the latter is at least $\frac{7}{8}$. Both parts together detect an error with probability $\geq 1 - 2 \cdot \frac{1}{8} = \frac{3}{4}$.

With β, δ being constants V uses as resources $O(k \cdot \log |F|) = O(\log n)$ many random bits for both sequences of tests underlying Theorems 2 and 3. It reads $O(d \cdot k)$ many real proof components in the low-degree test and $O(k \cdot |H|)$ many in the sum-check procedure. This results in $polylog(n)$ many components altogether. Finally, V's decision time is basically determined by evaluating $O(k)$ many times the sum of the values of a univariate polynomial of degree d in all points $x \in H$. Since $d = O(\log n)$ each single evaluation can be done in $O(\log n)$ many real number operations summing up to $O(k \cdot |H| \cdot \log n) = polylog(n)$ many operations in total. □

The following consequence of the above proof will be established in the paper's full version.

Corollary 1. $\mathrm{NEXP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(\mathrm{poly(n)}, \mathrm{poly(n)})$.

4.2 Conclusion and Open Questions

In this paper we have characterized the real number complexity class $NP_{\mathbb{R}}$ as containing precisely those problems in $PCP_{\mathbb{R}}(O(\log n), polylog(n))$. This is the first non-trivial characterization of $NP_{\mathbb{R}}$ by real number $PCP_{\mathbb{R}}$-classes. In the full version of this paper we shall present two further verifiers which are based on other low-degree tests. They prove the above characterization of $NP_{\mathbb{R}}$ once again but differ with respect to certain additional parameters. The latter classically played an important role to improve the results further using a technique called composition-of-verifiers. There are several open questions related. First, is there a real version of the composition lemma for verifiers? If yes, would it result in a further characterization of $NP_{\mathbb{R}}$ through verifiers using less many resources? In view of those other verifiers at least the equality $NP_{\mathbb{R}} = PCP_{\mathbb{R}}(O(\log n), O(\log n))$ seems a reasonable conjecture here. If the latter turns out to be true the next natural conjecture is $NP_{\mathbb{R}} = PCP_{\mathbb{R}}(O(\log n), O(1))$, i.e., a full real analogue of the classical PCP theorem. Whereas a real version of one ingredient of the proof given in [1], namely the inclusion $NP_{\mathbb{R}} \subset PCP_{\mathbb{R}}(poly(n), O(1))$ was shown to be true in [10], a major problem seems to be to establish a real low-degree test for polynomials defined on arbitrary finite subdomains of $\mathbb{R}$ which respects the parameter restrictions necessary to apply a composition lemma. The problems with such a test are a lacking structure of the real domains that occur. In the classical proof, as a major technical issue such tests are considered on suitable finite fields. It is then heavily used that the uniform distribution on these fields is invariant under scalar multiplication and shift operations. This is not at all true on the domains needed for our low-degree tests. There are papers dealing with such tests on more general structures, e.g., [12]. However, the test developed there enlarges the initial domain too much, thus requiring more random bits to address particular function values. It remains an open question whether more efficient low-degree tests can be designed. Of course as another attempt one might try to prove $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$ along the lines of [7]. We as well do not know how to do this at the time being.

Another interesting future question concerns consequences of the results with respect to approximation algorithms over the reals.

Acknowledgement. Thanks to K. Friedl for making available [8].

References

1. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and hardness of approximation problems. Journal of the ACM 45(3), 501–555 (1998); Preliminary version: Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, 14–23 (1992)
2. Arora, S., Safra, S.: Probabilistic checking proofs: A new characterization of NP.. Journal of the ACM 45(1), 70–122 (1998); Preliminary version: Proc. of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, 2–13 (1992)
3. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer, Heidelberg (1999)
4. Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity 1, 3–40 (1990)
5. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, Heidelberg (1998)
6. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Amer. Math. Soc. 21, 1–46 (1989)
7. Dinur, I.: The PCP theorem by gap amplification. Journal of the ACM 54(3) (2007)
8. Friedl, K., Hátsági, Z., Shen, A.: Low-degree tests. In: Proc. SODA, pp. 57–64 (1994)
9. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM 39(4), 859–868 (1992)
10. Meer, K.: Transparent long proofs: A first PCP theorem for $\mathrm{NP}_{\mathbb{R}}$.. Foundations of Computational Mathematics 5(3), 231–255 (2005)
11. Meer, K.: On some relations between approximation problems and PCPs over the real numbers. Theory of Computing Systems 41, 107–118 (2007)
12. Rubinfeld, R., Sudan, M.: Self-testing polynomial functions efficiently and over rational domains. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 23–32. ACM, Orlando (1992)

The Effect of Homogeneity on the Complexity of *k*-Anonymity

Robert Bredereck[1,⋆], André Nichterlein[1], Rolf Niedermeier[1], and Geevarghese Philip[2]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
[2] The Institute of Mathematical Sciences, Chennai, India
{robert.bredereck,andre.nichterlein,rolf.niedermeier}@tu-berlin.de, gphilip@imsc.res.in

Abstract. The NP-hard k-Anonymity problem asks, given an $n \times m$-matrix M over a fixed alphabet and an integer $s > 0$, whether M can be made k-anonymous by suppressing (blanking out) at most s entries. A matrix M is said to be k-anonymous if for each row r in M there are at least $k-1$ other rows in M which are identical to r. Complementing previous work, we introduce two new "data-driven" parameterizations for k-Anonymity—the number t_{in} of different input rows and the number t_{out} of different output rows—both modeling aspects of data homogeneity. We show that k-Anonymity is fixed-parameter tractable for the parameter t_{in}, and it is NP-hard even for $t_{\text{out}} = 2$ and alphabet size four. Notably, our fixed-parameter tractability result implies that k-Anonymity can be solved in *linear time* when t_{in} is a constant. Our results also extend to some interesting generalizations of k-Anonymity.

1 Introduction

Assume that data about individuals are represented by equal-length vectors consisting of attribute values. If all vectors are identical, then we have full homogeneity and thus full anonymity of all individuals. Relaxing full anonymity to k-anonymity, in this work we investigate how the degree of (in)homogeneity influences the computational complexity of the NP-hard problem of making sets of individuals k-anonymous.

Sweeney [24] devised the notion of k-anonymity to better quantify the degree of anonymity in sanitized data. This notion formalizes the intuition that entities who have identical sets of attributes cannot be distinguished from one another. For a positive integer k we say that a matrix M is k-anonymous if, for each row r in M, there are at least $k-1$ other rows in M which are identical to r. Thus k-anonymity provides a concrete optimization goal while sanitizing data: choose a value of k which would satisfy the relevant privacy requirements, and then try to modify—"at minimum cost"—the matrix in such a way that it becomes k-anonymous. The corresponding decision problem k-Anonymity asks,

⋆ Supported by the DFG, research project PAWS, NI 369/10.

additionally given an upper bound s for the number of suppressions allowed, whether a matrix can be made k-anonymous by suppressing (blanking out) at most s entries. While k-Anonymity is our central problem, our results also extend to several more general problems.

We focus on a better understanding of the computational complexity and on tractable special cases of these problems; see Machanavajjhala et al. [18] and Sweeney [24] for discussions on the pros and cons of these models in terms of privacy vs preservation of meaningful data. In particular, note that in the data privacy community "differential privacy" (cleverly adding some random noise) is now the most popular method [9,14]. However, k-Anonymity is a very natural combinatorial problem (with potential applications beyond data privacy), and—for instance—in the case of "one-time anonymization", may still be valuable for the sake of providing a simple model that does not introduce noise.

k-Anonymity and many related problems are NP-hard [19], even when the input matrix is highly restricted. For instance, it is APX-hard when $k = 3$, even when the alphabet size is just two [4]; NP-hard when $k = 4$, even when the number of columns in the input dataset is 8 [4]; MAX SNP-hard when $k = 7$, even when the number of columns in the input dataset is just three [7]; and MAX SNP-hard when $k = 3$, even when the number of columns in the input dataset is 27 [3].

Confronted with this computational hardness, we study the parameterized complexity of k-Anonymity as initiated by Evans et al. [10]. The central question here is how naturally occurring parameters influence the complexity of k-Anonymity. For example, is k-Anonymity polynomial-time solvable for constant values of k? The general answer is "no" since already 3-Anonymity is NP-hard [19], even on binary data sets [4]. Thus, k alone does not give a promising parameterization.[1]

k-Anonymity has a number of meaningful parameterizations beyond k, including the number of rows n, the alphabet size $|\Sigma|$, the number of columns m, and, in the spirit of multivariate algorithmics [21], various combinations of single parameters. Here the arity of $|\Sigma|$ may range from binary (such as gender) to unbounded. For instance, answering an open question of Evans et al. [10], Bonizzoni et al. [5] recently showed that k-Anonymity is fixed-parameter tractable with respect to the combined parameter $(m, |\Sigma|)$, whereas there is no hope for fixed-parameter tractability with respect to the single parameters m and $|\Sigma|$ [10]. We emphasize that Bonizzoni et al. [5] made use of the fact that the value $|\Sigma|^m$ is an upper bound on the number of different input rows, thus implicitly exploiting a very rough upper bound on input homogeneity. Clearly, $|\Sigma|^m$ denotes the maximum possible number of different input rows. In this work, we refine this view by asking how the "degree of homogeneity" of the input matrix influences the complexity of k-Anonymity. In other words, is k-Anonymity fixed-parameter tractable for the parameter "number of different input rows"? In a similar vein, we also study the effect of the degree of homogeneity of the output matrix on the complexity of k-Anonymity. Table 1, which extends similar tables due to Evans et al. [10] and Bonizzoni et al. [5], summarizes known and new results.

[1] However, it has been shown that 2-Anonymity is polynomial-time solvable [3].

Table 1. The parameterized complexity of k-Anonymity. Results proved in this paper are in **bold**. The column and row entries represent parameters. For instance, the entry in row "–" and column "s" refers to the (parameterized) complexity for the single parameter s whereas the entry in row "m" and column "s" refers to the (parameterized) complexity for the combined parameter (s, m).

	–	k	s	k, s
–	NP-hard [19]	NP-hard [19]	W[1]-hard [5]	W[1]-hard [5]
$\lvert\Sigma\rvert$	NP-hard [2]	NP-hard [2]	?	?
m	NP-hard [4]	NP-hard [4]	FPT [10]	FPT [10]
n	FPT [10]	FPT [10]	FPT [10]	FPT [10]
$\lvert\Sigma\rvert, m$	FPT [5]	FPT [10]	FPT [10]	FPT [10]
$\lvert\Sigma\rvert, n$	FPT [10]	FPT [10]	FPT [10]	FPT [10]
t_{in}	**FPT**	**FPT**	**FPT**	**FPT**
t_{out}	**NP-hard**	?	**FPT**	**FPT**

Our contributions. We introduce the "homogeneity parameters" t_{in}, the number of different input rows, and t_{out}, the number of different output rows, for studying the computational complexity of k-Anonymity and related problems. Typically, we expect $t_{\text{in}} \ll n$ and $t_{\text{in}} \ll |\Sigma|^m$. Indeed, t_{in} is a "data-driven parameterization" in the sense that one can efficiently measure in advance the instance-specific value of t_{in} whereas $|\Sigma|^m$ is a trivial upper bound for homogeneity.

First, we show that there is always an optimal solution (minimizing the number of suppressions) with $t_{\text{out}} \leq t_{\text{in}}$. Then, we derive an algorithm that solves k-Anonymity in $O(nm + 2^{t_{\text{in}}t_{\text{out}}} t_{\text{in}}(t_{\text{out}}m + t_{\text{in}}^2 \cdot \log(t_{\text{in}})))$ time, which compares favorably with Bonizzoni et al.'s [5] algorithm running in $O(2^{(|\Sigma|+1)^m} kmn^2)$ time. Since $t_{\text{out}} \leq t_{\text{in}}$, this shows that k-Anonymity is fixed-parameter tractable when parameterized by t_{in}. In particular, when t_{in} is a constant, our algorithm solves k-Anonymity in time *linear* in the size of the input. In contrast, when only t_{out} is fixed, then we show that the problem remains NP-hard. More precisely, opposing the trivial case $t_{\text{out}} = 1$, we show that k-Anonymity is already NP-hard when $t_{\text{out}} = 2$, even when $|\Sigma| = 4$. We remark that t_{out} is an interesting parameter since it is "stronger" than t_{in} and since, interpreting k-Anonymity as a (meta-)clustering problem (of Aggarwal et al. [1]), t_{out} may also be interpreted as the number of output "clusters".

Finally, we mention that all our results extend to more general problems, including ℓ-Diversity [18] and "k-Anonymity with domain generalization hierarchies" [23]; we defer the corresponding details to a full version of the paper.

Preliminaries and basic observations. Our inputs are datasets in the form of $n \times m$-matrices, where the n rows refer to the individuals and the m columns correspond to attributes with entries drawn from an alphabet Σ. Suppressing an entry $M[i,j]$ of an $n \times m$-matrix M over alphabet Σ with $1 \leq i \leq n$ and $1 \leq j \leq m$ means to simply replace $M[i,j] \in \Sigma$ by the new symbol "$\star$" ending up with a matrix over the alphabet $\Sigma \uplus \{\star\}$. A *row type* is a string from $(\Sigma \uplus \{\star\})^m$. We say that a row in a matrix has a certain row type if it coincides in all its entries with the row type. In what follows, synonymously, we sometimes also speak of a row "lying in a row type", and that the row type "contains" this row. We say that a matrix is *k-anonymous* if every row type contains none or at least k rows in the matrix. A natural objective when trying to achieve k-anonymity is to minimize the number of suppressed matrix entries. For a row y (with some entries suppressed) in the output matrix, we call a row x in the input matrix the *preimage of y* if y is obtained from x by suppressing in x the $\star$-entry positions of y. The central problem of this work reads as follows.

k-Anonymity
Input: An $n \times m$-matrix M and nonnegative integers k, s.
Question: Can at most s elements of M be suppressed to obtain a k-anonymous matrix M'?

Our algorithmic results mostly rely on concepts of parameterized algorithmics [8,12,20]. The fundamental idea herein is, given a computationally hard problem X, to identify a parameter p (typically a positive integer or a tuple of positive integers) for X and to determine whether a size-n input instance of X can be solved in $f(p) \cdot n^{O(1)}$ time, where f is an arbitrary computable function. If this is the case, then one says that X is *fixed-parameter tractable* for the parameter p. The corresponding complexity class is called FPT. If X could only be solved in polynomial running time where the degree of the polynomial depends on p (such as $n^{O(p)}$), then, for parameter p, problem X only lies in the parameterized complexity class XP.

We study two new parameters t_{in} and t_{out}, where t_{in} denotes the number of input row types in the given matrix and t_{out} denotes the number of row types in the output k-anonymous matrix. Note that using sorting t_{in} can be efficiently determined for a given matrix. To keep the treatment simple, we assume that t_{out} is a user-specified number which bounds the maximum number of output types. Clearly, one could have variations on this: for instance, one could ask to minimize t_{out} with at most a given number s of suppressions. We refrain from further exploration here and treat t_{out} as part of the input specifying an upper bound on the number of output types.[2] The following lemma says that without loss of generality one may assume $t_{\text{out}} \leq t_{\text{in}}$.

Lemma 1. *Let (M, k, s) be a* YES*-instance of k-Anonymity. If M has t_{in} row types, then there exists a k-anonymous matrix M' with at most t_{in} row types which can be obtained from M by suppressing at most s elements.*

2 Parameter t_{in}

In this section we show that k-Anonymity is fixed-parameter tractable with respect to the parameter number t_{in} of input row types. Since $t_{\text{in}} \leq |\Sigma|^m$ and $t_{\text{in}} \leq n$, k-Anonymity is also fixed-parameter tractable with respect to the

[2] Indeed, interpreting k-Anonymity as a clustering problem with a guarantee k for the minimum cluster size (as in the work by Aggarwal et al. [1]), t_{out} can also be seen as the number of "clusters" (that is, output row types) that are built.

Algorithm 1. Pseudo-code for solving k-Anonymity. The function `solveRowAssignment` solves Row Assignment in polynomial time, see Lemma 2.

```
 1: procedure SOLVEKANONYMITY($M$, $k$, $s$, $t_{\text{out}}$)
 2:     Determine the row types $\mathsf{R}_1, \ldots, \mathsf{R}_{t_{\text{in}}}$          ▷ Phase 1, Step 1
 3:     for each possible $A : [0,1]^{t_{\text{in}} \times t_{\text{out}}}$ do          ▷ Phase 1, Step 2
 4:         for $j \leftarrow 1$ to $t_{\text{out}}$ do                                     ▷ Phase 1, Step 3
 5:             if $A[1,j] = A[2,j] = \ldots = A[t_{\text{in}}, j] = 0$ then
 6:                 delete empty output row type $\mathsf{R}'_j$
 7:                 decrease $t_{\text{out}}$ by one
 8:             else
 9:                 Determine all entries of $\mathsf{R}'_j$
10:         if solveRowAssignment then                                          ▷ Phase 2
11:             return 'YES'
12:     return 'NO'
```

combined parameter $(m, |\Sigma|)$ and the single parameter n. Both these latter results were already proved quite recently; Bonizzoni et al. [5] demonstrated fixed-parameter tractability for the parameter $(m, |\Sigma|)$, and Evans et al. [10] showed the same for the parameter n. Besides achieving a fixed-parameter tractability result for a more general and typically smaller parameter, we improve their results by giving a simpler algorithm with a (usually) better running time.

Let (M, k, s) be an instance of k-Anonymity, and let M' be the (unknown) k-anonymous matrix which we seek to obtain from M by suppressing at most s elements. Our fixed-parameter algorithm works in two phases. In the first phase, the algorithm guesses the entries of each row type R'_j in M'. In the second phase, the algorithm computes an assignment of the rows of M to the row types R'_j in M'—see Algorithm 1 for an outline.

We now explain the two phases in detail, beginning with Phase 1. To determine the row types R_i of M (line 2 in Algorithm 1), the algorithm constructs a trie [13] on the rows of M. The leaves of the trie correspond to the row types of M. For later use, the algorithm also keeps track of the numbers $n_1, n_2, \ldots, n_{t_{\text{in}}}$ of each type of row that is present in M; this can clearly be done by keeping a counter at each leaf of the trie and incrementing it by one whenever a new row matches the path to a leaf. All of this can be done in a single pass over M.

For implementing the guess in Step 2 of Phase 1, the algorithm goes over all binary matrices of dimension $t_{\text{in}} \times t_{\text{out}}$; such a matrix A is interpreted as follows: A row of type R_i is mapped[3] to a row of type R'_j if and only if $A[i,j] = 1$ (see line 3). Note that we allow in our guessing step an output type to contain no row of any input row type. These "empty" output row types are deleted. Hence, with our guessing in Step 2, we guess not only output matrices M' with *exactly* t_{out} types, but also matrices M' with *at most* t_{out} types.

Now the algorithm computes the entries of each row type R'_j, $1 \leq j \leq t_{\text{out}}$, of M' (Step 3 of Phase 1). Assume for ease of notation that $\mathsf{R}_1, \ldots, \mathsf{R}_\ell$ are the

[3] Note that not all rows of an input type need to be mapped to the same output type.

row types of M which contribute (according to the guessing) at least one row to the (as yet unknown) output row type R'_j. Now, for each $1 \le i \le m$, if $\mathsf{R}_1[i] = \mathsf{R}_2[i] = \ldots = \mathsf{R}_\ell[i]$, then set $\mathsf{R}'_j[i] := \mathsf{R}_1[i]$; otherwise, set $\mathsf{R}'_j[i] := \star$. This yields the entries of the output type R'_j, and the number ω_j of suppressions required to convert any input row (if possible) to the type R'_j is the number of $\star$-entries in R'_j.

The guessing in Step 2 of Phase 1 takes time exponential in the parameter t_{in}, but Phase 2 can be done in polynomial time. To show this, we prove that Row Assignment is polynomial-time solvable. We do this in the next lemma, after formally defining the Row Assignment problem. To this end, we use the two sets $T_{\text{in}} = \{1, \ldots, t_{\text{in}}\}$ and $T_{\text{out}} = \{1, \ldots, t_{\text{out}}\}$.

Row Assignment
Input: Nonnegative integers k, s, $\omega_1, \ldots, \omega_{t_{\text{out}}}$ and $n_1, \ldots, n_{t_{\text{in}}}$ with $\sum_{i=1}^{t_{\text{in}}} n_i = n$, and a function $a : T_{\text{in}} \times T_{\text{out}} \to \{0, 1\}$.
Question: Is there a function $g : T_{\text{in}} \times T_{\text{out}} \to \{0, \ldots, n\}$ such that

$$a(i,j) \cdot n \ge g(i,j) \qquad \forall i \in T_{\text{in}} \forall j \in T_{\text{out}} \tag{1}$$

$$\sum_{i=1}^{t_{\text{in}}} g(i,j) \ge k \qquad \forall j \in T_{\text{out}} \tag{2}$$

$$\sum_{j=1}^{t_{\text{out}}} g(i,j) = n_i \qquad \forall i \in T_{\text{in}} \tag{3}$$

$$\sum_{i=1}^{t_{\text{in}}} \sum_{j=1}^{t_{\text{out}}} g(i,j) \cdot \omega_j \le s \tag{4}$$

Row Assignment formally defines the remaining problem in Phase 2: At this stage of the algorithm the input row types $\mathsf{R}_1, \ldots, \mathsf{R}_{t_{\text{in}}}$ and the number of rows $n_1, \ldots, n_{t_{\text{in}}}$ in these input row types are known. The algorithm has also computed the output row types $\mathsf{R}'_1, \ldots, \mathsf{R}'_{t_{\text{out}}}$ and the number of suppressions $\omega_1, \ldots, \omega_{t_{\text{out}}}$ in these output row types. Now, the algorithm computes an assignment of the rows of the input row types to output row types such that:

- The assignment of the rows respects the guessing in Step 2 of Phase 1. This is secured by Inequality (1).
- M' is k-anonymous, that is, each output row type contains at least k rows. This is secured by Inequality (2).
- All rows of each input row type are assigned. This is secured by Equation (3).
- The total cost of the assignment is at most s. This is secured by Inequality (4).

Note that in the definition of Row Assignment no row type occurs and, hence, the problem is independent of the specific entries of the input or output row types.

Lemma 2. Row Assignment *can be solved in* $O(t_{in}^3 \cdot \log(t_{in}))$ *time.*

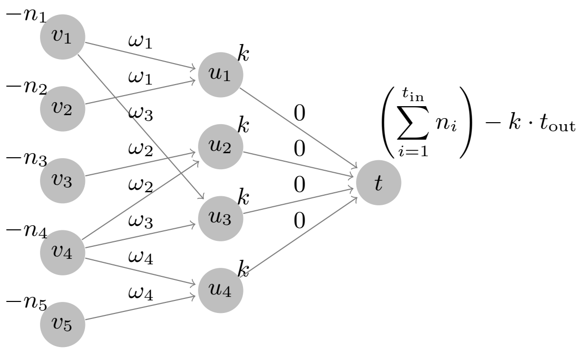


Fig. 1. Example of the constructed network with $t_{\text{in}} = 5$ and $t_{\text{out}} = 4$. The number on each arc denotes its cost. The number next to each node denotes its demand.

Proof (Sketch). We reduce Row Assignment to the Uncapacitated Minimum Cost Flow problem, which is defined as follows [22]:

Uncapacitated Minimum Cost Flow
Input: A network (directed graph) $D = (V, A)$ with demands $b : V \to \mathbb{Z}$ on the nodes and costs $c : V \times V \to \mathbb{N}$.
Task: Find a function f which minimizes $\sum_{(u,v)\in A} c(u,v) \cdot f(u,v)$ and satisfies:

$$\sum_{\{v|(u,v)\in A\}} f(u,v) - \sum_{\{v|(v,u)\in A\}} f(v,u) = b(u) \qquad \forall u \in V$$
$$f(u,v) \geq 0 \qquad \forall (u,v) \in A$$

We first describe the construction of the network with demands and costs. For each n_i, $1 \leq i \leq t_{\text{in}}$, add a node v_i with demand $-n_i$ (that is, a supply of n_i) and for each ω_j add a node u_j with demand k. If $a(i,j) = 1$, then add an arc (v_i, u_j) with cost ω_j. Finally, add a sink t with demand $(\sum n_i) - k \cdot t_{\text{out}}$ and the arcs (u_i, t) with cost zero. See Figure 1 for an example of the construction. Note that, although the arc capacities are unbounded, the maximum flow over one arc is implicitly bounded by n because the sum of all supplies is $\sum_{i=1}^{t_{\text{in}}} n_i = n$.

The Uncapacitated Minimum Cost Flow problem is solvable in $O(|V| \cdot \log(|V|)(|A| + |V| \cdot \log(|V|)))$ time in a network (directed graph) $D = (V, A)$ [22]. Since our constructed network has $t_{\text{in}} + t_{\text{out}}$ nodes and $t_{\text{in}} \cdot t_{\text{out}}$ arcs, we can solve our Uncapacitated Minimum Cost Flow-instance in $O((t_{\text{in}} + t_{\text{out}}) \cdot \log(t_{\text{in}} + t_{\text{out}})(t_{\text{in}} \cdot t_{\text{out}} + (t_{\text{in}} + t_{\text{out}}) \log(t_{\text{in}} + t_{\text{out}})))$ time. Since, by Lemma 1, $t_{\text{in}} \geq t_{\text{out}}$, the running time is $O(t_{\text{in}}^3 \cdot \log(t_{\text{in}}))$. □

Putting all these together, we arrive at the following theorem:

Theorem 1. k-Anonymity *can be solved in* $O(nm + 2^{t_{in}t_{out}} t_{in}(t_{out}m + t_{in}^2 \cdot \log(t_{in})))$ *time, and so, in* $O(nm + 2^{t_{in}^2} t_{in}^2(m + t_{in} \log(t_{in})))$ *time.*

We remark that the described algorithm can be modified to use *domain generalization hierarchies* (DGH) [23] or to solve ℓ-Diversity [18]. We defer the details to a full version of the paper.

We end this section by comparing our algorithmic results to the closely related ones by Bonizzoni et al. [5]. They presented an algorithm for k-Anonymity with a running time of $O(2^{(|\Sigma|+1)^m} kmn^2)$ which works—similarly to our algorithm—in two phases: First, their algorithm guesses all possible output row types together with their entries in $O(2^{(|\Sigma|+1)^m})$ time. In Phase 1 our algorithm guesses the output row types producible from M within $O(2^{t_{\text{in}} t_{\text{out}}} t_{\text{in}} t_{\text{out}} m + mn)$ time using a different approach. Note that, in general, t_{in} is much smaller than the number $|\Sigma|^m$ of all possible different input types. Hence, in general the guessing step of our algorithm is faster. For instances where $|\Sigma|^m \leq t_{\text{in}} \cdot t_{\text{out}}$, one can guess the output types like Bonizzoni et al. in $O(2^{(|\Sigma|+1)^m})$ time.

Next, we compare Phase 2 of our algorithm to the second step of Bonizzoni et al.'s algorithm. In both algorithms, the same problem Row Assignment is solved. Bonizzoni et al. did this using maximum matching on a bipartite graph with $O(n)$ nodes, while we do it using a flow network with $O(t_{\text{in}})$ nodes. Consequently, the running time of our approach depends only on t_{in}, and its proof of correctness is—arguably—simpler.

As we mentioned above, our algorithm can easily be modified to solve k-Anonymity using DGHs with no significant increase in the running time. Since in the DGH setting the alphabet size $|\Sigma|$ increases, it is not immediately clear how the algorithm due to Bonizzoni et al. can be modified to solve this problem without an exponential increase in the running time. Finally, when the parameters t_{in} and $(|\Sigma|, m)$ are constants, then Bonizzoni et al.'s algorithm runs in $O(kmn^2)$ time while our algorithm runs in linear time $O(mn)$.

3 Parameter t_{out}

There is a close relationship between k-Anonymity and clustering problems where one is also interested in grouping together similar objects. Such a relationship has already been observed in related work [1,6,11]. The clustering view on k-Anonymity makes the number t_{out} of output types (corresponding to the number of clusters) of a k-anonymous matrix an interesting parameter.

There is also a more algorithmic motivation for investigating the (parameterized) complexity of k-Anonymity for the parameter t_{out}. As we saw in Theorem 1, k-Anonymity is fixed-parameter tractable for the parameter t_{in}. Due to Lemma 1, we know that t_{out} is a stronger parameter than t_{in}, in the sense that $t_{\text{out}} \leq t_{\text{in}}$. Hence, it is a natural question whether k-Anonymity is already fixed-parameter tractable with respect to the number of output types. Answering this in the negative, we show that k-Anonymity is NP-hard even when there are only two output types and the alphabet has size four, destroying any hope for fixed-parameter tractability already for the combined parameter "number of output types and alphabet size". The hardness proof uses a polynomial-time many-one reduction from Balanced Complete Bipartite Subgraph:

BALANCED COMPLETE BIPARTITE SUBGRAPH (BCBS)
Input: A bipartite graph $G = (V, E)$ and an integer $k \geq 1$.
Question: Is there a complete bipartite subgraph of G whose partition classes are of size at least k each?

BCBS is NP-complete [15]. We provide a polynomial-time many-one reduction from a special case of BCBS with a *balanced* input graph, that is, a bipartite graph that can be partitioned into two independent sets of the same size, and $k = |V|/4$. This special case clearly is also NP-complete by a simple reduction from the general BCBS problem: Let $V := A \uplus B$ with A and B being the two vertex partition classes of the input graph. When the input graph is not balanced, that is, $|A| - |B| \neq 0$, add $||A| - |B||$ isolated vertices to the partition class of smaller size. If $k < |V|/4$, then repeat the following until $k = |V|/4$: Add one vertex to A and make it adjacent to each vertex from B and add one vertex to B and make it adjacent to each vertex from A. If $k > |V|/4$, then add $k - |V|/4$ isolated vertices to each of A and B.

We devise a reduction from BCBS with balanced input graph and $k = |V|/4$ to show the NP-hardness of k-ANONYMITY.

Theorem 2. *k-ANONYMITY is NP-complete for two output row types and alphabet size four.*

Proof (Sketch). We only have to prove NP-hardness, since containment in NP is clear. Let $(G, n/4)$ be a BCBS-instance with $G = (V, E)$ being a balanced bipartite graph and $n := |V|$. Let $A = \{a_1, a_2, \ldots, a_{n/2}\}$ and $B = \{b_1, b_2, \ldots, b_{n/2}\}$ be the two vertex partition classes of G. We construct an $(n/4 + 1)$-ANONYMITY-instance that is a YES-instance if and only if $(G, n/4) \in$ BCBS. To this end, the main idea is to use a matrix expressing the adjacencies between A and B as the input matrix. Partition class A corresponds to the rows and partition class B corresponds to the columns. The salient points of our reduction are as follows.

1. By making a matrix with $2x$ rows x-anonymous we ensure that there are at most two output types. One of the types, the *solution type*, corresponds to the solution set of the original instance: Solution set vertices from A are represented by rows that are preimages of rows in the solution type and solution set vertices from B are represented by columns in the solution type that are not suppressed.
2. We add one row that contains the $\square$-symbol in each entry. Since the $\square$-symbol is not used in any other row, this enforces the other output type to be *fully suppressed*, that is, each column is suppressed.
3. Since the rows in the solution type have to agree on the columns that are not suppressed, we have to ensure that they agree on adjacencies to model BCBS. This is done by using two different types of 0-symbols representing non-adjacency. The 1-symbol represents adjacency.

The matrix D is described in the following and illustrated in Figure 2. There is one row for each vertex in A and one column for each vertex in B. The value in the ith column of the jth row is 1 if a_j is adjacent to b_i and, otherwise, 0_1 if[4] $j \leq$

[4] We assume without loss of generality that n is divisible by four.

$$\begin{array}{c|c|c|c|c|c|}
 & b_1 & b_2 & \dots & b_{n/2-1} & b_{n/2} \\ \hline
a_1 & 1 & 1 & 0_1 & 1 & 1 \\ \hline
a_2 & 0_1 & 1 & 0_1 & 1 & 1 \\ \hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline
a_{n/4} & 1 & 1 & 0_1 & 0_1 & 1 \\ \hline
a_{n/4+1} & 1 & 1 & 0_2 & 0_2 & 1 \\ \hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline
a_{n/2} & 0_2 & 1 & 0_2 & 1 & 1 \\ \hline
\square & \square & \square & \square & \square & \square \\ \hline
1 & 1 & 1 & 1 & 1 & 1 \\ \hline
\end{array}$$

Fig. 2. Typical structure of the matrix D of the k-Anonymity instance obtained by the reduction from Balanced Complete Bipartite Subgraph

$n/4$ and 0_2 if $j > n/4$. Additionally, there are two further rows, one containing only 1s and one containing only $\square$-symbols. The number of allowed suppressions is $s := (n/4+1) \cdot n/2 + (n/4+1) \cdot n/4$. This completes the construction.

It remains to show that $(G, n/4)$ is a YES-instance of BCBS if and only if the constructed matrix can be transformed into an $(n/4+1)$-anonymous matrix by suppressing at most s elements; this part of the proof is deferred to a full version of the paper. $\square$

Deconstructing intractability. In the remainder of this section we briefly discuss the NP-hardness proof for k-Anonymity in the spirit of "deconstructing intractability" [16,21]. In our reduction the alphabet size $|\Sigma|$ and the number t_{out} of output row types are constants whereas the number n of rows, the number m of attributes, number s of suppressions, and the anonymity quality k are unbounded. This suggests a study of the computational complexity of those cases where at least one of these quantities is bounded. Some of the corresponding parameterizations have already been investigated, see Table 1 in Section 1. While for parameters $(|\Sigma|, m)$, $(|\Sigma|, n)$, and n k-Anonymity is fixed-parameter tractable, it is open whether combining t_{out} with m, k, or s helps to obtain fixed-parameter tractability. In particular, the parameterized complexity for the combined parameter $(|\Sigma|, s, k)$ is still open. In contrast, k-Anonymity is W[1]-hard for (s, k) [5], that is, it is presumably fixed-parameter intractable for this combined parameter.

Whereas k-Anonymity is NP-hard for constant m and unbounded t_{out}, one can easily construct an XP-algorithm with respect to the combined parameter (t_{out}, m): In $O(2^{m \cdot t_{\text{out}}} \cdot m \cdot t_{\text{out}})$ time guess the suppressed columns for all output row types. Then, guess in $n^{O(t_{\text{out}})}$ time one prototype for each output row type, that is, one input row that is a preimage of a row from the output row type. Now, knowing the entries for each output row, one can simply apply the Row Assignment algorithm from Section 2:

Proposition 1. *k-Anonymity parameterized by (t_{out}, m) is in XP.*

Next, we prove fixed-parameter tractability for k-Anonymity with respect to the combined parameter (t_{out}, s) by showing that the number t_{in} of input types is at most $(t_{\text{out}} + s)$. To this end, consider a feasible solution for an arbitrary k-Anonymity instance. We distinguish between input row types that have rows which have at least one suppressed entry in the solution (*suppressed input row types* in the following) and input row types that do only have rows that remain unchanged in the solution (*unsuppressed input row types* in the following). Clearly, every unsuppressed input row type needs at least one unsuppressed output row type. Thus, the number of unsuppressed input row type cannot exceed t_{out}. Furthermore, the number of rows that have at least one suppressed entry is at most s. Hence the number of suppressed input row types is also at most s. It follows that $t_{\text{in}} \leq t_{\text{out}} + s$. Now, fixed-parameter tractability follows from Theorem 1:

Proposition 2. k-Anonymity *is fixed-parameter tractable with respect to the combined parameter* (t_{out}, s).

However, to achieve a better running time one might want to develop a direct fixed-parameter algorithm for (t_{out}, s). Finally, we conjecture that an XP-algorithm for k-Anonymity can be achieved with respect to the combined parameter (t_{out}, k).

4 Conclusion

This paper adopts a data-driven approach towards the design of (exact) algorithms for k-Anonymity and related problems. More specifically, the parameter t_{in} measures an easy-to-determine input property. Our central message here is that if t_{in} is small or even constant, then k-Anonymity and its related problems are efficiently solvable—for constant t_{in} even in linear time. On the contrary, already for two output types k-Anonymity becomes computationally intractable.

We contributed to a refined analysis of the computational complexity of k-Anonymity. The state of the art including several research challenges concerning natural parameterizations for k-Anonymity is surveyed in Table 1 in the introductory section. Besides running time improvements in general and open questions indicated in Table 1 such as the parameterized complexity of k-Anonymity for the combined parameter "number of suppressions plus alphabet size", it would also be interesting to determine whether our fixed-parameter tractability result for k-Anonymity with respect to the parameter t_{in} not only extends to the already mentioned generalizations of k-Anonymity but also to the (in terms of privacy) more restrictive t-Closeness problem [17].

References

1. Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., Zhu, A.: Achieving anonymity via clustering. ACM Trans. Algorithms 6(3), 1–19 (2010)

2. Aggarwal, G., Feder, T., Kenthapadi, K., Motwani, R., Panigrahy, R., Thomas, D., Zhu, A.: Anonymizing tables. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 246–258. Springer, Heidelberg (2005)
3. Blocki, J., Williams, R.: Resolving the complexity of some data privacy problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 393–404. Springer, Heidelberg (2010)
4. Bonizzoni, P., DellaVedova, G., Dondi, R.: Anonymizing binary and small tables is hard to approximate. J. Comb. Optim. 22, 97–119 (2011)
5. Bonizzoni, P., Della Vedova, G., Dondi, R., Pirola, Y.: Parameterized complexity of k-anonymity: Hardness and tractability. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 242–255. Springer, Heidelberg (2011)
6. Bredereck, R., Nichterlein, A., Niedermeier, R., Philip, G.: Pattern-guided data anonymization and clustering. In: Proc. 36th MFCS. LNCS. Springer, Heidelberg (to appear, 2011)
7. Chakaravarthy, V.T., Pandit, V., Sabharwal, Y.: On the complexity of the k-anonymization problem. CoRR, abs/1004.4729 (2010)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Dwork, C.: A firm foundation for private data analysis. Commun. ACM 54, 86–95 (2011)
10. Evans, P.A., Wareham, T., Chaytor, R.: Fixed-parameter tractability of anonymizing data by suppressing entries. J. Comb. Optim. 18(4), 362–375 (2009)
11. Fard, A.M., Wang, K.: An effective clustering approach to web query log anonymization. In: Proc. SECRYPT, pp. 109–119. SciTePress (2010)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
13. Fredkin, E.: Trie memory. Commun. ACM 3(9), 490–499 (1960)
14. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Comput. Surv. 42(4), 14:1–14:53 (2010)
15. Johnson, D.S.: The NP-completeness column: An ongoing guide. J. Algorithms 8, 438–448 (1987)
16. Komusiewicz, C., Niedermeier, R., Uhlmann, J.: Deconstructing intractability–A multivariate complexity analysis of interval constrained coloring. J. Discrete Algorithms 9, 137–151 (2011)
17. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: Proc. 23rd ICDE, pp. 106–115. IEEE, Los Alamitos (2007)
18. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: ℓ-diversity: Privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data 1, 52 (2007)
19. Meyerson, A., Williams, R.: On the complexity of optimal k-anonymity. In: Proc. 23rd PODS, pp. 223–228. ACM, New York (2004)
20. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
21. Niedermeier, R.: Reflections on multivariate algorithmics and problem parameterization. In: Proc. 27th STACS. LIPIcs, vol. 5, pp. 17–32. IBFI, Dagstuhl (2010)
22. Orlin, J.: A faster strongly polynomial minimum cost flow algorithm. In: Proc. 20th STOC, pp. 377–387. ACM, New York (1988)
23. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. IJUFKS 10(5), 571–588 (2002)
24. Sweeney, L.: k-anonymity: A model for protecting privacy. IJUFKS 10(5), 557–570 (2002)

On the Optimal Compression of Sets in PSPACE

Marius Zimand*

Department of Computer and Information Sciences, Towson University, Baltimore, MD, USA

Abstract. We show that if DTIME$[2^{O(n)}]$ is not included in DSPACE $[2^{o(n)}]$, then, for every set B in PSPACE, all strings x in B of length n can be represented by a string *compressed*(x) of length at most $\log(|B^{=n}|) + O(\log n)$, such that a polynomial-time algorithm, given *compressed*(x), can distinguish x from all the other strings in $B^{=n}$. Modulo the $O(\log n)$ additive term, this achieves the information-theoretical optimum for string compression.

Keywords: compression, time-bounded Kolmogorov complexity, pseudo-random generator.

1 Introduction

In many practical and theoretical applications in computer science, it is important to represent information in a compressed way. If an application handles strings x from a finite set B, it is desirable to represent every x by another shorter string *compressed*(x) such that *compressed*(x) describes unambigously the initial x. Regarding the compression rate, ideally, one would like to achieve the information-theoretical bound $|compressed(x)| \leq \log(|B|)$, for all $x \in B$. If a set B is computably enumerable, a fundamental result in Kolmogorov complexity states that for all $x \in B^{=n}$, $C(x) \leq \log(|B^{=n}|) + O(\log n)$, where $C(x)$ is the Kolmogorov complexity of x, i.e., the shortest effective description of x ($B^{=n}$ is the set of strings of length n in B). The result holds because x can be described by its rank in the enumeration of $B^{=n}$. However enumeration is typically a slow operation and, in many applications, it is desirable that the unambiguous description is not merely effective, but also efficient. This leads to the idea of considering a time-bounded version of Kolmogorov complexity. An interesting line of research [Sip83, BFL01, BLvM05, LR05], which we also pursue in this paper, focuses on the time-bounded distinguishing Kolmogorov complexity, $\mathrm{CD}^{\mathrm{t}}(\cdot)$. We say that a program p distinguishes x if p accepts x and only x. $\mathrm{CD}^{\mathrm{t,A}}(\mathrm{x})$ is the size of the smallest program that distinguishes x and that runs in time $t(|x|)$ with access to the oracle A. Buhrman, Fortnow, and Laplante [BFL01] show that for some polynomial p, for every set B, and every string $x \in B^{=n}$, $\mathrm{CD}^{\mathrm{p,B^{=n}}}(\mathrm{x}) \leq 2\log(|\mathrm{B}^{=\mathrm{n}}|) + \mathrm{O}(\log \mathrm{n})$. This is an important and very general result but the upper bound for the compressed string length is roughly

* The author is supported in part by NSF grant CCF 1016158. URL: http://triton.towson.edu/~mzimand

$2\log(|B^{=n}|)$ instead of $\log(|B^{=n}|)$, that one may hope. In fact, Buhrman, Laplante, and Miltersen [BLM00] show that, for some sets B, the factor 2 is necessary. There are some results where the upper bound is asymptotically $\log(|B^{=n}|)$ at the price of weakening other parameters. Sipser [Sip83] shows that the upper bound of $\log(|B^{=n}|)$ can be achieved if we allow the distinguisher program to use polynomial advice: For every set B, there is a string w_B of length poly(n) such that for every $x \in B^{=n}$, $\mathrm{CD}^{\mathrm{poly},\mathrm{B}^{=n}}(\mathrm{x} \mid \mathrm{w_B}) \leq \log(|\mathrm{B}^{=n}|) + \log\log(|\mathrm{B}^{=n}|) + \mathrm{O}(1)$. Buhrman, Fortnow, and Laplante [BFL01] show that $\log(|B^{=n}|)$ can be achieved if we allow a few exceptions: For any B, any ϵ, for all except a fraction of ϵ strings $x \in B^{=n}$, $\mathrm{CD}^{\mathrm{poly},\mathrm{B}^{=n}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=n}|) + \mathrm{poly}\log(\mathrm{n} \cdot 1/\epsilon)$. Buhrman, Lee, and van Melkebeek [BLvM05] show that for all B and $x \in B^{=n}$, $\mathrm{CND}^{\mathrm{poly},\mathrm{B}^{=n}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=n}|) + \mathrm{O}((\sqrt{\log(|\mathrm{B}^{=n}|)} + \log \mathrm{n})\log \mathrm{n})$, wher CND is similar to CD except that the distinguisher program is nondeterministic.

Our main result shows that under a certain reasonable hardness assumption, the upper bound of $\log(|B^{=n}|)$ holds for every set B in PSPACE.

Main Result. Assume that there exists $f \in$ E that cannot be computed by circuits of size $2^{o(n)}$ with PSPACE gates. Then for any B in PSPACE, there exists a polynomial p such that for every $x \in B^{=n}$,

$$\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=n}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=n}|) + \mathrm{O}(\log \mathrm{n}).$$

The main result is a corollary of the following stronger result: Under the same hardness assumption, the distinguisher program p for x of length $\log(|B^{=n}|) + O(\log n)$ is simple conditioned by x, in the sense that $C^{\mathrm{poly}}(p \mid x) = O(\log n)$, where $C^{\mathrm{poly}}(\cdot)$ is the polynomial-time bounded Kolmogorov complexity.

We also consider some variations of the main result in which the set B is in P or in NP. We show that the hardness assumption can be somewhat weakened by replacing the PSPACE gates with Σ_3^p gates. We also show that the distinguisher program no longer needs oracle access to $B^{=n}$ in case we allow it to be nondeterministic and B is in NP.

The hardness assumption in the main result, which we call H1, states that there exists a function $f \in \mathrm{E} = \cup_{c \geq 0}\mathrm{DTIME}[2^{cn}]$ that cannot be computed by circuits of size $2^{o(n)}$ with PSPACE gates. This looks like a technical hypothesis; however, Miltersen [Mil01] shows that the more intuitive assumption "E is not contained in DSPACE$[2^{o(n)}]$" implies H1. We note that this assumption (or related versions) has been used before in somewhat similar contexts. Antunes and Fortnow[AF09] use a version of H1 (with the PSPACE gates replaced by Σ_2^p gates) to show that the semi-measure $m^p(x) = 2^{-C^p(x)}$ dominates all polynomial-time samplable distributions. Trevisan and Vadhan [TV00] use a version of H1 (with the PSPACE gates replaced by Σ_5^p gates) to build for each k a polynomial-time extractor for all distributions with min-entropy $(1-\delta)n$ that are samplable by circuits of size n^k.

1.1 Discussion of Technical Aspects

We present the main ideas in the proof of the main result. The method is reminiscent of techniques used in the construction of Kolmogorov extractors in [FHP$^+$06, HPV09, Zim10]. Let B in PSPACE. To simplify the argument suppose that $|B^{=n}|$ is a power of two, say $|B^{=n}| = 2^k$. If we would have a polynomial-time computable function $E : \{0,1\}^n \to \{0,1\}^k$, whose restriction on B is 1-to-1, then every $x \in B^{=n}$ could be distinguished from the other elements of $B^{=n}$ by $z = E(x)$ and we would obtain $\mathrm{CD}^{\mathrm{poly},\mathrm{B}^{=n}}(\mathrm{x}) \leq |\mathrm{z}| + \mathrm{O}(1) = \log(|\mathrm{B}^{=n}|) + \mathrm{O}(1)$. We do not know how to obtain such a function E, but, fortunately, we can afford a slack additive term of $O(\log n)$ and therefore we can relax the requirements for E. We can consider functions E of type $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$. More importantly, it is enough if E is computable in polynomial time given an advice string σ of length $O(\log n)$ and if every $z \in \{0,1\}^k$ has at most $O(n)$ preimages in $B \times \{0,1\}^{\log n}$. With such an E, the string $z = E(x, 0^{\log n})$ distinguishes x from strings that do not map into z and, using the general result of Buhrman, Fortnow, and Laplante [BFL01], with additional $2\log n + O(1)$ bits we can distinguish x from the other at most $O(n)$ strings that map into z. With such an E, we obtain for every $x \in B^{=n}$ the desired $\mathrm{CD}^{\mathrm{poly},\mathrm{B}^{=n}}(\mathrm{x}) \leq |\mathrm{z}| + |\sigma| + 2\log \mathrm{n} + \mathrm{O}(1) = \log(|\mathrm{B}^{=n}|) + \mathrm{O}(\log \mathrm{n})$.

Now it remains to build the function E. An elementary use of the probabilistic method shows that if we take $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$ at random, with high probability, every $z \in \{0,1\}^k$ has at most $7n$ preimages. The problem is that to compute a random E in polynomial-time we would need its table and the table of such a function has size poly(N), where $N = 2^n$. This is double exponentially larger than $O(\log n)$ which has to be the size of σ from our discussion above.

To reduce the size of advice information (that makes E computable in polynomial time) from poly(N) to $O(\log n)$, we derandomize the probabilistic construction in two steps.

In the first step we observe that counting (the number of preimages of z) can be done with sufficient accuracy by circuits of size poly(N) and constant-depth using the result of Ajtai [Ajt93]. We infer that there exists a circuit G of size poly(N) and constant depth such that $\{E \mid \text{every } z \text{ has} \leq 7n \text{ preimages in } B \times \{0,1\}^{\log n}\} \subseteq \{E \mid G(E) = 1\} \subseteq \{E \mid \text{every } z \text{ has} \leq 8n \text{ preimages in } B \times \{0,1\}^{\log n}\}$. Now we can utilize the Nisan-Wigderson [NW94] pseudo-random generator NW-gen against constant-depth circuits and we obtain that, for most seeds s (which we call good seeds for NW-gen), NW-gen(s) is the table of a function E where each element $z \in \{0,1\}^k$ has at most $8n$ preimages in $B \times \{0,1\}^{\log n}$. This method is inspired by the work of Musatov [Mus10], and it has also been used in [Zim11]. The seed s has size poly $\log(N) = \text{poly}(n)$, which is not short enough.

In the second step we use the Impagliazzo-Wigderson pseudo-random generator [IW97] as generalized by Klivans and van Melkebeek [KvM02]. We observe that checking that a seed s is good for NW-gen can be done in PSPACE, and we use the hardness assumption to infer the existence of a pseudo-random generator H such that for most seeds σ of H (which we call good seeds for H), $H(\sigma)$ is a good seed for NW-gen. We have $|\sigma| = \log |s| = O(\log n)$ as desired. Finally, we take our function E to be the function whose table is NW-gen$(H(\sigma))$, for some good seed σ for H. It follows that, given σ, E is computable in polynomial time and that every $z \in \{0,1\}^k$ has at most $8n$ preimages in $B^{=n} \times \{0,1\}^{\log n}$.

The idea of the 2-step derandomization has been used by Antunes and Fortnow [AF09] and later by Antunes, Fortnow, Pinto and Souza [AFPS07].

2 Preliminaries

2.1 Notation and Basic Facts on Kolmogorov Complexity

We work over the binary alphabet $\{0,1\}$. A string is an element of $\{0,1\}^*$. If x is a string, $|x|$ denotes its length; if B is a finite set, $|B|$ denotes its size. If $B \subseteq \{0,1\}^*$, then $B^{=n} = \{x \in B \mid |x| = n\}$.

The Kolmogorov complexity of a string x is the length of the shortest program that prints x. The t-time bounded Kolmogorov complexity of a string x is the length of the shortest program that prints x in at most $t(|x|)$ steps. The t-time bounded distinguishing Kolmogorov complexity of a string x is the length of the shortest program that accepts x and only x and runs in at most $t(|x|)$ steps. The formal definitions are as follows.

We fix an universal Turing machine U, which is able to simulate any other Turing machine with only a constant additive term overhead in the program length. The Kolmogorov complexity of the string x conditioned by string y, denoted $C(x \mid y)$, is the length of the shortest string p (called a program) such that $U(p, y) = x$. In case y is the empty string, we write $C(x)$.

For the time-bounded versions of Kolmogorov complexity, we fix an universal machine U, that, in addition to the above property, can also simulate any Turing machine M in time $t_M(|x|) \log t_M(|x|)$, where $t_M(|x|)$ is the running time of M on input x. For a time bound $t(\cdot)$, the t-bounded Kolmogorov complexity of x conditioned by y, denoted $C^t(x \mid y)$, is the length of the shortest program p such that $U(p, y) = x$ and $U(p, y)$ halts in at most $t(|x| + |y|)$ steps.

The t-time bounded distinguishing complexity of x conditioned by y, denoted $\mathrm{CD}^{\mathrm{t}}(\mathrm{x} \mid \mathrm{y})$ is the length of the shortest program p such that

(1) $U(p, x, y)$ accepts,
(2) $U(p, v, y)$ rejects for all $v \neq x$,
(3) $U(p, v, y)$ halts in at most $t(|v| + |y|)$ steps for all v and y.

In case y is the empty string λ, we write $\mathrm{CD}^{\mathrm{t}}(\mathrm{x})$ in place of $\mathrm{CD}^{\mathrm{t}}(\mathrm{x} \mid \lambda)$. If U is an oracle machine, we define in the similar way, $\mathrm{CD}^{\mathrm{t,A}}(\mathrm{x} \mid \mathrm{y})$ and $\mathrm{CD}^{\mathrm{t,A}}(\mathrm{x})$, by allowing U to query the oracle A.

For defining t-time bounded nondeterministic distinguishing Kolmogorov complexity, we fix U a nondeterministic universal machine, and we define $\mathrm{CND}^{\mathrm{t}}(\mathrm{x} \mid \mathrm{y})$ in the similar way.

Strings $x_1, x_2, \ldots, x_k$ can be encoded in a self-delimiting way (i.e., an encoding from which each string can be retrieved) using $|x_1| + |x_2| + \ldots + |x_k| + 2\log|x_2| + \ldots + 2\log|x_k| + O(k)$ bits. For example, x_1 and x_2 can be encoded as $\overline{(bin(|x_2|)}01x_1x_2$, where $bin(n)$ is the binary encoding of the natural number n and, for a string $u = u_1 \ldots u_m$, $\overline{u}$ is the string $u_1u_1 \ldots u_mu_m$ (i.e., the string u with its bits doubled).

2.2 Distinguishing Complexity for Strings in an Arbitrary Set

As mentioned, Buhrman, Fortnow and Laplante [BFL01], have shown that for any set B and for every $x \in B^{=n}$ it holds that $\mathrm{CD}^{\mathrm{poly},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x}) \leq 2\log(|\mathrm{B}^{=\mathrm{n}}|) + \mathrm{O}(\log \mathrm{n})$. We need the following more explicit statement of their result.

Theorem 1. *There exists a polynomial-time algorithm A such that for every set $B \subseteq \{0,1\}^*$, every n, every $x \in B^{=n}$, there exists a string p_x of length $|p_x| \leq 2\log(|B^{=n}|) + O(\log n)$ such that*

- *$A(p_x, x) =$ accept,*
- *$A(p_x, y) =$ reject, for every $y \in B^{=n} - \{x\}$.*

2.3 Approximate Counting via Polynomial-Size Constant-Depth Circuits

We will need to do counting with constant-depth polynomial-size circuits. Ajtai [Ajt93] has shown that this can be done with sufficient precision.

Theorem 2. *(***Ajtai's approximate counting with polynomial size constant-depth circuits**[1]*) There exists a uniform family of circuits $\{G_n\}_{n\in\mathbb{N}}$, of polynomial size and constant depth, such that for every n, for every $x \in \{0,1\}^n$, for every $a \in \{0,\ldots,n-1\}$, and for every constant $\epsilon > 0$,*

- *If the number of 1's in x is $\leq (1-\epsilon)a$, then $G_n(x, a, 1/\epsilon) = 1$,*
- *If the number of 1's in x is $\geq (1+\epsilon)a$, then $G_n(x, a, 1/\epsilon) = 0$.*

We do not need the full strength (namely, the uniformity of G_n) of this theorem; the required level of accuracy (just $\epsilon > 0$) can be achieved by non-uniform polynomial-size circuits of depth $d = 3$ (with a much easier proof, see [Vio10]).

2.4 Pseudo-Random Generator Fooling Bounded-Size Constant-Depth Circuits

The first step in the derandomization argument in the proof of Theorem 5 is done using the Nisan-Wigderson pseudo-random generator that "fools" constant-depth circuits [NW94].

Theorem 3 (Nisan-Wigderson pseudo random generator). *For every constant d there exists a constant $\alpha > 0$ with the following property. There exists a function NW-gen $: \{0,1\}^{O(\log^{2d+6} n)} \to \{0,1\}^n$ such that for any circuit G of size 2^{n^α} and depth d,*

$$|\mathrm{Prob}_{s\in\{0,1\}^{O(\log^{2d+6} n)}}[G(\mathit{NW\text{-}gen}(s)) = 1] - \mathrm{Prob}_{z\in\{0,1\}^n}[G(z) = 1]| < 1/100.$$

Moreover, there is a procedure that on inputs (n, i, s) produces the i-th bit of NW-gen(s) in time $\mathrm{poly}(\log n)$. [2]

[1] Ajtai's actual result is stronger in some aspects than what we state here.
[2] The "Moreover ..." part is not stated explicitly in [NW94], but follows easily from the proof.

2.5 Hardness Assumptions and Pseudo-Random Generators

The second step in the derandomization argument in the proof of Theorem 5 uses pseudo-random generators based on the assumption that hard functions exist in $\mathrm{E} = \cup_c \mathrm{DTIME}[2^{cn}]$. Such pseudo-random generators were constructed by Nisan and Wigderson [NW94]. Impagliazzo and Wigderson [IW97] strengthen the results in [NW94] by weakening the hardness assumptions. Klivans and van Melkebeek [KvM02] show that the Impagliazzo-Wigderson results hold in relativized worlds. We use in this paper some instantiations of a general result in [KvM02].

We need the following definition. For a function $f : \{0,1\}^* \to \{0,1\}$ and an oracle A, the circuit complexity $C_f^A(n)$ of f at length n relative to A is the smallest integer t such that there exists an A oracle circuit of size t that computes f on inputs of length n.

We use the following hardness assumptions.

Assumption H1

There exists $f \in \mathrm{E}$ such that for some $\epsilon > 0$ and for some PSPACE complete set A, $C_f^A(n) \geq 2^{\epsilon \cdot n}$.

Assumption H2

There exists $f \in \mathrm{E}$ such that for some $\epsilon > 0$ and for some Σ_3^p complete set A, $C_f^A(n) \geq 2^{\epsilon \cdot n}$.

If H1 holds, then for some oracle A that is PSPACE complete, for every k, there exists $H : \{0,1\}^{c \log n} \to \{0,1\}^n$ computable in time poly(n) such that for every oracle circuit C of size n^k,

$$|\mathrm{Prob}_{\sigma \in \{0,1\}^{c \log n}}[C^A(H(\sigma)) = 1] - \mathrm{Prob}_{s \in \{0,1\}^n}[C^A(s) = 1]| < o(1).$$

If H2 holds, then for some oracle A that is Σ_3^p complete, for every k, there exists $H : \{0,1\}^{c \log n} \to \{0,1\}^n$ computable in time poly(n) such that for every oracle circuit C of size n^k,

$$|\mathrm{Prob}_{\sigma \in \{0,1\}^{c \log n}}[C^A(H(\sigma)) = 1] - \mathrm{Prob}_{s \in \{0,1\}^n}[C^A(s) = 1]| < o(1).$$

3 Main Result

Theorem 4. *Assuming H1, the following holds: For every set B in PSPACE, there exists a polynomial p such that for every length n, and for every string $x \in B^{=n}$, there exists a string z with the following properties:*

(1) $|z| = \lceil \log(|B^{=n}|) \rceil$,
(2) $C^p(z \mid x) = O(\log n)$,
(3) $\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x} \mid \mathrm{z}) = \mathrm{O}(\log \mathrm{n})$.

Before proving the theorem, we note that (1) and (3) immediately imply the following theorem, which is our main result.

Theorem 5. *Assumming H1, the following holds: For every set B in PSPACE there exists a polynomial p, such that for every length n, and for every string $x \in B^{=n}$,*

$$\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=\mathrm{n}}|) + \mathrm{O}(\log \mathrm{n}).$$

Proof. (of Theorem 4) Let us fix a set B in PSPACE and let us focus on $B^{=n}$, the set of strings of length n in B. Let $k = \lceil \log |B^{=n}| \rceil$ and let $K = 2^k$. Also, let $N = 2^n$.

Definition 1. *Let $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$. We say that E is Δ-balanced if for every $z \in \{0,1\}^k$,*

$$\left|E^{-1}(z) \ \cap \ B^{=n} \times \{0,1\}^{\log n}\right| \leq \Delta \cdot \frac{|B^{=n}| \cdot n}{K}.$$

The definition says that if we view E as an N-by-n table with entries colored with colors in $[K]$, then no color z appears in the rectangle $B^{=n} \times \{0,1\}^{\log n}$ (formed by rows in $B^{=n}$ and all the columns of the table) more than $\Delta\times$ (number of occurrences of a color in a perfectly balanced coloring).

The plan for the proof is as follows. Suppose that we have a function $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$ that is Δ-balanced, for some constant Δ.

Furthermore assume that E can be "described" by a string σ, in the sense that given σ as an advice string, E is computable in time polynomial in n.

Fix x in $B^{=n}$ and let $z = E(x, 0^{\log n})$. Clearly, the string z satisfies requirement (1). It remains to show (2) and (3).

Consider the set

$$U = \{u \in B^{=n} \mid \exists v \in \{0,1\}^{\log n}, E(u,v) = z\}.$$

Since E is Δ-balanced, the size of U is bounded by $\Delta \cdot \frac{|B^{=n}| \cdot n}{K} \leq \Delta n$.

Now observe that for some polynomial p,

$$\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x} \mid \mathrm{z}) \leq |\sigma| + 2\log(\Delta \mathrm{n}) + \mathrm{O}(\log \mathrm{n}) + \text{self-delimitation overhead}.$$

Indeed, the following is a polynomial-time algorithm using oracle $B^{=n}$ that distinguishes x (it uses an algorithm A, promised by Theorem 1, that distinguishes x from the other strings in U, using a string p_x of length $2\log(|U^{=n}|) + O(\log n) \leq 2\log(\Delta n) + O(\log n)$).

```
Input: $y$; the strings $z, \sigma, p_x$, defined above, are also given.
If $y \notin B^{=n}$, then reject.
If for all $v \in \{0,1\}^{\log n}$, $E(y,v) \neq z$, then reject.
If $A(y, p_x) =$ reject, then reject.
Else accept.
```

Clearly, the algorithm accepts input y iff $y = x$. Also, since $z = E(x, 0^{\log n})$, $C^p(z \mid x) \leq |\sigma| + O(1)$. For a further application (Theorem 14), note that the above algorithm queries the oracle $B^{=n}$ a single time.

Therefore, if we manage to achieve $\sigma = O(\log n)$, we obtain that $\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x} \mid \mathrm{z}) \leq \mathrm{O}(\log \mathrm{n})$ and $C^p(z \mid x) \leq O(\log n)$.

Thus our goal is to produce a function $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$ that using an advice string σ of length $O(\log n)$ is computable in polynomial time and is Δ-balanced for some constant Δ. Let us call this goal $(*)$.

We implement the ideas exposed in Section 1.1 and the reader may find convenient to review that discussion.

Claim 6. *With probability at least* 0.99*, a random function* $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$ *is* 7*-balanced.*

Proof. For fixed $x \in B^{=n}$, $y \in \{0,1\}^{\log n}$, $z \in \{0,1\}^m$, if we take a random function $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$, we have that $\mathrm{Prob}[E(x,y) = z] = 1/K$. Thus the expected number of preimages of z in the rectangle $B^{=n} \times \{0,1\}^{\log n}$ is $\mu = (1/K) \cdot |B^{=n}| \cdot n$. By the Chernoff's bounds,[3]

$$\mathrm{Prob}[\text{number of preimages of } z \text{ in } B^{=n} \times \{0,1\}^{\log n} > 7\mu] < e^{-(6 \ln 2)\mu}.$$

Therefore, the probability of the event "there is some $z \in \{0,1\}^k$ such that the number of z-cells in $B^{=n} \times \{0,1\}^{\log n}$ is $> 7\mu$" is at most $K \cdot e^{-(6\ln 2)\mu} < 0.01$. ∎

Claim 7. *There exists a circuit G of size* $\mathrm{poly}(N)$ *and constant depth such that for any function $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$ (whose table is given to E as the input),*

(a) If $G(E) = 1$, then E is 8*-balanced,*

(b) If E is 7*-balanced, then $G(E) = 1$.*

Proof. By Theorem 2, there is a constant-depth, $\mathrm{poly}(N)$ size circuit that counts in an approximate sense the occurrences of a string z in $\{0,1\}^k$ in the rectangle $B^{=n} \times \{0,1\}^{\log n}$. If we make a copy of this circuit for each $z \in \{0,1\}^k$ and link all these copies to an AND gate we obtain the desired circuit G.

More precisely, let x_z be the binary string of length $|B^{=n}| \cdot n$, whose bits are indexed as (u,v) for $u \in B^{=n}, v \in \{0,1\}^{\log n}$, and whose (u,v)-bit is 1 iff $E(u,v) = z$. By Theorem 2, there is a constant-depth, $\mathrm{poly}(N)$ size circuit G' that behaves as follows:

- $G'(x_z) = 1$ if the number of 1's in x_z is $\leq 7\frac{|B^{=n}| \cdot n}{K}$,
- $G'(x_z) = 0$ if the number of 1's in x_z is $> 8\frac{|B^{=n}| \cdot n}{K}$,

[3] We use the following version of the Chernoff bound. If X is a sum of independent Bernoulli random variables, and the expected value $E[X] = \mu$, then $\mathrm{Prob}[X \geq (1+\Delta)\mu] \leq e^{-\Delta(\ln(\Delta/3))\mu}$. The standard Chernoff inequality $\mathrm{Prob}(X \geq (1+\Delta)\mu] \leq \left(\frac{e^{\Delta}}{(1+\Delta)^{(1+\Delta)}}\right)^{\mu}$ is presented in many textbooks. It can be checked easily that $\frac{e^{\Delta}}{(1+\Delta)^{(1+\Delta)}} < e^{-\Delta \ln(\Delta/3)}$, which implies the above form.

If the number of 1's is between the two bounds, the circuit G' outputs either 0 or 1.

The circuit G on input a table of E, will first build the string x_z for each $z \in \{0,1\}^k$, then has a copy of G' for each z, with the z-th copy running on x_z and then connects the outputs of all the copies to an AND gate, which is the output gate of G. ∎

Claim 8. *If we pick at random a function* $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$, *with probability at least* 0.99*,* $G(E) = 1$.

Proof. This follows from Claim 6 and from Claim 7 (b). ∎

Let $\tilde{N} = N \cdot n \cdot k$. Let d be the depth of the circuit G. We denote $\tilde{n} = log^{2d+6}\tilde{N}$. Note that $\tilde{n} = \text{poly}(n)$. We consider the Nisan-Wigderson pseudo-random generator for depth d given by Theorem 3. Thus,

$$\text{NW-gen} : \{0,1\}^{\tilde{n}} \to \{0,1\}^{\tilde{N}}.$$

For any string s of length $\tilde{n}$, we view NW-gen(s) as the table of a function $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$.

Claim 9. *If we pick at random $s \in \tilde{n}$, with probability of s at least* 0.9*, it holds that NW-gen(s) is 8-balanced.*

Proof. Since G is a circuit of constant depth and polynomial size, by Theorem 3, the probability of the event "G(NW-gen(s)) $= 1$" is within 0.01 of the probability of the event "$G(E) = 1$," and the second probability is at least 0.99 by Claim 8. Thus the first probability is greater than 0.9. Taking into account that if G(NW-gen(s)) $= 1$ then NW-gen(s) is 8-balanced, the conclusion follows. ∎

Claim 10. *Let $T = \{s \in \{0,1\}^{\tilde{n}} \mid$ NW-gen(s) is 8-balanced$\}$. Then T is in PSPACE.*

Proof. We need to count for every $z \in \{0,1\}^k$, the number of z-cells in the rectangle $B^{=n} \times \{0,1\}^{\log n}$ of the table of NW-gen(s). Since B is in PSPACE and since each entry in the table of NW-gen(s) can be computed in time polynomial in $\tilde{n}$, it follows that the above operation can be done in PSPACE. ∎

Claim 11. *Assume H1. There exists a constant c and a function $H : \{0,1\}^{c \log \tilde{n}} \to \{0,1\}^{\tilde{n}}$, computable in time* $\text{poly}(\tilde{n}) = \text{poly}(n)$*, such that if σ is a string chosen at random in $\{0,1\}^{c \log \tilde{n}}$, with probability at least* 0.8*, it holds that NW-gen($H(\sigma)$)) is 8-balanced.*

Proof. Under assumption H1, there exists a pseudo-random generator $H : \{0,1\}^{c \log \tilde{n}} \to \{0,1\}^{\tilde{n}}$ such that for any set A in PSPACE,

$$|\text{Prob}_{\sigma \in \{0,1\}^{c \log \tilde{n}}}[H(\sigma) \in A] - \text{Prob}_{s \in \{0,1\}^{\tilde{n}}}[s \in A]| < 0.1.$$

Since the set T is in PSPACE, $\text{Prob}_{\sigma \in \{0,1\}^{c \log \tilde{n}}}$[NW-gen($H(\sigma)$) is 8-balanced] is within 0.1 from $\text{Prob}_{s \in \{0,1\}^{\tilde{n}}}$[NW-gen($s$) is 8-balanced]. Since the latter probability is at least 0.9, the conclusion follows. ∎

We can now finish the proof of Theorem 5.

Fix $\sigma \in \{0,1\}^{c \log \tilde{n}}$ such that NW-gen($H(\sigma)$)) is 8-balanced. Note that $|\sigma| = O(\log n)$. Given σ, each entry in the table of NW-gen(($H(\sigma)$)) can be computed in time poly(n). Thus the function $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^k$, whose table is NW-gen(($H(\sigma)$)), satisfies the goal (∗). ∎

4 Variations around the Main Result

We analyze here the polynomial-time bounded distinguishing Kolmogorov of strings in a set B that is in P or in NP. We start with the case $B \in$ P. The following is the analog of Theorem 4 and its main point is that assumption H1 can be replaced by the presumably weaker assumption H2.

Theorem 12. *Assuming H2, the following holds: For every set B in* P*, there exists a polynomial p such that, for every length n, and for every string $x \in B^{=n}$, there exists a string z with the following properties:*
(1) $|z| = \lceil \log(|B^{=n}|) \rceil$,
(2) $C^p(z \mid x) = O(\log n)$,
(3) $\mathrm{CD}^p(x \mid z) = O(\log n)$.

Proof. We follow the proof of Theorem 4. First note that since $B \in$ P, the universal machine does not need oracle access to B. We still need to justify that assumption H1 can be replaced by the weaker assumption H2.

Assumption H1 was used in Claim 11. The point was that the set $T = \{s \mid$ NW-gen(s) is 8-balanced$\}$ is in PSPACE and H1 was used to infer the existence of a pseudo-random generator H that fools T. If $B \in$ P, we can check that NW-gen(s) is sufficiently balanced using less computational power than PSPACE. Basically we need to check that for all $z \in \{0,1\}^k$,

$$|\text{NW-gen}(s)^{-1}(z) \cap B^{=n} \times \{0,1\}^{\log n}| \le \Delta \cdot \frac{|B^{=n}| \cdot n}{K},$$

for some constant Δ. Using Sipser's method from [Sip83], there is a Σ_2^p predicate R such that

- $R(s,z) = 1$ implies $|\text{NW-gen}(s)^{-1}(z) \cap B^{=n} \times \{0,1\}^{\log n}| \le 16 \cdot n$,
- $R(s,z) = 0$ implies $|\text{NW-gen}(s)^{-1}(z) \cap B^{=n} \times \{0,1\}^{\log n}| \ge 64 \cdot n$.

Thus there is a set $T' \subseteq \{0,1\}^{\tilde{n}}$ in Σ_3^p such that for all $s \in T'$, NW-gen(s) is 64-balanced and T' contains all s such that NW-gen(s) is 8-balanced. Note that the second property implies that $|T'| \ge 0.99 \cdot 2^{\tilde{n}}$.

Now assumption H2 implies that there exists a pseudo-random generator $H : \{0,1\}^{c \log(\tilde{n})} \to \{0,1\}^{\tilde{n}}$ that fools T'. In particular it follows that with probability of $\sigma \in \{0,1\}^{c \log(\tilde{n})}$ at least 0.8, $H(\sigma) \in T'$ and thus NW-gen($H(\sigma)$) is 64-balanced. The rest of the proof is identical with the proof of Theorem 5. ∎

The next result is the analog of Theorem 5 for the case when the set B is in P.

Theorem 13. *Assuming H2, the following holds: For every $B \in \mathrm{P}$, there exists a polynomial p such that for all n, and for all $x \in B^{=n}$,*

$$\mathrm{CD}^{\mathrm{poly}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=\mathrm{n}}|) + \mathrm{O}(\log \mathrm{n}).$$

Proof. This is an immediate consequence of (1) and (3) in Theorem 12. ∎

Next we consider the case when the set B is in NP. The main point is that the assumption H1 can be replaced by H2, and that the distinguishing program does not need access to the oracle $B^{=n}$ provided it is nondeterministic.

Theorem 14. *Assuming H2, the following holds: For every set B in* NP, *there exists a polynomial p such that for every length n, and for every string $x \in B^{=n}$, there exists a string z with the following properties:*

(1) $|z| = \lceil \log(|B^{=n}|) \rceil$,
(2) $C^p(z \mid x) = O(\log n)$,
(3) $\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x} \mid \mathrm{z}) = \mathrm{O}(\log \mathrm{n})$.
(4) $\mathrm{CND}^{\mathrm{p}}(\mathrm{x} \mid \mathrm{z}) = \mathrm{O}(\log \mathrm{n})$.

Proof. (1), (2) and (3). We only need to show that in the proof of Theorem 4, in case $B \in \mathrm{NP}$, the assumption H1 can be replaced by the weaker assumption H2. This is done virtually in the same way as in the proof of Theorem 13. The predicate R also needs this time to check that certain strings are in B and this involves an additional quantifier, but that quantifier can be merged with the existing quantifiers and R remains in Σ_2^p.

(4). We need to show that, at the price of replacing CD by CND, the use of the oracle $B^{=n}$ is no longer necessary. Note that the distinguisher procedure given in the proof of Theorem 4, queries the oracle only once, and if the answer to that query is NO, then the algorithm rejects immediately. Thus, instead of making the query, a nondeterministic distinguisher can just guess a witness for the single query it makes. ∎

The following is the analog of Theorem 5 in case the set B is in NP.

Theorem 15. *Assuming H2, the following holds:*

(a) For every $B \in$ NP, *there exists a polynomial p, such that for all n, and for all $x \in B^{=n}$,*

$$\mathrm{CD}^{\mathrm{p},\mathrm{B}^{=\mathrm{n}}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=\mathrm{n}}|) + \mathrm{O}(\log \mathrm{n}).$$

(b) For every $B \in$ NP, *there exists a polynomial p, such that for all n, and for all $x \in B^{=n}$,*

$$\mathrm{CND}^{\mathrm{p}}(\mathrm{x}) \leq \log(|\mathrm{B}^{=\mathrm{n}}|) + \mathrm{O}(\log \mathrm{n}).$$

Proof. Statement (a) follows from (1) and (3) in Theorem 14, and (b) follows from (1) and (4) in Theorem 14. ∎

References

[AF09] Antunes, L., Fortnow, L.: Worst-case running times for average-case algorithms. In: Proceedings of the 24th Conference in Computational Complexity Conference, pp. 303–598. IEEE Computer Society Press, Los Alamitos (2009)

[AFPS07] Antunes, L., Fortnow, L., Pinto, A., Souto, A.: Low-depth witnesses are easy to find. In: IEEE Conference on Computational Complexity, pp. 46–51 (2007)

[Ajt93] Ajtai, M.: Approximate counting with uniform constant-depth circuits. In: Advances in computational complexity, pp. 1–20 (1993)

[BFL01] Buhrman, H., Fortnow, L., Laplante, S.: Resource-bounded Kolmogorov complexity revisited. SIAM J. Comput. 31(3), 887–905 (2001)

[BLM00] Buhrman, H., Laplante, S., Miltersen, P.B.: New bounds for the language compression problem. In: IEEE Conference on Computational Complexity, pp. 126–130 (2000)

[BLvM05] Buhrman, H., Lee, T.: D van Melkebeek. Language compression and pseudorandom generators. Computational Complexity 14(3), 228–255 (2005)

[FHP$^+$06] Fortnow, L., Hitchcock, J.M., Pavan, A., Vinodchandran, N.V., Wang, F.: Extracting Kolmogorov complexity with applications to dimension zero-one laws. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 335–345. Springer, Heidelberg (2006)

[HPV09] Hitchcock, J.M., Pavan, A., Vinodchandran, N.V.: Kolmogorov complexity in randomness extraction. In: FSTTCS, pp. 215–226 (2009)

[IW97] Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In: Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC 1997), pp. 220–229. Association for Computing Machinery, New York (1997)

[KvM02] Klivans, A., van Melkebeek, D.: Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. SIAM J. Comput. 31(5), 1501–1526 (2002)

[LR05] Lee, T., Romashchenko, A.E.: Resource bounded symmetry of information revisited. Theor. Comput. Sci. 345(2-3), 386–405 (2005)

[Mil01] Miltersen, P.B.: Derandomizing complexity classes. In: Pardalos, P., Reif, J., Rolim, J. (eds.) Handbook on Randomized Computing, vol. II, Kluwer Academic Publishers, Dordrecht (2001)

[Mus10] Musatov, D.: Improving the space-bounded version of Muchnik's conditional complexity theory via "naive" derandomization. CoRR, abs/1009.5108 (2010) (to appear in CSR 2011)

[NW94] Nisan, N., Wigderson, A.: Hardness vs. randomness. Journal of Computer and System Sciences 49, 149–167 (1994)

[Sip83] Sipser, M.: A complexity theoretic approach to randomness. In: Proceedings of the 15th ACM Symposium on Theory of Computing, pp. 330–335 (1983)

[TV00] Trevisan, L., Vadhan, S.: Extracting randomness from samplable distributions. In: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, pp. 32–42 (2000)

[Vio10] Viola, E.: Randomness buys depth for approximate counting. Electronic Colloquium on Computational Complexity (ECCC) 17, 175 (2010)

[Zim10] Zimand, M.: Possibilities and impossibilities in Kolmogorov complexity extraction. SIGACT News 41(4) (December 2010)

[Zim11] Zimand, M.: Symmetry of information and bounds on nonuniform randomness extraction via Kolmogorov complexity. In: Proceedings 26th IEEE Conference on Computational Complexity (to appear, June 2011)

Computational Randomness from Generalized Hardcore Sets

Chia-Jung Lee[1], Chi-Jen Lu[1], and Shi-Chun Tsai[2]

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan
{leecj,cjlu}@iis.sinica.edu.tw

[2] Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan
sctsai@csie.nctu.edu.tw

Abstract. The seminal hardcore lemma of Impagliazzo states that for any mildly-hard Boolean function f, there is a subset of input, called the hardcore set, on which the function is extremely hard, almost as hard as a random Boolean function. This implies that the output distribution of f given a random input looks like a distribution with some statistical randomness. Can we have something similar for hard functions with several output bits? Can we say that the output distribution of such a general function given a random input looks like a distribution containing several bits of randomness? If so, one can simply apply any statistical extractor to extract computational randomness from the output of f. However, the conventional wisdom tells us to apply extractors with some additional *reconstruction* property, instead of just any extractor. Does this mean that there is no analogous hardcore lemma for general functions?

We show that a general hard function does indeed have some kind of hardcore set, but it comes with the price of a security loss which is proportional to the number of output values. More precisely, consider a hard function $f : \{0,1\}^n \to [V] = \{1, \ldots, V\}$ such that any circuit of size s can only compute f correctly on at most $\frac{1}{L}(1-\gamma)$ fraction of inputs, for some $L \in [1, V-1]$ and $\gamma \in (0,1)$. Then we show that for some $I \subseteq [V]$ with $|I| = L+1$, there exists a hardcore set $H_I \subseteq f^{-1}(I)$ with density $\gamma/\binom{V}{L+1}$ such that any circuit of some size s' can only compute f correctly on at most $\frac{1+\varepsilon}{L+1}$ fraction of inputs in H_I. Here, s' is smaller than s by some $\text{poly}(V, 1/\varepsilon, \log(1/\gamma))$ factor, which results in a security loss of such a factor. We show that it is basically impossible to guarantee a much larger hardcore set or a much smaller security loss. Finally, we show how our hardcore lemma can be used for extracting computational randomness from general hard functions.

1 Introduction

Impagliazzo's hardcore lemma [9] is a fundamental result in complexity theory which states that any mildly-hard function has a subset of inputs on which it is extremely hard. More precisely, consider a function $f : \{0,1\}^n \to \{0,1\}$ such that any circuit of size s disagrees with f on at least δ fraction of inputs, and we call such a function (δ, s)-hard where the parameter δ is called the hardness of f. Then the hardcore lemma asserts that there exists a subset $H \subseteq \{0,1\}^n$ of density δ such that any circuit of size s' must disagree with f on at least $\frac{1-\varepsilon}{2}$ fraction of inputs from H, for some s' slightly smaller than s. This means that given a random input x in H, although the value of $f(x)$ is fixed and thus has no randomness at all in a statistical sense, it still looks like a random bit to small circuits. Because of this nice property, the hardcore lemma has become an important tool in the study of pseudo-randomness. For example, it was used in [9] for an alternative proof of Yao's XOR lemma [20], used in [17] for constructing a pseudo-random generator directly from a mildly-hard function without going through the XOR lemma, and more recently used in [15,18,19,6] for amplifying hardness of functions in NP. The parameters of the hardcore lemma were later improved by [10,7,2].

Note that Impagliazzo's hardcore lemma works for *Boolean* functions. It says that the output of a hard function given a random input looks like a random bit and thus contains statistical randomness, when the input falls in the hardcore set. When using the lemma, the hard function is usually evaluated at several inputs in order to obtain several output bits, which together can be argued to contain some sufficient amount of randomness. Usually, the amount of randomness in a distribution is measured by its min-entropy, where a distribution has min-entropy at least k if every element occurs with probability at most 2^{-k}. Then from a distribution with some min-entropy, one applies a so-called randomness extractor [21,14] to extract a distribution which looks almost random.

On the other hand, there are natural functions with many output bits which are believed to be hard, such as factoring and discrete logarithm, and one may be able to extract several bits at once from one output value. This is also related to the problem of extracting randomness from sources with computational randomness, studied in [3,8,12]. One may wonder if there is an analogous hardcore lemma for a general non-Boolean function, which can guarantee that the output distribution given a random input will look like one with some min-entropy, hopefully much larger than one. For example, assume that a one-way permutation $g : \{0,1\}^n \to \{0,1\}^n$ exists, whose inverse function $f = g^{-1}$ is hard to compute by small (say, polynomial-size) circuits. Then, if one could show that the distribution of $x = f(y)$ given a random y looks like having some min-entropy to small circuits, one could simply apply *any* extractor on x. However, the conventional wisdom does not suggest so and the following counter example seems to be known as a folklore. Given an efficiently-computable extractor E and a one-way permutation g, the function $\textsc{Ext}$ defined as $\textsc{Ext}(x, u) = \mathrm{E}(g(x), u)$ is still an extractor, but its output can be easily computed (and hence does not look random at all) given $y = g(x)$ and u. To extract such computational randomness, previous works all resorted to extractors with some *reconstruction* property, which roughly corresponds to error correcting codes with efficient decoders (see, e.g., [16] for a definition).

Does this mean that there is no analogous hardcore lemma for general functions? If we consider a hard function $f : \{0,1\}^n \to \{0,1\}^2$ with two, instead of one, output bits, it may be hard to believe that we can no longer have any kind of hardcore lemma for it. But can we guarantee the existence of a hardcore set H such that $f(x)$, for a random $x \in H$, looks like a random value in $\{0,1\}^2$? The answer is no in general because f may in fact have at most three possible output values, so we have to settle for something weaker. One approach is to see each output bit of a (δ, s)-hard function $f : \{0,1\}^n \to \{0,1\}^d$ as a Boolean function, so some of these d Boolean functions must have hardness $\Omega(\delta/d)$ and they have Boolean hardcore sets (with two output values) of density $\Omega(\delta/d)$ using Impagliazzo's lemma. Unfortunately, this only gives a very weak result because even if f is extremely hard, with δ close to $1 - 2^{-d}$, one may still only be able to guarantee one bit of randomness in the output of f if those Boolean hardcore sets are disjoint.

We are looking for something stronger, in which more bits of randomness can be guaranteed. We consider a general (δ, s)-hard function f of the form $f : \{0,1\}^n \to [V] = \{1, \ldots, V\}$. We discover that a good way to see its hardness is to express it in the form of $\delta = 1 - \frac{1}{L}(1-\gamma)$, for some $L \in [1, V-1]$ and $\gamma \in (0,1)$, and we obtain the following results.

First, we show that any function with such hardness has a hardcore set with $L+1$ output values. More precisely, we show that for such a hard function f, there exist some $I \subseteq [V]$ with $|I| = L+1$ and some $H_I \subseteq f^{-1}(I)$ of density $|H_I|/2^n \geq \gamma / \binom{V}{L+1}$ such that any circuit of size s' can only compute f correctly on $\frac{1+\varepsilon}{L+1}$ fraction of the inputs in H_I, where s' is smaller than s by a factor of $\text{poly}(V, 1/\varepsilon, \log(1/\gamma))$. Let us call such a set H_I an (I, ε, s') hardcore set, and let us take a close look at what our result says as L varies. At one end of the spectrum with $L = V-1$, our result guarantees the existence of a hardcore set H_I, with $I = [V]$, such that f restricting to the set H_I has almost the largest possible hardness and it looks like a random function from H_I to $[V]$. Note that when $V = 2$ (and $L = 1$), we have Impagliazzo's hardcore lemma as our special case. As L becomes smaller, the hardness decreases, and it is no longer possible to always have a hardcore set with V output values. Nevertheless, our result shows that one can still have a hardcore set H_I with $|I| = L+1$ output values, such that f restricting to H_I looks like a random function from H_I to I.

Notice that in our first result, we can only guarantee a hardcore set of density $\gamma / \binom{V}{L+1}$, and one may wonder if it is possible to guarantee a larger one. Our second result shows that this is basically impossible. More precisely, we show the existence of a (δ, s)-hard function $f : \{0,1\}^n \to [V]$, with $\delta = 1 - \frac{1}{L}(1-\gamma)$ and $s \geq \text{poly}(\gamma, 1/L, 2^n)$, which has no (I, ε, s') hardcore set of density $4(L+1)\gamma / \binom{V}{L+1}$ for any $I \subseteq [V]$ with $|I| = L+1$, where $s' = \text{poly}(n)$. Note that the density achieved by out first result and that ruled out by our second result are off by an $O(L)$ factor, and we believe that the bound of our second result may be improved. On the other hand, our second result is strong in the sense that even when we start from a function which is hard against very large circuits of exponential size, it is still impossible to have a hardcore set of a small density against small circuits of polynomial size.

With a small hardcore set, one can only say that the output of a hard function f looks somewhat random when the input falls into that small set. This alone is not good enough for the purpose of randomness extraction because the vast majority of inputs are outside of the hardcore set and may contribute a large error. Our next result shows that in fact we can have not just one but a collection of disjoint hardcore sets, and they together cover all but a small fraction of the inputs, which implies that the output of f looks somewhat random for most input. More precisely, we show that for a (δ, s)-hard function, with $\delta \geq 1 - 2^{-k}$, its output distribution given a random input looks close, within some distance ε, to a distribution with min-entropy $\Omega(k)$, by circuits of size $s' = s/\text{poly}(V, 1/\varepsilon)$. This implies that we can simply apply any seeded statistical extractor to extract computational randomness from the output of f as long as s is large (say, super-polynomial) or V is small (say, polynomial). This also works for seedless multi-source extractors, and in particular, it fits nicely with the setting of independent-symbol sources studied in [12] in which each symbol is considered to come from a small set. Therefore, we can generalize the result of [12] from a statistical setting to a computational one: given multiple independent sources, over a small set of symbols, which look slightly random to polynomial-size circuits but may have no min-entropy at all, the statistical extractor there can be used to produce an output which looks almost random to polynomial-size circuits.

Note that in our hardcore set result, there is a security loss of some factor $\text{poly}(V, 1/\varepsilon)$ in circuit size. That is, starting from a function which is hard against circuits of size s, we can only guarantee the hardness of a hardcore set against circuits of size s', with s' smaller than s by that factor. Consequently, with $s = \text{poly}(n)$, we can only extract randomness from a function with $V \leq \text{poly}(n)$ (or equivalently, with $O(\log n)$ output bits). One may wonder if such a security loss of circuit size can be avoided. Our final result shows that this is basically impossible, if the proof is done in a certain black box way. Here, we use the notion of black-box proofs for hardcore sets introduced in [13]. Informally speaking, a black-box proof is realized by an oracle algorithm R such that for any function f and any collection G of circuits, if G breaks the hardcore set condition, then R breaks the hardness of f by using G only as an oracle. In this black-box model, we show that any algorithm R must make at least $q = \Omega((Vk/\varepsilon^2)\log(1/\delta))$ queries in order to show the existence of a hardcore set with k output values. This translates to a loss of a q factor in circuit size, because the resulting circuit of R^G is larger than those in G by this factor. This explains the need of using *reconstructive* extractors, instead of just any extractors, on the input of a one-way permutation discussed before, since there we have a large $V = 2^n$. Finally, we would like to clarify a potential confusion with the security loss of using *reconstructive* extractors in previous works. When applying reconstructive extractors on the output of a hard function f, previous results also suffered some loss of circuit size in the same sense: the outputs of extractors only look random to smaller circuits compared to those which the hardness of f is measured against. However, the loss is in terms of the output length m of extractors, instead of the output length of f. More precisely, the loss factor is $\text{poly}(2^m)$, which again limits us to extracting $O(\log n)$ bits when f is hard against circuits of size $s = \text{poly}(n)$.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$, and let $\mathcal{U}_n$ denote the uniform distribution over $\{0,1\}^n$. For a set X, we let $|X|$ denote the number of elements in X, and for a subset $S \subseteq X$, we say that S has density $|S|/|X|$ in X. For a set X and an integer $n \in \mathbb{N}$, we use the notation $\binom{X}{n}$ to denote the collection of subsets $S \subseteq X$ such that $|S| = n$. When we sample from a finite set, the default distribution is the uniform one. All logarithms used in this paper will have base two. Let $\mathsf{SIZE}(s)$ be the class of functions computable by circuits of size s. We measure the hardness of computing a function in the following way.

Definition 1. *A function f is (δ, s)-hard if any circuit in $\mathsf{SIZE}(s)$ must fail to compute f correctly for at least a δ fraction of inputs.*

Impagliazzo [9] considered Boolean functions and show that any hard function must have a hardcore set such that the function restricted to the hardcore set is extremely hard. In this paper, we consider general functions of the form $f : X \to [V]$, for some input set X and for the output set $[V]$ with integer $V \geq 2$. For such general functions, we introduce our notion of generalized hardcore sets as follows.

Definition 2. *For a function $f : X \to [V]$ and some $I \subseteq [V]$, we say that a subset of inputs $H \subseteq f^{-1}(I)$ is an (I, ε, s) hardcore set if for any circuit $C \in \mathsf{SIZE}(s)$, $\Pr_{x \in H}[C(x) = f(x)] \leq (1+\varepsilon)/|I|$. We say that such an H is a hardcore set with $|I|$ output values.*

Note that $f(x) \in I$ for any $x \in H$, so the above probability bound says that f restricted to H looks like a random function from H to I. We say that a distribution looks like another one if there is no distinguisher for them, defined as follows.

Definition 3. *A function $D : X \to \{0,1\}$ is an ε-distinguisher for two distributions $\mathcal{X}$ and $\mathcal{Y}$ over X if $|\Pr[D(\mathcal{X}) = 1] - \Pr[D(\mathcal{Y}) = 1]| \geq \varepsilon$, and we call such a function D an (ε, s)-distinguisher if $D \in \mathsf{SIZE}(s)$.*

We measure the amount of randomness in a distribution $\mathcal{X}$ by its *min-entropy*, and we say that $\mathcal{X}$ has min-entropy at least k, denoted by $\mathrm{H}_\infty(\mathcal{X}) \geq k$, if for any x, $\Pr[\mathcal{X} = x] \leq 2^{-k}$. From a source which is only weakly random (with some min-entropy), we would like to have a procedure called an extractor to extract a distribution which is almost random. When trying to extract randomness from a single source, one usually needs an additional short seed, and such an extractor is called a seeded one, which is defined as follows.

Definition 4. *A function $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is called a (seeded) (k, ε)-extractor if for any distribution $\mathcal{X}$ over $\{0,1\}^n$ with $\mathrm{H}_\infty(\mathcal{X}) \geq k$, there is no ε-distinguisher for the distributions $\textsc{Ext}(\mathcal{X}, \mathcal{U}_d)$ and $\mathcal{U}_m$.*

When there are at least two independent sources which are weakly random, it becomes possible to have a seedless extractor, which is defined as follows.

Definition 5. *A function* $\text{EXT} : (\{0,1\}^n)^t \to \{0,1\}^m$ *is called a (seedless) t-source (k, ε)-extractor if for any t independent distributions $\mathcal{X}_1, \ldots, \mathcal{X}_t$ over $\{0,1\}^n$ with $\sum_{i \in [t]} \mathrm{H}_\infty(\mathcal{X}_i) \geq k$, there is no ε-distinguisher for the distributions $\text{EXT}(\mathcal{X}_1, \ldots, \mathcal{X}_t)$ and $\mathcal{U}_m$.*

3 Generalized Hardcore Set

In this section, we generalize Impagliazzo's hardcore lemma [9] from the case of Boolean functions to the case of general functions. More precisely, we have the following.

Lemma 1. *Let $f : X \to [V]$ be a (δ, s)-hard function, with $\delta \geq 1 - \frac{1}{L}(1-\gamma)$ for some $\gamma \in (0,1)$ and some integer $L \in [V-1]$. Then for any $\varepsilon > 0$, there exist $s' = s/\text{poly}(V, 1/\varepsilon, \log(1/\gamma))$ and $I \in \binom{[V]}{L+1}$ such that f has an (I, ε, s')-hardcore set H_I of density $|H_I|/|X| \geq \gamma / \binom{V}{L+1}$.*

To prepare for the proof of Lemma 1, let us first recall Nisan's proof of Impagliazzo's hardcore lemma (for Boolean functions) described in [9]. The proof is by contradiction, which starts by assuming that a (ρ, s)-hard function $f : X \to I$, with $|I| = 2$, has no hardcore set of density ρ. Then the key step there is to use the min-max theorem of von Neumann to show the existence of a subset of inputs $T \subseteq X$ of density less than ρ and a collection of circuits $A_I \subseteq \mathsf{SIZE}(s')$ with $|A_I| \leq O((1/\varepsilon^2)\log(1/\rho))$ such that for any $x \notin T$,

$$\Pr_{A \in A_I}[A(x) = f(x)] > \frac{1}{2}.$$

Then by letting C be the circuit computing the majority of those circuits in A_I, one has $C(x) = f(x)$ for every $x \notin T$, which contradicts the fact that f is (ρ, s)-hard.

We would like to extend this idea to a general function $f : X \to [V]$, with $V \geq 3$. First, it is straightforward to verify that a similar argument using the min-max theorem can also prove the following lemma.

Lemma 2. *Suppose $f : X \to [V]$ does not have an (I, ε, s')-hardcore set of density ρ in X, for some $I \subseteq [V]$. Then there exist a subset of inputs $T_I \subseteq f^{-1}(I)$ of density less than ρ in X and a collection of circuits $A_I \subseteq \mathsf{SIZE}(s')$ with $|A_I| \leq O((1/\varepsilon^2)\log(1/\rho))$ such that for any $x \in f^{-1}(I) \setminus T_I$,*

$$\Pr_{A \in A_I}[A(x) = f(x)] > \frac{1}{|I|}.$$

However, unlike the Boolean case, it is not clear how to construct a circuit C to approximate f from these collections of circuits. This is because for an input

x with $\Pr_{A \in A_I}[A(x) = f(x)] > \frac{1}{|I|}$, it is still possible that the majority value of $A(x)$, for $A \in A_I$, differs from $f(x)$. Moreover, given an input x, we do not know which subset I contains $f(x)$, so we do not even know which collection A_I of circuits might help. A more careful analysis is needed, and now we proceed to prove Lemma 1.

Proof. Assume for the sake of contradiction that there is no (I, ε, s')-hardcore set of density $\gamma/\binom{V}{L+1}$ in X for any $I \in \binom{[V]}{L+1}$. Then we know from Lemma 2 that for any $I \in \binom{[V]}{L+1}$, there exist a collection $A_I \subseteq \mathsf{SIZE}(s')$ with $|A_I| \leq O((1/\varepsilon^2)\log(\binom{V}{L+1}/\gamma))$ and a subset T_I of inputs with density less than $\gamma/\binom{V}{L+1}$ in X such that for any $x \in f^{-1}(I) \setminus T_I$, $\Pr_{A \in A_I}[A(x) = f(x)] > \frac{1}{L+1}$. Let T be the union of all such T_I's, and we have $\Pr_{x \in X}[x \in T] < \binom{V}{L+1} \cdot \gamma/\binom{V}{L+1} = \gamma$. Note that for any $x \notin T$, we can rule out the value v as a candidate for $f(x)$ if v is contained in some $I \in \binom{[V]}{L+1}$ such that the following condition holds:

$$\Pr_{A \in A_I}[A(x) = v] \leq \frac{1}{L+1}. \tag{1}$$

This suggests the randomized algorithm R described in Figure 1, which tries to compute $f(x)$ by ruling out the candidates one by one.

```
1. Let $Q = [V]$.
2. While $|Q| \geq L + 1$ do the following:
   (a) Choose any $I \subseteq Q$ with $|I| = L + 1$.
   (b) Delete from $Q$ any $v \in I$ such that the condition (1) holds.
3. Output a random element in $Q$.
```

Fig. 1. The randomized algorithm R

Observe that each iteration of the while loop in algorithm R has at least one v deleted from Q, because it is impossible that all the $L + 1$ outcomes have probability more than $\frac{1}{L+1}$. Thus, R exits the while loop after at most V iterations, and for any input $x \notin T$, the value $f(x)$ remains in the final Q, with $|Q| \leq L$, which implies that R outputs $f(x)$ correctly with probability at least $\frac{1}{L}$. By an averaging argument, one can fix the randomness of R to obtain a deterministic circuit C such that $C(x) = f(x)$ for at least $\frac{1}{L}$ fraction of $x \notin T$. As a result, we have

$$\Pr_{x \in X}[C(x) = f(x)] \geq \Pr_{x \in X}[x \notin T] \cdot \Pr_{x \in X}[C(x) = f(x) \mid x \notin T] > (1 - \gamma) \cdot \frac{1}{L} \geq 1 - \delta.$$

On the other hand, the size of the circuit C is at most

$$\mathrm{poly}(V) \cdot O\left((1/\varepsilon^2)\log\left(\binom{V}{L+1}/\gamma\right)\right) \cdot s' \leq s,$$

for some $s' = s/\mathrm{poly}(V, 1/\varepsilon, \log(1/\gamma))$, which contradicts the hardness condition of f. This implies that the assumption of no hardcore set at the beginning is false, which proves Lemma 1. □

4 Density of Hardcore Sets

For the generalized hardcore set lemma in Section 3, one may wonder whether it is possible to guarantee the existence of a much larger hardcore set. In this section, we show that this is basically impossible. Formally, we have the following.

Theorem 1. *For any $\delta = 1 - \frac{1}{L}(1-\gamma)$, with $\gamma \in (0, 1/2(L+1))$ and $L \leq V-1$, there is a (δ, s)-hard function $f : \{0,1\}^n \to [V]$, for some $s \geq \text{poly}(\gamma, 1/L, 2^n)$, such that the following condition holds:*

– *For any $I \in \binom{[V]}{L+1}$ and $\varepsilon < \frac{1}{2L}$, there exists some $s' \leq \text{poly}(n)$ such that f has no (I, ε, s')-hardcore set of density $4(L+1)\gamma / \binom{V}{L+1}$ in $\{0,1\}^n$.*

Note that the theorem says that even for a function which is hard against very large circuits of exponential size, one can only guarantee a hardcore set of a small density against small circuits of polynomial size. However, there is a gap of a $4(L+1)$ factor between the density of a hardcore set ruled out by Theorem 1 and the density achievable by our Lemma 1.

Proof. We show the existence of such a function f by a probabilistic method. Let T denote the first $2(L+1)\gamma$ fraction of the input space $\{0,1\}^n$, and let us divide T into $\binom{V}{L+1}$ disjoint parts of equal size (assuming for simplicity of presentation that T can be divided evenly), denoted by T_I, for $I \in \binom{[V]}{L+1}$. Then we choose the function $f : \{0,1\}^n \to [V]$ randomly in the way such that independently for each input x,

$$f(x) = \begin{cases} \text{a random value in } I, & \text{if } x \in T_I \text{ for some } I \in \binom{[V]}{L+1}; \\ \text{a random value in } [L], & \text{if } x \notin T. \end{cases}$$

We need the following lemma; the proof is by a standard probabilistic argument and is omitted here due to the page limit.

Lemma 3. $\Pr_f$ *[f is not (δ, s)-hard] < 1, for some $s = \text{poly}(\gamma, 1/L, 2^n)$.*

This lemma implies the existence of a function f which is (δ, s)-hard, and let us fix one such f. It remains to show that this f satisfies the condition of the theorem. For any $I \in \binom{[V]}{L+1}$ and any $H \subseteq f^{-1}(I)$ of density $4(L+1)\gamma/\binom{V}{L+1}$ in $\{0,1\}^n$, consider the algorithm A which outputs a random value in $I \cap J$ when $x \in T_J$ for some $J \in \binom{[V]}{L+1}$, and outputs a random value in $[L]$ when $x \notin T$. Then the probability, over $x \in H$ and the randomness of A, that $A(x) = f(x)$ is at least

$$\Pr_{x \in H}[x \in T_I] \cdot \frac{1}{L+1} + \Pr_{x \in H}[x \notin T_I] \cdot \frac{1}{L} = \frac{1}{L+1} + \Pr_{x \in H}[x \notin T_I] \cdot \left(\frac{1}{L} - \frac{1}{L+1}\right)$$

which is at least $\frac{1}{L+1} + \frac{1}{2} \cdot \frac{1}{L(L+1)} > \frac{1+\varepsilon}{L+1}$ for any $\varepsilon < \frac{1}{2L}$. This means that there exists a fixing of the randomness of A to get a deterministic circuit which preserves the above bound. Since we can do this for every I and H, the condition of the theorem is satisfied, which proves the theorem. □

5 Extracting Computational Randomness

In this section, we show that one can extract randomness from the output of a hard function. For this, we first show that the output of a hard function looks somewhat random, even given the input. More precisely, we have the following.

Lemma 4. *Let $f : X \to [V]$ be a (δ, s)-hard function, with $\delta \geq 1 - 2^{-k}$ for some positive $k \in \mathbb{R}$. Let $\mathcal{X}$ be the uniform distribution over X. Then for any $\varepsilon \in (0, 1)$, there exist some $s' \geq s/\mathrm{poly}(V, 1/\varepsilon)$ and a distribution $\mathcal{V}$ (correlated with $\mathcal{X}$) such that the following two conditions hold.*

- *The distributions $(\mathcal{X}, f(\mathcal{X}))$ and $(\mathcal{X}, \mathcal{V})$ have no (ε, s')-distinguisher.*
- $\Pr_{x \in \mathcal{X}} [\mathrm{H}_\infty(\mathcal{V}|\mathcal{X} = x) < \lceil k/3 \rceil] \leq 2^{-k/3}$.

The proof of Lemma 4 is omitted here due to the page limit. The basic idea is that by applying Lemma 1 repeatedly, we can find a collection of disjoint hardcore sets covering a large fraction of inputs. Then by extending the idea in [17], we can show that when a randomly sampled input x falls into one of these hardcore sets, its output looks like a random value from some set.

According to Lemma 4, the output distribution of a hard function looks like one with some min-entropy, given a randomly selected input. This suggests the possibility that one can simply apply any extractor to extract randomness from the output. Formally, we have the following theorem, for the case of seeded extractors. Due to the page limit, the proof is omitted here.

Theorem 2. *Let $f : X \to \{0, 1\}^\ell$ be a (δ, s)-hard function with $\delta \geq 1 - 2^{-k}$, and let $\mathcal{X}$ be the uniform distribution over X. Then for any seeded $(k/3, \varepsilon)$-extractor* $\textsc{Ext} : \{0, 1\}^\ell \times \{0, 1\}^d \to \{0, 1\}^m$ *computable in* $\mathsf{SIZE}(s_0)$*, the distributions $(\mathcal{X}, \textsc{Ext}(f(\mathcal{X}), \mathcal{U}_d))$ and $(\mathcal{X}, \mathcal{U}_m)$ have no $(\bar{\varepsilon}, \bar{s})$-distinguisher, for some $\bar{\varepsilon} \leq 2\varepsilon + 2^{-k/3}$ and $\bar{s} \geq s/\mathrm{poly}(2^\ell, 1/\varepsilon) - s_0$.*

Note that many constructions of seeded extractors are in fact computable by small circuits, of size $s_0 \leq \mathrm{poly}(\ell/\varepsilon)$. Then, for example, when $k \geq \Omega(\ell)$ and $\varepsilon = \mathrm{poly}(2^{-\ell})$, we have $\bar{\varepsilon} \leq 2^{-\Omega(\ell)}$ and $\bar{s} \geq s/2^{O(\ell)}$. This means that as long as s is large enough (or ℓ is small enough), any single-source seeded extractor can be used to extract randomness in the computational setting.

For the case of seedless extractors, we have the following. Due to the page limit, the proof is omitted here.

Theorem 3. *For $i \in [t]$, let $f^{(i)} : X^{(i)} \to \{0, 1\}^\ell$ be a $(\delta^{(i)}, s)$-hard function with $\delta^{(i)} \geq 1 - 2^{-k^{(i)}}$, and let $k = \sum_{i \in [t]} k^{(i)}$. Let $\mathcal{X} = (\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(t)})$, where each $\mathcal{X}^{(i)}$ is an independent uniform distribution over $X^{(i)}$, and let $f(\mathcal{X}) = (f^{(1)}(\mathcal{X}^{(1)}), \ldots, f^{(t)}(\mathcal{X}^{(t)}))$. Then for any seedless t-source $(k/7, \varepsilon)$-extractor* $\textsc{Ext} : (\{0, 1\}^\ell)^t \to \{0, 1\}^m$ *computable in* $\mathsf{SIZE}(s_0)$*, the distributions $(\mathcal{X}, \textsc{Ext}(f(\mathcal{X})))$ and $(\mathcal{X}, \mathcal{U}_m)$ have no $(\bar{\varepsilon}, \bar{s})$-distinguisher, for some $\bar{\varepsilon} \leq (t + 1)\varepsilon + 2^{-\Omega(k^2/t\ell^2)}$ and $\bar{s} \geq s/\mathrm{poly}(2^\ell, 1/\varepsilon) - s_0$.*

Let $\textsc{Ext}$ be the seedless t-source extractor in [11], which is computable by a small circuit, with $s_0 \leq \mathrm{poly}(t\ell/\varepsilon)$. Then, for example, when $t\varepsilon = \mathrm{poly}(2^{-\ell})$

and $k \geq \ell^c$ for a large enough constant c, we have $\bar{\varepsilon} \leq 2^{-\Omega(\ell)}$ and $\bar{s} \geq s/2^{O(\ell)}$. Again, this means that when s is large enough (or ℓ is small enough), any seedless multi-source extractor can also work in the computational setting.

6 Loss of Circuit Size

Recall that in our generalized hardcore set lemma (Lemma 1), there is a loss of circuit size by a factor of poly(V) for functions with V output values. That is, from a (δ, s)-hard function, we can only guarantee the existence of an (I, ε, s')-hardcore set with $s' \leq s/\text{poly}(V)$. In this section, we show that such a loss of circuit size is in fact unavoidable, if the proof is done in a black-box way. Before we can formally state our result, we need to introduce some definitions. Let $F_{X,V}$ denote the collection of functions from X to $[V]$.

Definition 6. *Given a collection $G \subseteq F_{X,V}$, we say that a function $f \in F_{X,V}$ is $(k, \rho, \varepsilon, G)$-easy if for any $I \in \binom{[V]}{k}$ and any $H \subseteq f^{-1}(I)$ of density $|H|/|X| \geq \rho$, there is a function $g \in G$ such that $\Pr_{x \in H}[g(x) = f(x)] \geq \frac{1+\varepsilon}{k}$.*

Next, we define our notion of a black-box proof, which is realized by some oracle algorithm $R^{(\cdot)}$. We allow R to be non-uniform and randomized, and we use the notation $R_r^{G;\alpha}(x)$ to denote that R is given an oracle G, an advice string α, a random string r, and an input x.

Definition 7. *We say that an oracle algorithm $R^{(\cdot)}$ realizes a $(\delta, k, \rho, \varepsilon, S)$ black-box proof of hardcore sets for functions in $F_{X,V}$, if the following holds. For any $f \in F_{X,V}$ and any $G \subseteq F_{X,V}$ with $|G| = S$, if f is $(k, \rho, \varepsilon, G)$-easy, then there exists some advice α such that*

$$\Pr_{x,r}\left[R_r^{G;\alpha}(x) \neq f(x)\right] < \delta.$$

Here we allow R to make adaptive queries, but for simplicity we consider only the case that R on input x queries functions in the oracle at all x. That is, R may first queries $g_i(x)$ for some $g_i \in G$, and depending on the answer, R next queries $g_j(x)$ for some $g_j \in G$, and so on. Note that our proof for the generalized hardcore sets is done in this black-box way, and so do all the known proofs for Impagliazzo's hardcore set lemma. Our result in this section shows that any algorithm realizing such a black-box proof must make many queries to the oracle.

Theorem 4. *Suppose $V \geq \omega(1)$, $0 < \delta \leq 1 - (4 \log V)/V$, $0 < \varepsilon \leq 1/3$, $0 < \rho < 1$, and $S \geq \Omega((V^{k+1}k^3/\varepsilon^2)\log(1/\rho))$. Consider any oracle algorithm which uses an advice of length $\tau \leq o(\delta|X|)$ and realizes a $(\delta, k, \varepsilon, S, \rho)$ black-box proof of hardcore sets for functions in $F_{X,V}$. Then it must make at least $\Omega((Vk/\varepsilon^2)\log(1/\delta))$ oracle queries.*

Note that the theorem says that even if we start from a very hard function, with δ close to one, and even if we only want a hardcore set with $k = 2$ output values, any algorithm realizing such a black-box proof still need to make many queries, which corresponds to a large loss of circuit size. In particular, a loss by a V factor is unavoidable. Now let us prove the theorem.

Proof. Consider any R which realizes such a black-box proof. Assume that R makes at most $q = o((Vk/\varepsilon^2)\log(1/\delta))$ oracle queries, and we will show that this leads to a contradiction. In particular, we will show the existence of a function f and a collection of functions $G = \{g_{I,i} : I \in \binom{[V]}{k}$ and $i \in [T]\}$, for some $T = \Omega((Vk^3/\varepsilon^2)\log(1/\rho))$, such that f is $(k, \rho, \varepsilon, G)$-easy but $\Pr_{x,r}[R_r^{G;\alpha}(x) \neq f(x)] \geq \delta$, for any advice α, which violates the requirement for a black-box proof. We will prove the existence of such f and G by a probabilistic argument.

We choose $f \in F_{X,V}$ randomly such that independently for each input x, $f(x)$ takes a uniformly random value in $[V]$. Then we choose each $g_{I,i}$ in the following way:

– Independently for each input $x \in f^{-1}(I)$, $g_{I,i}(x)$ takes the value $f(x)$ with probability $(1+2\varepsilon)/k$ and each other value in I with probability $(1-2\varepsilon/(k-1))/k$.
– Independently for each input $x \notin f^{-1}(I)$, $g_{I,i}(x)$ takes each value in I with probability $1/k$.

Our key lemma is the following.

Lemma 5. *For any advice α and any input x, we have* $\Pr_{f,G,r}[R_r^{G;\alpha}(x) \neq f(x)] \geq \sqrt{\delta}$.

Due to the page limit, we omit the formal proof of Lemma 5 and only sketch the proof idea here. Consider any advice α and any input x. Recall that in our model, any query made by $R^{G;\alpha}(x)$ is of the form $g_{I,i}(x)$, for some $g_{I,i} \in G$, and the outcome of such a query has a distribution close to the uniform over the k values in I, which is independent of f. When R makes only a small number of queries, we can show that the distribution of the sequence of outcomes corresponding to the queries is still close to a distribution which is independent of f, with the outcome of each query being independent and uniform over some k values. That is, such an R cannot fully exploit the small correlation between f and G, and hence it behaves similarly when the useful oracle G is replaced by a useless one which is independent of f. However, without a useful oracle, R cannot possibly predict a random f well, which implies that even given G, R cannot predict f well either.

Using this lemma together with a Hoeffding bound, one can show that for any advice α,

$$\Pr_{f,G,r}\left[\Pr_x\left[R_r^{G;\alpha}(x) \neq f(x)\right] < \delta\right] \leq 2^{-\Omega(\delta|X|)}.$$

Then, using a union bound, we have

$$\Pr_{f,G,r}\left[\exists\alpha \in \{0,1\}^\tau : \Pr_x\left[R_r^{G;\alpha}(x) \neq f(x)\right] < \delta\right] \leq 2^\tau \cdot 2^{-\Omega(\delta|X|)} \leq o(1), \quad (2)$$

when $\tau \leq o(\delta|X|)$. Next, we need the following; the proof is by a simple probabilistic argument and is omitted here due to the page limit.

Lemma 6. $\Pr_{f,G}[f$ *is not* $(k, \rho, \varepsilon, G)$*-easy*$] \leq o(1)$.

From Lemma 6 and the bound in (2), we can conclude the existence of some f and G such that f is $(k, \rho, \varepsilon, G)$-easy but $\Pr_{f,G,r}[R_r^{G;\alpha}(x) \neq f(x)] \geq \delta$ for any advice α, which contradicts the requirement for a black-box proof of hardcore sets. Therefore, any R realizing such a black-box proof must make at least $\Omega((Vk/\varepsilon^2)\log(1/\delta))$ queries, which proves Theorem 4. $\square$

References

1. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.: The non-stochastic multi-armed bandit problem. SIAM J. Comput. 32(1), 48–77 (2002)
2. Barak, B., Hardt, M., Kale, S.: The Uniform Hardcore Lemma via Approximate Bregman Projectionss. In: SODA 2008, pp. 1193–1200 (2008)
3. Barak, B., Shaltiel, R., Wigderson, A.: Computational analogues of entropy. In: Proc. APPROX-RANDOM, pp. 200–215 (2003)
4. Cover, T., Thomas, J.: Elements of Information Theory. Wiley, Chichester (1991)
5. Goldreich, O., Rubinfeld, R., Sudan, M.: Learning polynomials with queries: the highly noisy case. SIAM J. Disc. Math. 13(4), 535–570 (2000)
6. Healy, A., Vadhan, S., Viola, E.: Using nondeterminism to amplify hardness. SIAM J. Comput. 35(4), 903–931 (2006)
7. Holenstein, T.: Key agreement from weak bit agreement. In: STOC 2005, pp. 664–673 (2005)
8. Hsiao, C.-Y., Lu, C.-J., Reyzin, L.: Conditional computational entropy, or toward separating pseudoentropy from compressibility. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 169–186. Springer, Heidelberg (2007)
9. Impagliazzo, R.: Hard-core distributions for somewhat hard problems. In: FOCS 1995, pp. 538–545 (1995)
10. Klivans, A., Servedio, R.A.: Boosting and hard-core sets. Machine Learning 51(3), 217–238 (2003)
11. Lee, C.-J., Lu, C.-J., Tsai, S.-C.: Deterministic extractors for independent-symbol sources. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 84–95. Springer, Heidelberg (2006)
12. Lee, C.-J., Lu, C.-J., Tsai, S.-C.: Extracting computational entropy and learning noisy linear functions. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 338–347. Springer, Heidelberg (2009)
13. Lu, C.-J., Tsai, S.-C., Wu, H.-L.: On the complexity of hard-core set constructions. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 183–194. Springer, Heidelberg (2007)
14. Nisan, N., Zuckerman, D.: Randomness is linear in space. J. Comput.Syst. Sci. 52(1), 43–52 (1996)
15. O'Donnell, R.: Hardness amplification within NP. In: STOC, pp. 751–760 (2002)
16. Shaltiel, R.: Recent developments in explicit constructions of extractors. Bulletin of the EATCS 77, 67–95 (2002)
17. Sudan, M., Trevisan, L., Vadhan, S.: Pseudorandom generators without the XOR lemma. J. Comput.Syst. Sci. 62(2), 236–266 (2001)
18. Trevisan, L.: List decoding using the XOR lemma. In: FOCS, pp. 126–135 (2003)
19. Trevisan, L.: On uniform amplification of hardness in NP. In: STOC, pp. 31–38 (2005)
20. Yao, A.: Theory and applications of trapdoor functions. In: FOCS 1982, pp. 80–91 (1982)
21. Zuckerman, D.: General weak random sources. In: FOCS, pp. 534–543 (1990)

Data Reduction for Graph Coloring Problems[⋆]

Bart M.P. Jansen and Stefan Kratsch

Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{bart,kratsch}@cs.uu.nl

Abstract. This paper studies the kernelization complexity of graph coloring problems, with respect to certain structural parameterizations of the input instances. We are interested in how well polynomial-time data reduction can provably shrink instances of coloring problems, in terms of the chosen parameter. It is well known that deciding 3-colorability is already NP-complete, hence parameterizing by the requested number of colors is not fruitful. Instead, we pick up on a research thread initiated by Cai (DAM, 2003) who studied coloring problems parameterized by the modification distance of the input graph to a graph class on which coloring is polynomial-time solvable; for example parameterizing by the number k of vertex-deletions needed to make the graph chordal. We obtain various upper and lower bounds for kernels of such parameterizations of q-Coloring, complementing Cai's study of the time complexity with respect to these parameters.

1 Introduction

Graph coloring is one of the most well-studied and well-known topics in graph algorithmics and discrete mathematics; it hardly needs an introduction. In this work we study the kernelization complexity of graph coloring problems, or in other words the existence of efficient and provably effective preprocessing procedures, using the framework of parameterized complexity [10,15] (Section 2 contains definitions). Parameterized complexity enables us to study qualitatively and quantitatively how different properties of a graph coloring instance contribute to its difficulty.

The choice of parameter is therefore very important. If we consider the vertex coloring problem and parameterize by the requested number of colors, then this problem is already NP-complete for a constant value of 3 for the parameter, resulting in intractability; we should consider different parameterizations to obtain meaningful questions. In his study of the parameterized complexity of vertex coloring problems, Leizhen Cai [7] introduced a convenient notation to talk about *structural* parameterizations of graph problems. For a graph class $\mathcal{F}$ let $\mathcal{F} + kv$ denote the graphs which can be built by adding at most k vertices to a graph in G; the neighborhoods of these new vertices can be arbitrary. Equivalently the class $\mathcal{F} + kv$ contains those graphs which contain a *modulator* $X \subseteq V(G)$ of

⋆ This work was supported by the Netherlands Organization for Scientific Research (NWO), project "KERNELS: Combinatorial Analysis of Data Reduction".

size at most k such that $G - X \in \mathcal{F}$. Hence FOREST + kv is exactly the class of graphs which have a feedback vertex set of size at most k. Similarly one may define classes $\mathcal{F} + ke$ and $\mathcal{F} - ke$ where the structure is measured through the number of *edges* which were added or removed from a member of $\mathcal{F}$ to build the graph. Using this notation we can define a class of parameterized coloring problems with structural parameters.

q-COLORING ON $\mathcal{F} + kv$ GRAPHS
Input: An undirected graph G and a modulator $X \subseteq V(G)$ such that $G - X \in \mathcal{F}$.
Parameter: The size $k := |X|$ of the modulator.
Question: Is $\chi(G) \leq q$?

To decouple the existence of polynomial kernelizations from the difficulties of *finding* a modulator, we assume that a modulator is given in the input. The CHROMATIC NUMBER on $\mathcal{F} + kv$ graphs problem is defined similarly as q-COLORING, with the important exception that the value q is not fixed, but part of the input. For the purposes of kernelization, however, there is little left to explore for CHROMATIC NUMBER: a superset of the authors showed [2, Theorem 14] that CHROMATIC NUMBER does not admit a polynomial kernel when parameterized by the vertex cover number, or in Cai's notation: CHROMATIC NUMBER on INDEPENDENT + kv graphs does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses to the third level [23] (PH = Σ_3^p). The proof given in that paper shows that even a compound parameterization by the vertex cover number *plus* the number of colors that is asked for, does not admit a polynomial kernel. Hence it seems that the size of the kernel must depend superpolynomially on the number of colors. In this work we therefore focus on q-COLORING and consider how the structural parameterizations influence the complexity of the problem when keeping the number of colors q *fixed*.

When studying coloring problems with these structural parameterizations, we feel it is important to look at the relations between the parameters and consider the parameter space as a *hierarchy* (Fig. 1), rather than exploring a table of problems versus parameters one row at a time (cf. [12]). It is known that there are several coloring problems such as PRECOLORING EXTENSION and EQUITABLE COLORING which are W[1]-hard when parameterized by treewidth, but fixed-parameter tractable parameterized by the vertex cover number [13,11]. These parameters also yield differences in the *kernelization complexity* of q-COLORING. Our hierarchy includes these parameters, and several others which are sandwiched between them.

Our results. In this paper we pinpoint the boundary for polynomial kernelizability of q-COLORING in the given hierarchy, by exhibiting upper- and lower bounds for kernel sizes. For all parameters in Fig. 1 for which q-COLORING is in FPT we either give a polynomial kernel, or prove that the existence of such a kernel would imply NP $\subseteq$ coNP/poly and is therefore unlikely.

Upper bounds in the hierarchy. We derive a general theorem which associates the existence of polynomial kernels for q-COLORING on $\mathcal{F} + kv$ graphs to

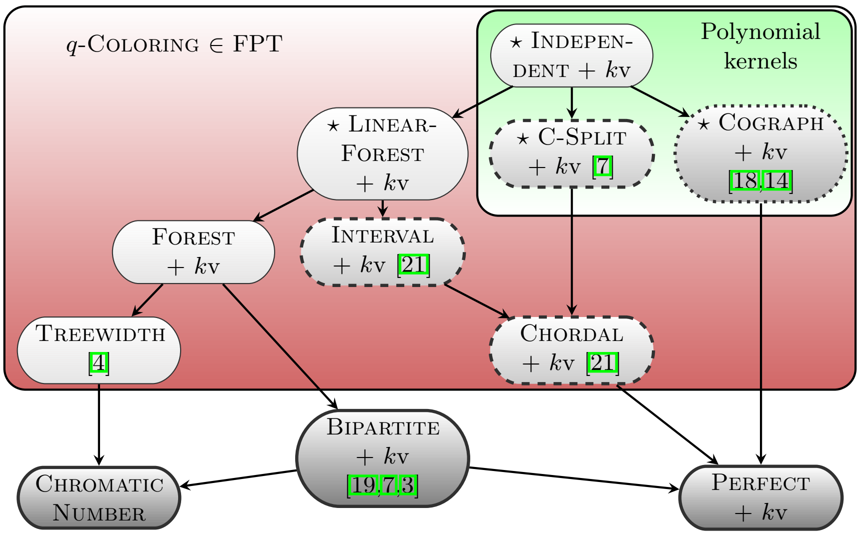


Fig. 1. The hierarchy of parameters used in this work. Arrows point from larger parameters to smaller parameters: an arc $P \rightarrow P'$ signifies that every graph G satisfies $P(G) + 2 \geq P'(G)$. For parameters with a star we obtain new parameterized complexity results in this work. The complexity of the Chromatic Number problem for a given parameterization is expressed through the shading and border style of the parameters: the complexity status can be NP-complete for fixed k ▬, or contained in XP but not known to be W[1]-hard ⬚, or contained in XP and W[1]-hard ⬚, or FPT but without polynomial kernel unless NP $\subseteq$ coNP/poly ▭ . The complexity of q-Coloring for a given parameterization is expressed through the containers: the status is either FPT with a polynomial kernel, FPT but no polynomial kernel unless NP $\subseteq$ coNP/poly, or NP-complete for fixed k.

properties of the q-List Coloring problem on graphs in $\mathcal{F}$: if the non-list-colorability of a graph in $\mathcal{F}$ is "local" in the sense that for any no-instance there is a small subinstance on $f(q)$ vertices to which the answer is no, then q-Coloring on $\mathcal{F} + kv$ graphs admits a polynomial kernel for every fixed q. We then apply this general theorem to give polynomial kernels for Cograph $+ kv$ and C-Split $+ kv$.

Lower bounds in the hierarchy. In the seminal paper on kernelization lower-bounds, Bodlaender et al. [1, Theorem 2] prove that 3-Coloring parameterized by treewidth does not admit a polynomial kernel unless all coNP-complete problems have distillation algorithms. We strengthen their result by showing that unless NP $\subseteq$ coNP/poly (an even less likely condition), the problem does not even admit a polynomial kernel parameterized by vertex-deletion distance to a single path: 3-Coloring on Path $+ kv$ graphs does not admit a polynomial kernel. Under the same assumption, this immediately excludes polynomial kernels on e.g. Forest $+ kv$ or Interval $+ kv$ graphs, since the latter are *smaller* parameters.

We also investigate the *degree* of the polynomial in the kernels that we obtain for q-Coloring parameterized by vertex cover. Our general scheme yields kernels with $k + k^q$ vertices, and a small insight allows us to encode these instances in $\mathcal{O}(k^q)$ bits. Using a connection to Not-All-Equal q-Satisfiability (q-nae-sat) we prove that for every $q \geq 4$ the q-Coloring problem parameterized by vertex cover does not admit a kernel which can be encoded in $\mathcal{O}(k^{q-1-\epsilon})$ bits for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly.

Domination-related parameters. It turns out that the difficulty of a 3-Coloring instance is intimately related to the domination-properties of the graph. We show the surprising (but not difficult) result that 3-Coloring on a general graph G can be solved in $\mathcal{O}^*(3^k)$ time when a dominating set X of size k is given along with the input. In contrast we show that 3-Coloring parameterized by the size of a dominating set does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. To obtain polynomial kernels by exploiting the domination structure of the graph, we must consider another parameter. Let Dominated be the graphs where each connected component has a dominating vertex. 3-Coloring on Dominated$+k$v graphs admits a polynomial kernel. This cannot be extended to arbitrary q, since 4-Coloring is NP-complete on Dominated graphs.

Related work. Structural parameterizations of graph coloring problems were first studied by Cai [7], and later by Marx [21]. An overview of their results relevant to this work can be found in Fig. 1. Chor, Fellows, and Juedes [8] considered the problem of coloring a graph on n vertices with $n - k$ colors and obtained an FPT algorithm. They also found a polynomial kernel for a related problem, which can be seen to imply that Chromatic Number on Complete$+$ kv has a linear-vertex kernel. Finally we observe that the q-Coloring problem on Interval$+k$v and Chordal$+k$v graphs is in FPT since yes-instances have treewidth $\mathcal{O}(k + q)$.

2 Preliminaries

Parameterized complexity and kernels. A parameterized problem Q is a subset of $\Sigma^* \times \mathbb{N}$, the second component being the *parameter* which expresses some structural measure of the input. A parameterized problem is (strongly uniform) *fixed-parameter tractable* if there exists an algorithm to decide whether $(x, k) \in Q$ in time $f(k)|x|^{\mathcal{O}(1)}$ where f is a computable function.

A *kernelization algorithm* (or *kernel*) for a parameterized problem Q is a polynomial-time algorithm which transforms an instance (x, k) into an equivalent instance (x', k') such that $|x'|, k' \leq f(k)$ for some computable function f, which is the *size* of the kernel. If $f \in k^{\mathcal{O}(1)}$ is a polynomial then this is a *polynomial kernel.* Intuitively a polynomial kernel for a parameterized problem is a polynomial-time preprocessing procedure which reduces the size of an instance to something which *only* depends (polynomially) on the parameter k, and does not depend at all on the input size $|x|$.

Graphs. All graphs we consider are finite, undirected and simple. We use $V(G)$ and $E(G)$ to denote the vertex- and edge set of a graph G. For $X \subseteq V(G)$ the subgraph induced by X is denoted by $G[X]$. The terms P_n, C_n and K_n denote the path, cycle, and complete graphs on n vertices, respectively. For natural numbers q we define $[q] := \{1, 2, \ldots, q\}$. A proper q-coloring of a graph G is a function $f : V(G) \to [q]$ such that adjacent vertices receive different colors. The *chromatic number* $\chi(G)$ of a graph is the smallest number q for which it has a proper q-coloring. For a vertex set $X \subseteq V(G)$ we denote by $G - X$ the graph obtained from G by deleting all vertices of X and their incident edges.

A graph G with $E(G) = \emptyset$ is called an *independent* graph. A graph is a *split graph* if there is a partition of $V(G)$ into sets X, Y such that X is a clique and Y is an independent set. We define the class C-Split of *component-wise split graphs* containing all graphs for which each connected component is a split graph. A graph is a *cograph* if it does not contain P_4 as an induced subgraph. A *linear forest* is a disjoint union of paths. The book by Brandstädt, Le, and Spinrad [6] contains more information about the graph classes used in this work. For a finite set X and non-negative integer i we write $\binom{X}{i}$ for the collection of all size-i subsets of X, $\binom{X}{\leq i}$ for all size *at most* i subsets of X, and X^i for the cartesian product of i copies of X. Proofs of statements marked with a star ★ can be found in the full version of this work [16].

3 Positive Results in the Hierarchy

Our positive results are obtained by a general theorem which connects the existence of polynomial kernelizations for q-Coloring on $\mathcal{F} + kv$ graphs with the existence of small induced certificates for no instances of q-List Coloring on graphs from $\mathcal{F}$. We introduce some terminology to state the theorem and proof precisely.

> q-List-Coloring
> **Input:** An undirected graph G and for each vertex $v \in V(G)$ a list $L(v) \subseteq [q]$ of allowed colors.
> **Question:** Is there a proper q-coloring $f : V(G) \to [q]$ such that $f(v) \in L(v)$ for each $v \in V(G)$?

An instance (G', L') of q-List Coloring is a *subinstance* of (G, L) if G' is an induced subgraph of G and $L'(v) = L(v)$ for all $v \in V(G')$. If (G', L') is a no-instance we say it is a no-subinstance.

Theorem 1. *Let $\mathcal{F}$ be a hereditary class of graphs for which there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for any* no*-instance (G, L) of q-*List Coloring *on a graph $G \in \mathcal{F}$, there is a* no*-subinstance (G', L') on at most $|V(G')| \leq f(q)$ vertices. The q-*Coloring *problem on $\mathcal{F} + kv$ graphs admits a polynomial kernel with $\mathcal{O}(k^{q \cdot f(q)})$ vertices for every fixed q.*

Proof. Consider an instance (G, X) of q-Coloring on a graph class $\mathcal{F} + kv$ which satisfies the stated requirements. We give the outline of the reduction algorithm.

```
1. For each undirected graph $H$ on $t \leq f(q)$ vertices $\{h_1, \ldots, h_t\}$, do:
     For each tuple $(S_1, \ldots, S_t) \in \binom{X}{\leq q}^t$, do:
       If there is an induced subgraph of $G - X$ on vertices $\{v_1, \ldots, v_t\}$
          which is isomorphic to $H$ by the mapping $h_i \mapsto v_i$, and $S_i \subseteq$
          $N_G(v_i)$ for $i \in [t]$, then mark the vertices $\{v_1, \ldots, v_t\}$ as impor-
          tant for one such subgraph, which can be chosen arbitrarily.
2. Let $Y$ contain all vertices of $G - X$ which were marked as important, and
   output the instance $(G', X)$ with $G' := G[X \cup Y]$.
```

Let us verify that this procedure can be executed in polynomial time for fixed q, and leads to a reduced instance of the correct size. The number of undirected graphs on $f(q)$ vertices is constant for fixed q. The number of considered tuples is bounded by $\mathcal{O}((q|X|^q)^{f(q)})$, and for each graph H, for each tuple, we mark at most $f(q)$ vertices. These observations imply that the algorithm outputs an instance of the appropriate size, and that it can be made to run in polynomial time for fixed q beacause we can just try all possible isomorphisms by brute-force. It remains to prove that the two instances are equivalent: $\chi(G) \leq q \Leftrightarrow \chi(G') \leq q$. The forward direction of this equivalence is trivial, since G' is a subgraph of G. We now prove the reverse direction.

Assume that $\chi(G') \leq q$ and let $f' : V(G') \to [q]$ be a proper q-coloring of G'. Obtain a partial q-coloring f of G by copying the coloring of f' on the vertices of X. Since $G'[X] = G[X]$ the function f is a proper partial q-coloring of G, which assigns all vertices of X a color. We will prove that f can be extended to a proper q-coloring of G, using an argument about list coloring. Consider the graph $H := G - X$ which contains exactly the vertices of G which are not yet colored by f. For each vertex $v \in V(H)$ define a list of allowed colors as $L(v) := [q] \setminus \{f(u) \mid u \in N_G(v)\}$, i.e. for every vertex we allow the colors which are not yet used on a colored neighbor in G. From this construction it is easy to see that any proper q-list-coloring of the instance (H, L) gives a valid way to augment f to a proper q-coloring of all vertices of G. Hence it remains to prove that (H, L) is a YES-instance of q-LIST COLORING.

Assume for a contradiction that (H, L) is a NO-instance. Since the problem definition ensures that $H = G - X \in \mathcal{F}$ the assumptions on $\mathcal{F}$ imply there is a NO-subinstance (H', L') on $t \leq f(q)$ vertices.

Let the vertices of H' be $h_1, \ldots, h_t$. Since (H', L') is a subinstance of (H, L) we know by construction of the latter that for every vertex h_i with $i \in [t]$ and for every color $j \in [q] \setminus L'(h_i)$ there is a vertex of $N_G(h_i)$ which is colored with j. Now choose sets $S_1, \ldots, S_t$ such that S_i contains for every $j \in [q] \setminus L'(h_i)$ exactly one neighbor $v \in N_G(h_i)$ with $f(v) = j$, which is be possible by the previous observation. Since f only colors vertices from X we have $S_i \subseteq X$ for all $i \in [t]$.

Because H' is an induced subgraph on at most $f(q)$ vertices of $H = G - X$, we must have considered graph H' during the outer loop of the reduction algorithm. Since each S_i contains at most q vertices from X, we must have considered the tuple $(S_1, \ldots, S_t)$ during the inner loop of the reduction algorithm, and because the existence of H' shows that there is at least one induced subgraph of $G - X$ which satisfies the if-condition, we must have marked some vertices $\{v_1, \ldots, v_t\}$ of an induced subgraph H^* of $G - X$ isomorphic to H' by some isomorphism $v_i \mapsto h_i$ as *important*, and hence these vertices exist in the graph G'. Recall that f' is a proper q-coloring of G', and that f and f' assign the same colors to vertices of X. By construction this shows that for each vertex h_i with $i \in [t]$ of the presumed NO-subinstance (H', L') of q-List Coloring, for each color $j \in [q] \setminus L'(h_i)$ which is not on the list of h_i, there is a neighbor of the corresponding vertex v_i (i.e. a vertex in $N_G(v_i)$) which is colored j. Using the fact that H^* is isomorphic to H' we find that we obtain a valid q-list-coloring of H' by using the colors assigned to H^* by f. But this shows that (H', L') is in fact a YES-instance of q-List Coloring, which contradicts our initial assumption. This proves that the instance (H, L) of q-List Coloring that we created must be a YES-instance, and by our earlier observations this implies that $\chi(G) \leq q$, which concludes the proof of the equivalence of the input- and output instance.

Hence we have shown that for each fixed q there is a polynomial-time algorithm which transforms an input of q-Coloring on $\mathcal{F} + kv$ graphs into an equivalent instance of bounded size, which concludes the proof. □

Next we will show how to apply Theorem 1 to obtain polynomial kernels for various structural parameterizations of q-Coloring. By noting that a NO-instance of q-List Coloring on an independent graph has an induced NO-subinstance on a single vertex, the proof of Theorem 1 gives the following corollary.

Corollary 1. *q-Coloring on Independent $+ k$v graphs (i.e. parameterized by vertex cover) admits a polynomial kernel with $k + k^q$ vertices for every fixed integer q.*

Unlike most kernels with $\mathcal{O}(k^c)$ vertices (which often require $\Omega(k^{c+1})$ bits to represent), we can prove that a kernel for q-Coloring on Independent $+ k$v graphs exists which can be encoded in $\mathcal{O}(k^q)$ bits.

Lemma 1 (★). *For every fixed $q \geq 3$, q-Coloring on Independent $+ k$v graphs (i.e. parameterized by vertex cover) admits a kernel which can be encoded in $\mathcal{O}(k^q)$ bits.*

We now move to more general graph classes and consider split graphs.

Theorem 2. *Component-wise split graphs satisfy the conditions of Theorem 1 and therefore q-Coloring on C-Split $+ k$v graphs admits a polynomial kernel for every fixed q.*

Proof. We prove that component-wise split graphs satisfy the conditions of Theorem 1 from which the theorem follows. It is well-known that split graphs are hereditary, and therefore this holds for the class C-Split as well. Consider a NO-instance of q-List Coloring (G, L) with $G \in$ C-Split. We may assume without loss of generality that G is connected, because a graph is list-colorable if and only if each connected component is list-colorable; hence in any disconnected NO-instance there is a connected NO-subinstance. Let X, Y be a partition of $V(G)$ such that X is a clique and Y is an independent set. If $|X| > q$ then for any subset $X' \subseteq X$ of size $q+1$ we know that $G[X']$ is not q-colorable, so in particular $G[X']$ is not q-list-colorable which proves the existence of a NO-subinstance on $q+1$ vertices. Now consider the more interesting case that $|X| \leq q$. We prove the existence of a small NO-subinstance by exploiting the structure of a simple algorithm to decide q-List Coloring for split graphs.

So consider the following algorithm. For each vertex $v \in X$, try all possible ways of assigning a color of $L(v)$ to v. Over all vertices of X there are at most $|X|^q \leq q^q$ ways to do this. Now observe that since Y is an independent set, each vertex in Y only has neighbors in X. For each possible assignment of colors to X that is proper, test for each vertex in Y if there is a color on its list which is not yet taken by a neighbor in X. If we can find such an unused color for each $u \in Y$ then we found a valid list coloring and the algorithm outputs YES; if all assignments to X yield a vertex in Y on which the coloring cannot be extended, the algorithm outputs NO.

It is easy to see that this algorithm is correct. Now consider running this algorithm on (G, L). By the assumption that (G, L) is a NO-instance, for each assignment of colors to X there must be a vertex $u \in Y$ for which all available colors are already used on a neighbor. Now remember one such vertex u for each possible color assignment to X, and let Z be the set of at most q^q remembered vertices. Now consider the subinstance on the graph $G[X \cup Z]$. It is easy to see that when we would execute the proposed algorithm, for each failed attempt at list coloring the graph, we remembered a witness which ensures that this attempt also fails on $G[X \cup Z]$. Hence the algorithm outputs NO, and since it is a correct algorithm this shows that the subinstance on graph $G[X \cup Z]$ on at most $|X| + q^q \leq q + q^q$ vertices is also NO; this shows that the class of componentwise split graphs satisfies the required condition with $f(q) := q + q^q$. □

The following theorem can be proven by a similar type of argument, finding small NO-subinstances using the existence of a dynamic programming algorithm on a cotree decomposition to solve q-List Coloring on cographs.

Theorem 3 (★). *Cographs satisfy the conditions of Theorem 1 and therefore q-*Coloring *on* Cograph + kv *graphs admits a polynomial kernel for every fixed q.*

4 Negative Results in the Hierarchy

Our main negative result is that 3-Coloring on Path + kv graphs does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. This fact nicely complements Theorem 1 since paths are arguably the simplest graphs where there are no $f(q)$ NO-subinstances of q-List Coloring. We first prove a slightly weaker lower bound. The reduction in the proof of the following theorem is inspired by a reduction of Lokshtanov et al. [20, Theorem 6.1]. We learned that Stefan Szeider independently found a similar result for Forest + kv graphs.

Theorem 4. 3-Coloring *on* LinearForest + kv *graphs does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.*

Proof. We give a polynomial-parameter transformation [5, Definition 3] from CNF-SAT parameterized by the number of variables n to 3-COLORING parameterized by deletion distance from a linear forest. Consider an input to CNF-SAT which consists of clauses $C_1, \ldots, C_m$ where each clause is a disjunction of literals of the form x_i or $\overline{x_i}$ for $i \in [n]$. We build a graph G and a modulator $X \subseteq V(G)$ such that $|X| = 2n + 3$ and $G - X \in$ LINEARFOREST.

Construct a clique on three vertices p_1, p_2, p_3; this clique will serve as our palette of three colors, since in any proper 3-coloring all three colors must occur on this clique. For each variable x_i for $i \in [n]$ we make vertices T_i and F_i and add the edge $\{T_i, F_i\}$ to G. We make the vertices T_i, F_i adjacent to the palette vertex p_1. Now we create gadgets for the clauses of the satisfiability instance.

For each clause C_j with $j \in [m]$, let n_j be the number of literals in C_j and create a path $(a_j^1, b_j^1, a_j^2, b_j^2, \ldots, a_j^{n_j}, b_j^{n_j})$ on $2n_j$ vertices. We call this the clause-path for C_j. Make the first and last vertices on the path a_j^1 and $b_j^{n_j}$ adjacent to the palette vertex p_1. Make the b-vertices $b_j^1, b_j^2, \ldots, b_j^{n_j}$ adjacent to palette vertex p_3. As the last step we connect the vertices on the path to the vertices corresponding to literals. For $r \in [n_j]$ if the r-th literal of C_j is x_i (resp. $\overline{x_i}$) then make vertex a_j^r adjacent to T_i (resp. F_i). This concludes the construction of the graph G. We use the modulator $X := \{T_i, F_i \mid i \in [n]\} \cup \{p_1, p_2, p_3\}$. It is easy to verify that $|X| = 2n + 3$ and therefore that the parameter of the 3-COLORING instance is polynomial in the parameter of CNF-SAT. Since vertices on a clause-path are not adjacent to other clause-paths, it follows that $G - X$ is a linear forest. It remains to prove that the two instances are equivalent. Using the fact that any proper 3-coloring of G must color at least one a-vertex on each clause-path with the same color as p_3, it is not hard to prove that the truth assignment which makes all literals true whose vertex has been colored with the color of p_2 is a satisfying assignment. We defer the remainder of the proof to full version [16] due to space restrictions.

Since the construction can be carried out in polynomial time and guarantees that the parameter of the output instance is bounded polynomially in the parameter of the input instance, the given reduction is indeed a polynomial-parameter transformation. The theorem now follows in a well-known manner from the fact that CNF-SAT parameterized by the number of variables does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [9] by applying the framework for kernelization lower-bounds of Bodlaender et al. [1,5]. □

By considering the proof of Theorem 4 we can obtain a corollary for a stronger parameterization. Consider the graph G and modulator X which is constructed in the proof: the remainder graph $G - X$ is a linear forest, a disjoint union of paths. By adding vertices of degree two to G we may connect all the paths in $G - X$ into a single path. Since degree-2 vertices do not affect the 3-colorability of a graph, this does not change the answer to the instance and ensures that $G - X$ is a single path. Hence we obtain:

Corollary 2. 3-COLORING *on* PATH $+ k$v *graphs does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.*

Using recent machinery developed by Dell and van Melkebeek [9] we can also prove lower bounds on the *coefficient* of the polynomial in the kernel size of q-Coloring on Independent + kv graphs for $q \geq 4$. Recall that an instance of q-nae-sat consists of a CNF formula with at most q literals per clause, which is satisfied if at least one literal in each clause evaluates to false, and at least one evaluates to true. By relating q-cnf-sat to $(q+1)$-nae-sat (both parameterized by the number of variables) through a reduction due to Knuth [17, Section 6] we obtain a compression lower bound for $(q+1)$-nae-sat, and by relating the latter to $(q+1)$-Coloring on Independent + kv graphs we can obtain the following theorem.

Theorem 5 (★). *For every $q \geq 4$, q-*Coloring *on* Independent + kv *graphs does not have a kernel of bitsize $\mathcal{O}(k^{q-1-\epsilon})$ for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly.*

The proof of Theorem 5 shows that an improved compression lower bound for q-nae-sat also yields a better lower bound for q-Coloring on Independent + kv graphs. In particular, if it would be proven that for $q \geq 3$ q-nae-sat on n variables cannot be compressed in polynomial time into an equivalent instance on $\mathcal{O}(n^{q-\epsilon})$ bits, then the kernel of Lemma 1 is optimal up to $k^{o(1)}$ factors.

5 Domination-Related Parameters

In this section we show that the complexity of 3-Coloring is strongly related to the domination-structure of the graph.

Theorem 6. 3-Coloring *on a general graph G can be solved in $\mathcal{O}^*(3^k)$ time when given a dominating set X of size k.*

Proof. Let X be a dominating set in graph G of size k. The algorithm proceeds as follows. For each of the 3^k possible assignments of colors to X, we check in linear time whether adjacent vertices received the same color. If the partial coloring is proper then we determine whether it can be extended to the remainder of the graph, and this check can be modeled as a 3-List Coloring instance on the graph $G - X$: for every vertex $v \in G - X$ the list of available colors is formed by those elements of $\{1, 2, 3\}$ which do not occur on neighbors in X. Since X is a dominating set, every vertex has at least one colored neighbor and therefore each vertex of $G - X$ has a list of at most two available colors. It has long been known that such 3-List Coloring instances can be solved in polynomial time by guessing a color for a vertex and propagating the implications; see for example the survey by Tuza [22, Section 4.3]. Hence for each assignment of colors to X we can test in polynomial time whether it can be extended to $G - X$ or not, and G is 3-colorable if and only if at least one of these attempts succeeds. □

The fixed-parameter tractability of 3-Coloring parameterized by the size of a given dominating set raises the question whether the problem admits a polynomial kernel. Assuming NP $\nsubseteq$ coNP/poly this is not the case, which can be seen from the proof of Theorem 4: the modulator X which is constructed in the proof is a dominating set in the graph, and therefore the given reduction serves as a polynomial-parameter transformation from CNF-SAT parameterized by n to 3-Coloring parameterized by the size of the dominating set X.

Having a restricted domination-structure in a 3-Coloring instance *does* make it more amenable to kernelization, which becomes clear when we use a different parameter. Recall that the class Dominated contains those graphs in which each connected component has a dominating vertex.

Theorem 7 (★). 3-Coloring *on* Dominated $+ k$v *graphs admits a polynomial kernel.*

6 Conclusions

We studied the kernelizability of q-Coloring within a hierarchy of structural parameterizations of the graph, obtaining several positive and negative results. It is interesting to note that in the parameter space we consider, the positive results obtained through Theorem 1 even hold for q-List Coloring on $\mathcal{F} + kv$ graphs. We can obtain a kernel for q-List Coloring by transforming a list-coloring instance into a q-Coloring instance by adding a clique on q vertices to the modulator and using adjacencies to this clique to enforce the color lists, increasing the parameter by the constant q. The resulting q-Coloring instance can then be reduced using Theorem 1.

The parameter hierarchy we considered uses vertex-deletion distance to well-studied graph classes as the parameter. The question of kernelizability can also been asked for the edge-deletion and edge-completion variants of these parameters [7,21] which will result in quite a different boundary between tractable and intractable: it is not hard to see that q-Coloring on LinearForest$\pm k$e graphs admits a polynomial kernel by deleting vertices of degree at most two, whereas Theorem 4 shows that this is not the case on LinearForest $+ k$v graphs.

As a final open question it will be interesting to settle the gap between the kernelization upper- and lower bounds of q-Coloring on Independent $+ k$v graphs.

Acknowledgments. We are grateful to an anonymous referee of STACS 2011 [2] who suggested to investigate Dell and van Melkebeek-type lower bounds for q-Coloring parameterized by vertex cover.

References

1. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
2. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Cross-composition: A new technique for kernelization lower bounds. In: Proc. 28th STACS, pp. 165–176 (2011)
3. Bodlaender, H.L., Jansen, K., Woeginger, G.J.: Scheduling with incompatible jobs. Discrete Applied Mathematics 55(3), 219–232 (1994)

4. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. Comput. J. 51(3), 255–269 (2008)
5. Bodlaender, H.L., Thomassé, S., Yeo, A.: Kernel bounds for disjoint cycles and disjoint paths. In: Proc. 17th ESA, pp. 635–646 (2009)
6. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: a survey. Society for Industrial and Applied Mathematics, Philadelphia (1999)
7. Cai, L.: Parameterized complexity of vertex colouring. Discrete Applied Mathematics 127(3), 415–429 (2003)
8. Chor, B., Fellows, M., Juedes, D.W.: Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 257–269. Springer, Heidelberg (2004)
9. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: Proc. 42nd STOC, pp. 251–260 (2010)
10. Downey, R., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
11. Fellows, M.R., Fomin, F.V., Lokshtanov, D., Rosamond, F.A., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. Inf. Comput. 209(2), 143–153 (2011)
12. Fellows, M.R., Lokshtanov, D., Misra, N., Mnich, M., Rosamond, F.A., Saurabh, S.: The complexity ecology of parameters: An illustration using bounded max leaf number. Theory Comput. Syst. 45(4), 822–848 (2009)
13. Fiala, J., Golovach, P.A., Kratochvíl, J.: Parameterized complexity of coloring problems: Treewidth versus vertex cover. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 221–230. Springer, Heidelberg (2009)
14. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Intractability of clique-width parameterizations. SIAM J. Comput. 39(5), 1941–1956 (2010)
15. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
16. Jansen, B.M.P., Kratsch, S.: Data reduction for graph coloring problems. CoRR, abs/1104.4229 (2011)
17. Knuth, D.E.: Axioms and hulls. Springer, Heidelberg (1992)
18. Kobler, D., Rotics, U.: Edge dominating set and colorings on graphs with fixed clique-width. Discrete Applied Mathematics 126(2-3), 197–221 (2003)
19. Kratochvíl, J.: Precoloring extension with fixed color bound. Acta Mathematica Universitatis Comenianae 62(2), 139–153 (1993)
20. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: Proc. 22nd SODA, pp. 777–789 (2011)
21. Marx, D.: Parameterized coloring problems on chordal graphs. Theoretical Computer Science 351(3), 407–424 (2006)
22. Tuza, Z.: Graph colorings with local constraints - a survey. Math. Graph Theory 17, 161–228 (1997)
23. Yap, C.-K.: Some consequences of non-uniform conditions on uniform classes. Theor. Comput. Sci. 26, 287–300 (1983)

Hunting Distributed Malware with the κ-Calculus

Mila Dalla Preda[1] and Cinzia Di Giusto[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
[2] INRIA Rhône Alpes, Grenoble, France

Abstract. The defense of computer systems from malicious software attacks, such as viruses and worms, is a key aspect of computer security. The analogy between malicious software and biological infections suggested us to use the κ-calculus, a formalism originally developed for the analysis of biological systems, for the formalization and analysis of malicious software. By modeling the different actors involved in a malicious code attack in the κ-calculus and by simulating their behavior, it is possible to extract important information that can drive in the choice of the defense technique to apply.

1 Introduction

The Challenge. According to Grimes, "Malicious code is any software program designed to move from computer to computer and network to network in order to intentionally modify computer systems without the consent of the owner" [16]. Malicious codes are classified according to their key features: the ability to propagate and the potential to perform a damage (or *payload*) [21]. The most popular classes of malicious code include viruses, worms, Trojan horses, spyware, trap doors and logic bombs. In general, the term *malware* refers to a malicious code regardless of its classification.

The enlarging size and complexity of modern information systems, together with the growing connectivity of computers through the Internet, have facilitated the spread of malware [21]. The past thirty years have seen a continuous growth of the threats of malware both in the danger and in the complexity of the malicious codes. Nowadays, one of the most sophisticated techniques used by hackers is to exploit the resources of some victim machines in order to create and control powerful networks of compromised nodes called *botnets*. A botnet consists of a collection of victim machines running specific programs, called bots, that are remotely controlled by the attacker, called botmaster [20]. Typical applications of botnets are the implementation of distributed denial of service attacks and of e-mail spamming. Botnets represent a dangerous, potent and quick evolving threat that is yet to be fully understood. Their potential resides in distributing the maliciousness over the network. This makes detection much harder and poses a global threat that can count on the cooperation of many machines.

Since the threat of malware attacks is an unavoidable problem in computer security, it is crucial to develop both sophisticated models for expressing and understanding distributed malware, i.e. botnets, and efficient techniques of defense from them. A standard defense technique is misuse malware detection which consists in detecting malicious code by scanning for predefined sequences of bytes that act like fingerprints. A novel promising approach regards the monitoring of malware propagation in a network both for studying properties of the considered malware and/or for finding ways to immunize the network. While misuse malware detection is based on an abstract representation of the inner working of a malicious code viewed as a single entity, the propagation monitoring approach is based on the characterization of the interactions of a malicious code with the environment. Indeed, the monitoring of malware propagation turns out to be particularly interesting for analyzing the topology of botnets and for understanding, detecting and stopping bot propagation [9].

The challenging behavior of malware suggests the development of a formal framework for the specification and analysis of malicious software. The standard formal definitions of viruses by Cohen [7] and Adleman [1] are not appropriate for modeling and analyzing the behavior and the propagation of modern distributed malware. Cohen characterizes viruses in terms of their ability to replicate with respect to a given Turing machine, while Adleman provides a more abstract definition of computer viruses based on recursive functions where an infected program either *imitates* the behavior of the original program, *injures* the machine or *propagates* infection. Interesting generalizations of the Adleman's definition are the ones given by Zou and Zhou [26] and Bonfante et al. [4]. All these abstract theoretical models based on Turing machines and recursive functions are used to prove that the malware detection problem is undecidable.

Observe that all these models, based on Turing machines or recursive functions, are not the most appropriate ones for describing the interaction and the cooperation between different systems. In fact, modern malware as botnets cannot be described and specified in a convenient and detailed way with the standard theoretical models. The advent of these interaction-based malware requires the development of new formal models for malware analysis. We mention here the theoretical model known as k-ary malware that distributes the malicious behavior into many concurrent codes [14]. At the moment there exist only trivial implementations of k-ary malware, but in principle, they could avert standard malware detection strategies. Another related work is the recent one of Jacob et al. [17], where the authors propose a model of malware based on the Join calculus. They found their definition of malware on the notion of virus given by Cohen [7] and extend it to support concurrent interactions. They focus on the malware detection problem and prove that in general it is undecidable in the Join calculus. Furthermore, they identify a fragment of the Join calculus where this problem becomes decidable even if it is not clear whether real malware can be modeled in the identified fragment.

The idea. The main objective of this work is to design a model for the specification and analysis of standard and distributed malware together with their propagation mechanisms. This accounts to consider formalisms able to express the concurrent interaction between the multiple actors involved in a malicious attack. When dealing with concurrency, languages such as CCS [22], ACP [3] and the π-calculus [24], are among the most natural choices. This is also the reason why those languages have largely inspired the design of calculi for the biological world. In this setting, where not only concurrent interactions but also spatial aspects such as containment or relative position (i.e. topology) are essential, new formal frameworks like brane calculus [5], Ambients [6] and κ-calculus [12] have been proposed. As biological virus are generally associated to malware, we find it natural to look for a suitable calculus among the ones designed for computational biology.

We propose to use the κ-calculus to model distributed malicious attacks. The κ-calculus is a language for the specification of systems that can be represented as graphs where nodes are equipped with an internal state and labeled ports, and edges specify connections between labeled ports of nodes. Rewriting rules associated to each system model the graph evolution. We believe that the κ-calculus is suitable for assessing the malware threat, also in the case of distributed malware, for the following reasons:

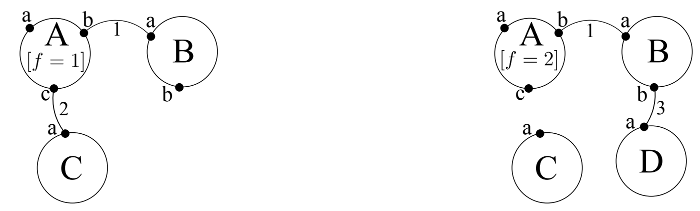
1. κ allows a detailed specification of the mechanisms of interactions between concurrent systems. One of the main advantages of using κ with respect to other process calculi as the π-calculus or the Join calculus, resides in its graphical nature that allows the description of spatial properties. Moreover, the set of rules defining the behavior of malware can be isolated and distinguished from the rules describing the normal evolution of a system.
2. κ offers a framework where the granularity of the model can be chosen depending on the specific problem to represent, i.e., one can model programs inside a single machine, or machines abstracting from the programs they are running.
3. Recent works have shown how topology plays a fundamental role in the analysis of malware propagation [15]. κ-systems are essentially graphs and this allows a straightforward definition of different network topologies. Moreover, since κ is a rewriting model, it is suitable for the specification of networks whose topology evolves over time as in [23].
4. The analysis of malware formalized in κ can exploit the tools for the simulation of κ-systems [8]. Moreover, these tools – developed in the biological setting – allow the specification of stochastic measures that can model the probability of a certain node to be infected, the rate of propagation of a given infection, etc.

In Section 2 we briefly introduce the κ-calculus. Section 3 formalizes the notion of malware with respect to an environment and shows how the proposed definition can be instantiated to the different kinds of standard and distributed malware. In Section 4 we present and discuss how the proposed model can be employed to study the propagation strategies of specific malware. Section 5 concludes by discussing future developments.

2 The κ-Calculus

In this section we recall the basic definitions of the κ-calculus [12] independently from its biological domain of application. Some terminology has been changed and some constraints (like the separation between destruction and construction rules, typical of biological settings) have been relaxed. The κ-calculus is a graph rewriting model where objects are represented by nodes and their relationships by edges. More formally, let us consider the following countable sets: $\texttt{Nodes}$ with elements $A, B, C, \ldots$; $\texttt{Gates}$ with elements $a, b, c \ldots$; $\texttt{Arms}$ with elements $1, 2, 3, \ldots$; $\texttt{Fields}$ with elements $f_1, f_2, \ldots$. Let $\mathit{Arms}(A) \subseteq \texttt{Arms}$, $\mathit{Gates}(A) \subseteq \texttt{Gates}$ and $\mathit{Fields}(A) \subseteq \texttt{Fields}$ denote, respectively, the set of arms, gates and fields of a node A. A *state* α is a map from $\texttt{Fields}$ to a finite set of possible values. An *interface* $\mathcal{S}$ is a map $\texttt{Gates} \mapsto \texttt{Arms} \cup \{\varepsilon\}$ where $\mathcal{S}(a) = \varepsilon$ means that no arm is associated to gate a.

The domain of a partial function $f : D \mapsto C$ is $\mathit{dom}(f) = \{d \in D \mid f(d) \neq \bot\}$. The *union* of two partial functions $f, g : D \mapsto C$ such that $\mathit{dom}(f) \cap \mathit{dom}(g) = \emptyset$,



(a) The start up condition of the system (b) The system after the evolution

Fig. 1. Graphical representation of the system $A[f = 1](a = \varepsilon, b = 1, c = 2), B(a = 1, b = \varepsilon), C(a = 2)$ and the application of rule $A[f = 1](c = 2), B(b = \varepsilon), C(a = 2) \rightarrow A[f = 2](c = \varepsilon), B(b = 3), C(a = \varepsilon), D(a = 3)$

is a partial function $f + g : D \mapsto C$ with domain $dom(f + g) = dom(f) \cup dom(g)$ such that $(f + g)(d) = f(d)$ if $d \in dom(f)$, $(f + g)(d) = g(d)$ if $d \in dom(g)$, and $f + g(d) = \perp$ otherwise. $f \leq g$ if $dom(f) \subseteq dom(g)$ and $\forall x \in dom(g) : f(x) = g(x)$ or $f(x)$ undefined.

A component is a node A with a state that specifies all the values of the fields of A, and an interface that specifies which gates of A are connected to some arms (edges).

Definition 1. *A* component $A[\alpha](\mathcal{S})$ *is a node A equipped with a state map α and an interface map $\mathcal{S}$ that are total w.r.t. A: $dom(\alpha) = Fields(A)$ and $dom(\mathcal{S}) = Gates(A)$.*

If a component has no fields we omit the part [ ]. A system is a multiset of components and it can be graphically represented as a non-oriented graph with nodes denoting components and edges modeling arms (see Fig. 1(a) for an example).

Definition 2. *A* system Σ *is defined as* $\Sigma ::= A[\alpha](\mathcal{S}) \mid \Sigma, \Sigma$, *where "," denotes the associative operator for composition and each arm occurs exactly twice.*

We speak about component-projection when we have a node with a partial specification of its state and interface.This notion can be naturally extended to systems by saying that a system-projection is a collection of component-projections. If we consider the system in Fig. 1(a), $B(a^1)$ is a component-projection of $B(a^1 + b^\varepsilon)$.

Definition 3. $A[\alpha](\mathcal{S})$ *is a* component-projection *(of A) if α and $\mathcal{S}$ are not total with respect to A i.e., $dom(\alpha) \subseteq Fields(A)$ and $dom(\mathcal{S}) \subseteq Gates(A)$.*

$A_1[\alpha_1](\mathcal{S}_1) \ldots A_n[\alpha_n](\mathcal{S}_n)$ *is a* system-projection *(of $A_1 \ldots A_n$) if for every $i \in [1..n]$ we have that $A_i[\alpha_i](\mathcal{S}_i)$ is a component projection (of A_i) and every arm occurs exactly twice.*

Definition 4. *A* rule *has the form $L \rightarrow R$ where the left hand side L and the right hand side R are system-projections:*

$$A_1[\alpha_1](\mathcal{S}_1), \ldots, A_n[\alpha_n](\mathcal{S}_n) \rightarrow A_{i_1}[\alpha'_{i_1}](\mathcal{S}'_{i_1}), \ldots, A_{i_m}[\alpha'_{i_m}](\mathcal{S}'_{i_m}),$$
$$B_1[\beta_1](\mathcal{F}_1), \ldots, B_k[\beta_k](\mathcal{F}_k)$$

where for all $j \in [1..m]$: $i_j \in J \subseteq \{1 \ldots n\}$, $dom(\alpha_{i_j}) = dom(\alpha'_{i_j})$ and $\mathcal{S}'_{i_j} \leq \mathcal{S}_{i_j}$, for every $i \in \{1 \ldots n\} \setminus J$ we have that α_i and $\mathcal{S}_i$ are total with respect to A_i and for every $l \in [1..k]$ we have that β_l and $\mathcal{F}_l$ are total with respect to B_l.

The intuition is that a rule can have the following effects: (i) modify the state of the given components by changing the values of the already specified fields; (ii) add new arms to the gates that the given components map to ε; (iii) add new components (i.e. fully specified terms), (iv) delete arms of given components by modifying the interface in such a way that the corresponding gate maps to ε; (v) remove the elements of the system-projection only if they are components (i.e., they are fully specified). As an example see Fig. 1(b).

Definition 5. *The* structural equivalence *relation between systems, denoted $\equiv$, is the least equivalence satisfying:*

- $\Sigma, \Pi \equiv \Pi, \Sigma$*;*
- $\Sigma \equiv \Pi$ *if there exists an injective renaming ι on arms such that Σ is obtained from Π by renaming its arms according to ι, denoted $\Sigma = \iota(\Pi)$.*

Two component-projections $A[\alpha_1](\mathcal{S}_1)$ and $A[\alpha_2](\mathcal{S}_2)$ on the same node A are *disjoint* when $dom(\alpha_1) \cap dom(\alpha_2) = \emptyset$ and $dom(\mathcal{S}_1) \cap dom(\mathcal{S}_2) = \emptyset$. We define the *sum* of disjoint component-projections as: $A[\alpha_1](\mathcal{S}_1) \oplus A[\alpha_2](\mathcal{S}_2) = A[\alpha_1 + \alpha_2](\mathcal{S}_1 + \mathcal{S}_2)$. This can be extended to systems in the expected way. A component-projection, as well as a system-projection, can be instantiated to a set of possible components, or systems, by specifying the values of the undefined fields and the missing gate-arm relations. The notion of projection allows us to define the reduction relation for the κ-calculus.

Definition 6. *Let us consider a set $\mathbb{R}$ of rules and the systems $\Sigma, \Sigma', \Pi, \Pi', \Gamma$. The* reduction relation *associated to $\mathbb{R}$, denoted $\rightarrow_{\mathbb{R}}$, is the least relation that satisfies the followings:*

- *if $L \rightarrow R \in \mathbb{R}$, $\Sigma = L \oplus \Delta$, and $\Pi = R \oplus \Delta$ then: $\Sigma \rightarrow_{\mathbb{R}} \Pi$;*
- *if $\Sigma \rightarrow_{\mathbb{R}} \Pi$ and $(Arms(\Pi) \smallsetminus Arms(\Sigma)) \cap Arms(\Gamma) = \emptyset$ then $\Sigma, \Gamma \rightarrow_{\mathbb{R}} \Pi, \Gamma$;*
- *if $\Sigma \equiv \Sigma'$, $\Sigma' \rightarrow_{\mathbb{R}} \Pi'$ and $\Pi' \equiv \Pi$, then $\Sigma \rightarrow_{\mathbb{R}} \Pi$.*

The first condition specifies how to apply rule $L \rightarrow R$ to a system Σ with a system-projection that matches L. The second condition states that rules can be applied to portions of the system only if the arms created by the rule are not present in the rest of the graph. The third condition says that the reduction relation between systems can be extended to structurally equivalent systems. We denote with $\Rightarrow$ the reflexive and transitive closure of $\rightarrow$. Given two systems Σ, Π with the respective sets of rules $\mathbb{R}_\Sigma, \mathbb{R}_\Pi$, we use notation $(\Sigma, \mathbb{R}_\Sigma), (\Pi, \mathbb{R}_\Pi) \rightarrow (\Sigma', \mathbb{R}_\Sigma), (\Pi', \mathbb{R}_\Pi)$ to refer to $\Sigma, \Pi \rightarrow_{\mathbb{R}_\Sigma \cup \mathbb{R}_\Pi} \Sigma', \Pi'$. Moreover, we denote with $\Sigma \overset{\mathbb{R}}{\Longrightarrow} \Sigma'$ the fact that during the evolution we have used only the rules in the set $\mathbb{R}$.

3 Modeling Malware in κ

Since this is the first time that the κ-calculus is used for modeling programs, we need to formally specify what is a program and what is an execution environment in the κ-framework.

Definition 7. *A* program $\mathtt{P}$ *in* κ *is a pair* $(\Sigma_{\mathtt{P}}, \mathbb{R}_{\mathtt{P}})$*, where* $\Sigma_{\mathtt{P}}$ *is a system and* $\mathbb{R}_{\mathtt{P}}$ *is the set of rules that describe the behaviour of* $\mathtt{P}$ *in the environment. Let Progr denote the set of programs in* κ*. An* environment $\mathcal{E}$ *is a collection of programs* $\mathtt{P}_1, \ldots, \mathtt{P}_n$ *with* $\mathtt{P}_i \in \mathit{Progr}$ *for all* $i \in [1, n]$*.*

We denote with $\mathcal{E}[\mathtt{P} \leftarrow \mathtt{P}']$, the environment $\mathcal{E}$ where program $\mathtt{P}$ has been replaced with program $\mathtt{P}'$. We extend the notion of structural equivalence to programs by saying that two programs $\mathtt{P} = (\Sigma_{\mathtt{P}}, \mathbb{R}_{\mathtt{P}})$ and $\mathtt{Q} = (\Sigma_{\mathtt{Q}}, \mathbb{R}_{\mathtt{Q}})$ are *structurally equivalent*, denoted $\mathtt{P} \dot{\equiv} \mathtt{Q}$, if they have systems that are structurally equivalent and the same set of rules, i.e., $\mathtt{P} \dot{\equiv} \mathtt{Q}$ if $\Sigma_{\mathtt{P}} \equiv \Sigma_{\mathtt{Q}}$ and $\mathbb{R}_{\mathtt{P}} = \mathbb{R}_{\mathtt{Q}}$ up to renaming. Thus, programs that are structurally equivalent cause structurally equivalent evolutions on structurally equivalent systems. We can now specify a formal definition of malware in the κ-calculus, where a malware is a program that behaves in a specific way in an environment. The proposed definition is inspired by the ones of Cohen [7] and Adleman [1]. In fact, according to Cohen, we provide a notion of malware with respect to a given environment and, according to Adleman, our notion of malware specifies the possible behaviors of the malicious code either as imitate, injure and infect. Moreover, by modeling the malicious behaviors in the κ-framework we can specify the interactions of the malware with the environment and we can describe the behavior of distributed malware.

Definition 8. *A program* $\mathtt{M} = (\Sigma_{\mathtt{M}}, \mathbb{R}_{\mathtt{M}})$ *is a* malware w.r.t. environment $\mathcal{E}$ *if it behaves in one of the following ways in environment* $\mathcal{E}$*:*

- Imitate*: If* $\mathtt{M}$ *assumes the presence of a host program then it has the following structure* $\mathtt{M} = h(\mathtt{MC}, \mathtt{Q})$ *where* $\mathtt{MC}$ *is the malicious code,* $\mathtt{Q} = (\Sigma_{\mathtt{Q}}, \mathbb{R}_{\mathtt{Q}})$ *is the program hosting it, and* $h : \mathit{Progr} \times \mathit{Progr} \rightarrow \mathit{Progr}$ *is the infection function that given* $\mathtt{MC}$ *and* $\mathtt{Q}$ *returns the program* $h(\mathtt{MC}, \mathtt{Q})$ *obtained by infecting* $\mathtt{Q}$ *with* $\mathtt{MC}$*. In this case we say that program* $h(\mathtt{MC}, \mathtt{Q})$ *is able to imitate the behavior of the host program* $\mathtt{Q}$ *if the following holds: if* $\mathtt{Q}, \mathcal{E} \overset{\mathbb{R}_{\mathtt{Q}}}{\Longrightarrow} \mathtt{Q}', \mathcal{E}'$ *then* $h(\mathtt{MC}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_{\mathtt{Q}}}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q})$ *and* $\mathtt{Im}(\mathcal{E}, \mathtt{Q}) = h(\mathtt{MC}, \mathtt{Q}'), \mathcal{E}''$ *such that* $\mathcal{E}'' \dot{\equiv} \mathcal{E}'$*.*
- Injure: $\mathtt{M}$ *performs the intended payload on* $\mathcal{E}$*:* $\mathtt{M}, \mathcal{E} \overset{\mathbb{R}_{\mathtt{M}}}{\Longrightarrow} \mathtt{M}', \mathtt{Pl}(\mathcal{E}; \mathtt{M})$*, where* $\mathtt{Pl} : \wp(\mathit{Progr}) \times \mathit{Progr} \rightarrow \wp(\mathit{Progr})$ *is the payload function that, given an environment* $\mathcal{E}$ *and a malware* $\mathtt{M}$*, returns the damaged environment, namely the environment obtained by performing the malicious actions of malware* $\mathtt{M}$ *on environment* $\mathcal{E}$*.*
- Infect: $\mathtt{M}$ *replicates itself in* $\mathcal{E}$*:* $\mathtt{M}, \mathcal{E} \overset{\mathbb{R}_{\mathtt{M}}}{\Longrightarrow} \mathtt{M}', \mathtt{Rep}(\mathcal{E}; \mathtt{M})$*, where* $\mathtt{Rep} : \wp(\mathit{Progr}) \times \mathit{Progr} \rightarrow \wp(\mathit{Progr})$ *is the replication function that, given an environment* $\mathcal{E}$ *and a malware* $\mathtt{M}$*, returns the infected environment, namely the environment obtained by infecting a program of environment* $\mathcal{E}$ *with malware* $\mathtt{M}$*.*

Let us show how by further specifying functions $\mathtt{Pl}$ and $\mathtt{Rep}$ we can instantiate the above general definition to the different kind of existing malware. Given a malicious code $\mathtt{V} = (\Sigma_{\mathtt{V}}, \mathbb{R}_{\mathtt{V}})$ and a host program $\mathtt{Q} = (\Sigma_{\mathtt{Q}}, \mathbb{R}_{\mathtt{Q}})$, sometimes we write $h(\mathtt{V}, \mathtt{Q}) = (\Sigma_{\mathtt{Q}}^{\mathtt{V}}, \mathbb{R}_{\mathtt{Q}}^{\mathtt{V}})$ where $\Sigma_{\mathtt{Q}}^{\mathtt{V}}$ denotes the system obtained by infecting the system $\Sigma_{\mathtt{Q}}$ with system $\Sigma_{\mathtt{V}}$ and $\mathbb{R}_{\mathtt{Q}}^{\mathtt{V}}$ denotes the rules obtained by infecting the rules of $\mathbb{R}_{\mathtt{Q}}$ with $\mathbb{R}_{\mathtt{V}}$.

Virus. A virus is a self-propagating program that attaches itself to host programs and propagates when the hosting program executes. Some viruses are designed to damage the machines on which they execute, while other viruses simply replicate themselves.

Definition 9. *Let $\mathcal{E} = \mathtt{P}_1 \dots \mathtt{P}_n$, with $\mathtt{P}_i = (\Sigma_i, \mathbb{R}_i)$ where every element $\mathtt{P}_i$ models a program. A program $\mathtt{V} = (\Sigma_\mathtt{V}, \mathbb{R}_\mathtt{V})$ is a* virus w.r.t. $\mathcal{E}$ *if, given a hosting program $\mathtt{Q} = (\Sigma_\mathtt{Q}, \mathbb{R}_\mathtt{Q})$, we have that $h(\mathtt{V}, \mathtt{Q}) = (\Sigma^\mathtt{V}_\mathtt{Q}, \mathbb{R}^\mathtt{V}_\mathtt{Q})$ is a malware w.r.t. $\mathcal{E}$ such that:*

– *Imitate: $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_\mathtt{Q}}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q})$.*
– *Injure (optional): $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}^\mathtt{V}_\mathtt{Q}}{\Longrightarrow} h(\mathtt{V}, \mathtt{Q})', \mathtt{Pl}(\mathcal{E}; \mathtt{V})$. Function $\mathtt{Pl}(\mathcal{E}; \mathtt{V})$ does not depend on $\mathtt{Q}$, and this models the fact that the payload of a virus does not depend on the program hosting it.*
– *Infect: $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}^\mathtt{V}_\mathtt{Q}}{\Longrightarrow} h(\mathtt{V}', \mathtt{Q}), \mathcal{E}[\mathtt{P}_j \leftarrow h(\mathtt{V}, \mathtt{P}_j)]$.*

Given the analogy between the proposed definition and the one of Adleman it is not surprising that we are able to prove that our notion of virus is at least as expressive as the one of Adleman. In particular, since κ is Turing equivalent [13] we show that for every recursive function that is a virus according to Adleman there exists a program in κ that computes it and that is a virus w.r.t. every environment.

Theorem 1. *Every program $\mathtt{V} = (\Sigma_\mathtt{V}, \mathbb{R}_\mathtt{V})$ in the κ-framework that computes a function T_v : Progr $\rightarrow$ Progr that is a virus according to Adleman is a virus with respect to every environment according to Definition 9.*

As done by Zuo and Zhou, Adleman's definition could be instantiated to different kinds of viruses by specifying the infection strategy [26]. The same approach can be applied to our model by further delineating the infection function h.

Trojan horse, Spyware, Trap Door and Logic Bomb. Here we group together those malware that do not exhibit an infection behavior. A *Trojan horse* is a non-replicating program that hides its malicious intent inside host programs that may look useful, or at least harmless. *Spyware* are malicious programs designed to monitor user's actions in order to collect private information and send them to an external entity over the Internet. A *trap door* is a malicious code that provides a secret entry point into a program that allows someone that is aware of the trap door to gain access without going through the usual security access procedure. A *logic bomb* is a malicious code embedded in a legitimate program that executes when a certain predefined event occurs.

Definition 10. $\mathtt{V} = (\Sigma_\mathtt{V}, \mathbb{R}_\mathtt{V})$ *is a* Trojan horse *or a* spyware *or a* trap door *or a* logic bomb w.r.t. $\mathcal{E}$ *if $\mathtt{M} = h(\mathtt{V}, \mathtt{Q}) = (\Sigma^\mathtt{V}_\mathtt{Q}, \mathbb{R}^\mathtt{V}_\mathtt{Q})$ is a malware w.r.t. to $\mathcal{E}$ such that:*

– *Imitate: $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_\mathtt{Q}}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q})$.*
– *Injure: $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}^\mathtt{V}_\mathtt{Q}}{\Longrightarrow} h(\mathtt{V}', \mathtt{Q}), \mathtt{Pl}(\mathcal{E}; \mathtt{Q})$, where Trojan horses, spyware, trap doors and logic bombs are differentiate by the specification of function $\mathtt{Pl}(\mathcal{E}; \mathtt{Q})$.*

k-ary virus. Interestingly, the theoretical model of k-ary virus can be seen as an instance of our definition of malware. A k-ary virus is a family of k files (some of them may not be executable) whose union constitutes a computer virus [14]. We can model the k fragments composing the k-ary virus as a set of k systems: $\{\mathtt{V}_1, \dots, \mathtt{V}_k\} = \{(\Sigma_{\mathtt{V}_1}, \mathbb{R}_{\mathtt{V}_1}), \dots, (\Sigma_{\mathtt{V}_k}, \mathbb{R}_{\mathtt{V}_k})\}$.

Definition 11. *Let $\mathcal{E} = P_1, \dots, P_n$ where each P_i models a program. We say that programs $\{V_1, \dots, V_k\}$ form a k-*ary virus* w.r.t. $\mathcal{E}$ if $h(V_1, Q_1) \dots h(V_k, Q_k)$ where $\mathbb{R}_V = \mathbb{R}_{V_1} \cup \dots \cup \mathbb{R}_{V_k}$ behave as follows:*

– *Imitate: $h(V_1, Q_1) \dots h(V_k, Q_k), \mathcal{E} \xRightarrow{\mathbb{R}_{Q_l}} \mathtt{Im}(\mathcal{E}; Q_l)$, for every $l \in [1, k]$.*
– *Injure: $h(V_1, Q_1) \dots h(V_k, Q_k), \mathcal{E} \xRightarrow{\mathbb{R}_V} h(V_1, Q_1)' \dots h(V_k, Q_k)', \mathtt{Pl}(\mathcal{E}; V_1 \dots V_k)$*
– *Infect: $h(V_1, Q_1) \dots h(V_k, Q_k), \mathcal{E} \xRightarrow{\mathbb{R}_V} h(V_1, Q_1)' \dots h(V_k, Q_k)', \mathcal{E}[P_j \leftarrow h(V_1, P_j) \dots P_{j+k} \leftarrow h(V_k, P_{j+k})]$.*

Worm. A worm is a malicious program that uses a network to send copies of itself to other systems and, unlike viruses, do not need a host program. In general, worms do not contain a specific payload but they are only designed to spread. In order to model worms in the κ-framework we have to observe their behavior in a network. Thus, we consider an environment $\mathcal{E} = S_1 \dots S_n$ where each system $S_i = (\Delta_i, \mathbb{R}_i)$ with $i \in [1, n]$, models a machine in a network. In this setting, when a machine $S_k = (\Delta_k, \mathbb{R}_k)$ hosts a program $W = (\Sigma_W, \mathbb{R}_W)$ we simply write $S_k^W = S_k, W$.

Definition 12. *Let $\mathcal{E} = S_i \dots S_n$ where each $S_i = (\Delta_i, \mathbb{R}_i)$ is a machine. A program $W = (\Sigma_W, \mathbb{R}_W)$ is a* worm *w.r.t. $\mathcal{E}$ if the machine $S_k^W = (\Delta_k, \mathbb{R}_k), (\Sigma_W, \mathbb{R}_W)$ is a malware w.r.t. $\mathcal{E}$ with the following behavior:*

– *Injure (optional): $(\Sigma_W, \mathbb{R}_W), (\Delta_k, \mathbb{R}_k), \mathcal{E} \xRightarrow{\mathbb{R}_W} (\Sigma'_W, \mathbb{R}_W), (\Delta'_k, \mathbb{R}_k), \mathtt{Pl}(\mathcal{E}; W)$.*
– *Infect: $(\Sigma_W, \mathbb{R}_W), (\Delta_k, \mathbb{R}_k), \mathcal{E} \xRightarrow{\mathbb{R}_W} (\Sigma'_W, \mathbb{R}_W), (\Delta_k, \mathbb{R}_k), \mathcal{E}[S_i \leftarrow S_i, (\Sigma_W, \mathbb{R}_W)]$.*

Botnet. The term *botnet* refers to a network of compromised machines. A botnet is controlled by the so called botmaster, that is an attacker machine that performs the following actions: (1) identifies a set of target machines; (2) installs a bot on each target machine such that the bot remains dormant until it receives a predefined command form the botmaster; (3) launches the attack by triggering the bots. Botnet are successfully used to implement distributed denial of service attacks.

Definition 13. *A program $B = (\Sigma_B, \mathbb{R}_B)$ is a* botmaster *w.r.t. $\mathcal{E} = S_1 \dots S_n$, where each S_i is a machine in a network, if it is a malware w.r.t. $\mathcal{E}$ that either infects: $B, \mathcal{E} \xRightarrow{\mathbb{R}_B} B', \mathcal{E}[S_i \leftarrow h(S_i, bot)]$, or injures: $B, \mathcal{E} \xRightarrow{\mathbb{R}_B} B', \mathtt{Pl}(\mathcal{E}; bot, B)$. A program bot is a bot if every machine $h(S_i, bot)$ hosting it is a malware w.r.t. $\mathcal{E}$ that behaves as follows:*

– *Imitate: $B, \mathcal{E}[S_i \leftarrow h(S_i, bot)] \xRightarrow{\mathbb{R}_{S_i}} \mathtt{Im}(B, \mathcal{E}[S_i \leftarrow h(S_i, bot)]; S_i)$.*
– *Injure: $B, \mathcal{E}[S_i \leftarrow h(S_i, bot)] \xRightarrow{\mathbb{R}_{bot}} \mathtt{Pl}(B, \mathcal{E}[S_i \leftarrow h(S_i, bot)]; B)$.*

Thus, the proposed definition of malware in κ allows for the specification of the main actors involved in a botnet attack. In particular, thanks to the different levels of granularity that we can use to model a κ-system, we are able to specify both the botnet as a whole and the single bots in the same framework, and we can tune the level of abstraction of these specifications according to the features that we want to analyze. For instance, since bots are given by the combination of standard malware, like viruses and worms, and a communication channel with the botmaster, we can precisely model in κ both the behavior of the bots and their interaction with the botmaster.

4 Hunting Malware in κ

The typical approach to malware defense is malware detection, namely the identification of the malicious code in order to avoid infection. In literature there are many results that show the intractability of the malware detection problem (e.g. [1,4,7,17,26]). All these results share a common pattern: (i) they provide a formalization of malware in a Turing complete model of computation (e.g., Turing machines, recursive functions); (ii) they prove the undecidability of the malware detection problem by reducing it to problems that are known to be undecidable. In Section 3 we have proposed a formalization of the notion of malware in κ, a Turing complete calculus [13]. In particular, according to our definition of malware, in order to decide whether a program $\mathtt{M}$ is a malware w.r.t. an environment $\mathcal{E}$ we have to verify that every evolution of $\mathcal{E}$ is also an evolution of $\mathcal{E}, \mathtt{M}$ (imitate); that $\mathcal{E}, \mathtt{M}$ allows more evolutions than $\mathcal{E}$ (injure); and that $\mathcal{E}, \mathtt{M}$ has evolutions that lead to the replication of $\mathtt{M}$, namely $\mathcal{E}, \mathtt{M} \Rightarrow \mathcal{E}', \mathtt{M}, \mathtt{M}'$ with $\mathtt{M} \doteq \mathtt{M}'$ (infect). Observe that the problem of deciding whether the evolutions of $\mathcal{E}$ are included in the ones of $\mathcal{E}, \mathtt{M}$ can be reduced to the problem of deciding program equivalence which is known to be undecidable in any Turing complete language. Moreover, the problem of malware replication can be reduced to a coverability problem which is known to be undecidable in κ [13]. Thus, as expected, also with our formalization of malware it is possible to conclude that malware detection is undecidable.

A more recent promising approach to malware defense regards the monitoring of malware propagation in a network both for studying properties of the considered malware and malware epidemiology and for finding ways to immunize the network.

Malware Propagation. The problem we want to tackle can be made clear with a similitude with the biological setting. Suppose that a certain infection is taking off and degenerating into an epidemics: What is the faster way to vaccinate people in order to minimize the spread of the infection? Rephrasing this problem in our setting: suppose that a network is under the attack of a fast spreading malware, how can we detect which are the weak nodes to be immunized? This is not a new approach to the field, several studies have been conducted to this aim. The first attempt dates back to 1991 when Kephart and White [18] propose an epidemiological model inspired by the biological setting. More recently, this work has been shown to be outdated as modern worms spread with different kinds of models usually guided by the topology of the network [15,19,23]. A key aspect of these techniques is that they employ stochastic measures to represent both the speed of propagation and the likelihood of a node to be infected (stochastic measures could also model network features such as link capacities, congestion, intermittent connectivity). Interestingly, as mentioned in the introduction, there exists simulation tools for κ-systems that take into account stochastic measurements (expressing the probability of a rule to be applied) [8]. Let us detail the main steps for the investigation of malware propagation by means of κ and its simulation tools:

1. *Modeling:* Choose the proper level of abstraction of the model and formalize the main actors as κ-systems and their activities as sets of rules. Then, set the stochastic metrics either as the result of personal investigation or derived from previous existing works (e.g. [10,15]). This leads to the specification of a stochastic κ-system that represents the initial condition of the network.

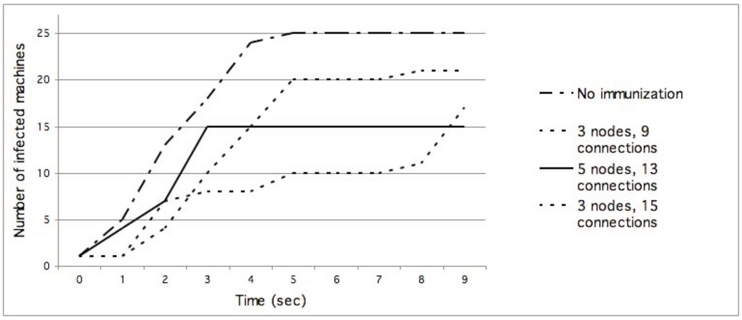


Fig. 2. Speed of infection of Example 1

2. *Simulation:* Choose one of the freely available tools in [8] that, given the specification of the model in κ, simulates its evolution and returns the resulting system.
3. *Analysis:* From the results of the simulation, we extract the *network of infected machines*. This accounts in showing which machines have been infected and which connections have been exploited for malware propagation (the topology of this network could be substantially different from the original one). By analyzing this graph it should be possible to determine the best defense strategy to obstruct the propagation of the considered malware. For instance, the machines to be immunized could be identified by running a discovery algorithm on the graph that counts the nodes reachable, and therefore corruptible, from each node.

Example 1. We have considered and modeled in the κ-framework a small network of 25 machines and a malware infecting it. In one step the malware infects all its sane neighbors that are not immunized. We have simulated the spread of the infection when no machines are immunized and in few seconds all the network has been compromised. In this simple case, in order to identify the best nodes to be immunized on the network of infected machines it is enough to count the number of connections of each nodes. We have considered different immunization scenarios and the experimental results confirm that the speed of infection decreases with the growth of the number of connections of the immunized nodes. Fig. 2 shows the speed of infections when different sets of nodes (and therefore connections) are immunized.

The special case of botnets. As depicted in several works [9,11,25], botnets are difficult to track. Although being a major threat, their life-cycle is still yet to be understood and it is almost impossible to prevent systems from being compromised. Lately, researchers have focused on the study of botnet propagation strategies, in order to make the botnet harmless by arresting the spread of bots.

The systematic strategy proposed above could be used for the study of the recruitment phase performed by bots infection. Indeed, botnets usually exploit standard malicious techniques to corrupt and therefore recruit target machines. For this reason, we claim that by modeling botnets in the κ-framework, we could shed light on the life-cycle of botnets and on their propagation strategies. Roughly speaking, we could represent different topologies of networks where nodes model the machines and the botmaster.

Machines are equipped with a special state that indicates if they are clean or infected and edges represent the physical connections between them. Rules describe both the choice of potential victims in the recruitment phase and how the network changes along time. Thus, from the analysis of the network of compromised machines obtained through the simulation process, we could extract important features about the botnet behavior.

5 Concluding Remarks

In this work we have proposed to apply biological concurrent calculi to the analysis of malicious code. We have focused on the κ-calculus and we have used it to provide an unifying formalization of existing malware. We have analyzed and discussed the potentialities of modeling malware and their propagation in the κ-framework. We believe that the use of κ will open new challenges in the field of malware analysis and defense. In particular, we claim that the proposed formalization of malware provides a powerful framework for understanding the behavior of distributed malware. In fact, we believe that κ and its tools of analysis provide the right means for investigating the main aspects of distributed malware: propagation over a network, machines and programs interactions, launch of a distributed attack. By simulating the distributed malware behavior it is possible to identify the countermeasures to take in order to defend an environment from the considered attack, or measure the goodness of existing defense strategies. As for future work in this direction, we plan to model with κ real types of botnets [2] and to use the metrics proposed in known works (e.g. [10,15]) in order to stochastically simulate their behavior. We believe that considerably big portions of the Internet network can be faithfully compiled into κ and by observing the results of the simulation, we will be able to better understand the behavior of the considered botnets and propose an efficient defense strategy.

In this work, we have focused on the distributed features of the κ-calculus to reason on network properties and spreading behaviors. However, the existence of decidability results in κ [13] opens the way for the study of interesting reachability malware properties that could be used for malware detection. This means that this investigation could lead to the identification of fragments of the κ-calculus where the detection of specific classes of malware becomes decidable.

Acknowledgements. We would like to thank Maurizio Gabbrielli and Roberto Giacobazzi for their support and helpful feedbacks. We are also grateful to Jean Krivine and Jorge A. Pérez for their remarks and suggestions.

References

1. Adleman, L.M.: An abstract theory of computer viruses. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 354–374. Springer, Heidelberg (1990)
2. Bächer, P., Holz, T., Kötter, M., Wicherski, G.: Know your enemy: Tracking botnet, http://www.honeynet.org/papers/bots
3. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control 60(1-3), 109–137 (1984)

4. Bonfante, G., Kaczmarek, M., Marion, J.: On abstract computer virology from a recursion theoretic perspective. Journal in Computer Virology 1(3-4), 45–54 (2006)
5. Cardelli, L.: Brane calculi. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–278. Springer, Heidelberg (2005)
6. Cardelli, L., Gordon, A.D.: Mobile ambients. TCS 240(1), 177–213 (2000)
7. Cohen, F.: Computer viruses: Theory and experiments. Computers and Security 6, 22–35 (1987)
8. Collection of kappa tools, `http://kappalanguage.org/tools`
9. Cooke, E., Jahanian, F., McPherson, D.: The zombie roundup: Understanding, detecting, and disrupting botnets. In: SRUTI 2005, pp. 39–44 (2005)
10. Dagon, D., Gu, G., Lee, C.P.: A taxonomy of botnet structures. In: *Botnet Detection*. Advances in Information Security, vol. 36, pp. 143–164. Springer, Heidelberg (2008)
11. Dagon, D., Zou, C.C., Lee, W.: Modeling botnet propagation using time zones. In: NDSS. The Internet Society (2006)
12. Danos, V., Laneve, C.: Formal molecular biology. TCS 325(1), 69–110 (2004)
13. Delzanno, G., Di Giusto, C., Gabbrielli, M., Laneve, C., Zavattaro, G.: The κ-lattice: Decidability boundaries for qualitative analysis in biological languages. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 158–172. Springer, Heidelberg (2009)
14. Filiol, E.: Formalisation and implementation aspects of k-ary (malicious) codes. Journal in Computer Virology 3(2), 75–86 (2007)
15. Ganesh, A.J., Massoulié, L., Towsley, D.F.: The effect of network topology on the spread of epidemics. In: INFOCOM, pp. 1455–1466. IEEE, Los Alamitos (2005)
16. Grimes, R.A.: Malicious mobile code: Virus protection for windows. O'Reilly & Associates, Inc., Sebastopol (2001)
17. Jacob, G., Filiol, E., Debar, H.: Formalization of viruses and malware through process algebras. In: ARES 2010, pp. 597–602. IEEE Computer Society, Los Alamitos (2010)
18. Kephart, J.O., White, S.R.: Directed-graph epidemiological models of computer viruses. In: IEEE Symposium on Security and Privacy, pp. 343–361 (1991)
19. Kim, J., Radhakrishnan, S., Dhall, S.K.: Measurement and analysis of worm propagation on internet network topology. In: ICCCN, pp. 495–500. IEEE, Los Alamitos (2004)
20. McCarty, B.: Botnets: Big and bigger. IEEE Security and Privacy 1, 87–90 (2003)
21. McGraw, G., Morrisett, G.: Attacking malicious code: Report to the Infosec resarch council. IEEE Software 17(5), 33–41 (2000)
22. Milner, R.: Communication and concurrency. Prentice Hall International, Englewood Cliffs (1989)
23. Prakash, B.A., Tong, H., Valler, N., Faloutsos, M., Faloutsos, C.: Virus propagation on time-varying networks: Theory and immunization algorithms. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS, vol. 6323, pp. 99–114. Springer, Heidelberg (2010)
24. Sangiorgi, D., Walker, D.: PI-Calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
25. Wang, Q., Chen, Z., Chen, C., Pissinou, N.: On the robustness of the botnet topology formed by worm infection. In: GLOBECOM, pp. 1–6. IEEE, Los Alamitos (2010)
26. Zuo, Z., Zhou, M.: Some further theoretical results about computer viruses. Computer Journal 47(6), 627–633 (2004)

Edge-Matching Problems with Rotations

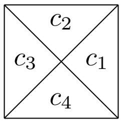
Martin Ebbesen, Paul Fischer, and Carsten Witt

DTU Informatics
Technical University of Denmark, DK-2800 Lyngby, Denmark

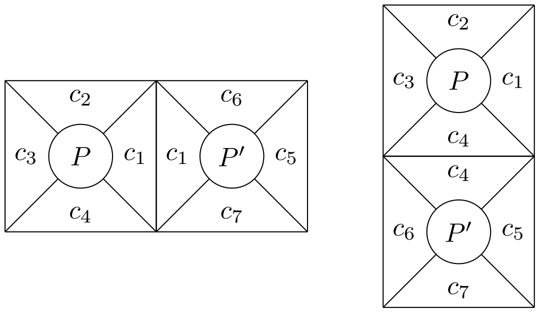
Abstract. Edge-matching problems, also called puzzles, are abstractions of placement problems with neighborhood conditions. Pieces with colored edges have to be placed on a board such that adjacent edges have the same color. The problem has gained interest recently with the (now terminated) Eternity II puzzle, and new complexity results. In this paper we consider a number of settings which differ in size of the puzzles and the manipulations allowed on the pieces. We investigate the effect of allowing rotations of the pieces on the complexity of the problem, an aspect that is only marginally treated so far. We show that some problems have polynomial time algorithms while others are NP-complete. Especially we show that allowing rotations in one-row puzzles makes the problem NP-hard. We moreover show that many commonly considered puzzles can be emulated by simple puzzles with quadratic pieces, so that one can restrict oneself to investigating those.

1 Introduction

The puzzles considered in this paper consist of quadratic *pieces* whose edges are colored. Let $c_0, c_1, \ldots, c_K$ denote the colors. The pieces are placed in the cells of a rectangular $N \times M$ grid. The edges of a piece are denoted *right*, *top*, *left*, *bottom* in the obvious way. A piece P can then be specified by its position (i, j) on the grid and the colors $c_{r,P}, c_{t,P}, c_{l,P}, c_{b,P}$ on its right, top, left, and bottom edge, in this order. The image below shows a piece specified by the color pattern (c_1, c_2, c_3, c_4).



Two pieces are *neighbors* if they are placed such that the difference of their row or column coordinates in the grid is 1, i.e., they share an edge. Given two pieces P and P' on neighboring positions, say at (i, j) and $(i, j+1)$ respectively, we say that they *match*, if the adjacent edges have the same color, that is $c_{r,P} = c_{l,P'}$. In this case we also say that the edge they have in common *matches*. Similarly, if they are vertically adjacent, say at (i, j) and $(i+1, j)$, they *match*, if $c_{b,P} = c_{t,P'}$, see example below.



We say that a board is *solved* if the pieces are placed in such a way that all edges match. At this point one has to specify rules for the “border edges”, that is, the edges facing the border of the board. We consider three cases here

Free. There is no restriction, any color matches the border of the board.
Monochrome. There is a single color for all border edges.
Cyclic. We assume that the board actually is a torus, that is, the top edges are aligned with the bottom ones and the left edges are aligned with the right ones.

An arrangement of the pieces which solves the board is called a *solution.*

The manipulations allowed in a board are:

Swap. Two pieces P and P' interchange their position, without being rotated.
Rotate. A piece P is rotated counter-clockwise in-place by 0 deg, 90 deg, 180 deg, or 270 deg.
Edge permutation. The colors on the edges are permuted in one of the 24 possible ways. The position of the piece is unchanged. Note that this manipulation includes rotations.
Flip. The colors of the left and right or up and down edges of piece P are interchanged (that is $P = (c_1, c_2, c_3, c_4)$ becomes $P_{flipped} = (c_3, c_2, c_1, c_4)$ or $P = (c_1, c_2, c_3, c_4)$ becomes $P_{flipped} = (c_1, c_4, c_3, c_2)$). Flipping is an in-place operation.

Combinations of the manipulations are possible.

The edge *matching problem* is then formulated as follows.

Problem 1. Given is a $N \times M$ board with K colors, NM pieces, a border rule and a set of manipulations. The decision problem is given by the question “is it possible to solve the board?”

In the optimization version of the problem, a solution has to be produced.

Edge-matching has found applications in biology where it is used in a method for DNA fragment assembly [1]. The problem has also gained interest recently with the Eternity II puzzle, a 16×16-puzzle, with a \$2 million prize for a solution (which was not found, though). Another area where this kind of problems appears is chip-layout, where interfaces have to be placed on a rectangular chip but their order is arbitrary.

1.1 Previous Work

Edge-matching with one row, swaps and no rotation corresponds to domino tiling [2] and is, as we discuss in this paper (and as covered in [2]), equivalent to the problem of finding an Eulerian path in a multi-graph. The computational complexity of edge-matching and related problems has been studied for several decades. Early results show that the more general tiling problem, i. e., solving the edge-matching problem with swaps for a quadratic board using only a subset of a given set of un-rotatable pieces is NP-complete [3]. Goles and Rapaport showed in [4] that the edge-matching problem with only in-place rotations (disallowing swaps) and free border rule is NP-complete. More recently, Demaine and Demaine [5] proved that the edge-matching variant considered here is NP-complete for quadratic boards with swaps and rotations. In 2010, Antoniadis and Lingas showed that this problem is even APX-hard, i. e., hard to approximate, already for boards with at least two rows [6].

1.2 Overview of the Paper

Most of the above-mentioned analyses of the edge-matching problem consider swaps, but do not allow rotating pieces. The NP-hardness proof in [5], even though it formally allowed rotations, forces the pieces to be used in a fixed orientation. Only recently, the APX-hardness proof in [6] explicitly made use of rotation and swaps at the same time. In this paper we address the question of what changes in the complexity of the problem occur when rotations of pieces are allowed.

In Section 2 we consider puzzles with only one row $(1 \times M)$. For these puzzles there is a known correspondence to Euler paths, that we describe along with some previous results. We then show that single-row puzzles where only in-place rotations are allowed can be efficiently solved. In contrast we show that solving single-row puzzles with rotations and swaps allowed is an NP-hard problem. The proofs implicitly use the Euler path formulation of the problem.

In Section 3 we strengthen a result of [5] by showing that already boards with two rows with swaps only are NP-hard to solve.

In Section 4 it is investigated how the number of solutions of puzzle depends on the board size, the number of colors, and the manipulations allowed. These results were used to construct hard to solve instances for the empirical tests.

The paper only considers puzzles with square pieces. In Section 5 we sketch how other shapes of pieces, e.g. triangular of hexagonal, can be simulated by square pieces. Moreover, we describe for the multi-row case how unrotatable pieces can be simulated by rotatable ones and vice versa.

2 Boards with One Row

Edge-matching with one row $(N = 1)$ without rotation is equivalent to the problem of finding an Eulerian trail in a multi-graph allowing loops. An Eulerian trail is a trail that visits each edge of the graph exactly once, and the concept is applicable to both directed and undirected graphs: An undirected graph is Eulerian (i.e. contains an Eulerian trail) iff it is connected and has no more than 2 vertices with an uneven number of edges. An Eulerian circuit (a trail starting and ending on the same vertex) requires that all vertices must have an even number of edges. A directed graph has an Eulerian circuit iff it is weakly connected and all vertices has equal in- and out-degree, while a path requires connectedness and 0 or 2 vertices with difference in in- and out-degree equal to 1. The above facts are shown in, e. g., [2].

2.1 Swaps and Flipping

An edge-matching instance with flipping, with M pieces and K colors is transformed to an undirected multigraph containing K vertices and M edges. Each piece corresponds to an edge connecting the vertices corresponding to the colors on opposing edges of the piece. A figure of the construction has been omitted due to space limitations, but can be found in [7] on page 35.

Now traversing a vertex using two different edges corresponds to matching two pieces having a common color. A trail in the graph corresponds to a matched chain of pieces, and an Eulerian cycle corresponds to a solution where all pieces are fitted, and vice versa. Hence this variant of the problem is efficiently solvable.

2.2 Swaps Only

Just like edge-matching with flipping correspond to Eulerian cycles in undirected multigraphs, edge-matching without flipping correspond to Eulerian cycles in directed multigraphs, and vice versa (see page 36 in [7]). Hence this variant of the problem is efficiently solvable.

2.3 Rotations Only

With free border rule, if the board consists of only one piece it will always be solved. Obviously, for a board with two pieces there will be a solution if the two pieces can be rotated such that their touching edges fit. This can be generalized:

Theorem 1. *Single-row edge matching puzzles with in-place rotations can be solved or determined to be unsolvable by an algorithm that has time complexity linear in the number of pieces.*

Proof. The proof is by induction. The pieces are numbered from left to right as $p_0, p_1, ..., p_{M-1}$. $L(p_i)$ is the set of colors that can be on the left edge of piece p_i which is equivalent to the set of unique colors on the 4 edges of p_i. $R(p_i)$ is the set of colors that can be on the right edge of p_i such that p_i fits p_{i-1}, that is, such that $L(p_i) \cap R(p_{i-1}) \neq \emptyset$. $L(p_i)$ and $R(p_i)$ can both have at most 4 members since a piece has 4 edges.

Base case: For p_0 there is no constraint, as one piece always represents a solution to a 1×1 board: $R(p_0) = L(p_0)$.

Induction step: Given that $R(p_i)$ is known and the board is solvable up to p_i then piece p_{i+1} can be fitted if $R(p_i) \cap L(p_{i+1}) \neq \emptyset$. If not, the board must be unsolvable. $R(p_{i+1})$ can be calculated by finding the color on the opposite edge of p_{i+1} for each member in $R(p_i) \cap L(p_{i+1})$. This operation takes constant time because of the bound on the sizes of the two sets.

The theorem holds because each step in the induction, i. e., each additional piece, only adds a constant time overhead. □

2.4 Swaps and Rotations

Theorem 2. *If both swaps and rotations are allowed, single-row edge-matching with free border is NP-complete.*

Proof. We use a polynomial-time reduction from the NP-complete Monotone 1-in-3-SAT problem [8]: given m monotone (i. e., disallowing negation) clauses $c_1, \ldots, c_m$ over n variables $x_1, \ldots, x_n$, the question is whether there exists an assignment which satisfies exactly one variable in each clause.

Let m_i, $i = 1, \ldots, n$, denote the number of occurrences of variable i, i. e., $m_1 + \cdots + m_n = 3m$. The instance of Monotone 1-in-3-SAT is mapped to a single-row board with free border at the top and bottom and cyclic border to the left and right (which can be simulated by a free border using a polynomial number of extra pieces). There are $M = 10m$ pieces and $K = 13m - n + 1$ colors. We denote colors by lower-case and pieces by upper-case letters. The set of pieces consists of all $V_{j,q}$, $V'_{j,q}$, S_j, where $j = 1, \ldots, m$ and $q = 1, 2, 3$, and $A_{i,k}$, where $i = 1, \ldots, n$ and $k = 1, \ldots, m_i$. The colors are ℓ, $t_{j,q}$, $t'_{j,q}$, $f_{j,q}$, s_j, and $a_{i,k}$, where $k = 1, \ldots, m_i - 1$. We define and name the different classes of pieces as follows:

Value. $V_{j,q} := (f_{j,q}, t_{j,q}, s_j, \ell)$; $V'_{j,q} = (\ell, t'_{j,q}, f_{j,q}, \ell)$.
Satisfying. $S_j := (s_j, s_j, s_j, s_j)$.
Accordance. If $m_i = 1$ then $A_{i,1} := (\ell, \ell, \ell, \ell)$. Otherwise, let the k-th occurrence of x_i be at position $q_k \in \{1, \ldots, 3\}$ in clause c_{j_k}, $k = 1, \ldots, m_i$. Then define $A_{i,k} := (t'_{j_k,q_k}, a_{i,k-1}, t_{j_k,q_k}, a_{i,k})$, where $a_{i,0} := a_{i,m_i} := \ell$.

This completes the transformation, which is obviously polynomial-time computable.

Rotations can deactivate up to two colors from a piece, namely those that are facing the border and thus need not be matched. Active colors must occur an even number of times in a solved board. When placing a piece $V_{j,q}$, either $f_{j,q}$ and s_j, or $t_{j,q}$ and ℓ will be deactivated. Actually, since $f_{j,q}$ only appears in $V_{j,q}$ and $V'_{j,q}$, a solution requires that these pieces either both deactivate $f_{j,q}$ or that both activate this color. If they both activate $f_{j,q}$ (which also activates s_j once), they will be adjacent in a solution. Later, active $f_{j,q}$ will model that the q-th variable in c_j is false, and inactive $f_{j,q}$ will model a true setting. Since s_j appears only in $V_{j,1}, V_{j,2}, V_{j,3}$ and S_j, but cannot be deactivated in S_j, where it will be used twice, it is necessary for a solution that an even, positive number of pieces from $V_{j,1}, V_{j,2}, V_{j,3}$ use s_j. This forces two of the variables in c_j to a "false" and one to a "true" setting.

Finally, the interplay of the $t_{j,q}$ and $t'_{j,q}$ within the V- and A-pieces will ensure the consistency of the truth assignment. We fix a variable x_i and assume $m_i > 1$ as there is nothing to show otherwise. Since color $a_{i,k}$, $k = 1, \ldots, m_i - 1$, is only used in $A_{i,k}$ and $A_{i,k+1}$, these two pieces must either both be rotated to activate the t and t' colors on their edges, or the pieces must be adjacent, which inductively requires that the solution contains all A_i-pieces as a chain $A_{i,1}, A_{i,2}, \ldots, A_{i,m_i}$ in this or reverse order. In the case of a chain, the colors t_{j_k,q_k} and t'_{j_k,q_k} must be inactive for all $k = 1, \ldots, m_i$, which means that the pieces V_{j_k,q_k} and V'_{j_k,q_k} are adjacent by virtue of color f_{j_k,q_k}, hence a "false" setting of x_i. Otherwise, the colors are all active, which is (by the uniqueness of the colors) only possible if $A_{i,k}$ is fit between the pieces V_{j_k,q_k} and V'_{j_k,q_k}, both of which are rotated to a "true" setting. Hence, all m_i occurrences of x_i must be consistent.

We now prove that there is a solution to Monotone 1-in-3-SAT if and only if there is one to the puzzle. Let us first prove the implication $\Rightarrow$, i. e., we consider an assignment to $x_1, \ldots, x_n$ which satisfies exactly one variable in each clause. We lay down the pieces as subsequences in clause-wise order. When considering a clause $c_j = (u_{j,1} \vee u_{j,2} \vee u_{j,3})$, $j = 1, \ldots, m$, let $q \in \{1, \ldots, 3\}$ be the index of its unique satisfied variable, say this variable is x_i in its k-th occurence. The subsequence for c_j starts with $V_{j,q}$ rotated by 270° (left-hand side has color ℓ), followed by $A_{i,k}$ unrotated and $V'_{j,q}$ rotated by 90°. Note that the right-hand side is ℓ. Let q' and q'' be the indices of the other two unsatisfied variables in c_j. We proceed by placing $V'_{j,q'}$ rotated by 180°, then $V_{j,q'}$ rotated by 180° and afterwards S_j. The construction for clause j is continued by placing $V_{j,q''}$ and $V'_{j,q''}$ unrotated, which again ends in color ℓ. Note that we have placed all $V_{j,\cdot}$-pieces and the S_j-piece as well as a single A-piece corresponding to the occurrence of the satisfied variable. Finally, if they have not been placed before, we use all A-pieces corresponding to the unsatisfied variables as follows: If $u_{j,q'} = x_r$ then $A_{r,1}, A_{r,2}, \ldots, A_{r,m_r}$ are appended in the chained way described above (again ending in ℓ), completed by the chain for variable $u_{j,q''}$. The construction ends with ℓ on the right-hand side and is continued with the next clause, resulting in all pieces being used and the puzzle being solved.

For the implication $\Leftarrow$, assume now that the puzzle is solved. We have already argued that for any j, there must be exactly one q such that the pair $(V_{j,q}, V'_{j,q})$ is rotated according a "true" setting and two other q such that the pair is in a "false" setting. We set the variables in c_j accordingly. If a variable contains in another clause, we already know that the corresponding pieces must be rotated in a consistent way, which proves that we have a solution to Monotone 1-in-3-SAT. □

3 Boards with at Least Two Rows

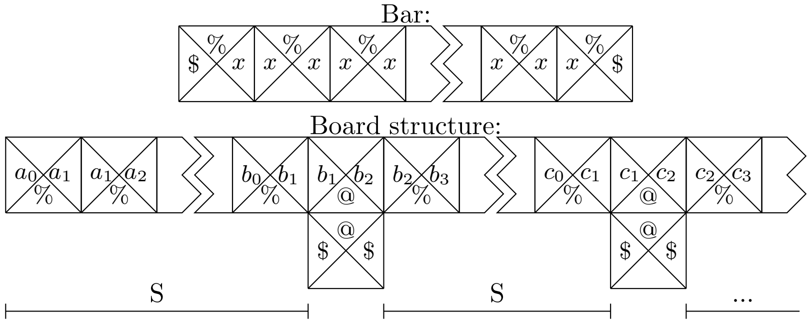
We consider boards with two rows and arbitrarily many columns. Of course, the roles of columns and rows can be interchanged.

In [5] edge-matching is shown to be NP-complete for quadratic boards. Recently, [6] showed in a much more involved proof that the problem is even

APX-complete, already for rectangular boards with only two rows. We focus on Demaine's and Demaine's technique from [5], which does not use rotations, and show that it can be strengthened to also include boards with row-count two – by extension edge-matching with any width/height ratio is NP-complete, since it is trivial to force a board to contain more rows by adding uniformly colored pieces. However, as stated in Section 2, edge-matching without rotation is efficiently solvable when the row-count is 1.

Theorem 3. *Two-row edge matching puzzles with swaps and free or monochromatic border is NP-complete.*

Proof. The transformation is from 3-partition in which the task is to partition a set of $3m$ positive integers into m sets, each consisting of 3 integers such that the integers in each set sum to the same value S. This problem can be visualized as the task of, given a collection of bars of varying length, placing the bars in rows, such that each row has 3 bars and all rows have the width S. Importantly 3-partition is also NP-complete when all the integers are limited to values in the range $(S/4, S/2)$, meaning that any row with width S must contain exactly 3 bars – this is the version of the problem used in this proof. The problem remains NP-complete when S and m are polynomially related.



Converting this problem into an edge-matching puzzle with height 2 proceeds as follows: A section of pieces (a 'bar') is defined for each of the $3m$ integers. The internal left-right edge-color (shown as 'x' in the figure) is unique for each bar, and every bar starts and ends on the color '\$'. A board with height $N = 2$ and width $M = mS + m - 1$ is constructed, where the upper row is forced to have a particular layout as illustrated in the figure by using unique colors for every edge pair, and the lower row is separated into areas of length S, each of which can contain 3 bars. All separators will fit any bar left and right (color '\$'), and all bars will fit in any position below the fixed upper row (color '%'). If the 3-partition has a solution then it will be possible to place the m bars into the board giving a solution to the edge matching puzzle. If the edge matching puzzle can be solved it will be because all sections can be placed into the forced layout of the grid, meaning that there is a solution to the corresponding 3-partition problem. □

4 Number of Solutions

This section presents an analysis of the number of solutions depending on the number NM of pieces and the number K of colors. This also lead to a conjecture of what settings of these parameters will result in hard instances. Below, some results are stated for boards with cyclic border, swaps and flipping and no rotations; we refer to the full version of the paper for detailed derivation.

For small values of NM and K the results are listed in Table 1. The instances were small enough to count the number of solvable and unsolvable boards by complete enumeration. Obviously a higher color count K leads to a much faster growth in number of combinations for increasing NM. It also seems intuitive that higher K results in a higher proportion of unsolvable boards since the number of ways colors can be combined without leading to solvable boards increases.

Table 1. Development of the number of single-row boards with given M and K and of the number of these that are solvable. This table is generated by enumerating all combinations for each pair (M, K) and deciding for each if a solution exists.

Width (M)	Combinations			Solvable/Unsolvable		
	K=2	K=3	K=4	K=2	K=3	K=4
1	3	6	10	2/1	3/3	4/6
2	6	21	55	3/3	6/15	10/45
3	10	56	220	4/6	10/46	20/200
4	15	126	715	6/9	21/105	55/660
5	21	252	2002	8/13	39/213	136/1866
6	28	462	5005	11/17	74/388	346/4659
7	36	792	11440	14/22	129/663	812/10628
8	45	1287	24310	18/27	219/1068	1823/22487
9	55	2002	48620	22/33	351/1651	3832/44788
10	66	3003	92378	27/39	546/2457	7666/84712

By counting Euler paths single-row boards ($N = 1$) and swaps only (no rotations) the expected number of solutions E_{1swap} can be shown to be

$$E_{1swap} = \frac{M!}{K^M} \tag{1}$$

For multi-row boards ($N > 1$) and swaps only (no rotations) the expected number of solutions E_{2swap} can be shown to be

$$E_{2swap} = \frac{(NM)!}{K^{2NM}} \tag{2}$$

For a given N, M, if K is "too small" then E_{1swap}, E_{2swap} are very large, and if K is "too large" then these quantities become small. Analyzing the expression for E_{1swap} and E_{2swap} we conclude that there is an almost linear relation between NM and K that yield boards which have a small number of solutions. The area where the expected number of solutions is close to 1 is known as a phase transition. For NP-hard problems the hardest variants are generally found around this phase transition [9]. For edge-matching this seems intuitive: if the number K of colors is low relative to the board size NM, then there are many opportunities for fitting pieces together and forming partial solutions (i.e. the problems are under-constrained) - if K is very small then the problems will tend to become trivial. Conversely if K is large relative to NM, there will be very few ways of forming partial solutions (i.e. the problems are over-constrained). When K becomes very large (almost NM) it tends to become trivial again, in the sense that no solutions can exist for a particular instance. By setting $E_{1swap} = 1$ and $E_{2swap} = 1$ in (1) and (2), respectively, and solving for K we derive the color count values which are likely to result in hard puzzles. These are after some algebraic simplifications:

$$K^*_{1Dswap} \sim \frac{M}{e} \qquad K^*_{2Dswap} \sim \sqrt{\frac{NM}{e}}$$

For multi-row boards with rotations and swapping the corresponding numbers are

$$E_{2Drotswap} \sim \frac{(NM)!\,4^{NM}}{K^{2NM}} \qquad K^*_{2Drotswap} \sim 2\sqrt{\frac{NM}{e}} \tag{3}$$

The Eternity II puzzle mentioned above used a 16×16 board with 22 colors (only 17 on the non-border pieces), while Formula (3) with $N = M = 16$ yields $K^*_{2rotswap} \sim 19.6$.

Finally for the multi-row case of rotation without swapping:

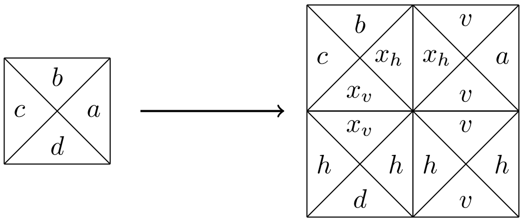
$$E_{2Drot} = \frac{4^{NM}}{K^{2NM}}$$

Which means that setting $E_{2Drot} = 1$ will simply yield $K^*_{2Drot} = 2$. Experimentation confirms that boards with two colors (and free border-rule) are indeed much more difficult to solve than boards with larger number of colors - both with respect to success-rate of attempted heuristic algorithms, average running time of a back-tracking algorithm and indeed even for a human player. We say more about heuristic approaches in Section 6.

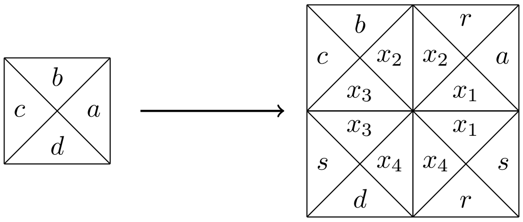
5 Simulating Other Puzzles

It turns out, that there are a number simulations that allow one border rule to be replaced by a different one, or which can prevent rotations and other kinds of edge-permutations. These simulations are implemented by replacing pieces by "super-pieces" composed of a number of normal pieces having also additional colors. Below are given two examples. Details will be given in the full paper.

An edge-matching instance where rotation is forbidden can be simulated by an instance where rotation is allowed by exchanging each piece $p_x = (a, b, c, d)$ with the following construct:



Similarly an edge-matching instance where rotation is forbidden can be simulated by an instance where flipping is allowed by exchanging each piece $p_x = (a, b, c, d)$ with the following construct:



Note that the above-described constructions increase the number of rows, columns and colors.

Also puzzles with hexagonal or triangular pieces can be simulated by puzzles with quadratic pieces, as shown in the images below. The construction introduces a number of new colors, all of which are shown as a single color (pink in the electronic version) in the pictures.

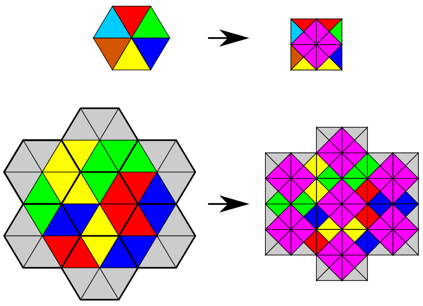


Fig. 1. On top a sketch simulation of a single hexagonal piece by an arrangement of four quadratic ones is shown. Below is the resulting simulation of a board with 7 hexagonal pieces with monochrome border.

6 Experiments

On the experimental side we have also implemented a number of heuristics, namely backtracking, local search and evolutionary algorithms, for solving the

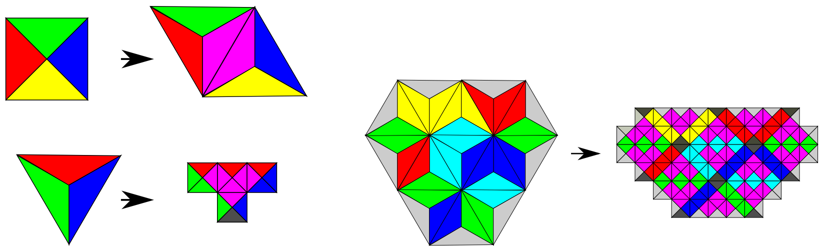


Fig. 2. Sketch of a simulation of square pieces by triangular ones (top left) and vice versa (bottom left). A simulation of a larger puzzle with triangular pieces is shown.

various types of edge-matching problems. The results achieved so far indicate that an "almost solution" (one with a small fraction non-matching edges) can be found in surprisingly fast, however, complete solutions are hard to find for most parameter settings. The tests were run on randomly generated instances, where the color of each edge was chosen under uniform distribution. Afterwards a random permutation was applied to the pieces, or every piece was randomly rotated, or both.

The experiments are continuing and detailed results will be given in the full paper.

7 Summary and Conclusions

Motivated by the Eternity-II puzzle, the computational complexity of edge-matching problems has recently gained increasing interest. In this paper, we have focused on several problem aspects whose impact on complexity was unknown or only marginally treated before, in particular rotating pieces, in-place pieces, border rules, number of colors and shapes of pieces. With regard to single-row boards, it has been shown that introducing rotations makes the otherwise easy problem NP-hard. Furthermore, we have studied the expected number of solutions for single- and multi-row boards and studied the impact of the number of colors. Finally, we have argued why the problem is polynomial-time equivalent under different border rules and shapes of pieces.

The table below summarizes the known results for boards with one row compared to boards with at least two rows.

Rows:	1	≥ 2
Swap	P	NP-complete
Rotation	P	NP-complete
Both	NP-complete	NP-complete

This paper has raised further questions concerning the hardness of edge-matching problems. For example, is the problem still hard for a fixed number of colors? Furthermore, it is unknown whether the single-row case with rotations is hard to approximate.

References

1. Pevzner, P., Tang, H., Waterman, M.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences of the United States of America 98(17), 9748–9753 (2001)
2. Lesniak, L., Oellermann, O.R.: An Eulerian exposition. Journal of Graph Theory 10, 277–297 (1986)
3. Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
4. Goles, E., Rapaport, I.: Complexity of tile rotation problems. Theoretical Computer Science 188, 129–159 (1997)
5. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics 23, 195–208 (2007)
6. Antoniadis, A., Lingas, A.: Approximability of edge matching puzzles. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 153–164. Springer, Heidelberg (2010)
7. Ebbesen, M.: Analysis of restricted edge-matching problems (2011) Master thesis, Technical University of Denmark, reference no. IMM-M.Sc.-2011-08, http://tinyurl.com/6jcf7kf
8. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226. ACM, New York (1978)
9. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems, pp. 331–337. Morgan Kaufmann, San Francisco (1991)

On the Link between Strongly Connected Iteration Graphs and Chaotic Boolean Discrete-Time Dynamical Systems

Jacques M. Bahi[1], Jean-Francois Couchot[1], Christophe Guyeux[1], and Adrien Richard[2]

[1] Computer Science Laboratory (LIFC), University of Franche-Comté, France
{jacques.bahi,jean-francois.couchot,christophe.guyeux}@univ-fcomte.fr
[2] I3S, UMR CNRS 6070 Sophia Antipolis, France
richard@i3s.unice.fr

Abstract. Chaotic functions are characterized by sensitivity to initial conditions, transitivity, and regularity. Providing new functions with such properties is a real challenge. This work shows that one can associate with any Boolean network a continuous function, whose discrete-time iterations are chaotic if and only if the iteration graph of the Boolean network is strongly connected. Then, sufficient conditions for this strong connectivity are expressed on the interaction graph of this network, leading to a constructive method of chaotic function computation. The whole approach is evaluated in the chaos-based pseudo-random number generation context.

Keywords: Boolean network, discrete-time dynamical system, topological chaos.

1 Introduction

Chaos has attracted a lot of attention in various domains of science and engineering, *e.g.*, hash function [1], steganography [2], pseudo random number generation [3]. All these applications capitalize fundamental properties of the chaotic maps [4], namely: sensitive dependence on initial conditions, transitivity, and density of periodic points. A system is sensitive to initial conditions if any point contains, in any neighborhood, another point with a completely different future trajectory. Topological transitivity is established when, for any element, any neighborhood of its future evolution eventually overlaps with any other open set. On the contrary, a dense set of periodic points is an element of regularity that a chaotic dynamical system has to exhibit.

Chaotic discrete-time dynamical systems are iterative processes defined by a chaotic map f from a domain E to itself. Starting from any configurations $x \in E$, the system produces the sequence $x, f(x), f^2(x), f^3(x), \dots$, where $f^k(x)$ is the k-th iterate of f at x. Works referenced above are instances of that scheme: they iterate *tent* or *logistic* maps known to be chaotic on $\mathbb{R}$.

As far as we know, no result so far states that the chaotic properties of a function that has been theoretically proven on $\mathbb{R}$ remain valid on the floating-point numbers, which is the implementation domain. Thus, to avoid the loss of chaos this work presents an alternative: to construct, from Boolean networks $f : \mathbb{B}^n \to \mathbb{B}^n$, continuous fonctions G_f defined on the domain $[\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$, where $[\![1;n]\!]$ is the interval of integers $\{1, 2, \ldots, n\}$ and $\mathbb{B}$ is the Boolean domain $\{0, 1\}$. Due to the discrete nature of f, theoretical results obtained on G_f are preserved in implementation.

Furthermore, instead of finding an example of such maps and to prove the chaoticity of their discrete-time iterations, we tackle the problem of characterizing all the maps with chaotic iterations according to Devaney's chaos definition [4]. This is the first contribution. This characterization is expressed on the asynchronous iteration graph of the Boolean map f, which contains 2^n vertices. To extend the applicability of this characterization, sufficient conditions that ensure this chaoticity are expressed on the interaction graph of f, which only contains n vertices. This is the second contribution. Starting thus with an interaction graph with required properties, all the maps resulting from a Boolean network constructed on this graph have chaotic iterations. Eventually, the approach is applied on a pseudo random number generation (PRNG). Uniform distribution of the output, which is a necessary condition for PRNGs is then addressed. Functions with such property are thus characterized again on the asynchronous iteration graph. This is the third contribution. The relevance of the approach and the application to pseudo random number generation are evaluated thanks to a classical test suite.

The rest of the paper is organized as follows. Section 2 recalls discrete-time Boolean dynamical systems. Their chaoticity is characterized in Sect. 3. Sufficient conditions to obtain chaoticity are presented in Sect. 4. The application to pseudo random number generation is formalized, maps with uniform output are characterized, and PRNGs are evaluated in Sect. 5. The paper ends with a conclusion section where intended future work is presented.

2 Preliminaries

Let n be a positive integer. A Boolean network is a discrete dynamical system defined from a *Boolean map*

$$f : \mathbb{B}^n \to \mathbb{B}^n, \qquad x = (x_1, \ldots, x_n) \mapsto f(x) = (f_1(x), \ldots, f_n(x)),$$

and an *iteration scheme* (*e.g.*, parallel, sequential, asynchronous...). For instance, with the parallel iteration scheme, given an initial configuration $x^0 \in \mathbb{B}^n$, the dynamics of the system are described by the recurrence $x^{t+1} = f(x^t)$. The retained scheme only modifies one element at each iteration and is further referred by *asynchronous*. In other words, at the t^{th} iteration, only the s_t-th component is "iterated", where $s = (s_t)_{t \in \mathbb{N}}$ is a sequence of indices taken in $[\![1;n]\!]$ called "strategy". Formally, let $F_f : [\![1;n]\!] \times \mathbb{B}^n$ to $\mathbb{B}^n$ be defined by

$$F_f(i, x) = (x_1, \ldots, x_{i-1}, f_i(x), x_{i+1}, \ldots, x_n).$$

With the asynchronous iteration scheme, given an initial configuration $x^0 \in \mathbb{B}^n$ and a strategy $s \in [\![1;n]\!]^{\mathbb{N}}$, the dynamics of the network are described by the recurrence

$$x^{t+1} = F_f(s_t, x^t). \tag{1}$$

Let G_f be the map from $[\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$ to itself defined by

$$G_f(s,x) = (\sigma(s), F_f(s_0, x)),$$

where $\forall t \in \mathbb{N}, \sigma(s)_t = s_{t+1}$. The parallel iteration of G_f from an initial point $X^0 = (s, x^0)$ describes the "same dynamics" as the asynchronous iteration of f induced by x^0 and the strategy s (this is why G_f has been introduced).

Consider the space $\mathcal{X} = [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$. The distance d between two points $X = (s,x)$ and $X' = (s',x')$ in $\mathcal{X}$ is defined by

$$d(X,X') = d_H(x,x') + d_S(s,s'), \text{ where } \begin{cases} d_H(x,x') = \displaystyle\sum_{i=1}^{n} |x_i - x'_i| \\ d_S(s,s') = \dfrac{9}{n} \displaystyle\sum_{t\in\mathbb{N}} \dfrac{|s_t - s'_t|}{10^{t+1}}. \end{cases}$$

Thus, $\lfloor d(X,X') \rfloor = d_H(x,x')$ is the Hamming distance between x and x', and $d(X,X') - \lfloor d(X,X') \rfloor = d_S(s,s')$ measures the differences between s and s'. More precisely, this floating part is less than 10^{-k} if and only if the first k terms of the two strategies are equal. Moreover, if the k^{th} digit is nonzero, then $s_k \neq s'_k$.

Let f be any map from $\mathbb{B}^n$ to itself, and $\neg : \mathbb{B}^n \to \mathbb{B}^n$ defined by $\neg(x) = (\overline{x_1}, \dots, \overline{x_n})$. Considering this distance d on $\mathcal{X}$, it has already been proven that [5]:

- G_f is *continuous*,
- the parallel iteration of $G_\neg$ is *regular* (periodic points of $G_\neg$ are dense in $\mathcal{X}$),
- $G_\neg$ is *topologically transitive* (for all $X, Y \in \mathcal{X}$, and for all open balls B_X and B_Y centered in X and Y respectively, there exist $X' \in B_X$ and $t \in \mathbb{N}$ such that $G_\neg^t(X') \in B_Y$),
- $G_\neg$ has *sensitive dependence on initial conditions* (there exists $\delta > 0$ such that for any $X \in \mathcal{X}$ and any open ball B_X, there exist $X' \in B_X$ and $t \in \mathbb{N}$ such that $d(G_\neg^t(X), G_\neg^t(X')) > \delta$).

Particularly, $G_\neg$ is *chaotic*, according to the Devaney's definition recalled below:

Definition 1 (Devaney [4]). *A continuous map f on a metric space $(\mathcal{X}, d)$ is chaotic if it is regular, sensitive, and topologically transitive.*

In other words, quoting Devaney in [4], a chaotic dynamical system "is unpredictable because of the sensitive dependence on initial conditions. It cannot be broken down or simplified into two subsystems which do not interact because of topological transitivity. And in the midst of this random behavior, we nevertheless have an element of regularity". Let us finally remark that the definition above is redundant: Banks *et al.* have proven that sensitivity is indeed implied by regularity and transitivity [6].

3 Characterization of Chaotic Discrete-Time Dynamical Systems

In this section, we give a characterization of Boolean networks f making the iterations of any induced map G_f chaotic. This is achieved by establishing inclusion relations between the transitive, regular, and chaotic sets defined below:

- $\mathcal{T} = \{f : \mathbb{B}^n \to \mathbb{B}^n / G_f \text{ is transitive}\}$,
- $\mathcal{R} = \{f : \mathbb{B}^n \to \mathbb{B}^n / G_f \text{ is regular}\}$,
- $\mathcal{C} = \{f : \mathbb{B}^n \to \mathbb{B}^n / G_f \text{ is chaotic (Devaney)}\}$.

Let f be a map from $\mathbb{B}^n$ to itself. The *asynchronous iteration graph* associated with f is the directed graph $\Gamma(f)$ defined by: the set of vertices is $\mathbb{B}^n$; for all $x \in \mathbb{B}^n$ and $i \in [\![1;n]\!]$, the graph $\Gamma(f)$ contains an arc from x to $F_f(i,x)$. The relation between $\Gamma(f)$ and G_f is clear: there exists a path from x to x' in $\Gamma(f)$ if and only if there exists a strategy s such that the parallel iteration of G_f from the initial point (s,x) reaches the point x'. Finally, in what follows the term *iteration graph* is a shortcut for asynchronous iteration graph.

We can thus characterize $\mathcal{T}$:

Proposition 1. *G_f is transitive if and only if $\Gamma(f)$ is strongly connected.*

Proof. $\Longleftarrow$ Suppose that $\Gamma(f)$ is strongly connected. Let (s,x) and (s',x') be two points of $\mathcal{X}$, and let $\varepsilon > 0$. We will define a strategy $\tilde{s}$ such that the distance between $(\tilde{s},x)$ and (s,x) is less than ε, and such that the parallel iterations of G_f from $(\tilde{s},x)$ reaches the point (s',x').

Let $t_1 = \lfloor -\log_{10}(\varepsilon)\rfloor$, and let x'' be the configuration of $\mathbb{B}^n$ that we obtain from (s,x) after t_1 iterations of G_f. Since $\Gamma(f)$ is strongly connected, there exists a strategy s'' and $t_2 \in \mathbb{N}$ such that, x' is reached from (s'',x'') after t_2 iterations of G_f.

Now, consider the strategy $\tilde{s} = (s_0, \ldots, s_{t_1-1}, s''_0, \ldots, s''_{t_2-1}, s'_0, s'_1, s'_2, s'_3 \ldots)$. It is clear that (s',x') is reached from $(\tilde{s},x)$ after t_1+t_2 iterations of G_f, and since $\tilde{s}_t = s_t$ for $t < t_1$, by the choice of t_1, we have $d((s,x),(\tilde{s},x)) < \epsilon$. Consequently, G_f is transitive.

$\Longrightarrow$ If $\Gamma(f)$ is not strongly connected, then there exist two configurations x and x' such that $\Gamma(f)$ has no path from x to x'. Let s and s' be two strategies, and let $0 < \varepsilon < 1$. Then, for all (s'',x'') such that $d((s'',x''),(s,x)) < \varepsilon$, we have $x'' = x$, so that iteration of G_f from (s'',x'') only reaches points in $\mathcal{X}$ that are at a greater distance than one with (s',x'). So G_f is not transitive.

We now prove that:

Proposition 2. *$\mathcal{T} \subset \mathcal{R}$.*

Proof. Let $f : \mathbb{B}^n \to \mathbb{B}^n$ such that G_f is transitive (f is in $\mathcal{T}$). Let $(s,x) \in \mathcal{X}$ and $\varepsilon > 0$. To prove that f is in $\mathcal{R}$, it is sufficient to prove that there exists a strategy $\tilde{s}$ such that the distance between $(\tilde{s},x)$ and (s,x) is less than ε, and such that $(\tilde{s},x)$ is a periodic point.

Let $t_1 = \lfloor -\log_{10}(\varepsilon) \rfloor$, and let x' be the configuration that we obtain from (s, x) after t_1 iterations of G_f. According to the previous proposition, $\Gamma(f)$ is strongly connected. Thus, there exists a strategy s' and $t_2 \in \mathbb{N}$ such that x is reached from (s', x') after t_2 iterations of G_f.

Consider the strategy $\tilde{s}$ that alternates the first t_1 terms of s and the first t_2 terms of s': $\tilde{s} = (s_0, \dots, s_{t_1-1}, s'_0, \dots, s'_{t_2-1}, s_0, \dots, s_{t_1-1}, s'_0, \dots, s'_{t_2-1}, s_0, \dots)$. It is clear that $(\tilde{s}, x)$ is obtained from $(\tilde{s}, x)$ after $t_1 + t_2$ iterations of G_f. So $(\tilde{s}, x)$ is a periodic point. Since $\tilde{s}_t = s_t$ for $t < t_1$, by the choice of t_1, we have $d((s, x), (\tilde{s}, x)) < \epsilon$.

Remark 1. Inclusion of proposition 2 is strict, due to the identity map (which is regular, but not transitive).

We can thus conclude that $\mathcal{C} = \mathcal{R} \cap \mathcal{T} = \mathcal{T}$, which leads to the following characterization:

Theorem 1. *Let $f : \mathbb{B}^n \to \mathbb{B}^n$. G_f is chaotic (according to Devaney) if and only if $\Gamma(f)$ is strongly connected.*

4 Generating Strongly Connected Iteration Graph

The previous section has shown the interest of strongly connected iteration graphs. This section presents two approaches to generate functions with such property. The first is algorithmic (Sect. 4.1) whereas the second gives a sufficient condition on the interaction graph of the Boolean map f to get a strongly connected iteration graph (Sect. 4.2).

4.1 Algorithmic Generation of Strongly Connected Graphs

This section presents a first solution to compute a map f with a strongly connected graph of iterations $\Gamma(f)$. It is based on a generate and test approach.

We first consider the negation function $\neg$ whose iteration graph $\Gamma(\neg)$ is obviously strongly connected. Given a graph Γ, initialized with $\Gamma(\neg)$, the algorithm iteratively does the two following stages:

1. randomly select an edge of the current iteration graph Γ and
2. check whether the current iteration graph without that edge remains strongly connected (by a Tarjan algorithm [7], for instance). In the positive case the edge is removed from Γ,

until a rate r of removed edges is greater than a threshold given by the user. If r is close to 0% (*i.e.*, few edges are removed), there should remain about $n \times 2^n$ edges. In the opposite case, if r is close to 100%, there are about 2^n edges left. In all cases, this step returns the last graph Γ that is strongly connected. It is now then obvious to return the function f whose iteration graph is Γ.

Even if this algorithm always returns functions with stroncly connected component (SCC) iteration graph, it suffers from iteratively verifying connexity on the whole iteration graph, *i.e.*, on a graph with 2^n vertices. Next section tackles this problem: it presents sufficient conditions on a graph reduced to n elements that allow to obtain SCC iteration graph.

4.2 Sufficient Conditions to Strongly Connected Graph

We are looking for maps f such that interactions between x_i and f_j make its iteration graph $\Gamma(f)$ strongly connected. We first need additional notations and definitions. For $x \in \mathbb{B}^n$ and $i \in [\![1;n]\!]$, we denote by $\overline{x}^i$ the configuration that we obtain be switching the $i-$th component of x, that is, $\overline{x}^i = (x_1, \ldots, \overline{x_i}, \ldots, x_n)$. Information interactions between the components of the system are obtained from the *discrete Jacobian matrix* f' of f, which is defined as being the map which associates to each configuration $x \in \mathbb{B}^n$, the $n \times n$ matrix

$$f'(x) = (f_{ij}(x)), \qquad f_{ij}(x) = \frac{f_i(\overline{x}^j) - f_i(x)}{\overline{x}^j_j - x_j} \qquad (i, j \in [\![1;n]\!]).$$

More precisely, interactions are represented under the form of a signed directed graph $G(f)$ defined by: the set of vertices is $[\![1;n]\!]$, and there exists an arc from j to i of sign $s \in \{-1, 1\}$, denoted (j, s, i), if $f_{ij}(x) = s$ for at least one $x \in \mathbb{B}^n$. Note that the presence of both a positive and a negative arc from one vertex to another is allowed.

Let P be a sequence of arcs of $G(f)$ of the form

$$(i_1, s_1, i_2), (i_2, s_2, i_3), \ldots, (i_r, s_r, i_{r+1}).$$

Then, P is said to be a path of $G(f)$ of length r and of sign $\Pi_{i=1}^r s_i$, and i_{r+1} is said to be reachable from i_1. P is a *circuit* if $i_{r+1} = i_1$ and if the vertices $i_1, \ldots i_r$ are pairwise distinct. A vertex i of $G(f)$ has a positive (resp. negative) *loop*, if $G(f)$ has a positive (resp. negative) arc from i to itself.

Let $\alpha \in \mathbb{B}$. We denote by f^α the map from $\mathbb{B}^{n-1}$ to itself defined for any $x \in \mathbb{B}^{n-1}$ by

$$f^\alpha(x) = (f_1(x, \alpha), \ldots, f_{n-1}(x, \alpha)).$$

We denote by $\Gamma(f)^\alpha$ the subgraph of $\Gamma(f)$ induced by the subset $\mathbb{B}^{n-1} \times \{\alpha\}$ of $\mathbb{B}^n$. Let us give and prove the following technical lemma:

Lemma 1. *$G(f^\alpha)$ is a subgraph of $G(f)$: every arc of $G(f^\alpha)$ is an arc of $G(f)$. Furthermore, if $G(f)$ has no arc from n to a vertex $i \neq n$, then $G(f^\alpha) = G(f) \backslash n$: one obtains $G(f^\alpha)$ from $G(f)$ by removing vertex n as well as all the arcs with n as initial or final vertex.*

Proof. Suppose that $G(f^\alpha)$ has an arc from j to i of sign s. By definition, there exists $x \in \mathbb{B}^{n-1}$ such that $f^\alpha_{ij}(x) = s$, and since it is clear that $f^\alpha_{ij}(x) = f_{ij}(x, \alpha)$, we deduce that $G(f)$ has an arc from j to i of sign s. This proves the first assertion. To demonstrate the second assertion, it is sufficient to prove that if $G(f)$ has an arc from i to j of sign s, with $i, j \neq n$, then $G(f^\alpha)$ also contains this arc. So suppose that $G(f)$ has an arc from i to j of sign s, with $i, j \neq n$. Then, there exists $x \in \mathbb{B}^{n-1}$ and $\beta \in \mathbb{B}$ such that $f_{ij}(x, \beta) = s$. If $f_{ij}(x, \beta) \neq f_{ij}(x, \alpha)$, then f_i depends on the $n-$th component, in contradiction with the assumptions. So $f_{ij}(x, \alpha) = s$. It is then clear that $f^\alpha_{ij}(x) = s$, that is, $G(f^\alpha)$ has an arc from j to i of sign s.

Lemma 2. *$\Gamma(f^\alpha)$ and $\Gamma(f)^\alpha$ are isomorphic.*

Proof. Let h be the bijection from $\mathbb{B}^{n-1}$ to $\mathbb{B}^{n-1} \times \{\alpha\}$ defined by $h(x) = (x, \alpha)$ for all $x \in \mathbb{B}^{n-1}$. It is easy to see that h is an isomorphism between $\Gamma(f^\alpha)$ and $\Gamma(f)^\alpha$ that is: $\Gamma(f^\alpha)$ has an arc from x to y if and only if $\Gamma(f)^\alpha$ has an arc from $h(x)$ to $h(y)$.

Theorem 2. *Let f be a map from $\mathbb{B}^n$ to itself such that:*

1. *$G(f)$ has no cycle of length at least two;*
2. *every vertex of $G(f)$ with a positive loop has also a negative loop;*
3. *every vertex of $G(f)$ is reachable from a vertex with a negative loop.*

Then, $\Gamma(f)$ is strongly connected.

Proof. By induction on n. Let f be a map from $\mathbb{B}^n$ to itself satisfying the conditions of the statement. If $n = 1$ the result is obvious: according to the third point of the statement, $G(f)$ has a negative loop; so $f(x) = \overline{x}$ and $\Gamma(f)$ is a cycle of length two. Assume that $n > 1$ and that the theorem is valid for maps from $\mathbb{B}^{n-1}$ to itself. According to the first point of the statement, $G(f)$ contains at least one vertex i such that $G(f)$ has no arc from i to a vertex $j \neq i$. Without loss of generality, assume that n is such a vertex. Then, according to Lemma 1, f^0 and f^1 satisfy the conditions of the statement. So, by induction hypothesis, $\Gamma(f^0)$ and $\Gamma(f^1)$ are strongly connected. So, according to Lemma 2, $\Gamma(f)^0$ and $\Gamma(f)^1$ are strongly connected. To prove that $\Gamma(f)$ is strongly connected, it is sufficient to prove that $\Gamma(f)$ contains an arc $x \to y$ with $x_n = 0 < y_n$ and an arc $x \to y$ with $x_n = 1 > y_n$. In other words, it is sufficient to prove that:

$$\forall \alpha \in \mathbb{B},\ \exists x \in \mathbb{B}^n, \qquad x_n = \alpha \neq f_n(x). \tag{$*$}$$

Assume first that n has a negative loop. Then, by the definition of $G(f)$, there exists $x \in \mathbb{B}^n$ such that $f_{nn}(x) < 0$. Consequently, if $x_n = 0$, we have $f_n(x) > f_n(\overline{x}^n)$, so $x_n = 0 \neq f_n(x)$ and $\overline{x}^n_n = 1 \neq f_n(\overline{x}^n)$; and if $x_n = 1$, we have $f_n(x) < f_n(\overline{x}^n)$, so $x_n = 1 \neq f_n(x)$ and $\overline{x}^n_n = 0 \neq f_n(\overline{x}^n)$. In both cases, the condition $(*)$ holds.

Now, assume that n has no negative loop. According to the second point of the statement, n has no loop, *i.e.*, the value of $f_n(x)$ does not depend on the value of x_n. According to the third point of the statement, n is not of in-degree zero in $G(f)$, *i.e.*, f_n is not a constant. Consequently, there exists $x, y \in \mathbb{B}^n$ such that $f_n(x) = 1$ and $f_n(y) = 0$. Let $x' = (x_1, \ldots, x_{n-1}, 0)$ and $y' = (y_1, \ldots, y_{n-1}, 1)$. Since the value of $f_n(x)$ (resp. $f_n(y)$) does not depend on the value of x_n (resp. y_n), we have $f_n(x') = f_n(x) = 1 \neq x'_n$ (resp. $f_n(y') = f_n(y) = 0 \neq y'_n$). So the condition $(*)$ holds, and the theorem is proven.

```
Input: a function $f$, an iteration number $b$, an initial configuration $x^0$ ($n$ bits)
Output: a configuration $x$ ($n$ bits)
$x \leftarrow x^0$;
$k \leftarrow b + XORshift(b+1)$;
for $i = 0, \dots, k-1$ do
    $s \leftarrow XORshift(n)$;
    $x \leftarrow F_f(s, x)$;
end
return $x$;
```

Algorithm 1. PRNG with chaotic functions

5 Application to Pseudo Random Number Generator

This section presents a direct application of the theory developed above.

5.1 Boolean and Chaos Based PRNG

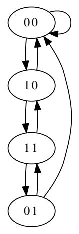
We have proposed in [8] a new family of generators that receives two PRNGs as inputs. These two generators are mixed with chaotic iterations, leading thus to a new PRNG that improves the statistical properties of each generator taken alone. Furthermore, our generator possesses various chaos properties that none of the generators used as input present. This former family of PRNGs was reduced to chaotic iterations of the negation function, *i.e.*, reduced to $G_{\neg}$. However, it is possible to use any function f such that G_f is chaotic (s.t. the graph $\Gamma(f)$ is strongly connected).

```
Input: the internal configuration $z$ (a 32-bit word)
Output: $y$ (a 32-bit word)
$z \leftarrow z \oplus (z \ll 13)$;
$z \leftarrow z \oplus (z \gg 17)$;
$z \leftarrow z \oplus (z \ll 5)$;
$y \leftarrow z$;
return $y$;
```

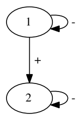
Algorithm 2. An arbitrary round of *XORshift* algorithm

This generator is synthesized in Algorithm 1. It takes as input: a function f; an integer b, ensuring that the number of executed iterations is at least b and at most $2b+1$; and an initial configuration x^0. It returns the new generated configuration x. Internally, it embeds two *XORshift*(k) PRNGs [9] that returns integers uniformly distributed into $[\![1; k]\!]$. *XORshift* is a category of very fast PRNGs designed by George Marsaglia, which repeatedly uses the transform of exclusive or (XOR, $\oplus$) on a number with a bit shifted version of it. This PRNG, which has a period of $2^{32} - 1 = 4.29 \times 10^9$, is summed up in Algorithm 2. It is used in our PRNG to compute the strategy length and the strategy elements.

We are then left to instantiate the function f in Algorithm 1 according to approaches detailed in Sect. 4. Next section shows how the uniformity of distribution has been taken into account.



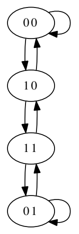
(a) $\Gamma(g)$.



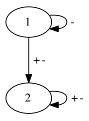
(b) $G(g)$.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

(c) $\check{M}_g$.



(d) $\Gamma(h)$.



(e) $G(h)$.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

(f) $\check{M}_h$.

Fig. 1. Graphs of candidate functions with $n = 2$

5.2 Uniform Distribution of the Output

Let us firstly recall that a stochastic matrix is a square matrix where all entries are nonnegative and all rows sum to 1. A double stochastic matrix is a stochastic matrix where all columns sum to 1. Finally, a stochastic matrix M of size n is regular if $\exists k \in \mathbb{N}^*, \forall i,j \in [\![1;n]\!], M_{ij}^k > 0$. The following theorem is well-known:

Theorem 3. *If M is a regular stochastic matrix, then M has an unique stationary probability vector π. Moreover, if π^0 is any initial probability vector and $\pi^{k+1} = \pi^k.M$ for $k = 0, 1, \dots$ then the Markov chain π^k converges to π as k tends to infinity.*

Let us explain on a small example with 2 elements that the application of such a theorem allows to verify whether the output is uniformly distributed or not. Let then g and h be the two functions from $\mathbb{B}^2$ to itself defined in Fig. 1 and whose iteration graphs are strongly connected. As the *XORshift* PRNG is uniformly distributed, the strategy is uniform on $[\![1,2]\!]$, and each edge of $\Gamma(g)$ and of $\Gamma(h)$ has a probability $1/2$ to be traversed. In other words, $\Gamma(g)$ is the oriented graph of a Markov chain. It is thus easy to verify that the transition matrix of such a process is $M_g = \frac{1}{2}\check{M}_g$, where $\check{M}_g$ is the adjacency matrix given in Fig. 1c, and similarly for M_h.

Both M_g and M_h are (stochastic and) regular since no element is null either in M_g^4 or in M_h^4. Furthermore, the probability vectors $\pi_g = (0.4, 0.1, 0.3, 0.2)$ and $\pi_h = (0.25, 0.25, 0.25, 0.25)$ verify $\pi_g M_g = \pi_g$ and $\pi_h M_h = \pi_h$. Thus, due to Theorem 3, for any initial probability vector π^0, we have $\lim_{k\to\infty} \pi^0 M_g^k = \pi_g$ and $\lim_{k\to\infty} \pi^0 M_h^k = \pi_h$. So the Markov process associated to h tends to the uniform distribution whereas the one associated to g does not. It induces that g shouldn't be iterated in a PRNG. On the contrary, h can be embedded into the PRNG Algorithm 1, provided the number b of iterations between two successive values is sufficiently large so that the Markov process becomes close to the uniform distribution.

Let us first prove the following technical lemma.

Lemma 3. *Let $f : \mathbb{B}^n \to \mathbb{B}^n$, $\Gamma(f)$ its iteration graph, $\check{M}$ the adjacency matrix of $\Gamma(f)$, and M a $n \times n$ matrix defined by $M_{ij} = \frac{1}{n}\check{M}_{ij}$ if $i \neq j$ and $M_{ii} = 1 - \frac{1}{n}\sum\limits_{j=1,j\neq i}^{n} \check{M}_{ij}$ otherwise. Then M is a regular stochastic matrix iff $\Gamma(f)$ is strongly connected.*

Proof. Notice first that M is a stochastic matrix by construction. If there exists k s.t. $M_{ij}^k > 0$ for any $i, j \in [\![1; 2^n]\!]$, the inequality $\check{M}_{ij}^k > 0$ is thus established. Since $\check{M}_{ij}^k$ is the number of paths from i to j of length k in $\Gamma(f)$ and since such a number is positive, thus $\Gamma(f)$ is strongly connected.

Conversely, if $\Gamma(f)$ is SCC, then for all vertices i and j, a path can be found to reach j from i in at most 2^n steps. There exists thus $k_{ij} \in [\![1, 2^n]\!]$ s.t. $\check{M}_{ij}^{k_{ij}} > 0$. As all the multiples $l \times k_{ij}$ of k_{ij} are such that $\check{M}_{ij}^{l\times k_{ij}} > 0$, we can conclude that, if k is the least common multiple of $\{k_{ij} / i, j \in [\![1, 2^n]\!]\}$ thus $\forall i, j \in [\![1, 2^n]\!], \check{M}_{ij}^k > 0$. So, $\check{M}$ and thus M are regular.

With such a material, we can formulate and prove the following theorem.

Theorem 4. *Let $f : \mathbb{B}^n \to \mathbb{B}^n$, $\Gamma(f)$ its iteration graph, $\check{M}$ its adjacency matrix and M a $n \times n$ matrix defined as in the previous lemma. If $\Gamma(f)$ is SCC then the output of the PRNG detailed in Algorithm 1 follows a law that tends to the uniform distribution if and only if M is a double stochastic matrix.*

Proof. M is a regular stochastic matrix (Lemma 3) that has a unique stationary probability vector (Theorem 3). Let π be $\left(\frac{1}{2^n}, \ldots, \frac{1}{2^n}\right)$. We have $\pi M = \pi$ iff the sum of values of each column of M is one, *i.e.*, iff M is double stochastic.

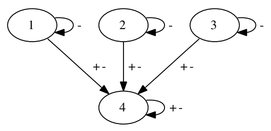
5.3 Experiments

Let us consider the interaction graph $G(f)$ given in Fig. 2a. It verifies Theorem 2: all the functions f whose interaction graph is $G(f)$ have then a strongly connected iteration graph $\Gamma(f)$. Practically, a simple constraint solving has found 520 non isomorphic functions and only 16 of them have a double stochastic matrix. Figure 2b synthesizes them by defining the images of 0,1,2,…,14,15. Let e_j be the unit vector in the canonical basis, the third column gives

$$\max_{j\in[\![1,2^n]\!]} \{\min\{k \mid k \in \mathbb{N}, \left|\left|\pi_j M_f^k - \pi\right|\right|_2 < 10^{-4}\}\}\text{where } \pi_j \text{ is } 1/n.e_j,$$

that is the smallest iteration number that is sufficient to obtain a deviation less than 10^{-4} from the uniform distribution. Such a number is the parameter b in Algorithm 1.

Quality of produced random sequences have been evaluated with the NIST Statistical Test Suite SP 800-22 [10]. For all 15 tests of this battery, the significance level α is set to 1%: a *p-value* which is greater than 0.01 is equivalent that the keystream is accepted as random with a confidence of 99%. Synthetic results in Table. 1 show that all these functions successfully pass this statistical battery of tests.



(a) Interaction Graph

Name	Function image	b
$\mathcal{F}_1$	14, 15, 12, 13, 10, 11, 8, 9, 6, 7, 4, 5, 2, 3, 1, 0	206
$\mathcal{F}_2$	14, 15, 12, 13, 10, 11, 8, 9, 6, 7, 5, 4, 3, 2, 0, 1	94
$\mathcal{F}_3$	14, 15, 12, 13, 10, 11, 8, 9, 6, 7, 5, 4, 3, 2, 1, 0	69
$\mathcal{F}_4$	14, 15, 12, 13, 10, 11, 9, 8, 6, 7, 5, 4, 3, 2, 0, 1	56
$\mathcal{F}_5$	14, 15, 12, 13, 10, 11, 9, 8, 6, 7, 5, 4, 3, 2, 1, 0	48
$\mathcal{F}_6$	14, 15, 12, 13, 10, 11, 9, 8, 7, 6, 4, 5, 2, 3, 0, 1	86
$\mathcal{F}_7$	14, 15, 12, 13, 10, 11, 9, 8, 7, 6, 4, 5, 2, 3, 1, 0	58
$\mathcal{F}_8$	14, 15, 12, 13, 10, 11, 9, 8, 7, 6, 4, 5, 3, 2, 1, 0	46
$\mathcal{F}_9$	14, 15, 12, 13, 10, 11, 9, 8, 7, 6, 5, 4, 3, 2, 0, 1	42
$\mathcal{F}_{10}$	14, 15, 12, 13, 10, 11, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0	69
$\mathcal{F}_{11}$	14, 15, 12, 13, 11, 10, 9, 8, 7, 6, 5, 4, 2, 3, 1, 0	58
$\mathcal{F}_{12}$	14, 15, 13, 12, 11, 10, 8, 9, 7, 6, 4, 5, 2, 3, 1, 0	35
$\mathcal{F}_{13}$	14, 15, 13, 12, 11, 10, 8, 9, 7, 6, 4, 5, 3, 2, 1, 0	56
$\mathcal{F}_{14}$	14, 15, 13, 12, 11, 10, 8, 9, 7, 6, 5, 4, 3, 2, 1, 0	94
$\mathcal{F}_{15}$	14, 15, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 0, 1	86
$\mathcal{F}_{16}$	14, 15, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0	206

(b) Double Stochastic Functions

Fig. 2. Chaotic Functions Candidates with $n = 4$

Table 1. NIST Test Evaluation of PRNG instances

Property	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	$\mathcal{F}_6$	$\mathcal{F}_7$	$\mathcal{F}_8$	$\mathcal{F}_9$	$\mathcal{F}_{10}$	$\mathcal{F}_{11}$	$\mathcal{F}_{12}$	$\mathcal{F}_{13}$	$\mathcal{F}_{14}$	$\mathcal{F}_{15}$	$\mathcal{F}_{16}$
Frequency	77.9	15.4	83.4	59.6	16.3	38.4	20.2	29.0	77.9	21.3	65.8	85.1	51.4	35.0	77.9	92.4
BlockFrequency	88.3	36.7	43.7	81.7	79.8	5.9	19.2	2.7	98.8	1.0	21.3	63.7	1.4	7.6	99.1	33.5
CumulativeSums	76.4	86.6	8.7	66.7	2.2	52.6	20.8	80.4	9.8	54.0	73.6	80.1	60.7	79.7	76.0	44.7
Runs	5.2	41.9	59.6	89.8	23.7	76.0	77.9	79.8	45.6	59.6	89.8	2.4	96.4	10.9	72.0	11.5
LongestRun	21.3	93.6	69.9	23.7	33.5	30.4	41.9	43.7	30.4	17.2	41.9	51.4	59.6	65.8	11.5	61.6
Rank	1.0	41.9	35.0	45.6	51.4	20.2	31.9	83.4	89.8	38.4	61.6	4.0	21.3	69.9	47.5	95.6
FFT	40.1	92.4	97.8	86.8	43.7	38.4	76.0	57.5	36.7	35.0	55.4	57.5	86.8	76.0	31.9	7.6
NonOverlappingTemplate	49.0	45.7	50.5	51.0	48.8	51.2	51.6	50.9	50.9	48.8	45.5	47.3	47.0	49.2	48.6	46.4
OverlappingTemplate	27.6	10.9	53.4	61.6	16.3	2.7	59.6	94.6	88.3	55.4	76.0	23.7	47.5	91.1	65.8	81.7
Universal	24.9	35.0	72.0	51.4	20.2	74.0	40.1	23.7	9.1	72.0	4.9	13.7	14.5	1.8	93.6	65.8
ApproximateEntropy	33.5	57.5	65.8	53.4	26.2	98.3	53.4	63.7	38.4	6.7	53.4	19.2	20.2	27.6	67.9	88.3
RandomExcursions	29.8	35.7	40.9	36.3	54.8	50.8	43.5	46.0	39.1	40.8	29.6	42.0	34.8	33.8	63.0	46.3
RandomExcursionsVariant	32.2	40.2	23.0	39.6	47.5	37.2	56.9	54.6	53.3	31.5	23.0	38.1	52.3	57.1	47.7	40.8
Serial	56.9	58.5	70.4	73.2	31.3	45.9	60.8	39.9	57.7	21.2	6.4	15.6	44.7	31.4	71.7	49.1
LinearComplexity	24.9	23.7	96.4	61.6	83.4	49.4	49.4	18.2	3.5	76.0	24.9	97.2	38.4	38.4	1.1	8.6

6 Conclusion and Future Work

This work has shown that discrete-time dynamical systems G_f are chaotic iff embedded Boolean maps f have strongly connected iteration graph $\Gamma(f)$. Sufficient conditions on its interaction graph $G(f)$ have been further proven to ensure this strong connexity. Finally, we have proven that the output of such a function is uniformly distributed iff the induced Markov chain can be represented as a double stochastic matrix. We have applied such a complete theoretical work on chaos to pseudo random number generation and all experiments have confirmed theoretical results. As far as we know, this work is the first one that allows to *compute* new functions whose chaoticity is proven and preserved during implementation. The approach relevance has been shown on PRNGs but is not limited to that domain. In fact, this whole work has applications everywhere chaoticity is a possible answer, e.g., in hash functions, digital watermarking...

In a future work, we will investigate whether the characterization of uniform distribution may be expressed in terms of interaction graph, avoiding thus to generate functions and to check later whether they induce double stochastic Markov matrix. The impact of the description of chaotic iterations as Markov processes will be studied more largely. We will look for new characterizations concerning other relevant topological properties of disorder, such as topological entropy, expansivity, Lyapunov exponent, instability, etc. Finally, the relation between these mathematical definitions and intended properties for each targeted application will be investigated too, specifically in the security field.

References

1. Yi, X.: Hash function based on chaotic tent maps. IEEE Transactions on Circuits and Systems II: Express Briefs 52(6), 354–357 (2005)
2. Dawei, Z., Guanrong, C., Wenbo, L.: A chaos-based robust wavelet-domain watermarking algorithm. Chaos, Solitons and Fractals 22, 47–54 (2004)
3. Stojanovski, T., Kocarev, L.: Chaos-based random number generators-part I: analysis [cryptography]. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 48(3), 281–288 (2001)
4. Devaney, R.L.: An Introduction to Chaotic Dynamical Systems, 2nd edn. Addison-Wesley, Redwood City (1989)
5. Bahi, J.M., Guyeux, C.: Topological chaos and chaotic iterations, application to hash functions. In: WCCI 2010 IEEE World Congress on Computational Intelligence, Barcelona, Spain, pp. 1–7 (2010)
6. Banks, J., Brooks, J., Cairns, G., Stacey, P.: On devaney's definition of chaos. Amer. Math. Monthly 99, 332–334 (1992)
7. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
8. Bahi, J., Guyeux, C., Wang, Q.: A novel pseudo-random generator based on discrete chaotic iterations. In: INTERNET 2009 1-st Int. Conf. on Evolving Internet, Cannes, France, pp. 71–76 (2009)
9. Marsaglia, G.: Xorshift rngs. Journal of Statistical Software 8(14), 1–6 (2003)
10. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. National Institute of Standards and Technology (2010), http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

A New Bound for 3-Satisfiable Maxsat and Its Algorithmic Application

Gregory Gutin, Mark Jones, and Anders Yeo

Royal Holloway, University of London, United Kingdom
{gutin,markj,anders}@cs.rhul.ac.uk

Abstract. Let F be a CNF formula with n variables and m clauses. F is t-*satisfiable* if for any t clauses in F, there is a truth assignment which satisfies all of them. Lieberherr and Specker (1982) and, later, Yannakakis (1994) proved that in each 3-satisfiable CNF formula at least $\frac{2}{3}$ of its clauses can be satisfied by a truth assignment. Yannakakis's proof utilizes the fact that $\frac{2}{3}m$ is a lower bound on the expected number of clauses satisfied by a random truth assignment over a certain distribution. A CNF formula F is called *expanding* if for every subset X of the variables of F, the number of clauses containing variables of X is not smaller than $|X|$. In this paper we strengthen the $\frac{2}{3}m$ bound for expanding 3-satisfiable CNF formulas by showing that for every such formula F at least $\frac{2}{3}m+\rho n$ clauses of F can be satisfied by a truth assignment, where $\rho(> 0.0019)$ is a constant. Our proof uses a probabilistic method with a sophisticated distribution for truth values. We use the bound $\frac{2}{3}m + \rho n$ and results on matching autarkies to obtain a new lower bound on the maximum number of clauses that can be satisfied by a truth assignment in any 3-satisfiable CNF formula.

We use our results above to show that the following parameterized problem is fixed-parameter tractable and, moreover, has a kernel with a linear number of variables. In 3-S-MaxSat-AE, we are given a 3-satisfiable CNF formula F with m clauses and asked to determine whether there is an assignment which satisfies at least $\frac{2}{3}m + k$ clauses, where k is the parameter. Note that Mahajan and Raman (1999) asked whether 2-S-MaxSat-AE, the corresponding problem for 2-satisfiable formulas, is fixed-parameter tractable. Crowston and the authors of this paper proved in [9] that 2-S-MaxSat-AE is fixed-parameter tractable and, moreover, has a kernel with a linear number of variables. 2-S-MaxSat-AE appears to be easier than 3-S-MaxSat-AE and, unlike this paper, [9] uses only deterministic combinatorial arguments.

1 Introduction

For a formula F in conjunctive normal form (CNF), let $\mathcal{C}(F) = \{C_1, \dots, C_m\}$ denote the set of clauses in F. Each clause $C \in \mathcal{C}(F)$ has an associated positive integral weight $w(C)$, and we let $w(\mathcal{C}(F))$ denote the total weight of clauses in $\mathcal{C}(F)$. (We use weighted clauses rather than letting F be a multiset of clauses; see [13] for a discussion.) Let $\mathrm{sat}(F)$ be the maximum weight of clauses in $\mathcal{C}(F)$ that can be satisfied by a truth assignment. The set of variables of F will be denoted by $V(F)$. In what follows, we assume that no clause has a variable and its negation and no clause is empty.

For any integer t, we say F is *t-satisfiable* if for any t clauses in $\mathcal{C}(F)$ there exists a truth assignment that satisfies all of them. Thus, if F is is 2-satisfiable then $\mathcal{C}(F)$

contains no pair of clauses of the form $\{x\}, \{\bar{x}\}$; if F is 3-satisfiable then the forbidden sets of clauses are pairs of the form $\{x\}, \{\bar{x}\}$ and triplets of the form $\{x\}, \{y\}, \{\bar{x}, \bar{y}\}$ or $\{x\}, \{\bar{x}, y\}, \{\bar{x}, \bar{y}\}$, as well as any triplets that can be derived from these by switching positive literals with negative literals.

It is well-known that for any 1-satisfiable CNF formula (i.e., any CNF formula) F, $\mathrm{sat}(F) \geq \frac{1}{2}w(\mathcal{C}(F))$. Lieberherr and Specker [18,19] and, later, Yannakakis [26] proved the following: if F is 2-satisfiable then $\mathrm{sat}(F) \geq \hat{\phi}w(\mathcal{C}(F))$ (where $\hat{\phi} \approx 0.61803$ is the positive root of $x^2 + x = 1$); if F is 3-satisfiable then $\mathrm{sat}(F) \geq \frac{2}{3}w(\mathcal{C}(F))$. These bounds are asymptotically tight (that is, for any $\epsilon > 0$, there exists a 3-satisfiable CNF formula F such that $\mathrm{sat}(F) < (\frac{2}{3} + \epsilon)w(\mathcal{C}(F))$ and similar inequalities hold for 1-satisfiable and 2-satisfiable formulas).

Mahajan and Raman [20] considered the following parameterized problem SAT-AE[1]: we are given a (1-satisfiable) CNF formula F and asked to determine whether there is an assignment which satisfies at least $\frac{1}{2}w(\mathcal{C}(F)) + k$ clauses, where k is the parameter. (Basic notions on parameterized algorithms and complexity are given in Section 2.) For SAT-AE, Mahajan and Raman [20] obtained a kernel with at most $6k + 3$ variables and $10k$ clauses. Crowston et al. [9] improved this to $4k$ variables and $(2\sqrt{5} + 4)k$ clauses.

Crowston et al. [9] strengthened the bound $\mathrm{sat}(F) \geq \hat{\phi}w(\mathcal{C}(F))$ for 2-satisfiable CNF formulas to $\mathrm{sat}(F) \geq \hat{\phi}w(\mathcal{C}(F)) + \gamma|V(F)|$ (where $\gamma \approx 0.072949$) using deterministic combinatorial arguments. The stronger bound allowed them to solve an open problem of Mahajan and Raman [20] by proving that the following parameterized problem is fixed-parameter tractable and, moreover, has a kernel with a linear number of variables. In 2-S-MAXSAT-AE, we are given a 2-satisfiable CNF formula F and asked to determine whether there is an assignment which satisfies at least $\hat{\phi}w(\mathcal{C}(F)) + k$ clauses, where k is the parameter.

In this paper, we strengthen the bound $\mathrm{sat}(F) \geq \frac{2}{3}w(\mathcal{C}(F))$ for 3-satisfiable CNF formulas. The deterministic approach of Crowston et al. [9] cannot be readily extended to the 3-satisfiability case (which appears to be more complicated) and we use probabilistic arguments instead. The stronger bound is given in Corollary 2 and mainly follows from Theorems 1 and 2. To prove both theorems, we use a probabilistic argument based on bounding the expected number of clauses satisfied by a random truth assignment. The difficulty lies in choosing an appropriate probabilistic distribution. While the distribution in the proof of Theorem 1 is relatively simple and was already used by Yannakakis [26], the distribution in the proof of Theorem 2 is quite complicated and, to the best of our knowledge, has never been used before. To formulate and prove Theorems 1 and 2, we use a new classification of clauses based on a notion of a *hard clause*. Another important ingredient is *autarkies*, a notion well-studied for the Satisfiability problem, see, e.g., [5], [17]; we introduce autarkies in the next section and use one of the key results on matching autarkies in our proofs.[2]

[1] AE stands for Above Expectation.

[2] Autarkies were first introduced in [22]. Autarkies in general are the subject of much study, see, e.g., [11], [17], [24], and see [5] for an overview. In this paper we only make use of a small part of the research on autarkies, as we may limit ourselves to the concept of matching autarkies for our proofs.

The improved bound allows us to prove that the following parameterized problem is fixed-parameter tractable and, moreover, has a kernel with a linear number of variables. In 3-S-MAXSAT-AE, we are given a 3-satisfiable CNF formula F and asked to determine whether there is an assignment which satisfies at least $\frac{2}{3}w(\mathcal{C}(F)) + k$ clauses, where k is the parameter. This answers a question from [9].

A parameterization of MAX-r-SAT above a tight lower bound was recently studied in [3,7,8,15]. Approaches used there are completely different from the one used in this paper.

Our paper is organized as follows. In Section 2, we provide additional terminology and notation. In Section 3, we prove our main results. A CNF formula F is *expanding* if for every subset X of the variables of F, the weight of clauses containing variables of X is not smaller than $|X|$. We show that for each expanding 3-satisfiable CNF formula F, we have $\text{sat}(F) \geq \frac{2}{3}w(\mathcal{C}(F)) + \rho|V|$, where ρ is a positive constant. In Section 3, we also prove that 3-S-MAXSAT-AE is fixed parameter tractable, and has a kernel with a linear number of variables. Our lower bound for $\text{sat}(F)$ above is derived from Theorems 1 and 2. This lower bound implies a lower bound for $\text{sat}(F)$ for every 3-satisfiable CNF formula F and the latter lower bound is given in Corollary 2. Theorem 2 is the main technical result in this paper, and we leave its proof for Section 4.

Two open problems on t-satisfiable CNF formulas for any t can be found in [13].

2 Terminology and Notation

For a clause C, we let $V(C)$ be the set of variables such that $x \in V(C)$ if $x \in C$ or $\bar{x} \in C$. We assume that every clause C appears only once in $\mathcal{C}(F)$. If at any stage we have two clauses C_1, C_2 containing exactly the same literals, we remove one of them, say C_2, and add the weight $w(C_2)$ to $w(C_1)$. For a set of clauses $\mathcal{C}' \subseteq \mathcal{C}(F)$, the *weight* $w(\mathcal{C}')$ of $\mathcal{C}'$ is $\sum_{C \in \mathcal{C}'} w(C)$. We will often write $w(F)$ instead of $w(\mathcal{C}(F))$.

Definition 1. *For any* 3*-satisfiable CNF formula, F, define $V_u(F)$, $\mathcal{C}_h(F)$, $V_{h\setminus u}(F)$ and $V_{\bar{h}}(F)$ as follows:*

$V_u(F)$ *denotes the variables appearing in unit clauses in F. That is, $s \in V_u(F)$ if and only if $\{s\} \in \mathcal{C}(F)$ or $\{\bar{s}\} \in \mathcal{C}(F)$.*

$\mathcal{C}_h(F)$ *denotes the set of* hard clauses *in F, which are defined as follows: Every unit clause in F is a hard clause. For $s \in V_u(F), t \notin V_u(F)$, every clause of the form $\{\bar{s}, t\}$ or $\{\bar{s}, \bar{t}\}$ is a hard clause if $\{s\} \in \mathcal{C}(F)$, and every clause of the form $\{s, t\}$ or $\{s, \bar{t}\}$ is a hard clause if $\{\bar{s}\} \in \mathcal{C}(F)$.*

$V_{h\setminus u}(F)$ *contains all variables which belong to a hard clause but not to a unit clause.*

$V_{\bar{h}}(F) = V(F) \setminus (V_u(F) \cup V_{h\setminus u}(F))$. *That is, $V_{\bar{h}}(F)$ contains all variables which do not belong to any hard clause.*

Remark 1. The hard clauses are so called because they are the clauses that make it difficult to strengthen the $\text{sat}(F) \geq \frac{2}{3}w(\mathcal{C})$ bound. If F contains only non-hard clauses, then it is easy to improve the bound to $\text{sat}(F) \geq \frac{19}{27}w(\mathcal{C})$ (see the proof of Theorem 1). For a formula containing hard clauses, it is much harder to strengthen the bound, and this is what makes up the main technical part of the paper.

For any $x \in V(F)$, let $w_h(x)$ denote the weight of all hard clauses containing the literal x and let $w_h(\bar{x})$ denote the weight of all hard clauses containing the literal $\bar{x}$.

Any formula F' obtained from F by deleting some clauses is called a *subformula* of F. If F' is a subformula of F then $F \setminus F'$ denotes the formula obtained from F by deleting all clauses of F'. Let X be a subset of the variables of a CNF formula F. Then F_X denotes the subformula obtained from F by deleting all clauses not containing variables from X. Recall that a CNF formula F is *expanding* if $|X| \le w(F_X)$ for each $X \subseteq V(F)$.

A *truth assignment* is a function $\alpha : V(F) \to \{\text{TRUE}, \text{FALSE}\}$. A truth assignment α *satisfies* a clause C if there exists $x \in V(F)$ such that $x \in C$ and $\alpha(x) = \text{TRUE}$, or $\bar{x} \in C$ and $\alpha(x) = \text{FALSE}$. We will denote, by $\text{sat}_\alpha(F)$, the sum of the weight of clauses in F satisfied by α. We denote the maximum value of $\text{sat}_\alpha(F)$, over all α, by $\text{sat}(F)$.

A function $\beta : U \to \{\text{TRUE}, \text{FALSE}\}$, where U is a subset of $V(F)$, is called a *partial truth assignment*. A partial truth assignment $\beta : U \to \{\text{TRUE}, \text{FALSE}\}$ is an *autarky* if β satisfies all clauses of F_U. Autarkies are of interest, in particular, due to the following simple fact whose trivial proof is omitted.

Lemma 1. *Let $\beta : U \to \{\text{TRUE}, \text{FALSE}\}$ be an autarky for a CNF formula F. Then given any truth assignment γ on $F \setminus F_U$, we can find, in polynomial time, a truth assignment τ such that* $\text{sat}_\tau(F) = w(F_U) + \text{sat}_\gamma(F \setminus F_U)$.

In Lemma 1, such a τ can be found by combining the partial truth assignments β and γ, and letting τ have arbitrary values for the variables not covered by β or γ. A version of Lemma 1 can be traced back to Monien and Speckenmeyer [22].

A *parameterized problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$ over a finite alphabet Σ. L is *fixed-parameter tractable* if the membership of an instance (I, k) in $\Sigma^* \times \mathbb{N}$ can be decided in time $f(k)|I|^{O(1)}$ where f is a function of the *parameter* k only [10,12,23]. Given a parameterized problem L, a *kernelization of* L is a polynomial-time algorithm that maps an instance (x, k) to an instance (x', k') (the *kernel*) such that (i) $(x, k) \in L$ if and only if $(x', k') \in L$, (ii) $k' \le h(k)$, and (iii) $|x'| \le g(k)$ for some functions h and g. It is well-known [10,12,23] that a decidable parameterized problem L is fixed-parameter tractable if and only if it has a kernel. By replacing Condition (ii) in the definition of a kernel by $k' \le k$, we obtain a definition of a *proper kernel* (sometimes, it is called a *strong kernel*); cf. [1,6].

3 Main Results

Let F be a CNF formula, $C_1, \dots, C_m$ the clauses of F, and $x_1, \dots, x_n$ the variables of F. Let p_i be the probability that x_i is assigned TRUE independently of all other variables, $i = 1, \dots, n$. Let α be a random truth assignment corresponding to this probability distribution. Then the probability that a clause C_j with variables $\{x_{i_1}, \dots, x_{i_s}\}$ is satisfied, is $P_j = 1 - \prod_{\ell=1}^{s} p^*_{i_\ell}$, where $p^*_{i_\ell} = 1 - p_{i_\ell}$ if $x_{i_\ell} \in C_j$ and $p^*_{i_\ell} = p_{i_\ell}$ if $\bar{x}_{i_\ell} \in C_j$. Thus, $\mathbb{E}(\text{sat}_\alpha(C_j)) = w(C_j)P_j$, and by linearity of expectation, $\mathbb{E}(\text{sat}_\alpha(F)) = \sum_{j=1}^{m} w(C_j)P_j$. We can find, in polynomial time, a (deterministic) truth assignment τ such that $\mathbb{E}(\text{sat}_\tau(F)) \ge \sum_{j=1}^{m} w(C_j)P_j$ using the well-known derandomization method of conditional expectations, cf. [4,26].

Assumption 1. *Let F be a 3-satisfiable CNF formula. In what follows, we may assume without loss of generality that all unit clauses in F are of the form $\{x\}$, where $x \in V(F)$.*

Indeed, suppose that $\{\bar{x}\} \in \mathcal{C}(F)$, then $\{x\} \notin \mathcal{C}(F)$ as F is 3-satisfiable. Thus, we may replace $\bar{x}$ by x and x by $\bar{x}$ in all clauses of F without changing $\text{sat}(F)$. Note that by Assumption 1, the only non-unit hard clauses are those of the form $\{\bar{s}, t\}$ or $\{\bar{s}, \bar{t}\}$, for $s \in V_u(F), t \notin V_u(F)$.

The following result is an easy extension of the $\frac{2}{3}w(F)$ bound on $\text{sat}(F)$. The proof is almost exactly the same as Yannakakis's proof in [26]; in particular the probability distribution involved is the same. The only difference is that our proof involves extra analysis to get the addition of $\frac{1}{27}w(\mathcal{C}(F)\backslash\mathcal{C}_h(F))$.

Theorem 1. *Let F be a 3-satisfiable CNF formula. Then we can find, in polynomial time, a truth assignment τ such that $\text{sat}_\tau(F) \geq \frac{2}{3}w(F) + \frac{1}{27}w(\mathcal{C}(F)\backslash\mathcal{C}_h(F))$.*

Proof. Let $\mathcal{C}^* = \mathcal{C}(F)\backslash\mathcal{C}_h(F)$. We will construct a random truth assignment α such that $\mathbb{E}(\text{sat}_\alpha(F)) \geq \frac{2}{3}w(F) + \frac{1}{27}w(\mathcal{C}^*)$. This implies that there exists an assignment which satisfies at least $\frac{2}{3}w(F) + \frac{1}{27}w(\mathcal{C}^*)$ clauses, as required.

We define a random truth assignment α as follows. For $x \in V_u(F)$, we let $\alpha(x)$ be TRUE with probability $\frac{2}{3}$. For $x \in V(F)\backslash V_u(F)$, we let $\alpha(x)$ be TRUE with probability $\frac{1}{2}$. The values are assigned to the variables independently from each other.

Let C be a clause and let α be the random truth assignment above. We will now bound $\mathbb{E}(\text{sat}_\alpha(C))$. We first consider a hard clause C and, to simplify notation, assume that $w(C) = 1$. By Assumption 1, we have the following cases.

$C = \{x\}$: In this case the probability that C is satisfied is exactly $\frac{2}{3}$ and, thus, $\mathbb{E}(\text{sat}_\alpha(C)) = \frac{2}{3}$.
$C = \{\bar{x}, y\}$ **or** $C = \{\bar{x}, \bar{y}\}$ **for** $x \in V_u(F), y \notin V_u(F)$: Then $\mathbb{E}(\text{sat}_\alpha(C)) = 1 - \frac{2}{3} \times \frac{1}{2} = \frac{2}{3}$.

Thus, for every hard clause C with $w(C) \geq 1$, we have $\mathbb{E}(\text{sat}_\alpha(C)) \geq \frac{2}{3}w(C)$. We will now consider a non-hard clause C and, to simplify notation, assume that $w(C) = 1$. The following cases cover all possibilities.

$|C| = 2$ **and** $|V(C) \cap V_u(F)| = 2$: $C = \{\bar{x}, \bar{y}\}$ is not possible as F is 3-satisfiable and we cannot satisfy the three clauses $\{x\}$, $\{y\}$ and $\{\bar{x}, \bar{y}\}$ simultaneously. Therefore we may assume that a positive literal x belongs to $\mathcal{C}(F)$, which implies that $\mathbb{E}(\text{sat}_\alpha(C)) \geq 1 - \frac{1}{3} \times \frac{2}{3} = \frac{7}{9}$, as $x \in V_u(F)$.
$|C| = 2$ **and** $|V(C) \cap V_u(F)| = 1$: Then $C = \{x, y\}$ or $C = \{x, \bar{y}\}$ for $x \in V_u(F), y \notin V_u(F)$. Then $\mathbb{E}(\text{sat}_\alpha(C)) = 1 - \frac{1}{2} \times \frac{1}{3} = \frac{5}{6}$
$|C| = 2$ **and** $|V(C) \cap V_u(F)| = 0$: Then $\mathbb{E}(\text{sat}_\alpha(C)) = 1 - \frac{1}{2} \times \frac{1}{2} = \frac{3}{4}$.
$|C| \geq 3$: Since for each literal the probability of it being assigned FALSE is at most $\frac{2}{3}$, we have $\mathbb{E}(\text{sat}_\alpha(C)) \geq 1 - (\frac{2}{3})^3 = \frac{19}{27}$.

Thus, for every non-hard clause C with weight $w(C)$, we have $\mathbb{E}(\text{sat}_\alpha(C)) \geq \frac{19}{27}w(C)$. Therefore,

$$\mathbb{E}(\text{sat}_\alpha(F)) \geq \frac{2}{3}w(\mathcal{C}_h(F)) + \frac{19}{27}w(\mathcal{C}^*) = \frac{2}{3}w(F) + \frac{1}{27}w(\mathcal{C}^*).$$

To find the required assignment τ, it remains to apply derandomization as described in the beginning of this section. □

This theorem and the definition of an expanding formula imply the following:

Corollary 1. *Let F be an expanding 3-satisfiable CNF formula. Then we can find, in polynomial time, a truth assignment τ such that* $\mathrm{sat}_\tau(F) \geq \frac{2}{3}w(F) + \frac{1}{27}|V_{\bar{h}}(F)|$.

Our aim is to prove a lower bound on $\mathrm{sat}(F)$ that includes a multiple of the number of variables as a term. It is clear that for general 3-satisfiable F such a bound is impossible. For consider a formula containing a single clause C containing a large number of variables. We can arbitrarily increase the number of variables in the formula, and the maximum number of satisfiable clauses will always be 1. We therefore need a reduction rule that cuts out 'excess' variables. Our reduction rule is based on the following lemma proved in Fleischner et al. [11] (Lemma 10), Kullmann [17] (Lemma 7.7) and Szeider [24] (Lemma 9).

Lemma 2. *Let F be a CNF formula. Define a bipartite graph, B_F, associated with F as follows: $V(F)$ and $\mathcal{C}(F)$ are partite sets of B_F and there is an edge between $v \in V(F)$ and $C \in \mathcal{C}(F)$ in B_F if and only if $v \in V(C)$. Given a maximum matching in B_F, in time $O(|\mathcal{C}(F)|)$ we can find an autarky $\beta : U \rightarrow \{\text{TRUE}, \text{FALSE}\}$ such that $F \setminus F_U$ is expanding.*

The papers [11], [17] and [24] actually show that $F \setminus F_U$ is 1-expanding (see [11] or [24] for a definition), which is a slightly stronger result. For our results it is enough that $F \setminus F_U$ is expanding. An autarky found by the algorithm of Lemma 2 is of a special kind, called a matching autarky; such autarkies were used first by Aharoni and Linial [2]. Note that the autarky found in Lemma 2 can be empty, i.e., $U = \emptyset$.

Lemmas 1 and 2 immediately imply the following:

Lemma 3. *Let F be a CNF formula and let $\beta : U \rightarrow \{\text{TRUE}, \text{FALSE}\}$ be an autarky found by the algorithm of Lemma 2. Then given any truth assignment γ on $F \setminus F_U$, we can find, in polynomial time, a truth assignment τ such that* $\mathrm{sat}_\tau(F) = w(F_U) + \mathrm{sat}_\gamma(F \setminus F_U)$*, and $F \setminus F_U$ is an expanding formula.*

In Section 4 we will give a proof of the following theorem, which is the main technical result of this paper. Hereafter, we let $q = \sqrt[3]{\frac{1}{3}} \approx 0.69336$ and $q' = \sqrt{\frac{1}{3}} \approx 0.57735$.

Theorem 2. *Let F be a 3-satisfiable CNF formula. Then we can find, in polynomial time, a truth assignment τ such that*

$$\mathrm{sat}_\tau(F) \geq \frac{2}{3}w(F) + (q - \frac{2}{3})(1 - q')|V_u(F)| + (q - \frac{2}{3})(q' - \frac{1}{2})|V_{h\setminus u}(F)|.$$

Using Corollary 1 and Theorem 2, we can show the following main bound:

Theorem 3. *Let F be an expanding 3-satisfiable CNF formula. Then there exists a constant $\rho > 0.0019$ such that* $\mathrm{sat}_\tau(F) \geq \frac{2}{3}w(F) + \rho|V(F)|$ *for some truth assignment τ that can be found in polynomial time.*

Proof. First note that $|V(F)| = |V_u(F)| + |V_{h\setminus u}(F)| + |V_{\bar{h}}(F)|$. Let $\beta = (q - \frac{2}{3})(q' - \frac{1}{2})$, and observe that Theorem 2 implies $\mathrm{sat}_{\tau'}(F) \geq \frac{2}{3}w(F) + \beta(|V_u(F)| + |V_{h\setminus u}(F)|)$ for some truth assignment τ' that can be found in polynomial time. Multiplying this inequality by $\frac{1}{27}$, we obtain $\frac{1}{27}\mathrm{sat}_{\tau'}(F) \geq \frac{1}{27} \times \frac{2}{3}w(F) + \frac{\beta}{27}(|V_u(F)| + |V_{h\setminus u}(F)|)$.

Similarly, from Corollary 1 we obtain $\beta\mathrm{sat}_{\tau''}(F) \geq \beta\frac{2}{3}w(F) + \frac{\beta}{27}|V_{\bar{h}}(F)|$ for some truth assignment τ'' that can be found in polynomial time. Let $\tau \in \{\tau', \tau''\}$ such that $\mathrm{sat}_\tau(F) = \max\{\mathrm{sat}_{\tau'}(F), \mathrm{sat}_{\tau''}(F)\}$.

The inequalities for $\frac{1}{27}\mathrm{sat}_{\tau'}(F)$ and $\beta\mathrm{sat}_{\tau''}(F)$ and the definition of τ imply

$$(\frac{1}{27} + \beta)\mathrm{sat}_\tau(F) \geq (\frac{1}{27} + \beta)\frac{2}{3}w(F) + \frac{\beta}{27}|V(F)|.$$

Therefore, we have that $\mathrm{sat}_\tau(F) \geq \frac{2}{3}w(F) + \rho|V(F)|$, where $\rho = \frac{\beta/27}{1/27+\beta} > 0.0019$. This completes the proof. □

As a direct consequence of Theorem 3 and Lemma 3, we also have the following bound on $\mathrm{sat}(F)$ for *any* 3-satisfiable CNF formula F.

Corollary 2. *Let F be a 3-satisfiable CNF formula and let $\beta : U \to \{$*TRUE, FALSE*$\}$ be an autarky found by the algorithm of Lemma 2. Then there exists a constant $\rho > 0.0019$ such that*

$$\mathrm{sat}_\tau(F) \geq \frac{2}{3}w(F) + \frac{1}{3}w(F_U) + \rho|V(F \setminus F_U)|$$

for some truth assignment τ that can be found in polynomial time.

Corollary 3. 3-S-MAXSAT-AE *is fixed-parameter tractable. Moreover, it has a proper kernel with $O(k)$ variables.*

Proof. Let F be a 3-satisfiable CNF formula, let $\beta : U \to \{$TRUE, FALSE$\}$ be an autarky found by the algorithm of Lemma 2 and let $F' = F \setminus F_U$. We are to decide whether $\mathrm{sat}(F) \geq \frac{2}{3}w(F) + k$, where k (an integer) is the parameter.

By Lemma 3, $\mathrm{sat}(F) = w(F_U) + \mathrm{sat}(F')$. Thus, $\mathrm{sat}(F) \geq \frac{2}{3}w(F) + k$ if and only if $\mathrm{sat}(F') \geq \frac{2}{3}w(F') + k'$, where $k' = \lceil\frac{k - w(F_U)}{3}\rceil$. Since F' is an expanding 3-satisfiable formula, by Theorem 3 we have $\mathrm{sat}_\tau(F') \geq \frac{2}{3}w(F') + \rho|V(F')|$ for some truth assignment τ that can be found in polynomial time, where $\rho > 0.0019$. Thus, if $\rho|V(F')| \geq k'$, then the answer to 3-S-MAXSAT-AE is YES and the corresponding truth assignment can be found in polynomial time. Otherwise, $|V(F')| < \frac{k'}{\rho}$ and, thus, $|V(F')| = O(k)$, and so we can find the optimal assignment in time $2^{0(k)}m^{O(1)}$, where $m = |\mathcal{C}(F)|$.

Let $m' = |\mathcal{C}(F')|$. If $m' \geq 2^{|V(F')|}$, we can find $\mathrm{sat}(F')$ and, thus, $\mathrm{sat}(F)$ in polynomial time. Therefore, we may assume that $m' < 2^{|V(F')|}$ and, thus, $m' = 2^{O(k)}$ implying that F' is a kernel. Since $k' \leq k$, F' is a proper kernel. □

4 Proof of Theorem 2

Due to the space limitation, we omit proofs of the lemmas in this section; the proofs can be found in [13].

Assumption 2. *In what follows, we may assume that for each $x \in V_u(F)$, x is not a literal in any non-unit clause of F.*

Indeed, let F' be the formula obtained from F by replacing every clause C with a literal $x \in V_u(F)$ by $\{x\}$. Note that $\mathrm{sat}(F') \le \mathrm{sat}(F)$, $V_u(F') = V_u(F)$, and since no non-unit hard clause contains the literal x for $x \in V_u(F')$, we have $V_{h\setminus u}(F') = V_{h\setminus u}(F)$. Note that while F' is 3-satisfiable, it is not necessarily expanding, even if F is expanding.

Since in this section we will only be considering a fixed 3-satisfiable CNF formula F, we will denote $V(F)$, $\mathcal{C}(F)$, $V_u(F)$, $\mathcal{C}_h(F)$, $V_{h\setminus u}(F)$ and $V_{\bar{h}}(F)$ by V, $\mathcal{C}$, V_u, $\mathcal{C}_h$, $V_{h\setminus u}$ and $V_{\bar{h}}$, respectively.

In order to prove Theorem 2 we define the following random truth assignment, α. Recall that $q = \sqrt[3]{\frac{1}{3}} \approx 0.69336$ and $q' = \sqrt{\frac{1}{3}} \approx 0.57735$.

Definition 2. *Let $V_{h\setminus u} = \{t_1, t_2, \ldots, t_a\}$, $a = |V_{h\setminus u}|$, and let $p_1, p_2, \ldots, p_a$ be arbitrary real values from the interval $[1-q', q']$, i.e., $1-q' \le p_i \le q'$ for all $i = 1, 2, \ldots, a$. Set the probability of a variable being* TRUE *as follows, independently of all other variables.*

Let $\mathbb{P}(\alpha(t_i) = \text{TRUE}) = p_i$ for all $i = 1, 2, \ldots, a$ (that is, the probability that t_i is given the truth value TRUE *by α is p_i). Let $\mathbb{P}(\alpha(r) = \text{TRUE}) = \frac{1}{2}$ for all $r \in V_{\bar{h}}$.*

Before we assign probabilities to the variables in $V_u = \{s_1, s_2, \ldots, s_r\}$, we will define random variables A_i and B_i, for all $i = 1, 2, \ldots, r$ as follows. Let A_i be the weight of the hard clauses containing the literal $\bar{s_i}$ which are satisfied by the above partial truth assignment. Analogously let B_i be the weight of the hard clauses containing the literal $\bar{s_i}$ which are not satisfied by the above partial truth assignment.

Observe that $w_h(s_i)$ is the weight of the singleton clause $\{s_i\}$. If $\mathbb{E}(B_i) \ge w_h(s_i)$, then let $\mathbb{P}(\alpha(s_i) = \text{TRUE}) = 1-q$. If $\mathbb{E}(B_i) < w_h(s_i)$, then let $\mathbb{P}(\alpha(s_i) = \text{TRUE}) = q$.

Lemma 4. *Given the probability distribution in Definition 2, the following holds.*

$$\mathbb{E}(\mathrm{sat}_\alpha(F)) \ge \frac{2}{3}w(F) + \frac{1}{3}\sum_{i=1}^{r}[\mathbb{E}(A_i) - \mathbb{E}(B_i)] + (q - \frac{2}{3})\sum_{i=1}^{r}\mathbb{E}(|B_i - w_h(s_i)|).$$

We will now assign values to $p_1, p_2, \ldots, p_a$ in Definition 2 such that the right-hand side of the inequality in Lemma 4 becomes as large as possible.

Definition 3. *Partition $V_{h\setminus u}$ into sets $T^*, T_1, T_2, T_3, \ldots, T_r$, such that the following holds: $T^* = \{t_i \in V_{h\setminus u} \mid w_h(t_i) \neq w_h(\bar{t_i})\}$ and $t_i \in T_j$ if and only if $t_i \in V_{h\setminus u} \setminus T^*$ and j is the minimum value such that $\{\bar{s_j}, t_i\} \in \mathcal{C}$ or $\{\bar{s_j}, \bar{t_i}\} \in \mathcal{C}$.*

For all $t_i \in T^*$ let $p_i = q'$ if $w_h(t_i) > w_h(\bar{t_i})$ and let $p_i = 1-q'$ if $w_h(t_i) < w_h(\bar{t_i})$.

First let $p_i = \frac{1}{2}$ for all $t_i \in V_{h\setminus u} \setminus T^*$, and then modify the values of p_i as follows. For each $j = 1, 2, 3, \ldots, r$ (in this order), if $\mathbb{E}(B_j) > w_h(s_j)$, then for all $t_i \in T_j$, we will let $p_i = q'$ if $\{\bar{s_j}, \bar{t_i}\} \in \mathcal{C}$ and $p_i = 1-q'$ if $\{\bar{s_j}, t_i\} \in \mathcal{C}$. Analogously, if $\mathbb{E}(B_j) \le w_h(s_j)$, then for all $t_i \in T_j$ we will let $p_i = q'$ if $\{\bar{s_j}, t_i\} \in \mathcal{C}$ and $p_i = 1-q'$ if $\{\bar{s_j}, \bar{t_i}\} \in \mathcal{C}$. (Note that $\{\bar{s_j}, \bar{t_i}\}$ and $\{\bar{s_j}, t_i\}$ cannot both be in $\mathcal{C}$, as F is 3-satisfiable.)

Note that after having modified the values of p_i for $t_i \in T_j$ these modifications may affect the values of $\mathbb{E}(B_{j'})$ for $j' > j$. But they do not affect any values $\mathbb{E}(B_{j''})$

for any $j'' < j$. Note also that modifying p_i for $t_i \in T_j$ will not change whether $\mathbb{E}(B_j) \leq w_h(s_j)$ or $\mathbb{E}(B_j) > w_h(s_j)$. In fact if $\mathbb{E}(B_j) \leq w_h(s_j)$ then modifying p_i from $\frac{1}{2}$ will decrease $\mathbb{E}(B_j)$ by $(q' - \frac{1}{2})$, and if $\mathbb{E}(B_j) > w_h(s_j)$ then modifying p_i will increase $\mathbb{E}(B_j)$ by $(q' - \frac{1}{2})$.

The following lemmas now hold.

Lemma 5. $\sum_{i=1}^{r}[\mathbb{E}(A_i) - \mathbb{E}(B_i)] \geq (2q' - 1)|T^*|$, *where T^* is defined in Definition 3.*

Lemma 6. $\mathbb{E}(|B_i - w_h(s_i)|) \geq 1 - q' + (q' - \frac{1}{2})|T_i|$ *for all $i = 1, 2, \ldots, r$.*

The following corollary is equivalent to Theorem 2, and therefore completes the proof of this theorem.

Corollary 4. *We can find, in polynomial time, a truth assignment τ such that*

$$\mathrm{sat}_\tau(F) \geq \frac{2}{3}w(F) + (q - \frac{2}{3})(1 - q')|V_u| + (q - \frac{2}{3})(q' - \frac{1}{2})|V_{h\setminus u}|.$$

Proof. Under the probability distribution given in Definition 2 (which uses Definition 3) the following holds by Lemmas 4, 5 and 6 and the fact that $\frac{2q'-1}{3} > (q - \frac{2}{3})(q' - \frac{1}{2})$: $\mathbb{E}(\mathrm{sat}(F))$ is at least

$$\begin{aligned} &\tfrac{2}{3}w(F) + \tfrac{1}{3}\textstyle\sum_{i=1}^{r}[\mathbb{E}(A_i) - \mathbb{E}(B_i)] + (q - \tfrac{2}{3})\textstyle\sum_{i=1}^{r}\mathbb{E}(|B_i - w_h(s_i)|) \\ \geq &\tfrac{2}{3}w(F) + \tfrac{1}{3}(2q' - 1)|T^*| + (q - \tfrac{2}{3})\left[|V_u|(1 - q') + (q' - \tfrac{1}{2})\textstyle\sum_{i=1}^{r}|T_i|\right] \\ \geq &\tfrac{2}{3}w(F) + (q - \tfrac{2}{3})(1 - q')|V_u| + (q - \tfrac{2}{3})(q' - \tfrac{1}{2})|V_{h\setminus u}|. \end{aligned}$$

This completes the proof as $\mathrm{sat}(F) \geq \mathbb{E}(\mathrm{sat}(F))$ and the required deterministic truth assignment τ can be found using the method of conditional expectations. Although the probability distribution given in Definition 2 uses irrational probabilities, we can apply the method of conditional expectations without performing any calculations on irrationals, by using the identities $q^3 = \frac{1}{3}$ and $q'^2 = \frac{1}{3}$. For a full explanation, see [13]. □

Acknowledgments. This research was partially supported by an International Joint grant of Royal Society.

References

1. Abu-Khzam, F.N., Fernau, H.: Kernels: Annotated, proper and induced. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 264–275. Springer, Heidelberg (2006)
2. Aharoni, R., Linial, N.: Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. J. Comb. Theory, Ser. A 43(2), 196–204 (1986)
3. Alon, N., Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: Solving MAX-k-SAT Above a Tight Lower Bound. Algorithmica, (in press), doi:10.1007/s00453-010-9428-7; A preliminary version in Proc. SODA 2010, pp. 511–517
4. Alon, N., Spencer, J.: The Probabilistic Method, 2nd edn. Wiley, Chichester (2000)

5. Büning, H.K., Kullmann, O.: Minimal Unsatisfiability and Autarkies. In: Handbook of Satisfiability, ch. 11, pp. 339–401., doi:10.3233/978-1-58603-929-5-339
6. Chen, Y., Flum, J., Müller, M.: Lower bounds for kernelizations and other preprocessing procedures. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 118–128. Springer, Heidelberg (2009)
7. Crowston, R., Gutin, G., Jones, M.: Note on Max Lin-2 above Average. Inform. Proc. Lett. 110, 451–454 (2010)
8. Crowston, R., Gutin, G., Jones, M., Kim, E.J., Ruzsa, I.Z.: Systems of linear equations over $\mathbb{F}_2$ and problems parameterized above average. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 164–175. Springer, Heidelberg (2010)
9. Crowston, R., Gutin, G., Jones, M., Yeo, A.: A New Lower Bound on the Maximum Number of Satisfied Clauses in Max-SAT and Its Algorithmic Application. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 84–94. Springer, Heidelberg (2010)
10. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
11. Fleischner, H., Kullmann, O., Szeider, S.: Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. Theoret. Comput. Sci. 289(1), 503–516 (2002)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
13. Gutin, G., Jones, M., Yeo, A.: A new bound for 3-Satisfiable MaxSat and its algorithmic application, arXiv:1104.2818 (April 2011)
14. Huang, M.A., Lieberherr, K.J.: Implications of forbidden structures for extremal algorithmic problems. Theoret. Comput. Sci. 40, 195–210 (1985)
15. Kim, E.J., Williams, R.: Improved Parameterized Algorithms for Constraint Satisfaction, arXiv:1008.0213 (August 2010)
16. Král, D.: Locally satisfiable formulas. In: Proc. SODA 2004, pp. 330–339 (2004)
17. Kullmann, O.: Lean clause-sets: Generalizations of minimally unsatisfiable clause-sets. Discrete Applied Mathematics 130, 209–249 (2003)
18. Lieberherr, K.J., Specker, E.: Complexity of partial satisfaction. J. ACM 28(2), 411–421 (1981)
19. Lieberherr, K.J., Specker, E.: Complexity of partial satisfaction, II. Tech. Report 293 of Dept. of EECS, Princeton Univ. (1982)
20. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSat and MaxCut. J. Algorithms 31(2), 335–354 (1999)
21. Mahajan, M., Raman, V., Sikdar, S.: Parameterizing above or below guaranteed values. J. Comput. Syst. Sci. 75(2), 137–153 (2009)
22. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discr. Appl. Math. 10, 287–295 (1985)
23. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
24. Szeider, S.: Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. J. Comput. Syst. Sci. 69(4), 656–674 (2004)
25. Trevisan, L.: On local versus global satisfiability. SIAM J. Discret. Math. 17(4), 541–547 (2004)
26. Yannakakis, M.: On the approximation of maximum satisfiability. J. Algorithms 17, 475–502 (1994)

On Memoryless Quantitative Objectives*

Krishnendu Chatterjee[1], Laurent Doyen[2], and Rohit Singh[3]

[1] Institute of Science and Technology(IST) Austria
[2] LSV, ENS Cachan & CNRS, France
[3] Indian Institute of Technology(IIT) Bombay

Abstract. In two-player games on graph, the players construct an infinite path through the game graph and get a reward computed by a payoff function over infinite paths. Over weighted graphs, the typical and most studied payoff functions compute the limit-average or the discounted sum of the rewards along the path. Besides their simple definition, these two payoff functions enjoy the property that memoryless optimal strategies always exist.

In an attempt to construct other simple payoff functions, we define a class of payoff functions which compute an (infinite) weighted average of the rewards. This new class contains both the limit-average and the discounted sum functions, and we show that they are the only members of this class which induce memoryless optimal strategies, showing that there is essentially no other simple payoff functions.

1 Introduction

Two-player games on graphs have many applications in computer science, such as the synthesis problem [7], and the model-checking of open reactive systems [1]. Games are also fundamental in logics, topology, and automata theory [17,14,20]. Games with quantitative objectives have been used to design resource-constrained systems [27,9,3,4], and to support quantitative model-checking and robustness [5,6,26].

In a two-player game on a graph, a token is moved by the players along the edges of the graph. The set of states is partitioned into player-1 states from which player 1 moves the token, and player-2 states from which player 2 moves the token. The interaction of the two players results in a play, an infinite path through the game graph. In qualitative zero-sum games, each play is winning exactly for one of the two players; in quantitative games, a payoff function assigns a value to every play, which is paid by player 2 to player 1. Therefore, player 1 tries to maximize the payoff while player 2 tries to minimize it. Typically, the edges of the graph carry a reward, and the payoff is computed as a function of the infinite sequences of rewards on the play.

Two payoff functions have received most of the attention in literature: the *mean-payoff* function (for example, see [11,27,15,19,12,21]) and the *discounted-sum* function (for example, see [24,12,22,23,9]). The mean-payoff value is the long-run average of

* This work was partially supported by FWF NFN Grant S11407-N23 (RiSE) and a Microsoft faculty fellowship.

the rewards. The discounted sum is the infinite sum of the rewards under a discount factor $0 < \lambda < 1$. For an infinite sequence of rewards $w = w_0 w_1 \dots$, we have:

$$\mathsf{MeanPayoff}(w) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} w_i \qquad \mathsf{DiscSum}_\lambda(w) = (1-\lambda) \cdot \sum_{i=0}^{\infty} \lambda^i \cdot w_i$$

While these payoff functions have a simple, intuitive, and mathematically elegant definition, it is natural to ask why they are playing such a central role in the study of quantitative games. One answer is perhaps that *memoryless* optimal strategies exist for these objectives. A strategy is memoryless if it is independent of the history of the play and depends only on the current state. Related to this property is the fact that the problem of deciding the winner in such games is in NP $\cap$ coNP, while no polynomial time algorithm is known for this problem. The situation is similar to the case of parity games in the setting of qualitative games where it was proved that the parity objective is the only prefix-independent objective to admit memoryless winning strategies [8], and the parity condition is known as a canonical way to express ω-regular languages [25].

In this paper, we prove a similar result in the setting of quantitative games. We consider a general class of payoff functions which compute an infinite weighted average of the rewards. The payoff functions are parameterized by an infinite sequence of rational coefficients $\{c_n\}_{n\geq 0}$, and defined as follows:

$$\mathsf{WeightedAvg}(w) = \liminf_{n\to\infty} \frac{\sum_{i=0}^{n} c_i \cdot w_i}{\sum_{i=0}^{n} c_i}.$$

We consider this class of functions for its simple and natural definition, and because it generalizes both mean-payoff and discounted-sum which can be obtained as special cases, namely for $c_i = 1$ for all[1] $i \geq 0$, and $c_i = \lambda^i$ respectively. We study the problem of characterizing which payoff functions in this class admit memoryless optimal strategies for both players. Our results are as follows:

1. If the series $\sum_{i=0}^{\infty} c_i$ converges (and is finite), then the discounted sum is the *only* payoff function that admits memoryless optimal strategies for both players.
2. If the series $\sum_{i=0}^{\infty} c_i$ does not converge, but the sequence $\{c_n\}_{n\geq 0}$ is bounded, then for memoryless optimal strategies the payoff function is equivalent to the mean-payoff function (equivalent for the optimal value and optimal strategies of both players).

Thus our results show that the discounted sum and mean-payoff functions, besides their elegant and intuitive definition, are the only members from a large class of natural payoff functions such that both players have memoryless optimal strategies. In other words, there is essentially no other simple payoff functions in the class of weighted infinite average payoff functions. This further establishes the canonicity of the mean-payoff and discounted-sum functions, and suggests that they should play a central role in the emerging theory of quantitative automata and languages [10,16,2,5].

In the study of games on graphs, characterizing the classes of payoff functions that admit memoryless strategies is a research direction that has been investigated in [13]

[1] Note that other sequences also define the mean-payoff function, such as $c_i = 1 + 1/2^i$.

which give general conditions on the payoff functions such that both players have memoryless optimal strategies, and [18] which presents similar results when only one player has memoryless optimal strategies. The conditions given in these previous works are useful in this paper, in particular the fact that it is sufficient to check that memoryless strategies are sufficient in one-player games [13]. However, conditions such as sub-mixing and selectiveness of the payoff function are not immediate to establish, especially when the sum of the coefficients $\{c_n\}_{n\geq 0}$ does not converge. We identify the necessary condition of boundedness of the coefficients $\{c_n\}_{n\geq 0}$ to derive the mean-payoff function. Our results show that if the sequence is convergent, then discounted sum (specified as $\{\lambda^n\}_{n\geq 0}$, for $\lambda < 1$) is the only memoryless payoff function; and if the sequence is divergent and bounded, then mean-payoff (specified as $\{\lambda^n\}_{n\geq 0}$ with $\lambda = 1$) is the only memoryless payoff function. However we show that if the sequence is divergent and unbounded, then there exists a sequence $\{\lambda^n\}_{n\geq 0}$, with $\lambda > 1$, that does not induce memoryless optimal strategies.

2 Definitions

Game graphs. A two-player *game graph* $G = \langle Q, E, w\rangle$ consists of a finite set Q of states partitioned into player-1 states Q_1 and player-2 states Q_2 (i.e., $Q = Q_1 \cup Q_2$), and a set $E \subseteq Q \times Q$ of edges such that for all $q \in Q$, there exists (at least one) $q' \in Q$ such that $(q, q') \in E$. The weight function $w : E \to \mathbb{Q}$ assigns a rational valued reward to each edge. For a state $q \in Q$, we write $E(q) = \{r \in Q \mid (q, r) \in E\}$ for the set of successor states of q. A *player*-1 *game* is a game graph where $Q_1 = Q$ and $Q_2 = \emptyset$. Player-2 games are defined analogously.

Plays and strategies. A game on G starting from a state $q_0 \in Q$ is played in rounds as follows. If the game is in a player-1 state, then player 1 chooses the successor state from the set of outgoing edges; otherwise the game is in a player-2 state, and player 2 chooses the successor state. The game results in a *play* from q_0, i.e., an infinite path $\rho = \langle q_0 q_1 \dots\rangle$ such that $(q_i, q_{i+1}) \in E$ for all $i \geq 0$. We write Ω for the set of all plays. The prefix of length n of ρ is denoted by $\rho(n) = q_0 \dots q_n$. A strategy for a player is a recipe that specifies how to extend plays. Formally, a *strategy* for player 1 is a function $\sigma : Q^* Q_1 \to Q$ such that $(q, \sigma(\rho \cdot q)) \in E$ for all $\rho \in Q^*$ and $q \in Q_1$. The strategies for player 2 are defined analogously. We write Σ and Π for the sets of all strategies for player 1 and player 2, respectively.

An important special class of strategies are *memoryless* strategies which do not depend on the history of a play, but only on the current state. Each memoryless strategy for player 1 can be specified as a function σ: $Q_1 \to Q$ such that $\sigma(q) \in E(q)$ for all $q \in Q_1$, and analogously for memoryless player 2 strategies.

Given a starting state $q \in Q$, the *outcome* of strategies $\sigma \in \Sigma$ for player 1, and $\pi \in \Pi$ for player 2, is the play $\omega(q, \sigma, \pi) = \langle q_0 q_1 \dots\rangle$ such that : $q_0 = q$ and for all $k \geq 0$, if $q_k \in Q_1$, then $\sigma(q_0, q_1, \dots, q_k) = q_{k+1}$, and if $q_k \in Q_2$, then $\pi(q_0, q_1, \dots, q_k) = q_{k+1}$.

Payoff functions, optimal strategies. The objective of player 1 is to construct a play that maximizes a *payoff function* $\phi : \Omega \to \mathbb{R} \cup \{-\infty, +\infty\}$ which is a measurable function that assigns to every value a real-valued payoff. The value for player 1 is the maximal payoff that can be achieved against all strategies of the other player. Formally the value for player 1 for a starting state q is defined as $val_1(\phi) = \sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \phi(\omega(q, \sigma, \pi))$. A strategy σ^* is *optimal* for player 1 from q if the strategy achieves at least the value of the game against all strategies for player 2, i.e., $\inf_{\pi \in \Pi} \phi(\omega(q, \sigma^*, \pi)) = val_1(\phi)$. The values and optimal strategies for player 2 are defined analogously.

The mean-payoff and discounted-sum functions are examples of payoff functions that are well studied, probably because they are simple in the sense that they induce memoryless optimal strategies and that this property yields conceptually simple fix-point algorithms for game solving [24,11,27,12]. In an attempt to construct other simple payoff functions, we define the class of *weighted average payoffs* which compute (infinite) weighted averages of the rewards, and we ask which payoff functions in this class induce memoryless optimal strategies.

We say that a sequence $\{c_n\}_{n \geq 0}$ of rational numbers has *no zero partial sum* if $\sum_{i=0}^{n} c_i \neq 0$ for all $n \geq 0$. Given a sequence $\{c_n\}_{n \geq 0}$ with no zero partial sum, the *weighted average payoff function* for a play $\langle q_0 q_1 q_2 \dots \rangle$ is

$$\phi(q_0 q_1 q_2 \dots) = \liminf_{n \to \infty} \frac{\sum_{i=0}^{n} c_i \cdot w(q_i, q_{i+1})}{\sum_{i=0}^{n} c_i}.$$

Note that we use $\liminf_{n \to \infty}$ in this definition because the plain limit may not exist in general. The behavior of the weighted average payoff functions crucially depends on whether the series $S = \sum_{i=0}^{\infty} c_i$ converges or not. In particular, the plain limit exists if S converges (and is finite). Accordingly, we consider the cases of converging and diverging sum of weights to characterize the class of weighted average payoff functions that admit memoryless optimal strategies for both players. Note that the case where $c_i = 1$ for all $i \geq 0$ gives the mean-payoff function (and S diverges), and the case $c_i = \lambda^i$ for $0 < \lambda < 1$ gives the discounted sum with discount factor λ (and S converges). All our results hold if we consider $\limsup_{n \to \infty}$ instead of $\liminf_{n \to \infty}$ in the definition of weighted average objectives.

In the sequel, we consider payoff functions $\phi : \mathbb{Q}^\omega \to \mathbb{R}$ that maps an infinite sequence of rational numbers to a real value with the implicit assumption that the value of a play $q_0 q_1 q_2 \dots \in Q^\omega$ according to ϕ is $\phi(w(q_0, q_1) w(q_1, q_2) \dots)$ since the sequence of rewards determines the payoff value.

We recall the following useful necessary condition for memoryless optimal strategies to exist [13]. A payoff function ϕ is *monotone* if whenever there exists a finite sequence of rewards $x \in \mathbb{Q}^*$ and two sequences $u, v \in \mathbb{Q}^\omega$ such that $\phi(xu) \leq \phi(xv)$, then $\phi(yu) \leq \phi(yv)$ for all finite sequence of rewards $y \in \mathbb{Q}^*$.

Lemma 2.1 ([13]). *If the payoff function ϕ induces memoryless optimal strategy for all two-player game graphs, then ϕ is monotone.*

3 Weighted Average with Converging Sum of Weights

The main result of this section is that for converging sum of weights (i.e., if $\lim_{n \to \infty} \sum_{i=0}^{n} c_i = c^* \in \mathbb{R}$), the only weighted average payoff function that induce memoryless optimal strategies is the discounted sum.

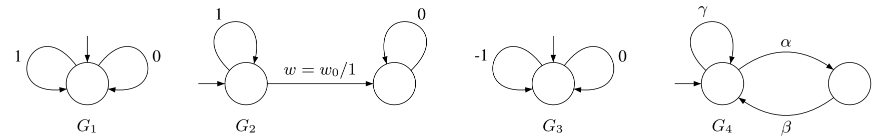


Fig. 1. Examples of one-player game graphs

Theorem 3.1. *Let $(c_n)_{n\in\mathbb{N}}$ be a sequence of real numbers with no zero partial sum such that $\sum_{i=0}^{\infty} c_i = c^* \in \mathbb{R}$. The weighted average payoff function defined by $(c_n)_{n\in\mathbb{N}}$ induces optimal memoryless strategies for all two-player game graphs if and only if there exists $0 < \lambda < 1$ such that $c_{i+1} = \lambda \cdot c_i$ for all $i \geq 0$.*

To prove Theorem 3.1, we first use its assumptions to obtain necessary conditions for the weighted average payoff function defined by $(c_n)_{n\in\mathbb{N}}$ to induce optimal memoryless strategies. By *assumptions of Theorem 3.1*, we refer to the fact that $(c_n)_{n\in\mathbb{N}}$ is a sequence of real numbers with no zero partial sum such that $\sum_{i=0}^{\infty} c_i = c^* \in \mathbb{R}$, and that it defines a weighted average payoff function that induces optimal memoryless strategies for all 2-player game graphs. All lemmas of this section use the the assumptions of Theorem 3.1, but we generally omit to mention them explicitly.

Let $d_n = \sum_{i=0}^{n-1} c_i$, $l = \liminf_{n\to\infty} \frac{1}{d_n}$ and $L = \limsup_{n\to\infty} \frac{1}{d_n}$. The assumption that $\sum_{i=0}^{\infty} c_i = c^* \in \mathbb{R}$ implies that $l \neq 0$. Note that $c_0 \neq 0$ since $(c_n)_{n\in\mathbb{N}}$ is a sequence with no zero partial sum. We can define the sequence $c'_n = \frac{c_n}{c_0}$ which defines the same payoff function ϕ. Therefore we assume without loss of generality that $c_0 = 1$.

Discussion about following three lemmas. In the following three lemmas we prove properties of a sequence $(c_n)_{n\in\mathbb{N}}$ with the assumption that the sequence induces optimal memoryless strategies in all game graphs. However note that the property we prove is about the sequence, and hence in all the lemmas we need to show witness game graphs where the sequence must satisfy the required properties.

Lemma 3.1. *If the weighted average payoff function defined by $(c_n)_{n\in\mathbb{N}}$ induces optimal memoryless strategies for all two-player game graphs, then $0 \leq l \leq L \leq 1$.*

Proof. Consider the one-player game graph G_1 shown in Fig. 1. In one-player games, strategies correspond to paths. The two memoryless strategies give the paths 0^ω and 1^ω with payoff value 0 and 1 respectively. The strategy which takes the edge with reward 1 once, and then always the edge with reward 0 gets the payoff $\phi(10^\omega) = \liminf_{n\to\infty} \frac{1}{d_n} = l$. Similarly, the path 01^ω has the payoff $\phi(01^\omega) = \liminf_{n\to\infty} \left(1 - \frac{1}{d_n}\right) = 1 - \limsup_{n\to\infty} \frac{1}{d_n} = 1 - L$. As all such payoffs must be between the payoffs obtained by the only two memoryless strategies, we have $l \geq 0$ and $L \leq 1$, and the result follows ($L \geq l$ follows from their definition). □

Lemma 3.2. *There exists $w_0 \in \mathbb{N}$ such that $w_0 > 1$, $w_0 l > 1$ and the following inequalities hold, for all $k \geq 0$: $c_k l \leq 1 - d_k L$ and $c_k w_0 l \geq 1 - d_k L$.*

Proof. Since $1 \geq l > 0$ (by Lemma 3.1), we can choose $w_0 \in \mathbb{N}$ such that $w_0 l > 1$ (and $w_0 > 1$). Consider the game graph G_2 shown in Fig. 1 and the case when $w = 1$. The optimal memoryless strategy is to stay on the starting state forever because $\phi(10^\omega) = l \leq \phi(1^w) = 1$. Using Lemma 2.1, we conclude that since $\phi(10^\omega) \leq \phi(1^\omega)$, we must have $\phi(0^k10^\omega) \leq \phi(0^k1^\omega)$ i.e. $c_kl \leq 1 - \left(\sum_{i=0}^{k-1} c_i\right) L$ which implies $c_kl \leq 1 - d_kL$.

Consider the case when $w = w_0$ in Fig. 1. The optimal memoryless strategy is to choose the edge with reward w_0 from the starting state since $\phi(w_00^\omega) = w_0l > \phi(1^\omega) = 1$. Using Lemma 2.1, we conclude that since $\phi(w_00^\omega) > \phi(1^\omega)$, we must have $\phi(0^kw_00^\omega) \geq \phi(0^k1^\omega)$ i.e. $c_kw_0l \geq 1 - \left(\sum_{i=0}^{k-1} c_i\right) L$ which implies $c_kw_0l \geq 1 - d_kL$. $\square$

From the inequalities in Lemma 3.2, it follows that for all k we have $c_kl \leq c_kw_0l$; and since $w_0 > 1$ and $l > 0$ we must have $c_k \geq 0$ for all k.

Corollary 3.1. *Assuming $c_0 = 1$, we have $c_k \geq 0$ for all $k \geq 0$.*

It follows from Corollary 3.1 that the sequence $(d_n)_{n\geq 0}$ is increasing and bounded from above (if d_n was not bounded, then there would exist a subsequence (d_{n_k}) which diverges, implying that the sequence $\{\frac{1}{d_{n_k}}\}$ converges to 0 in contradiction with the fact that $\liminf_{n\to\infty} \frac{1}{d_n} = l > 0$). Therefore, d_n must converge to some real number say $c^* > 0$ (since $c_0 = 1$). We need a last lemma to prove Theorem 3.1. Recall that we have $c_i \geq 0$ for all i and $\sum_{i=0}^{\infty} c_i = c^* > 0$. Given a finite game graph G, let W be the largest reward in absolute value. For any sequence of rewards (w_n) in a run on G, the sequence $\chi_n = \sum_{i=0}^{n} c_i(w_i + W)$ is increasing and bounded from above by $2 \cdot W \cdot d_n$ and thus by $2 \cdot W \cdot c^*$. Therefore, χ_n is a convergent sequence and $\sum_{i=0}^{\infty} c_iw_i$ converges as well. Now, we can write the payoff function as $\phi(w_0w_1\dots) = \frac{\sum_{i=0}^{\infty} c_iw_i}{c^*}$. We decompose c^* into $S_0 = \sum_{i=0}^{\infty} c_{2i}$ and $S_1 = \sum_{i=0}^{\infty} c_{2i+1}$, i.e. $c^* = S_0 + S_1$. Note that S_0 and S_1 are well defined.

Lemma 3.3. *For all reals α, β, γ, if $\alpha S_0 + \beta S_1 \leq \gamma(S_0 + S_1)$, then $(\gamma - \alpha)c_i \geq (\beta - \gamma)c_{i+1}$ for all $i \geq 0$.*

Proof. Consider the game graph G_4 as shown in Fig. 1. The condition $\alpha S_0 + \beta S_1 \leq \gamma(S_0 + S_1)$ implies that the optimal memoryless strategy is to always choose the edge with reward γ. This means that $\phi(\gamma^i\alpha\beta\gamma^\omega) \leq \phi(\gamma^\omega)$ hence $\alpha c_i + \beta c_{i+1} \leq \gamma(c_i + c_{i+1})$, i.e. $(\gamma - \alpha)c_i \geq (\beta - \gamma)c_{i+1}$ for all $i \geq 0$. $\square$

We are now ready to prove the main theorem of this section.

Proof (of Theorem 3.1). First, we show that $S_1 \leq S_0$. By contradiction, assume that $S_1 > S_0$. Choosing $\alpha = 1$, $\beta = -1$, and $\gamma = 0$ in Lemma 3.3, and since $S_0 - S_1 \leq 0$, we get $-c_i \geq -c_{i+1}$ for all $i \geq 0$ which implies $c_n \geq c_0 = 1$ for all n, which contradicts that $\sum_{i=0}^{\infty} c_i$ converges to $c^* \in \mathbb{R}$.

Now, we have $S_1 \leq S_0$ and let $\lambda = \frac{S_1}{S_0} \leq 1$. Consider a sequence of rational numbers $\frac{l_n}{k_n}$ converging to λ from the right, i.e., $\frac{l_n}{k_n} \geq \lambda$ for all n, and $\lim_{n\to\infty} \frac{l_n}{k_n} = \lambda$. Taking $\alpha = 1$, $\beta = k_n + l_n + 1$, and $\gamma = l_n + 1$ in Lemma 3.3, and since the condition $S_0 + (k_n + l_n + 1)S_1 \leq (l_n + 1)(S_0 + S_1)$ is equivalent to $k_nS_1 \leq l_nS_0$ which holds since $\frac{l_n}{k_n} \geq \lambda$, we obtain $l_nc_i \geq k_nc_{i+1}$ for all $n \geq 0$ and all $i \geq 0$, that is $c_{i+1} \leq \frac{l_n}{k_n}c_i$ and in the limit for $n \to \infty$, we get $c_{i+1} \leq \lambda c_i$ for all $i \geq 0$.

Similarly, consider a sequence of rational numbers $\frac{r_n}{s_n}$ converging to λ from the left. Taking $\alpha = r_n + s_n + 1$, $\beta = 1$, and $\gamma = s_n + 1$ in Lemma 3.3, and since the condition $(r_n + s_n + 1)S_0 + S_1 \leq (s_n + 1)(S_0 + S_1)$ is equivalent to $r_n S_0 \leq s_n S_1$ which holds since $\frac{r_n}{s_n} \leq \lambda$, we obtain $r_n c_i \leq s_n c_{i+1}$ for all $n \geq 0$ and all $i \geq 0$, that is $c_{i+1} \geq \frac{r_n}{s_n} c_i$ and in the limit for $n \to \infty$, we get $c_{i+1} \geq \lambda c_i$ for all $i \geq 0$.

The two results imply that $c_{i+1} = \lambda c_i$ for all $i \geq 0$ where $0 \leq \lambda < 1$. Note that $\lambda \neq 1$ because $\sum_{i=0}^{\infty} c_i$ converges. □

Since it is known that for $c_i = \lambda^i$, the weighted average payoff function induces memoryless optimal strategies in all two-player games, Theorem 3.1 shows that discounted sum is the only memoryless payoff function when the sum of weights $\sum_{i=0}^{\infty} c_i$ converges.

4 Weighted Average with Diverging Sum of Weights

In this section we consider weighted average objectives such that the sum of the weights $\sum_{i=0}^{\infty} c_i$ is divergent. We first consider the case when the sequence $(c_n)_{n\in\mathbb{N}}$ is bounded and show that the mean-payoff function is the only memoryless one.

4.1 Bounded Sequence

We are interested in characterizing the class of weighted average objectives that are memoryless, under the assumption the sequence (c_n) is *bounded*, i.e., there exists a constant c such that $|c_n| \leq c$ for all n. The boundedness assumption is satisfied by the important special case of regular sequence of weights which can be produced by a deterministic finite automaton. We say that a sequence $\{c_n\}$ is *regular* if it is eventually periodic, i.e. there exist $n_0 \geq 0$ and $p > 0$ such that $c_{n+p} = c_n$ for all $n \geq n_0$. Recall that we assume the partial sum to be always non-zero, i.e., $d_n = \sum_{i=0}^{n-1} c_i \neq 0$ for all n. We show the following result.

Theorem 4.1. *Let $(c_n)_{n\in\mathbb{N}}$ be a sequence of real numbers with no zero partial sum such that $\sum_{i=0}^{\infty}|c_i| = \infty$ (the sum is divergent) and there exists a constant c such that $|c_i| \leq c$ for all $i \geq 0$ (the sequence is bounded). The weighted average payoff function ϕ defined by $(c_n)_{n\in\mathbb{N}}$ induces optimal memoryless strategies for all two-player game graphs if and only if ϕ coincides with the mean-payoff function over regular words.*

Remark. From Theorem 4.1, it follows that all mean-payoff functions ϕ over bounded sequences that induce optimal memoryless strategies are equivalent to the mean-payoff function, in the sense that the optimal value and optimal strategies for ϕ are the same as for the mean-payoff function. This is because memoryless strategies induce a play that is a regular word. We also point out that it is not necessary that the sequence $(c_n)_{n\geq 0}$ consists of a constant value to define the mean-payoff function. For example, the payoff function defined by the sequence $c_n = 1 + 1/(n+1)^2$ also defines the mean-payoff function.

We prove Theorem 4.1 through a sequence of lemmas (using the the assumptions of Theorem 4.1, but we generally omit to mention them explicitly). In the following lemma we prove the existence of the limit of the sequence $\{\frac{1}{d_n}\}_{n\geq 0}$.

Lemma 4.1. *If* $\liminf_{n\to\infty} \frac{1}{d_n} = 0$*, then* $\limsup_{n\to\infty} \frac{1}{d_n} = 0$.

Proof. Since $l = \liminf_{n\to\infty} \frac{1}{d_n} = 0$, there is a subsequence $\{d_{n_k}\}$ which either diverges to $+\infty$ or $-\infty$.

1. If the subsequence $\{d_{n_k}\}$ diverges to $+\infty$, assume without loss of generality that each $d_{n_k} > 0$. Consider the one-player game graph G_3 shown in Figure 1. We consider the run corresponding to taking the edge with weight -1 for the first n_k steps followed by taking the 0 edge forever. The payoff for this run is given by

$$\liminf_{n\to\infty} \frac{-d_{n_k}}{d_n} = -d_{n_k} \cdot \limsup_{n\to\infty} \frac{1}{d_n} = -d_{n_k} \cdot L.$$

Since we assume the existence of memoryless optimal strategies this payoff should lie between -1 and 0. This implies that $d_{n_k} \cdot L \leq 1$ for all k. Since $L \geq l \geq 0$ and the sequence d_{n_k} is unbounded, we must have $L = 0$.

2. If the subsequence $\{d_{n_k}\}$ diverges to $-\infty$, assume that each $d_{n_k} < 0$. Consider the one-player game graph G_1 shown in Figure 1. We consider the run corresponding to taking the edge with weight 1 for the first n_k steps followed by taking the 0 edge forever. The payoff for this run is given by

$$\liminf_{n\to\infty} \frac{d_{n_k}}{d_n} = -|d_{n_k}| \cdot \limsup_{n\to\infty} \frac{1}{d_n} = -|d_{n_k}| \cdot L.$$

This payoff should lie between 0 and 1 (optimal strategies being memoryless), and this implies $L = 0$ as above. □

Since $\limsup_{n\to\infty} d_n = \infty$, Lemma 4.1 concludes that the sequence $\{\frac{1}{d_n}\}$ converges to 0 i.e. $\lim_{n\to\infty} \frac{1}{d_n} = 0$. It also gives us the following corollaries which are a simple consequence of the fact that $\liminf_{n\to\infty}(a_n + b_n) = a + \liminf_{n\to\infty} b_n$ if a_n converges to a.

Corollary 4.1. *If* $l = 0$*, then the payoff function* ϕ *does not depend upon any finite prefix of the run, i.e.,* $\phi(a_1a_2\ldots a_ku) = \phi(0^ku) = \phi(b_1b_2\ldots b_ku)$ *for all* a_i*'s and* b_i*'s.*

Corollary 4.2. *If* $l = 0$*, then the payoff function* ϕ *does not change by modifying finitely many values in the sequence* $\{c_n\}_{n\geq 0}$.

By Corollary 4.1, we have $\phi(xa^\omega) = a$ for all $a \in \mathbb{R}$. For $0 \leq i \leq k-1$, consider the payoff $S_{k,i} = \phi\left((0^i10^{k-i-1})^\omega\right)$ for the infinite repetition of the finite sequence of k rewards in which all rewards are 0 except the $(i+1)$th which is 1. We show that $S_{k,i}$ is independent of i.

Lemma 4.2. *We have* $S_{k,0} = S_{k,1} = \cdots = S_{k,k-1} \leq \frac{1}{k}$.

Proof. If $S_{k,0} \leq S_{k,1}$ then by prefixing by the single letter word 0 and using Lemma 2.1 we conclude that $S_{k,1} \leq S_{k,2}$. We continue this process until we get $S_{k,k-2} \leq S_{k,k-1}$. After applying this step again we get

$$S_{k,k-1} \leq \phi\left(0(0^{k-1}1)^\omega\right) = \phi\left(1(0^{k-1}1)^\omega\right) = \phi\left((10^{k-1})^\omega\right) = S_{k,0}.$$

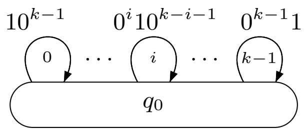


Fig. 2. The game $G(k,i)$

Hence, we have $S_{k,0} \leq S_{k,1} \leq \cdots \leq S_{k,k-1} \leq S_{k,0}$. Thus we have $S_{k,i}$ is a constant irrespective of the value of i. A similar argument works in the other case when $S_{k,0} \geq S_{k,1}$.

We will show that $S_{k,i} \leq \frac{1}{k}$ for $0 \leq i \leq k-1$. For this, we take $a_{i,n}$ to be the n^{th} term of the sequence whose $\liminf$ is the value $\phi((0^i i 0^{k-i-1})^\omega) = S_{k,i}$ i.e. $a_{i,n} = \frac{\sum_{j \in \{j \geq 0 \mid jk+i \leq n\}} c_{jk+i}}{\sum_{j=0}^{n} c_j} = \frac{\sum_{j \in \{0 \leq j \leq n \mid j \equiv i \pmod{k}\}} c_j}{\sum_{j=0}^{n} c_j}$. Clearly, $\sum_{i=0}^{k-1} a_{i,n} = 1$ and hence using the fact that $\liminf_{n\to\infty}(a_{0,n} + a_{1,n} + \cdots + a_{k-1,n}) \geq \liminf_{n\to\infty} a_{0,n} + \cdots + \liminf_{n\to\infty} a_{k-1,n}$, we have $1 \geq \sum_{i=0}^{k-1} S_{k,i} = kS_{k,i}$ (since all $S_{k,i}$'s are constant with respect to i) and therefore, $S_{k,i} \leq \frac{1}{k}$ for $0 \leq i \leq k-1$. □

Let $T_{k,i} = -\phi\left((0^i(-1)0^{k-i-1})^\omega\right)$. By similar argument as in the proof of Lemma 4.2, we show that $T_{k,0} = T_{k,1} = \cdots = T_{k,k-1} \geq \frac{1}{k}$.

We now show that (d_n) must eventually have always the same sign, i.e., there exists n_0 such that $\mathsf{sign}(d_m) = \mathsf{sign}(d_n)$ for all $m, n \geq n_0$. Note that by the assumption of non-zero partial sums, we have $d_n \neq 0$ for all n.

Lemma 4.3. *The d_n's eventually have the same sign.*

Proof. Let $c > 0$ be such that $|c_n| < c$ for all n. Since (d_n) is unbounded, there exists n_0 such that $|d_n| > c$ for all $n > n_0$ and then if there exists $m > n_0$ such that $d_m > 0$ and $d_{m+1} < 0$, we must have $d_m > c$ and $d_{m+1} < -c$. Thus we have $c_{m+1} = d_{m+1} - d_m < -2c$, and hence $|c_{m+1}| > 2c$ which contradicts the boundedness assumption on (c_n). □

If the d_n's are eventually negative then we use the sequence $\{c'_n = -c_n\}$ to obtain the same payoff and in this case $d_n = -\sum_{i=0}^{\infty} c_i$ will be eventually positive. Therefore we assume that there is some n_0 such that $d_n > 0$ for all $n > n_0$. Let $\beta = max\{|c_0|, |c_1|, \ldots, |c_{n_0}|\}$. We replace c_0 by 1 and all c_i's with β for $1 \leq i \leq n_0$. By corollary 4.2 we observe that the payoff function will still not change. Hence, we can also assume that $d_n > 0$ for all $n \geq 0$.

Lemma 4.4. *We have $S_{k,i} = \frac{1}{k} = T_{k,i}$ for all $0 \leq i \leq k-1$.*

Proof. Consider the game graph $G(k,i)$ which consists of state q_0 in which the player can choose among k cycles of length k where in the ith cycle, all rewards are 0 except on the $(i+1)$th edge which has reward 1 (see Fig. 2).

Consider the strategy in state q_0 where the player after every $k \cdot r$ steps ($r \geq 0$) chooses the cycle which maximizes the contribution for the next k edges. Let i_r be the index such that $kr \leq i_r \leq kr + k - 1$ and $c_{i_r} = \max\{c_{kr}, \ldots, c_{kr+k-1}\}$ for

$r \geq 0$. The payoff for this strategy is $\liminf_{n\to\infty} t_n$ where $t_n = \frac{c_{i_0}+c_{i_1}+\cdots+c_{i_{r-1}}}{d_n}$ for $i_{r-1} \leq n < i_r$.

Note that $c_{i_r} \geq \frac{\sum_{i=kr}^{kr+k-1} c_i}{k}$ (the maximum is greater than the average), and we get the following (where c is a bound on $(|c_n|)_{n\geq 0}$):

$$t_n \geq \frac{\sum_{i=0}^{n-1} c_i}{k \cdot d_n} - \frac{c}{d_n}, \quad \text{hence } \liminf_{n\to\infty} t_n \geq \frac{1}{k} - \liminf_{n\to\infty} \frac{c}{d_n} = \frac{1}{k}.$$

By Lemma 4.2, the payoff of all memoryless strategies in $G(k,i)$ is $S_{k,0}$, and the fact that memoryless optimal strategies exist entails that $S_{k_0} = \liminf_{n\to\infty} t_n \geq \frac{1}{k}$, and thus $S_{k,0} = \frac{1}{k} = S_{k,i}$ for all $0 \leq i \leq k-1$.

Using a similar argument on the graph $G(k,i)$ with reward -1 instead of 1, we obtain $T_{k,0} = \frac{1}{k} = T_{k,i}$ for all $0 \leq i \leq k-1$. □

From Lemma 4.4, it follows that $S_{k,i} = \phi((0^i 10^{k-i-1})^\omega) = \lim_{n\to\infty} \frac{\sum_{r=0}^{\left[\frac{n}{k}\right]} c_{kr+i}}{d_n} = \frac{1}{k}$, and hence,

$$\phi\left((a_0 a_1 \ldots a_{k-1})^\omega\right) = \liminf_{n\to\infty} \sum_{i=0}^{k-1} \left(a_i \cdot \frac{\sum_{r=0}^{\left[\frac{n}{k}\right]} c_{kr+i}}{d_n} \right) = \sum_{i=0}^{k-1} \left(a_i \cdot \lim_{n\to\infty} \frac{\sum_{r=0}^{\left[\frac{n}{k}\right]} c_{kr+i}}{d_n} \right)$$
$$= \frac{\sum_{i=0}^{k-1} a_i}{k}.$$

We show that the payoff of a regular word $u = b_1 b_2 \ldots b_m (a_0 a_1 \ldots a_{k-1})^\omega$ matches the mean-payoff value.

Lemma 4.5. *If $u := b_1 b_2 \ldots b_m (a_0 a_1 \ldots a_{k-1})^\omega$ and $v = (a_0 a_1 \ldots a_{k-1})^\omega$ are two regular sequences of weights then $\phi(u) = \phi(v) = \frac{\sum_{i=0}^{k-1} a_i}{k}$.*

Proof. Let $r \in \mathbb{N}$ be such that $kr > m$. If $\phi(v) \leq \phi(0v)$ then using Lemma 2.1 we obtain $\phi(0v) \leq \phi(0^2 v)$. Applying the lemma again and again, we get, $\phi(v) \leq \phi(0^m v) \leq \phi(0^{kr} v)$. From Corollary 4.1 we obtain $\phi(0^m v) = \phi(b_1 b_2 \ldots b_m v) = \phi(u)$ (hence $\phi(v) \leq \phi(0^m v) = \phi(u)$) and $\phi(0^{kr} v) = \phi\left((a_1 a_2 \ldots a_k)^r v\right) = \phi(v)$ (hence $\phi(u) = \phi(0^m v) \leq \phi(0^{kr} v) = \phi(v)$). Therefore, $\phi(u) = \phi(v) = \frac{\sum_{i=0}^{k-1} a_i}{k}$. The same argument goes through for the case $\phi(v) \geq \phi(0v)$. □

Proof (of Theorem 4.1). In Lemma 4.5 we have shown that the payoff function ϕ must match the mean-payoff function for regular words, if the sequence $\{c_n\}_{n\geq 0}$ is bounded. Since memoryless strategies in game graphs result in regular words over weights, it follows that the only payoff function that induces memoryless optimal strategies is the mean-payoff function which concludes the proof. □

As every regular sequence is bounded, Corollary 4.3 follows from Theorem 4.1.

Corollary 4.3. *Let $(c_n)_{n\in\mathbb{N}}$ be a regular sequence of real numbers with no zero partial sum such that $\sum_{i=0}^{\infty} |c_i| = \infty$ (the sum is divergent). The weighted average payoff function ϕ defined by $(c_n)_{n\in\mathbb{N}}$ induces optimal memoryless strategies for all two-player game graphs if and only if ϕ is the mean-payoff function.*

4.2 Unbounded Sequence

The results of Section 3 and Section 4.1 can be summarized as follows: (1) if the sum of c_i's is convergent, then the sequence $\{\lambda^i\}_{i\geq 0}$, with $\lambda < 1$ (discounted sum), is the only class of payoff functions that induce memoryless optimal strategies; and (2) if the sum is divergent but the sequence (c_n) is bounded, then the mean-payoff function is the only payoff function with memoryless optimal strategies (and the mean-payoff function is defined by the sequence $\{\lambda^i\}_{i\geq 0}$, with $\lambda = 1$). The remaining natural question is that if the sum is divergent and unbounded, then is the sequence $\{\lambda^i\}_{i\geq 0}$, with $\lambda > 1$, the only class that has memoryless optimal strategies. Below we show with an example that the class $\{\lambda^i\}$, with $\lambda > 1$, need not necessarily have memoryless optimal strategies.

We consider the payoff function given by the sequence $c_n = 2^n$. It is easy to verify that the sequence satisfies the partial non-zero assumption. We show that the payoff function does not result into memoryless optimal strategies. To see this, we observe that the payoff for a regular word $w = b_0 b_1 \dots b_t (a_0 a_1 \dots a_{k-1})^\omega$ is given by $\min_{0\leq i\leq k-1} \left(\frac{a_i + 2a_{i+1} + \cdots + 2^{k-1} a_{i+k-1}}{1+2+\cdots+2^{k-1}} \right)$ i.e., the payoff for a regular word is the least possible weighted average payoff for its cycle considering all possible cyclic permutations of its indices (note that the addition in indices is performed modulo k).

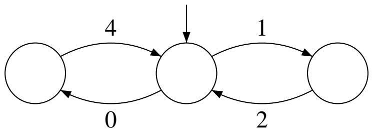


Fig. 3. The game $\mathcal{G}_{1024}$

Now, consider the game graph $\mathcal{G}_{1024}$ shown in figure 3. The payoffs for both the memoryless strategies (choosing the left or the right edge in the start state) are $\min\left(\frac{5}{3}, \frac{4}{3}\right)$ and $\min\left(\frac{4}{3}, \frac{8}{3}\right)$ which are both equal to $\frac{4}{3}$. Although, if we consider the strategy which alternates between the two edges in the starting state then the payoff obtained is $\min\left(\frac{37}{15}, \frac{26}{15}, \frac{28}{15}, \frac{14}{15}\right) = \frac{14}{15}$ which is less than payoff for both the memoryless strategies. Hence, the player who minimizes the payoff does not have a memoryless optimal strategy in the game $\mathcal{G}_{1024}$. The example establishes that the sequence $\{2^n\}_{n\geq 0}$ does not induce optimal strategies.

Open question. Though weighted average objectives such that the sequence is divergent and unbounded may not be of the greatest practical relevance, it is an interesting theoretical question to characterize the subclass that induce memoryless strategies. Our counter-example shows that $\{\lambda^n\}_{n\geq 0}$ with $\lambda > 1$ is not in this subclass.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49, 672–713 (2002)
2. Bojańczyk, M.: Beyond omega-regular languages. In: Proc. of STACS. LIPIcs, vol. 5, pp. 11–16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)

3. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
4. Chatterjee, K., Doyen, L.: Energy parity games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 599–610. Springer, Heidelberg (2010)
5. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Transactions on Computational Logic 11(4) (2010)
6. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Singh, R.: Measuring and synthesizing systems in probabilistic environments. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 380–395. Springer, Heidelberg (2010)
7. Church, A.: Logic, arithmetic, and automata. In: Proceedings of the International Congress of Mathematicians, pp. 23–35. Institut Mittag-Leffler (1962)
8. Colcombet, T., Niwiński, D.: On the positional determinacy of edge-labeled games. Theor. Comput. Sci. 352(1-3), 190–196 (2006)
9. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the future in systems theory. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1022–1037. Springer, Heidelberg (2003)
10. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. 380(1-2) (2007)
11. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. Int. Journal of Game Theory 8(2), 109–113 (1979)
12. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
13. Gimbert, H., Zielonka, W.: Games where you can play optimally without any memory. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 428–442. Springer, Heidelberg (2005)
14. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
15. Gurvich, V.A., Karzanov, A.V., Khachiyan, L.G.: Cyclic games and an algorithm to find minimax cycle means in directed graphs. USSR Computational Mathematics and Mathematical Physics 28, 85–91 (1988)
16. Henzinger, T.A.: From boolean to quantitative notions of correctness. In: Proc. of POPL: Principles of Programming Languages, pp. 157–158. ACM, New York (2010)
17. Kechris, A.: Classical Descriptive Set Theory. Springer, Heidelberg (1995)
18. Kopczyński, E.: Half-positional determinacy of infinite games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 336–347. Springer, Heidelberg (2006)
19. Liggett, T.A., Lippman, S.A.: Stochastic games with perfect information and time average payoff. Siam Review 11, 604–607 (1969)
20. Martin, D.A.: Borel determinacy. Annals of Mathematics 102(2), 363–371 (1975)
21. Mertens, J.F., Neyman, A.: Stochastic games. International Journal of Game Theory 10, 53–66 (1981)
22. Puri, A.: Theory of Hybrid Systems and Discrete Event Systems. PhD thesis, University of California, Berkeley (1995)
23. Puterman, M.L.: Markov Decision Processes. John Wiley and Sons, Chichester (1994)
24. Shapley, L.S.: Stochastic games. Proc. Nat. Acad. Sci. USA 39, 1095–1100 (1953)
25. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) New Trends in Formal Languages, vol. 3, ch. 7, pp. 389–455. Springer, Heidelberg (1997)
26. Velner, Y., Rabinovich, A.: Church synthesis problem for noisy input. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 275–289. Springer, Heidelberg (2011)
27. Zwick, U., Paterson, M.: The complexity of mean-payoff games on graphs. Theoretical Computer Science 158, 343–359 (1996)

Principal Types for Nominal Theories

Elliot Fairweather, Maribel Fernández, and Murdoch J. Gabbay

Abstract. We define rank 1 polymorphic types for nominal rewrite rules and equations. Typing environments type atoms, variables, and function symbols, and since we follow a Curry-style approach there is no need to fully annotate terms with types. Our system has principal types, and we give rule and axiom formats to guarantee preservation of types under both rewriting and equality reasoning. This is non-trivial because substitution does not avoid capture, so a substituted symbol can—if we are not careful—appear in inconsistent typing contexts.

Keywords: binding, polymorphism, type inference, rewriting.

1 Introduction

Suppose we want to specify λ-calculus $\beta\eta$-equivalence. We might well write this:

$$(\lambda \overset{\substack{\textit{object-variable } x\\ \downarrow}}{x}.\,(\lambda y.s))t =_\beta \lambda y.((\lambda x.s)t) \qquad \text{provided } y \overset{\substack{\textit{freshness side-condition}\\ \downarrow}}{\text{not}} \text{ free in } t$$

$$\underset{\substack{\uparrow\\ \textit{binding term-former } \lambda}}{\lambda} x.\,(rx) =_\eta \underset{\substack{\uparrow\\ \textit{meta-variable } r}}{r} \qquad \text{provided } x \text{ not free in } r$$

Equalities like this are typical, and appear in specifications of the λ-calculus (as above), substitution, logic with quantifiers, π-calculus, and more. They have common features as annotated above. To formalise and reason with this kind of systems, in this paper we follow the nominal approach:

- Nominal terms directly represent syntax with binding using term-formers, two levels of variable, and freshness side-conditions [20]. The terms and side-conditions above can be represented symbol-for-symbol in a formal syntax.
- Equality on nominal terms can be formalised in *nominal rewriting* [10] (for oriented equality) and *nominal algebra* [14] (for unoriented equality).
- Nominal terms can be given Hindley-Milner style types [9]. Given a type environment, principal types for nominal terms can be automatically deduced.

This paper combines types and equality reasoning—so we type terms *and* equations between them. This is not trivial because equality gives terms a dynamic behaviour, and type systems must ensure types are robust with respect to it.

For a large class of *closed* theories, corresponding to what can be defined in higher-order rewriting and equality reasoning, nominal rewriting is sound and complete for nominal algebra [11]. The technical contributions of this paper are notions of *typeable closed* rewrite rule and equality axiom such that types are preserved under rewriting and equality reasoning respectively. To ensure this, closedness and principal types play a key role, as our examples will demonstrate.

Related work. Nominal terms support a capturing substitution and the notation, though formal, is close to standard informal practice. For example β-reduction may be represented as $\mathsf{app}(\mathsf{lam}[a]X, Y) \to \mathsf{sub}([a]X, Y)$ where $\mathsf{sub}([a]X, Y)$ is a term which may be given the behaviour of 'non-capturing substitution' by rewrite rules [12,10].

Now consider *static* semantics, i.e. types like $\mathbb{N}$ for **numbers** and $\tau \to \tau$ for **functions**. Assigning types to terms partitions the language into 'numbers', or 'functions between numbers', and so on. This paper will not make the case for types but the interested reader can find compelling practical and theoretical motivations elsewhere, e.g. Java [17], ML [7], and System F [16].

Two approaches have been used to give static semantics for nominal terms: In e.g. [20,19,18] atoms have a special type of atoms $\mathbb{A}$. But, when we write $\mathsf{lam}[a]X$ or $\mathsf{lam}[a]a$ to represent $\lambda x.$- (a *term with a hole*, or *context*) or $\lambda x.x$, we might not expect a to be forbidden from having any type other than $\mathbb{A}$. We can use explicit casting function symbols to inject $\mathbb{A}$ into other types; but the a in $\mathsf{lam}[a]X$ still has type $\mathbb{A}$, so we cannot infer more about a until X is instantiated. This notion of typing is useful for terms without variables or in systems without polymorphism. So an alternative approach was proposed in [9] which allows atoms to inhabit any type. It follows a Curry-style approach, and has rank 1 polymorphism (*ML-style polymorphism* or *Hindley-Milner types* [7]); that is, types are assigned to terms without requiring type annotations for atoms or variables. Thus, we can write $\mathsf{lam}[a]X$, or $\mathsf{fix}[f]X$, or $\mathsf{forall}[a]X$, or $\mathsf{app}(\mathsf{lam}[a]X, \mathsf{lam}[b]Y)$, and so on, and expect the type system to make sense of this. The syntax-directed typing rules for nominal terms given in [9] guarantee that every typeable term has a *principal type* (one which subsumes all others in a suitable sense) in a given environment, and types are compatible with α-equivalence on nominal terms. Type inference is decidable and an algorithm to compute principal types has been implemented [8].

In [9], a notion of typed nominal rewriting rule is given, with a condition to ensure type preservation under rewriting. In this paper we consider also nominal equational axioms, and give sufficient conditions for type preservation under rewriting and equational deduction that are intuitive and easy to implement.

Overview of the paper. In Section 2 we recall nominal rewriting and equational reasoning, as well as the Curry-style type system for nominal terms. Section 3 gives examples to motivate the design of the type system for rules and equations. Section 4 contains the main results of the paper: it defines closed rules and the closed rewrite relation (which is sound and complete for nominal equational reasoning when the axioms are closed), and presents a notion of typeable rule and typeable axiom that guarantee that rewriting and equational reasoning preserve types. Finally, in Section 5 we draw conclusions and discuss future work.

2 Background

In this section we recall the main notions underlying the nominal approach to the specification of systems with binders. We focus on equational specifications.

Fix denumerable sets of **atoms**, **variables**, and **term-formers**. a, b, c will range over distinct atoms, $X, Y, Z, \ldots$ over distinct variables, and $\mathsf{f}, \mathsf{g}, \ldots$ over distinct term-formers. A **permutation** π is a bijection on atoms such that $\mathit{nontriv}(\pi) = \{a \mid \pi(a) \neq a\}$ is finite. **Nominal terms** r, s, t are defined by:

$$t ::= a \mid \pi \cdot X \mid \mathsf{f}\, t \mid (t, \ldots, t) \mid [a]t$$

$\pi \cdot X$ is a **(moderated) variable**; $[a]t$ is an **atom-abstraction**. We write $V(t)$ for the variables in t and $A(t)$ for the atoms in t (so $A([a]a) = \{a\}$ and $A(\pi \cdot X) = \mathit{nontriv}(\pi)$). We write $(a\ b)$ for the **swapping** permutation mapping a to b, b to a, and all other c to themselves; id for the identity permutation (so $\mathit{id}(a) = a$); $\pi \circ \pi'$ for composition (so $(\pi \circ \pi')(a) = \pi(\pi'(a))$ and π^{-1} for inverse).

A **substitution** σ is a map from variables to terms. Write $[X \mapsto r]$ for the substitution mapping X to r and all other Y to $\mathit{id} \cdot Y$.

Permutation and **substitutions** act on terms:

$$\begin{array}{lll} \pi \cdot a = \pi(a) & \pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X & \pi \cdot (\mathsf{f}\, r) = \mathsf{f}\, \pi \cdot r \\ \pi \cdot (r_1, \ldots, r_n) = (\pi \cdot r_1, \ldots, \pi \cdot r_n) & \pi \cdot [a]r = [\pi(a)]\pi \cdot r & \end{array}$$

$$\begin{array}{lll} a\sigma = a & (\pi \cdot X)\sigma = \pi \cdot (\sigma(X)) & (\mathsf{f}\, r)\sigma = \mathsf{f}(r\sigma) \\ (r_1, \ldots, r_n)\sigma = (r_1\sigma, \ldots, r_n\sigma) & ([a]r)\sigma = [a](r\sigma) & \end{array}$$

Definition 1. **α-equality** and **freshness** are defined by the following rules, where $ds(\pi, \pi') = \{a \mid \pi(a) \neq \pi'(a)\}$ is the **difference set** of π, π':

$$\frac{}{a\#b} \quad \frac{a\#s}{a\#\mathsf{f}s} \quad \frac{a\#s_i \quad (1 \leq i \leq n)}{a\#(s_1, \ldots, s_n)} \quad \frac{}{a\#[a]s} \quad \frac{a\#s}{a\#[b]s} \quad \frac{\pi^{-1}(a)\#X}{a\#\pi \cdot X} \quad \frac{s \approx_\alpha t}{\mathsf{f}\, s \approx_\alpha \mathsf{f}\, t}$$

$$\frac{}{a \approx_\alpha a} \quad \frac{s \approx_\alpha t}{[a]s \approx_\alpha [a]t} \quad \frac{s \approx_\alpha (a\ b) \cdot t \quad a\#t}{[a]s \approx_\alpha [b]t} \quad \frac{ds(\pi, \pi')\#X}{\pi \cdot X \approx_\alpha \pi' \cdot X} \quad \frac{s_i \approx_\alpha t_i \quad (1 \leq i \leq n)}{(s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)}$$

The intuition of $a\#t$ is 'a is not free in t', the intuition of $s \approx_\alpha t$ is 's and t are α-equivalent'. Both are conditional on freshness assumptions on the variables X. So **freshness contexts** Δ, ∇, ..., are finite sets of **freshness constraints** $a\#X$, and we write $\Delta \vdash a\#r$ when $a\#r$ follows from Δ; similarly for $\Delta \vdash r \approx_\alpha s$.

For example, $[a]a \approx_\alpha [b]b$ and $ds((a\ b), \mathit{id}) = \{a, b\}$ so $a\#X, b\#X \vdash (a\ b) \cdot X \approx_\alpha X$; see [20,10] for more examples.

We now recall the Curry-style type system from [9], which can typecheck nominal terms even if they contain variables (representing unknown subterms); see [8] for an implementation in Haskell.

Fix denumerable sets of **base data types** (e.g. δ, $\mathbb{N}$), **type variables** (e.g. α), and **type-formers** (e.g. C, List). **Types** τ, **type-schemes** σ, and **type-declarations** (**arities**) ρ are generated by:

$$\tau ::= \delta \mid \alpha \mid \tau_1 \times \ldots \times \tau_n \mid C\ \tau \mid [\tau']\tau \qquad \sigma ::= \forall \overline{\alpha}.\tau \qquad \rho ::= (\tau')\tau$$

$\overline{\alpha}$ denotes any finite set of type variables (if empty we omit $\forall$); call them **bound** in σ and call **free** type variables mentioned in σ not in $\overline{\alpha}$. Write $TV(\tau)$ for the set of type variables in τ, and $\equiv$ for **equality** modulo α-equivalence.[1]

We associate a type declaration to each term-former. E.g. we can have $\mathsf{succ} : (\mathbb{N})\mathbb{N}$ and $0 : ()\mathbb{N}$ (we may write just $0 : \mathbb{N}$ in this case).

Type substitutions S, T, U are finite partial functions from type variables to types. Write $[\alpha{\mapsto}\tau']$ for the partial function just mapping α to τ', and id for the **identity** substitution. Substitutions act on types, type schemes and arities as usual; application is written τS and composition SS' (apply S then S'). Write $\sigma \succcurlyeq \tau$ when $\sigma \equiv \forall\overline{\alpha}.\tau'$ and $\tau' S \equiv \tau$ for S with domain in $\overline{\alpha}$. τ may contain other type variables; only *bound* type variables in σ may be instantiated; so $\forall\alpha.(\alpha{\times}\beta) \succcurlyeq (\beta{\times}\beta)$ but $(\alpha{\times}\beta) \not\succcurlyeq (\beta{\times}\beta)$. Write $\rho \succcurlyeq \rho'$ when $\rho S \equiv \rho'$ for some S. All variables in arities are bound, but since they are *all* bound we omit $\forall$.

Definition 2. A **typing context** Γ is a partial function from atoms and variables to type schemes, with finite domain. We write $\Gamma, a{:}\,\sigma$ for Γ **updated with** $a{:}\,\sigma$, that is, $(\Gamma \setminus \{a{:}\,\sigma'\}) \cup \{a{:}\,\sigma\}$. Similarly we write $\Gamma, X : \sigma$. We write ΓS for the typing context obtained by applying S to the types in Γ. Similarly, $\pi{\cdot}\Gamma$ denotes the context obtained by applying π to the atoms in Γ. $TV(\Gamma)$ denotes the set of type variables occurring free in Γ.

A **typing judgement** is a tuple $\Gamma; \Delta \vdash s : \tau$ where Γ is a typing context, Δ a freshness context, s a term and τ a type (when Δ is empty we omit the separating ';'). Define **derivable typing judgements** by:

$$\frac{\sigma \succcurlyeq \tau}{\Gamma, a : \sigma; \Delta \vdash a : \tau} \qquad \frac{\sigma \succcurlyeq \tau \quad \Gamma; \Delta \vdash \pi{\cdot}X : \diamond}{\Gamma, X : \sigma; \Delta \vdash \pi{\cdot}X : \tau} \qquad \frac{\Gamma, a : \tau; \Delta \vdash t : \tau'}{\Gamma; \Delta \vdash [a]t : [\tau]\tau'}$$

$$\frac{\Gamma; \Delta \vdash t_i : \tau_i \quad (1 \le i \le n)}{\Gamma; \Delta \vdash (t_1, \ldots, t_n) : \tau_1 \times \ldots \times \tau_n} \qquad \frac{\Gamma; \Delta \vdash t : \tau' \quad \mathsf{f} : \rho \succcurlyeq (\tau')\tau}{\Gamma; \Delta \vdash \mathsf{f}\, t : \tau}$$

$\Gamma; \Delta \vdash \pi{\cdot}X : \diamond$ holds if, for any a such that $\pi{\cdot}a \neq a$, $\Delta \vdash a\#X$ or $a{:}\,\sigma,\ \pi{\cdot}a{:}\,\sigma \in \Gamma$ for some σ. The condition $\mathsf{f} : \rho \succcurlyeq (\tau')\tau$ is shorthand for $\mathsf{f} : \rho$ and $\rho \succcurlyeq (\tau')\tau$. If $\Gamma; \Delta \vdash s{:}\tau$ for some τ then call $\Gamma; \Delta \vdash s$ **typeable**; otherwise call it **untypeable**.

$\pi{\cdot}X$ represents an unknown term in which π permutes atoms. The second rule ensures compatibility between this and α-equivalence (see Theorem 4 below) since substitution does not avoid capture: e.g. $([a]X)[X{\mapsto}s] = [a]s$.

We refer the reader to [9] for examples and give just the typing of one term here: $([a](a, b), [a][b](a, b), a, b) : [\alpha_1](\alpha_1 \times \beta) \times [\alpha_2][\beta_2](\alpha_2 \times \beta_2) \times \alpha \times \beta)$.

Lemma 3. *If* $\Gamma; \Delta \vdash t : \tau$ *then* $\Gamma[\alpha{\mapsto}\tau']; \Delta \vdash t : \tau[\alpha{\mapsto}\tau']$.
If $\Gamma;\Delta \vdash t{:}\tau$ *and* $a,b{:}\sigma \in \Gamma$ *for some* σ, *then* $\Gamma;\Delta \vdash (a\ b){\cdot}t{:}\tau$.

Theorem 4. $\Delta \vdash s \approx_\alpha t$ *and* $\Gamma; \Delta \vdash s : \tau$ *imply* $\Gamma; \Delta \vdash t : \tau$.

Definition 5. A **typing problem** is a triple $\Gamma; \Delta \vdash s$, and its **solution** is a pair (S, τ) of a type-substitution S and a type τ such that $\Gamma S; \Delta \vdash s{:}\tau$. Write $S|_\Gamma$ for

[1] Types could be expressed in nominal syntax; our focus here is the term language.

the restriction of S to $TV(\Gamma)$. Solutions are naturally ordered by instantiation of substitutions; call a minimal element in a set of solutions **principal**.

Principal solutions for $\Gamma;\Delta \vdash s$ are unique modulo renaming type-variables. A function pt to **compute principal types**, generalising Hindley-Milner [7] to nominal terms, is given in [9, Definition 4]: $pt(\Gamma;\Delta \vdash s)$ is a principal solution (S,τ) such that $\Gamma S;\Delta \vdash s\colon \tau$, and is undefined if s is untypeable in $\Gamma;\Delta$.

Lemma 6. *1. **Type weakening**: If $\Gamma;\Delta \vdash s : \tau$ then $\Gamma,\Gamma';\Delta \vdash s : \tau$, provided that Γ' and Γ are disjoint.*
*2. **Type strengthening for atoms**: If $\Gamma, a : \tau';\Delta \vdash s : \tau$ then $\Gamma;\Delta \vdash s : \tau$ provided that $\Delta \vdash a\#s$.*
*3. **Type strengthening for variables**: If $\Gamma, X : \tau';\Delta \vdash s : \tau$ then $\Gamma;\Delta \vdash s : \tau$ provided X does not occur in s.*

We now introduce nominal algebra [14] and nominal rewriting [10]. See [11] for a detailed treatment of their connections.

An **equality judgement** is a tuple $\Delta \vdash s = t$ of a freshness context and two terms. Similarly a **rewrite judgement** is a tuple $\Delta \vdash s \to t$. An **equational theory** $\mathsf{T} = (\Sigma, Ax)$ is a pair of a signature Σ and a possibly infinite set of equality judgements Ax (**axioms**) in Σ. A **rewrite theory** $\mathsf{R} = (\Sigma, Rw)$ is a pair of Σ and a possibly infinite set of rewrite judgements Rw (**rewrite rules**) in Σ. Where Σ is clear, we identify T with Ax and R with Rw.

Definition 7. Write $\Delta \vdash (\phi_1,\ldots,\phi_n)$ for $\Delta \vdash \phi_1, \ldots, \Delta \vdash \phi_n$. A **position** C is a pair (s, X) of a term and a distinguished unknown X that occurs precisely once in s, as $id{\cdot}X$. If $C = (s, X)$ then we write $C[t]$ for $s[X{\mapsto}t]$.

Nominal rewriting: The *one-step rewrite relation* $\Delta \vdash s \overset{R}{\to} t$ is the least relation such that for every $(\nabla \vdash l \to r) \in \mathsf{R}$, freshness context Δ, position C, term s', permutation π, and substitution θ,

$$\frac{s \equiv C[s'] \qquad \Delta \vdash \big(\nabla\theta, \quad s' \approx_\alpha \pi{\cdot}(l\theta), \quad C[\pi{\cdot}(r\theta)] \approx_\alpha t\big)}{\Delta \vdash s \overset{R}{\to} t}\,(\mathbf{Rew}_{\nabla\vdash\mathbf{l}\to\mathbf{r}}). \qquad (1)$$

The *rewrite relation* $\Delta \vdash_{\mathsf{R}} s \to t$ is the reflexive transitive closure of the one-step rewrite relation. So, in particular, $\Delta \vdash s \approx_\alpha s'$ implies $\Delta \vdash_{\mathsf{R}} s \to s'$.

Nominal algebra: $\Delta \vdash_{\mathsf{T}} s = t$ is the least transitive reflexive symmetric relation such that for every $(\nabla \vdash l = r) \in \mathsf{T}$, freshness context Δ, position C, permutation π, substitution θ, and fresh Γ (so if $a\#X \in \Gamma$ then $a \notin A(\Delta, s, t)$),

$$\frac{\Delta,\Gamma \vdash \big(\nabla\theta, \quad s \approx_\alpha C[\pi{\cdot}(l\theta)], \quad C[\pi{\cdot}(r\theta)] \approx_\alpha t\big)}{\Delta \vdash_{\mathsf{T}} s = t}\,(\mathbf{Axi}_{\nabla\vdash\mathbf{l}=\mathbf{r}}). \qquad (2)$$

3 Examples

Consider a type Λ and term-formers $\mathsf{lam} : ([\Lambda]\Lambda)\Lambda$, $\mathsf{app} : (\Lambda \times \Lambda)\Lambda$ and $\mathsf{sub} : ([\Lambda]\Lambda \times \Lambda)\Lambda$, sugared to $\lambda[a]s$, $s\ t$, and $s[a{\mapsto}t]$. Then $X, Y{:}\Lambda \vdash (\lambda[a]X)Y : \Lambda$

by Definition 2. The function pt can automatically infer this type and a most general type for a, namely Λ, since $\mathsf{lam} : ([\Lambda]\Lambda)\Lambda$. Anticipating Section 4, here are rewrite rules for β-reduction (note typing *and* freshness conditions in rules); types of abstracted atoms can be inferred and are Λ:

$$X,Y{:}\Lambda \vdash (\lambda[a]X)Y \to X[a{\mapsto}Y]{:}\Lambda \qquad X,Y{:}\Lambda; a\#X \vdash X[a{\mapsto}Y] \to X{:}\Lambda$$
$$Y{:}\Lambda \vdash a[a{\mapsto}Y] \to Y{:}\Lambda \qquad X,Y{:}\Lambda; b\#Y \vdash (\lambda[b]X)[a{\mapsto}Y] \to \lambda[b](X[a{\mapsto}Y]){:}\Lambda$$
$$X,Y,Z{:}\Lambda \vdash (XY)[a{\mapsto}Z] \to (X[a{\mapsto}Z])\,Y[a{\mapsto}Z]{:}\Lambda$$

And here is an equality axiom for η: $X{:}\Lambda; a\#X \vdash \lambda[a](Xa) = X{:}\Lambda$.

Rewriting and axioms are used to compute and reason with terms—not only ground terms, but also terms with variables. Types should be compatible with the resulting dynamics on terms. Since a variable might occur several times in a term, under different abstractions, the interaction between types and rewriting or equational reasoning is subtle. *Closedness* [10] ensures the interaction between abstractions and variables is safe. Later we leverage this to conditions on closed rewrite rules and axioms to ensure also type preservation. E.g. the rules for β and η (as a rewrite or an equality) preserve types. Thus, the type Λ of $b : \Lambda \vdash \lambda[a](ba)$ is preserved in the reduced term $b : \Lambda \vdash b$.

We get more interesting types for λ-terms with a type-former $\Rightarrow$ of arity 2 and term-formers $\lambda : ([\alpha]\beta)(\alpha{\Rightarrow}\beta)$, $\circ : ((\alpha{\Rightarrow}\beta)\times\alpha)\beta$, and $\sigma : (([\alpha]\beta)\times\alpha)\beta$. As before, write $\sigma([a]s,t)$ as $s[a{\mapsto}t]$. Then the following rules preserve types:

$$X{:}\alpha, Y{:}\beta;\ a\#X \vdash X[a{\mapsto}Y] \to X : \alpha \qquad Y{:}\gamma \vdash a[a{\mapsto}Y] \to Y : \gamma$$
$$X{:}\alpha \Rightarrow \beta,\ Y{:}\alpha,\ Z{:}\gamma \vdash (XY)[a{\mapsto}Z] \to (X[a{\mapsto}Z])(Y[a{\mapsto}Z]) : \beta$$
$$X{:}\beta,\ Y{:}\gamma;\ b\#Y \vdash (\lambda[b]X)[a{\mapsto}Y] \to \lambda[b](X[a{\mapsto}Y]) : \alpha{\Rightarrow}\beta$$
$$X{:}\beta, Y{:}\alpha \vdash (\lambda[a]X)Y \to \sigma([a]X, Y) : \beta$$

Assume types $\mathbb{B}$ and $\mathbb{N}$. Then $B{:}\mathbb{B}, N{:}\mathbb{N} \vdash ((\lambda[a]a)B, (\lambda[a]a)N) : \mathbb{B}\times\mathbb{N}$ and $B{:}\mathbb{B},\ N{:}\mathbb{N} \vdash ((\lambda[a]a)B, (\lambda[a]a)N) \to (B,N) : \mathbb{B}\times\mathbb{N}$.

$\lambda[a]a$ takes different types just like $\lambda x.x$ in the Hindley-Milner type system; $pt(\vdash \lambda[a]a) = (id, \alpha \Rightarrow \alpha)$. Our system will not type $B{:}\mathbb{B}, N{:}\mathbb{N} \vdash BN$ or $\lambda[a]aa$—the previous system with a unique type Λ types the second term.

Below we show that η is compatible with this notion of type: $X{:}\alpha{\Rightarrow}\beta; a\#X \vdash \lambda[a](Xa) = X{:}\ \alpha \Rightarrow \beta$. A term obtained from a typeable term by typed equational reasoning is also typeable.

4 Types for (Closed) Nominal Rewriting and Algebra

We recall closed rewrite rules [10] and closed equational axioms [11], then give conditions to ensure nominal rewriting and equational reasoning preserve types.

4.1 Closed Rules and Closed Rewriting

Intuitively, a closed term has no unabstracted atoms and all occurrences of X (representing an unknown subterm) appear under *the same* abstracted atoms.

So $\mathsf{f}([a]X, X)$ is not closed since a is abstracted in the first X but not the second, and $a\#X \vdash \mathsf{f}([a]X, X)$ is closed, since the freshness constraint ensures that a cannot occur in X. Closedness was introduced in [12]. Definition 8 tests to check for closedness using *freshened variants* [10] (inductive definitions are also possible [6,13]).

A **freshened variant** $t^{\text{\tiny N}}$ of t is t in which atoms and unknowns have been replaced by 'fresh' atoms and unknowns (not in $A(t)$ and $V(t)$, and perhaps also fresh for some atoms and unknowns from other syntax, which we will always specify). We omit an inductive definition. Similarly, if ∇ is a freshness context then $\nabla^{\text{\tiny N}}$ will denote a freshened variant of ∇ (so if $a\#X \in \nabla$ then $a^{\text{\tiny N}}\#X^{\text{\tiny N}} \in \nabla^{\text{\tiny N}}$, where $a^{\text{\tiny N}}$ and $X^{\text{\tiny N}}$ are fresh for the atoms and unknowns of ∇). We may extend this to other syntax, like equality and rewrite judgements. If $\nabla^{\text{\tiny N}} \vdash l^{\text{\tiny N}} \to r^{\text{\tiny N}}$ is a freshened variant of $\nabla \vdash l \to r$ then $V(\nabla^{\text{\tiny N}} \vdash l^{\text{\tiny N}} \to r^{\text{\tiny N}}) \cap V(\nabla \vdash l \to r) = \varnothing$.

For example: $[a^{\text{\tiny N}}][b^{\text{\tiny N}}]X^{\text{\tiny N}}$ is a freshened variant of $[a][b]X$, $a^{\text{\tiny N}}\#X^{\text{\tiny N}}$ is a freshened variant of $a\#X$, and $\varnothing \vdash a^{\text{\tiny N}} \to b^{\text{\tiny N}}$ is a freshened variant of $\varnothing \vdash a \to b$.

Definition 8 (Closedness). Call $\nabla \vdash l$ **closed** when there exists a substitution θ with $dom(\theta) \subseteq V(\nabla^{\text{\tiny N}} \vdash l^{\text{\tiny N}})$ such that $\nabla, A(\nabla^{\text{\tiny N}}, l^{\text{\tiny N}})\#V(\nabla, l) \vdash (\nabla^{\text{\tiny N}}\theta,\ l \approx_\alpha l^{\text{\tiny N}}\theta)$.

Call $R = (\nabla \vdash l \to r)$ and $A = (\nabla \vdash l = r)$ **closed** when $\nabla \vdash (l, r)$ is closed.

Given a rewrite rule $R = (\nabla \vdash l \to r)$ and a term-in-context $\Delta \vdash s$, the (one-step) **closed rewriting** relation $\Delta \vdash s \overset{R}{\to}_c t$ holds if there are C and θ, such that, for a freshened variant $R^{\text{\tiny N}}$ of R (fresh for R, Δ, s, t):

$$s \equiv C[s'] \quad \text{and} \quad \Delta, A(R^{\text{\tiny N}})\ \#\ V(\Delta, s, t) \vdash (\nabla^{\text{\tiny N}}\theta,\ s' \approx_\alpha l^{\text{\tiny N}}\theta,\ C[r^{\text{\tiny N}}\theta] \approx_\alpha t). \qquad (3)$$

Closed rewriting $\Delta \vdash_{\mathsf{R}} s \to_c t$ is the reflexive transitive closure, as in Def. 7.

The choice of freshened variant in Definition 8 does not matter. Closed rewriting is sound and complete for nominal algebra, if all axioms are closed [11].

4.2 Typed Closed Rewriting

We define a notion typeable rule such that rewriting preserves types:

Definition 9. A **typeable closed rewrite rule** $R \equiv \Phi; \nabla \vdash l \to r : \tau$ is a tuple of a type context Φ which only types the variables in l and has no type-schemes (in particular, Φ mentions no atoms), a freshness context ∇, and terms l and r such that

1. $V(r, \nabla, \Phi) \subseteq V(l)$,
2. $pt(\Phi; \nabla \vdash l) = (id, \tau)$ and $\Phi; \nabla \vdash r : \tau$,
3. R^T is closed, where R^T is a variant of R in which each abstracted atom is decorated by appending the type of that atom to its name; a renamed atom occurring in a freshness constraint in ∇ is replaced by new freshness constraints, one for each differently decorated variant of the atom.

For example, under the signature $\mathsf{f}\colon ([N]N)N$, $\mathsf{g}\colon ([Bool]N)N$, the variant of the rule $X : N; a\#X \vdash \mathsf{f}([a]X) \to \mathsf{g}([a]X) : N$ used in the third condition in Definition 9 is $X\colon N; aN\#X, aBool\#X \vdash \mathsf{f}([aN]X) \to \mathsf{g}([aBool]X)$. The condition is satisfied: although one occurrence of X is under an abstraction for aN and the other is under an abstraction for $aBool$, we have freshness assumptions for both atoms: $aN\#X, aBool\#X$.

The type decorations for R^T can be determined mechanically by running pt to compute principal types on the left-hand side and right-hand side of R.

The first condition in the definition of typeable closed rule is standard. The second condition says that l is typeable using Φ and ∇, and r is typeable with a type *at least* as general. The third condition ensures that atom abstractions are typed in a consistent manner throughout the rule. To check this, we compute types for the abstracted atoms using the type inference algorithm, and annotate atom abstractions with types. In a Church-style system, we would use the type annotations provided by the user; thanks to the type inference algorithm, in our system users need not provide explicit type annotations for every abstraction.

The typed closed rewriting relation is defined using typed matching.

Definition 10. A **(typed) matching problem** $(\Phi; \nabla \vdash l) \; {}_?{\approx} \; (\Gamma; \Delta \vdash s)$ is a pair of tuples (Φ and Γ are typing contexts, ∇ and Δ are freshness contexts, l and s are terms) such that variables and type-variables on the left are disjoint from those in Γ, Δ, s, and Φ mentions no atoms or type-schemes.

A **solution** to this matching problem, if it exists, is the least pair (S, θ) of a type- and term-substitution (the ordering on substitutions extends to pairs component-wise) such that:

1. $X\theta \equiv X$ for $X \notin V(\Phi,\nabla,l)$, $\alpha S \equiv \alpha$ for $\alpha \notin TV(\Phi)$[2], $\Delta \vdash l\theta \approx_\alpha s$ and $\Delta \vdash \nabla\theta$.
2. $pt(\Phi; \nabla \vdash l) = (id, \tau)$, $pt(\Gamma; \Delta \vdash s) = (id, \tau S)$, and for each $X : \phi' \in \Phi$, we have $\Gamma, \Gamma'; \Delta \vdash X\theta : \phi' S$, where $\Gamma' = \Phi' S$ and Φ' contains the typings for the atoms abstracted above an occurrence of X in l and which may occur in instances of X.

That is: we want to make l on the left α-equivalent to s on the right. The first condition defines a matching solution for untyped nominal terms [20,10]. The last condition enforces type consistency: terms should have compatible types, and the solution should instantiate the variables consistent with the typing assumptions. For typeable closed rules, the typing context Φ' is uniquely defined.

The assumption that Φ mentions no atoms or type-schemes may seem strong, but is all we need: we give applications in Section 4.4.

For example, $(X{:}\alpha \vdash X) \; {}_?{\approx} \; (a{:}\mathbb{B} \vdash a)$ has solution $([\alpha \mapsto \mathbb{B}], [X \mapsto a])$, whereas $(X{:}\mathbb{B} \vdash X) \; {}_?{\approx} \; (a{:}\alpha \vdash a)$ has no solution—α on the right is too general.

To see why we must check θ in the second condition, consider $\mathsf{g}(\mathsf{f}\,\mathsf{True})$ where $\mathsf{g}\colon (\alpha)\mathbb{N}$ and $\mathsf{f}\colon (\beta)\mathbb{N}$ (i.e. both functions are polymorphic and produce a result of type $\mathbb{N}$) and a rule $X : \mathbb{N} \vdash \mathsf{g}(\mathsf{f}\ X) \to \mathsf{suc}\,X$, where $\mathsf{suc}\colon (\mathbb{N})\mathbb{N}$. Then the untyped matching problem $\mathsf{g}(\mathsf{f}\,X)) \; {}_?{\approx} \; \mathsf{g}(\mathsf{f}\,\mathsf{True})$ has solution $[X \mapsto \mathsf{True}]$, but the typed matching problem $(X : \mathbb{N} \vdash \mathsf{g}(\mathsf{f}\,X)) \; {}_?{\approx} \; (\vdash \mathsf{g}(\mathsf{f}\,\mathsf{True}))$ has none: $[X \mapsto \mathsf{True}]$ fails the last condition since X should have type $\mathbb{N}$ but is instantiated with a Boolean. If matching did not fail, this rewrite would not preserve types.

[2] So in particular, by the side-conditions on variables being disjoint between left and right of the problem, $X\theta \equiv X$ for $X \in V(\Gamma, \Delta, s)$ and $\alpha S \equiv \alpha$ for $\alpha \in TV(\Gamma)$.

Definition 11 (Typed closed rewriting). Take a typeable term $\Gamma;\Delta \vdash s{:}\mu$ and a typeable closed rule $R \equiv \Phi;\nabla \vdash l \to r : \tau$. Assume $s \equiv C[s']$ and $\Gamma';\Delta \vdash s' : \mu'$ is the typing of s' at the corresponding position in s.[3] We say s **closed rewrites with** R **to** t **in the context** $\Gamma;\Delta$ and write $\Gamma;\Delta \vdash s \overset{R}{\to}_c t$ when there is some $R^{\text{\tiny{N}}}$ a freshened variant of R (fresh for R, Γ, Δ, s, and t), where Δ' is the freshness context $A(R^{\text{\tiny{N}}}) \mathrel{\#} V(\Gamma,\Delta,s,t)$, such that:

1. The typed matching problem $(\Phi^{\text{\tiny{N}}};\nabla^{\text{\tiny{N}}}\vdash l^{\text{\tiny{N}}}) \ {}_?{\approx}\ (\Gamma';\Delta,\Delta'\vdash s')$ has solution (S,θ).
2. $\Delta,\Delta' \vdash C[r^{\text{\tiny{N}}}\theta] \approx_\alpha t$.

The conditions of typed closed rewriting extend those of closed nominal rewriting, with types. The following lemma, proved by induction, is used to prove that rewriting preserves types:

Lemma 12. *Consider $\Phi';\nabla \vdash r'$, where r' is either the right-hand side of a typeable closed rule $R \equiv \Phi;\nabla \vdash l \to r : \tau$ or a subterm of the right hand side; if r' is under atom abstractions in r, we assume $\Phi' = \Phi \cup \Phi''$ where Φ'' contains the atom typings of the removed atom abstractions. Assume $\Phi';\nabla \vdash r' : \tau'$ (so $\tau' = \tau$ if $r' = r$) and let (S,θ) be the substitutions that solve the typed matching problem $(\Phi;\nabla \vdash l) \ {}_?{\approx}\ (\Gamma;\Delta \vdash s)$. Then $\Gamma,\Phi''S;\Delta \vdash r'\theta : \tau'S$.*

Theorem 13 (Subject Reduction). *Let $R \equiv \Phi;\nabla \vdash l \to r : \tau$ be a typeable closed rule. If $\Gamma;\Delta \vdash s : \mu$ and $\Gamma;\Delta \vdash s \overset{R}{\to}_c t$ then $\Gamma;\Delta \vdash t : \mu$.*

Proof. It suffices to prove that if $pt(\Gamma;\Delta \vdash s) = (id,\nu)$ and $\Gamma;\Delta \vdash s \overset{R}{\to}_c t$ then $\Gamma;\Delta \vdash t : \nu$. Suppose $\Gamma;\Delta \vdash s \overset{R}{\to}_c t$. Then (using the notation in the definition of matching and rewriting above) we know that:

1. $R^{\text{\tiny{N}}}$ is a freshened variant of R (fresh for R, Γ, Δ, s, and t) and Δ' is the freshness context $A(R^{\text{\tiny{N}}}) \mathrel{\#} V(\Gamma,\Delta,s,t)$.
2. $s \equiv C[s']$, $\Delta,\Delta' \vdash l^{\text{\tiny{N}}}\theta \approx_\alpha s'$, and $\Delta,\Delta' \vdash a\#X\theta$ for each $a\#X$ in $\nabla^{\text{\tiny{N}}}$.
3. $\Gamma';\Delta \vdash s' : \nu'$ is the typing of s', and by the Weakening Lemma 6, also $\Gamma';\Delta,\Delta' \vdash s' : \nu'$.
4. $pt(\Phi^{\text{\tiny{N}}};\nabla^{\text{\tiny{N}}} \vdash l^{\text{\tiny{N}}}) = (id,\tau)$ and $pt(\Gamma',\Delta \vdash s') = (id,\tau S)$ so there is some S' such that $\Gamma'S' = \Gamma'$ and $\tau SS' = \nu'$.
5. $\Delta,\Delta' \vdash C[r^{\text{\tiny{N}}}\theta] \approx_\alpha t$.

By Theorem 4 (α-equivalence respects types), and 1, 2 and 3, we deduce $\Gamma';\Delta,\Delta' \vdash l^{\text{\tiny{N}}}\theta : \tau SS'$. Since $pt(\Phi^{\text{\tiny{N}}};\nabla^{\text{\tiny{N}}} \vdash l^{\text{\tiny{N}}}) = (id,\tau)$, by our assumptions on rewrite rules also $\Phi^{\text{\tiny{N}}};\nabla^{\text{\tiny{N}}} \vdash r^{\text{\tiny{N}}} : \tau$, and by Lemma 3 also $\Phi^{\text{\tiny{N}}}SS';\nabla \vdash r^{\text{\tiny{N}}} : \tau SS'$. Since θ respects the context $\Phi^{\text{\tiny{N}}}SS';\nabla^{\text{\tiny{N}}}$ by definition of typed matching, then, by the auxiliary lemma 12, $\Gamma';\Delta,\Delta' \vdash r^{\text{\tiny{N}}}\theta : \tau SS' = \nu'$. Hence $\Gamma;\Delta,\Delta' \vdash C[r^{\text{\tiny{N}}}\theta] : \nu$. By Theorem 4, also $\Gamma;\Delta,\Delta' \vdash t : \nu$, and since $A(\Delta') \notin t$ (by definition of closed rewriting), then $\Gamma;\Delta \vdash t : \nu$ as required. □

[3] $\Gamma \subseteq \Gamma'$, and $\Gamma \subset \Gamma'$ if C contains abstractions over s'.

Closure ensures that for each occurrence of X in l and r, instantiation behaves uniformly with respect to abstraction. This alone does not ensure preservation of types under reduction: consider $\mathsf{f} : ([N]N)N$, $\mathsf{g} : ([Bool]N)N$, $\mathsf{suc} : (N)N$. The rule $X : N \vdash \mathsf{f}[a]X \to \mathsf{g}[a]X : N$ is closed, and the principal type of the left-hand side is also the type of the right-hand side, but the term $\mathsf{f}[a]\mathsf{suc}(a) : N$ rewrites to $\mathsf{g}[a]\mathsf{suc}(a)$, which is untypeable. Each X in l and r must be under the same abstractions, *typed in the same way*. So condition 3 of Definition 9 annotates abstracted atoms in $l \to r$ with types.

4.3 Typed Equational Reasoning

Definition 14. A **typeable closed axiom** $A \equiv \Phi; \nabla \vdash l = r : \tau$ is a tuple of a type context Φ which only types the variables in l and r and has no type-schemes (so Φ mentions no atoms), a freshness context ∇, and terms l and r such that $pt(\Phi; \nabla \vdash l) = pt(\Phi; \nabla \vdash r) = (id, \tau)$ and A^T is closed, where A^T is a decorated version of A from Definition 11.

Typed nominal equational reasoning is defined as in Definition 7, but with the additional requirement that the substitution θ applied to l and r solves *typed* matching problems, that is, it must respect the types given in the typing context. Thus, using Lemma 12, equational deduction preserves types.

Theorem 15 (Preservation of types). *Let* T *be a nominal theory with only typeable closed axioms. If* $\Gamma; \Delta \vdash s{:}\mu$ *and* $\Gamma; \Delta \vdash_{\mathsf{T}} s{=}t$ *is in the typeable nominal equality relation generated from* T *by typed nominal reasoning, then* $\Gamma; \Delta \vdash t : \mu$.

Proof. By induction on nominal algebra equality. The interesting case is the use of the rule $(\mathbf{Axi}_{\mathbf{\Phi};\nabla\vdash\mathbf{l=r}})$, for a typeable closed axiom $A \equiv \Phi; \nabla \vdash l = r : \tau$ in T. The conditions in Definition 14 ensure that $pt(\Phi; \nabla \vdash l) = pt(\Phi; \nabla \vdash r)$. Since the substitution used to instantiate l and r to derive $\Gamma; \Delta \vdash_{\mathsf{T}} s = t$ respects $\Phi; \nabla$ (it solves typed matching problems), we can proceed as for Theorem 13. $\square$

4.4 Examples

Rewrites for **surjective pairing** cannot be implemented by compositional translation to λ-calculus [3]. We can define it; assume $\mathsf{fst} : (\alpha\times\beta)\alpha$ and $\mathsf{snd} : (\alpha\times\beta)\beta$:

$$X{:}\alpha,\ Y{:}\beta \vdash \mathsf{fst}(X, Y) \to X{:}\alpha \qquad X{:}\alpha,\ Y{:}\beta \vdash \mathsf{snd}(X, Y) \to Y{:}\beta$$
$$Z{:}\alpha\times\beta \vdash (\mathsf{fst}\,Z, \mathsf{snd}\,Z) \to Z{:}\alpha\times\beta$$

Consider a type o and term-formers $\top, \bot\colon o$, $\wedge, \Rightarrow, \Leftrightarrow\colon (o \times o)o$, $\approx\colon (\alpha\times\alpha)o$, $\forall\colon ([\alpha]o)o$, and substitution $\sigma\colon ([\alpha]o, \alpha)o$ sugared as before. Rewrite rules for equality and simplification for (a fragment of) **first-order logic** are typeable closed; we use infix notation for the binary connectives:

$$X{:}\alpha \vdash X \approx X \to \top{:}o \qquad X{:}o; a\#X \vdash \forall[a]X \to X{:}o$$
$$X, Y{:}o \vdash \forall[a](X \wedge Y) \to \forall[a]X \wedge \forall[a]Y{:}o$$

Using $P[a{\mapsto}T]$ as syntactic sugar for the term $\sigma([a]P, T)$ representing the substitution of a by T in P, we also have the following typeable closed axioms:

$$\begin{array}{l} P\colon o, T\colon \alpha \vdash\ \forall[a]P{\Rightarrow}P[a{\mapsto}T] = \top\colon o \\ P, Q\colon o;\ a\#P\ \vdash\ \forall[a](P{\Rightarrow}Q) \Leftrightarrow P{\Rightarrow}\forall[a]Q = \top\colon o \\ U, T\colon \alpha, P\colon o \vdash\ U \approx T \wedge P[a{\mapsto}T]{\Rightarrow}P[a{\mapsto}U] = \top\colon o \end{array}$$

We can extend this with arithmetic. Consider a type $\mathbb{N}$, term-formers $0 : \mathbb{N}$, $\mathsf{succ} : (\mathbb{N})\mathbb{N}$, $+ : (\mathbb{N} \times \mathbb{N})\mathbb{N}$. Then $[a](a \approx a)$ has principal type $[\alpha]o$, and $\forall[a](a \approx 0)$ is typeable (with type o) whereas $\forall[a](\top \approx 0)$ is not typeable.

We can also axiomatise **the λ-calculus** by these equalities [15], using the notation introduced in Section 3:

$$a[a{\mapsto}X] = X \quad a\#Z \vdash Z[a{\mapsto}X] = Z \qquad c\#X \vdash (\lambda[c]Z)[a{\mapsto}X] = \lambda[c](Z[a{\mapsto}X])$$
$$(Z'Z)[a{\mapsto}X] = (Z'[a{\mapsto}X])(Z[a{\mapsto}X]) \qquad Z[a{\mapsto}a] = Z$$

All are typeable closed axioms except for the last, which is not closed: it contains a free atom a on the left hand side and the atoms in an instance of Z on the left are under the scope of an abstraction for a, but they escape the abstraction on the right. The results presented here do not apply to this rule. Non-closed rules are arguably rare (and the standard higher-order rewriting formalisms do not accept them), but interesting examples exist—in this case, the rule is needed for completeness with respect to the models [15]. We conjecture that an extension with *essential typings* [9] (type assignments need not be unique but every type assignment for an occurrence of a variable X on the right must have already been made for an occurrence of X on the left) would let us deal with non-closed rules or axioms like this.

5 Conclusions and Future Work

Type inference is well-studied for the λ-calculus and Curry-style systems also exist for first-order rewriting systems [1] and algebraic λ-calculi (which combine term rewriting and λ-calculus) [2]. We know of no type assignment system for the standard higher-order rewriting formats (HRSs use a typed metalanguage, and restrict rewrite rules to base types). Our system is in Curry style; type annotations on terms are not required. We do rely on type declarations for term-formers (arities) and in future we may investigate conditions under which this assumption and the closedness assumptions could be relaxed.

The types produced resemble the Hindley-Milner polymorphic type system for the λ-calculus, but are acting on nominal terms which include variables X representing context holes as well as atoms a representing program variables.

The function *pt* is implemented [8]. The conditions on typeable closed rules and typeable closed axioms are decidable and easily mechanisable using *pt* and nominal matching, which is also implemented [5].

Theorem 4 proves our types compatible with the powerful notion of α-equivalence inherited from nominal terms [20]. Theorem 13 shows that a notion of typeable closed nominal rewrite rule exists which guarantees the preservation of types under closed nominal rewriting. Theorem 15 shows that types are also preserved under equational reasoning using a notion of typeable closed axioms.

Our type system has rank 1 polymorphism. More powerful systems, e.g. rank 2 polymorphic types or intersection types [4], should be possible. The latter have been used to characterise normalisation properties of λ-terms. Normalisation of nominal rewriting using type systems is a subject for future work, and one of our long-term goals is to come closer to applying logical semantics such as intersection types, to nominal rewriting.

Acknowledgements. Thanks to the anonymous referees.

References

1. van Bakel, S., Fernández, M.: Normalization results for typeable rewrite systems. Information and Computation 133(2), 73–116 (1997)
2. Barbanera, F., Fernández, M.: Intersection type assignment systems with higher-order algebraic rewriting. Theoretical Computer Science 170, 173–207 (1996)
3. Barendregt, H.P.: Pairing without conventional constraints. Zeitschrift für mathematischen Logik und Grundlagen der Mathematik 20, 289–306 (1974)
4. Barendregt, H.P., Coppo, M., DezaniCiancaglini, M.: A filter lambda model and the completeness of type assignment. Journal of Symbolic Logic 48(4), 931–940 (1983)
5. Calvés, C.: Complexity and implementation of nominal algorithms, Ph.D. thesis, King's College London (2010)
6. Clouston, R.: Closed terms (unpublished notes) (2007), `http://users.cecs.anu.edu.au/~rclouston/closedterms.pdf`
7. Damas, L., Milner, R.: Principal type-schemes for functional programs. In: Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1982), pp. 207–212. ACM, New York (1982)
8. Fairweather, E.: Principal types for nominal terms: tool description, `http://www.inf.kcl.ac.uk/pg/elliotf/research`
9. Fernández, M., Gabbay, M.J.: Curry-style types for nominal terms. In: Altenkirch, T., McBride, C. (eds.) TYPES 2006. LNCS, vol. 4502, pp. 125–139. Springer, Heidelberg (2007), `http://www.gabbay.org.uk/papers.html#curstn`
10. Fernández, M., Gabbay, M.J.: Nominal rewriting (journal version). Information and Computation 205(6), 917–965 (2007), `http://www.gabbay.org.uk/papers.html#nomr-jv`
11. Fernández, M., Gabbay, M.J.: Closed nominal rewriting and efficiently computable nominal algebra equality. In: Proceedings of the 5th International Workshop on Logical Frameworks and Meta-Languages (LFMTP 2010) (2010), `http://www.gabbay.org.uk/papers.html#clonre`
12. Fernández, M., Gabbay, M.J., Mackie, I.: Nominal Rewriting Systems. In: Proceedings of the 6th ACM SIGPLAN symposium on Principles and Practice of Declarative Programming (PPDP 2004), pp. 108–119. ACM Press, New York (2004), `http://www.gabbay.org.uk/papers.html#nomr`
13. Gabbay, M.J.: Nominal terms and nominal logics: from foundations to meta-mathematics. In: Handbook of Philosphical Logic, vol. 17, Kluwer, Dordrecht (2011), `http://www.gabbay.org.uk/papers.html#nomtnl`

14. Gabbay, M.J., Mathijssen, A.: Nominal universal algebra: equational logic with names and binding. Journal of Logic and Computation 19(6), 1455–1508 (2009), http://www.gabbay.org.uk/papers.html#nomuae
15. Gabbay, M.J., Mathijssen, A.: A nominal axiomatisation of the lambda-calculus. Journal of Logic and Computation 20(2), 501–531 (2010), http://www.gabbay.org.uk/papers.html#nomalc
16. Girard, J.-Y.: The system F of variable types, fifteen years later. Theoretical Computer Science 45 (1986)
17. Gosling, J., Joy, B., Steele, G.: The Java language specification. Addison-Wesley, Reading (1996)
18. Pitts, A.M.: Nominal system T. In: Proceedings of the 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2010), pp. 159–170. ACM Press, New York (2010)
19. Shinwell, M.R., Pitts, A.M., Gabbay, M.J.: FreshML: Programming with Binders Made Simple. In: Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), vol. 38, pp. 263–274. ACM Press, New York (2003), http://www.gabbay.org.uk/papers.html#frepbm
20. Urban, C., Pitts, A.M., Gabbay, M.J.: Nominal Unification. Theoretical Computer Science 323(1-3), 473–497 (2004), http://www.gabbay.org.uk/papers.html#nomu-jv

Modifying the Upper Bound on the Length of Minimal Synchronizing Word

A.N. Trahtman

Bar-Ilan University, Dep. of Math., 52900, Ramat Gan, Israel
trakht@macs.biu.ac.il

Abstract. A word w is called synchronizing (recurrent, reset, magic, directable) word of deterministic finite automaton (DFA) if w sends all states of the automaton to a unique state. In 1964 Jan Černy found a sequence of n-state complete DFA possessing a minimal synchronizing word of length $(n-1)^2$. He conjectured that it is an upper bound on the length of such words for complete DFA. Nevertheless, the best upper bound $(n^3-n)/6$ was found almost 30 years ago.

We reduce the upper bound on the length of the minimal synchronizing word to $n(7n^2+6n-16)/48$.

An implemented algorithm for finding synchronizing word with restricted upper bound is described. The work presents the distribution of all synchronizing automata of small size according to the length of an almost minimal synchronizing word.

Keywords: deterministic finite automaton, synchronizing word, Černy conjecture.

Introduction

The problem of synchronization of DFA is natural and various aspects of this problem were touched upon the literature. Synchronization makes the behavior of an automaton resistant against input errors since, after detection of an error, a synchronizing word can reset the automaton back to its original state, as if no error had occurred. Therefore different problems of synchronization draw the attention.

A problem with a long story is the estimation of the minimal length of synchronizing word. In 1964 Jan Černy found [3] n-state complete DFA with shortest synchronizing word of length $(n-1)^2$ for alphabet size $q = 2$. He conjectured that it is an upper bound on the length of the shortest synchronizing word for any n-state complete DFA. Best known now as a Černy's conjecture, it was raised independently not once.

The problem encourages a lot of investigations and generalizations [2] and together with Road Coloring problem [10], [14] was considered as a most fascinating old problem in finite automata theory.

The conjecture holds true for a lot of automata, but in general the problem still remains open in spite the fact that over hundred papers consider this problem from different points of view. Moreover, two conferences (”Workshop on Synchronizing Automata” (Turku, 2004) and ”Around the Černy conjecture” (Wroclaw,2008) were dedicated to this longstanding conjecture. The problem was discussed in ”Wikipedia” - the popular Internet Encyclopedia and on some other sites.

The problem can be reduced to automata with strongly connected graph [3]. The best known upper bound is now equal to $\frac{n^3-n}{6}$ [5], [9], [10]. This estimation was not improved almost 30 years.

We reduce the upper bound on the length of a minimal reset word. This length of n-state strongly connected automaton (and also for not necessary strongly connected) is not greater than

$$\frac{n(7n^2+6n-16)}{48}$$

The crucial estimation makes here the value $7n^3/48$. So the obtained result improves known upper bound $\frac{n^3-n}{6}$. A modification of the old bound makes here the coefficient $\frac{7}{8}$.

The search is essentially based on lemmas from [5] and [9]. The same lemmas were used in a polynomial time algorithm described below for finding synchronizing word. The algorithm is implemented in the package TESTAS [15]. The time complexity of the algorithm is $O(n^3)$ and the space complexity is quadratic. An important feature of the algorithm is that the length of the obtained synchronizing word is restricted by some given upper bound. We propose a modification of the algorithm that reduces this bound to the above-mentioned value of $n(7n^2+6n-16)/48$.

There are no examples of automata such that the length of the shortest synchronizing word is greater than $(n-1)^2$. Moreover, the examples of automata with synchronizing word of length $(n-1)^2$ are infrequent. After the sequence found by Černy and the example of Černy, Piricka and Rosenauerova [4] of 1971 for alphabet size $q=2$, a next example was found by Kari [7] only in 2001 for $n=6$ and $q=2$. Roman [11] had found an analogous example for $n=5$ and $q=3$ in 2004.

The package TESTAS has studied all automata with strongly connected transition graph of size $n \leq 10$ for $q=2$, of size $n \leq 8$ for $q \leq 3$ and of size $n \leq 7$ for $q \leq 4$ [15]. Our work presents the distribution of all considered synchronizing automata of small size according to the length of an almost minimal synchronizing word.

Five new examples of DFA with shortest synchronizing word of length $(n-1)^2$ from this class of automata were found. The size of the alphabet of these and all presently known examples is two or three.

Preliminaries

We consider a complete n-state DFA with state transition graph Γ and transition semigroup S over alphabet Σ ($|\Sigma|=q$). Let us exclude the trivial cases $n \leq 2$ and $q=1$.

The states of the automaton are considered also as vertices of the transition graph Γ and let $|\Gamma| = n$ be the number of states.

If there exists a path in an automaton from the state $\mathbf{p}$ to the state $\mathbf{q}$ and the edges of the path are consecutively labelled by $\sigma_1, ..., \sigma_k$, then for $s = \sigma_1...\sigma_k \in \Sigma^+$ let us write $\mathbf{q} = \mathbf{p}s$.

Let Ps be the set of states $\mathbf{q} = \mathbf{p}s$ for all $\mathbf{p}$ from the subset P of states and $s \in \Sigma^+$. For the transition graph Γ of an automaton let Γs denote the set Ps for the set P of all states of the automaton.

A word $v \in \Sigma^+$ is called *synchronizing word* of an automaton A with transition graph Γ if $|\Gamma v| = 1$. An automaton (and its transition graph) possessing a synchronizing word is called *synchronizing.*

A state $\mathbf{p}$ will be called *empty* state by mapping of the word s if $\mathbf{p} \in \Gamma \setminus \Gamma s$.

The direct product Γ^2 of two copies of the transition graph Γ over an alphabet Σ consists of pairs $(\mathbf{p}, \mathbf{q})$ and edges $(\mathbf{p}, \mathbf{q}) \rightarrow (\mathbf{p}\sigma, \mathbf{q}\sigma)$ labelled by σ. Here $\mathbf{p}, \mathbf{q} \in \Gamma$, $\sigma \in \Sigma$ [13].

1 A State Outside the Image

Lemma 1. *Suppose $\mathbf{p}_i \notin \Gamma s$. Then $\mathbf{p}_i \notin \Gamma us$ for any word u.*

Proof follows from $\Gamma u \subseteq \Gamma$.

Lemma 2. *Let Γ be transition graph of a DFA. If there are words s and t such that $\mathbf{p}s \notin \Gamma ts$ for some $\mathbf{p}$ from $\Gamma \setminus \Gamma t$ then Γts is a proper subset of Γs.*

Proof. One has $\Gamma s = ((\Gamma \setminus \Gamma t) \cup \Gamma t)s = (\Gamma \setminus \Gamma t)s \cup \Gamma ts$. The state $\mathbf{p}s$ from Γs is outside Γts and also outside $(\Gamma \setminus \Gamma t)s$. Now from $\Gamma ts \subseteq \Gamma s$ follows $\Gamma ts \subset \Gamma s$.

Lemma 3. *Let Γ be a transition graph of a synchronizing strongly connected n-state DFA. Then for any state $\mathbf{q}$ there exists a word t of length not greater than n such that $\mathbf{q} \notin \Gamma t$. For any $k < n$ there are at least k states $\mathbf{q}_k$ and a words u_k of length not greater than k such that $\mathbf{q}_k \notin \Gamma u_k$.*

Proof. The automaton is synchronizing, whence for some letter β, $\Gamma\beta \subset \Gamma$ and at least one state is empty by mapping β. The set $\Gamma \setminus \Gamma s$ is the set of empty states by mapping of the word s. Let R_k be a union of all $\Gamma \setminus \Gamma t$ for all words t such that $|t| \leq k$. Obviously that $R_k \subseteq R_m$ for $k \leq m$. From $\Gamma\beta \subset \Gamma$ follows that R_1 is non-empty.

For complement C_k of the set R_k we have $C_k = \cap\Gamma s$ for all words s of length not greater than k.

The graph Γ is strongly connected. Therefore for non-empty complement C_k of R_k there exists a letter γ such that $C_k\gamma \not\subseteq C_k$, whence $C_k \setminus C_k\gamma$ is not empty. Suppose $\mathbf{r} \in C_k \setminus C_k\gamma$.

$C_{k+1} = \cap\Gamma s$ for all words s of length not greater than $k+1$ and $\mathbf{r} \notin \mathbf{C_k}\gamma$. Therefore $\mathbf{r} \notin C_{k+1}$. Thus $\mathbf{r} \in R_{k+1}$, whence $R_k \subset R_{k+1}$ and $|R_k| < |R_{k+1}|$.

Consequently, for any $k \leq n$ there exists a state $\mathbf{q}$ and a word u of length not greater than k such that $\mathbf{q} \notin \Gamma u$. One has $|R_k| \geq k$, whence for given k there are at least k such states $\mathbf{q}$.

Lemma 4. *Let Γ be a transition graph of synchronizing strongly connected n-state DFA. Then for every $k \leq \frac{n+1}{2}$ there exists a word s such that $|s| \leq k^2$ and $|\Gamma s| \leq n-k$.*

Proof. If $|\Gamma s| > \frac{n+1}{2}$ for a word s then there is at least one state $\mathbf{p} \in \Gamma s$ having only one preimage $\mathbf{q}$ by mapping s. In opposite case every state $\mathbf{p} \in \Gamma s$ has at least two preimages by mapping s, whence $|\Gamma s| \leq \frac{n}{2}$.

Let us consider for a word s_i the states from Γs_i having only single preimage by mapping s_i and let Q_i be the set of such single preimages.

Our aim is now to find a short word s such that $|\Gamma s| \leq \frac{n+1}{2}$. We construct a sequence of mappings s_i that reduce the size of the set Q_i and the size of Γs_i on every step i.

By Lemma 3 for every state $\mathbf{q}$ there exists a word t_q such that $\mathbf{q} \notin \Gamma t_q$ and for $k \leq n$ there are at least k states $\mathbf{q}$ with t_q of length not greater than k.

There exists a letter α such that $|\Gamma\alpha| < |\Gamma|$. Let α be the word $s_1 = t_1$. Let Q_1 be the set of single preimages of Γt_1. Then $|\Gamma t_1| < n$ and $|Q_1| \leq n-2$. If $n - |\Gamma\alpha| = m$ then $m < n - |Q_1| \leq 2m$.

On every next step, let us take the state $\mathbf{q}$ from Q_{i-1} with t_q of minimal length. Suppose $t_{i-1} = t_q$ and $s_i = t_{i-1}s_{i-1}$. By Lemma 2, $\Gamma s_i \subset \Gamma s_{i-1}$ and so $|\Gamma s_i| < |\Gamma s_{i-1}|$. Also $|Q_i| < |Q_{i-1}|$, at least one state leaves Q_{i-1} on the step.

If $|\Gamma s_{i-1}| - |\Gamma s_i| = j$ then at most $2j$ states leave Q_{i-1}. One has $2j \geq |Q_{i-1}| - |Q_i| \geq j$. In the worst case, such states $\mathbf{q}$ leaving Q_{i-1} have minimal t_q among the states of Q_{i-1}. So $\min(|t_q|) - |t_{i-1}| \leq 2j$ for $\mathbf{q}$ in Q_i. For $j = 1$ we have $\min(|t_q|) - |t_{i-1}| \leq 2$. The first $t_q = t_1$ is a letter (a word of length one). Consequently, for $n - |\Gamma s_i| = m$ there exists $\mathbf{q} \in Q_{i-1}$ such that $|t_q| \leq 2m-1$.

The process continues until the set Q_i is not empty (in particular, if $|\Gamma s_i| > n/2$). Thus the length of s_k is restricted by the sum of k (or less) odd integers for every $k \leq \frac{n+1}{2}$.

Consequently, for some word s and $k \leq \frac{n+1}{2}$ the length of s in view of Lemma 3 is restricted by the sum $\sum_{i=1}^{k}(2i-1) = k^2$ and $|\Gamma s| \leq n-k$.

2 Pairs of States

The next our step is based on the following result of Frankl and Klyachko et al.

Theorem 1. *[5], [9] Let N be set of size n with subset D of size $i > 1$. Then there exists a word s of length at most*

$$C^2_{n-i+2} = (n-i+2) * (n-i+1)/2$$

such that $|Ds| < |D|$.

The next lemma follows the ideas from [9].

Lemma 5. *Let Γ be a transition graph of a strongly connected n-state automaton and let D_k of size k be a subset of states of the automaton.*

Then the word of length at most $C^3_{n+1} - C^3_{n-k+2}$ synchronizes the set D_k.

Proof. By Theorem 1, every D_i has a pair of states with a minimal synchronizing word of length not greater than $C^2_{n-i+2} = (n-i+2)(n-i+1)/2$. So D_k has synchronizing word of length at most $S = \sum_{i=2}^{k} C^2_{n-i+2}$.

Suppose $j = n - i + 2$. Then $n \geq j \geq n - k + 2$. Now $S = \sum_{j=n-k+2}^{n} C^2_j = \sum_{j=2}^{n} C^2_j - \sum_{j=2}^{n-k+1} C^2_j$.

For every $m > 2$, $\sum_{j=2}^{m} C^2_j = C^3_{m+1}$. So $\sum_{j=2}^{n} C^2_j = C^3_{n+1}$ and $\sum_{j=2}^{n-k+1} C^2_j = C^3_{n-k+2}$. Therefore $S = C^3_{n+1} - C^3_{n-k+2}$.

Theorem 2. *Let Γ be a transition graph of a strongly connected n-state automaton. Then a word of length not greater than*

$$\frac{n(7n^2+6n-16)}{48}$$

synchronizes the automaton.

Proof. Let us combine the quadratic estimation from Lemma 4 and the cubic estimation of Lemma 5. The length of some synchronizing word is not greater than the sum of $S_1 = C^3_{n+1} - C^3_{n-k+2}$ (Lemma 5) and $S_2 = k^2$ (Lemma 4) for $k \leq \frac{n+1}{2}$.

We must consider $k \leq \frac{n+1}{2}$. Hence the maximum of $S = S_1 + S_2$ exists for even n and $k = \frac{n}{2}$ (the case of odd n and $k = \frac{n+1}{2}$ also will be calculated for clarity). In the case of even n

$S_1 = C^3_{n+1} - C^3_{n-k+2} = C^3_{n+1} - C^3_{n/2+2} = \frac{n^3-n}{6} - \frac{(n/2+1)^3-n/2-1}{6} = \frac{8n^3-8n-n^3-6n^2-12n-8+4n+8}{48} = \frac{n(7n^2-6n-16)}{48}$.

$S_2 = k^2 = \frac{n^2}{4}$.

So the length of a minimal synchronizing word has in the case of even n the following upper bound $S_1 + S_2 = \frac{n(7n^2-6n-16)}{48} + \frac{n^2}{4} = \frac{n(7n^2+6n-16)}{48}$.

In the case of odd n

$S_1 = C^3_{n+1} - C^3_{n-k+2} = C^3_{n+1} - C^3_{(n+1)/2+2} = \frac{n^3-n}{6} - \frac{((n+1)/2+1)^3-(n+1)/2-1}{6} = \frac{8n^3-8n-n^3-9n^2-27n-27+4n+12}{48} = \frac{7n^3-9n^2-31n-15}{48}$.

$S_2 = k^2 = \frac{n^2+2n+1}{4}$.

So the length of a minimal synchronizing word has in the case of odd n the following upper bound $S_1 + S_2 = \frac{7n^3-9n^2-31n-15}{48} + \frac{n^2+2n+1}{4} = \frac{7n^3+3n^2-7n-3}{48}$. This value is less than $\frac{n(7n^2+6n-16)}{48}$ for $n > 2$.

Thus the value $\frac{n(7n^2+6n-16)}{48}$ is an upper bound on the length of the minimal synchronizing word. The obtained result improves the old upper bound $\frac{n(n^2-1)}{6}$ by factor $\frac{7}{8}$.

Remark 1. *For odd n a word of length not greater than $\frac{7n^3+3n^2-7n-3}{48}$ synchronizes the automaton.*

2.1 An Algorithm for Finding Synchronizing Word of Restricted Length

The algorithm presents another useful application of the combinatorial ideas from [9]. The Theorem 1 gives us an estimation of the length of the reset word.

Let us consider the inverse of the graph Γ^2. So the incoming edges of every pair $(\mathbf{p},\mathbf{q})$ from Γ^2 together with its ancestors are known. Then let us enumerate the pairs of vertices. There are vertices $\mathbf{p},\mathbf{q}$ from Γ such that for some letter α $\mathbf{p}\alpha = \mathbf{q}\alpha$. For every such pair $(\mathbf{p},\mathbf{q})$ from Γ^2 suppose $n(\mathbf{p},\mathbf{q}) = 1$. Let us connect with the pair $(\mathbf{p},\mathbf{q})$ the letter α.

Then for every enumerated pair $(\mathbf{p},\mathbf{q})$ from Γ^2 with $n(\mathbf{p},\mathbf{q}) = k$ let us consider all its ancestors without enumeration. These pairs obtain the number $k+1$ and are connected with the letter on the edge of the graph Γ^2 from this pair to the pair $(\mathbf{p},\mathbf{q})$.

We find a sequence of mappings of the graph of the automaton induced by the letters on the labels. Let us consider the graph Γs for some word s and find a pair $(\mathbf{p},\mathbf{q})$ from Γs with a minimal number. The letter α of the pair is the first letter of the word w we build. The next letter of the word w is the letter of the pair $\mathbf{p}\alpha,\mathbf{q}\alpha$. The number of this pair is less than the number of $(\mathbf{p},\mathbf{q})$. We proceed on this way until the number of the pair exists. The last pair is synchronizing by a letter. The obtained word w synchronizes the vertices $\mathbf{p}$ and $\mathbf{q}$. The length of the word w is at most $\frac{(n-|\Gamma s|+2)(n-|\Gamma s|+1)}{2}$ (theorem 1).

The search of the first letter of the word w needs $O(|\Gamma|(|\Gamma|-1)/2)$ steps. Then the building of the word w needs $|\Gamma|$ steps. The number of the words w is less than n. Therefore the time complexity of considered procedure can be estimated by $O(|\Gamma|^3)$ in the worst case. The space complexity of the algorithm is $O(|\Gamma|^2)$ because of the size of Γ^2. The algorithm is correct in view of the Lemma 5 and is implemented in the package TESTAS.

2.2 A Modification of the Algorithm

The modification is based on Lemmas 3 and 4. There exists a letter α and a state $\mathbf{p}$ such that $\mathbf{p} \notin \Gamma\alpha$. Let R_1 be the set of such states $\mathbf{p}$ (as in Lemma 3). We associate the word $u_1 = \alpha$ of length one with every state.

Suppose the set R_k of states and its non-empty complement C_k with corresponding words exist. From the proof of Lemma 3 follows that for some letter β there exists a state $\mathbf{q}$ in $C_k \setminus C_k\beta$. For every state $\mathbf{r}$ from C_k with corresponding word u we associate the word $u_{k+1} = u\beta$ and add $\mathbf{q}$ to R_{k+1}.

Let us keep with every state $\mathbf{p}s \in \Gamma s$ its preimage by mapping s and fix the case of more than one preimage. If $\mathbf{p}s_i$ has only one preimage by mapping s_i then suppose $\mathbf{p} \in Q_i$.

All states of the graph belong to Q_0, after the mapping v_1 we have $|Q_1| \leq n-2$. The set Q_i lost states on every step. If $|\Gamma s| > |\Gamma|/2)$ then there exists a state in Γs having only one preimage and so Q_i is not empty by mapping s. We proceed until Q_i is not empty.

Let us choose a state $\mathbf{p} \in Q_i$ with word u_k of minimal length and suppose $v_{i+1} = u_k v_i$. The length of the word v_i is restricted according to Lemma 3. We continue until Γv_i has states with only one preimage (Q_i is not empty). So we obtain the set of states Γv_i of size less than $(n+1)/2$ (Lemma 4) and then proceed by the main algorithm.

The upper bound on the length of the synchronizing word in virtue of Theorem 2 is $\frac{n(7n^2+6n-16)}{48}$.

3 Distribution of the Length of Synchronizing Word of Small Automata

A program based on the synchronization algorithms of the package TESTAS was used for a search of automata with a minimal reset word of relatively great length. The program has investigated all complete DFA for $n \leq 10$ over an alphabet of size 2, $n \leq 8$ over an alphabet of size 3 and for $n \leq 7$ over an alphabet of size 4 [14].

Maximal value of the length of a synchronizing word for $n = 10$ found by the algorithm on the set of considered automata of size n is 93. The length found by the minimal length algorithm is 81 ($Err < 0.13$). So the shift of the size of the synchronizing word is relatively small.

The program consistently sifts non-synchronizing automata, the automata with a very short reset word and a part of isomorphic automata. The following table presents the distribution of all remaining automata of size 10 over an alphabet of two letters (see also [1]).

interval of size of the automata	n - 2n	2n - 3n	3n - 4n	4n - 5n	5n - 6n	6n -7n
percent of automata in interval	81.01	16.2	1.82	0.8	0.05	0.006

The distribution for three and four letters does not differ noticeable and is omitted.

The synchronizing words of minimal length are found only for automata having great minimal reset words. The maximal number of considered n-state automata has its length of the reset word near n+1.

Thus one can conclude that the polynomial synchronizing algorithms of the package find synchronizing words of a length not far of the minimal, especially for automata with very great reset words. The presented distribution does not differ essentially from the distribution of the lengths of the minimal synchronizing words.

Acknowledgments. I would like to express my gratitude to the referees for many helpful and useful remarks and for improving the calculation of the result.

References

1. Ananichev, D., Gusev, V., Volkov, M.: Slowly Synchronizing Automata and Digraphs. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 55–65. Springer, Heidelberg (2010)
2. Béal, M.P., Czeizler, E., Kari, J., Perrin, D.: Unambiguous automata. Math. Comput. Sci. 1, 625–638 (2008)
3. Černy, J.: Poznamka k homogenym eksperimentom s konechnymi automatami. Math.-Fyz. Čas 14, 208–215 (1964)

4. Černy, J., Piricka, A., Rosenauerova, B.: On directable automata. Kybernetika 7, 289–298 (1971)
5. Frankl, P.: An extremal problem for two families of sets. Eur. J. Comb. 3, 125–127 (1982)
6. Friedman, J.: On the road coloring problem. Proc. of the Amer. Math. Soc. 110, 1133–1135 (1990)
7. Kari, J.: A counter example to a conjecture concerning synchronizing word in finite automata. EATCS Bulletin 73, 146–147 (2001)
8. Kari, J.: Synchronizing Finite Automata on Eulerian Digraphs. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 432–438. Springer, Heidelberg (2001)
9. Kljachko, A.A., Rystsov, I.K., Spivak, M.A.: An extremely combinatorial problem connected with the bound on the length of a recurrent word in an automata. Kybernetika, 216–225 (1987)
10. Pin, J.-E.: On two combinatorial problems arising from automata theory. Annals of Discrete Math 17, 535–548 (1983)
11. Roman, A.: Experiments on Synchronizing Automata. Schedae Informaticae 19, 35–51 (2010)
12. Steinberg, B.: The Averaging Trick and the Čern'y Conjecture. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 423–431. Springer, Heidelberg (2010)
13. Trahtman, A.N.: The Cerny Conjecture for Aperiodic Automata. Discr. Math. and Theoret. Comput. Sci. 9(2), 3–10 (2007)
14. Trahtman, A.N.: The Road Coloring and Černy Conjecture. In: Proc. of Prague Stringology Conference, pp. 1–12 (2008)
15. Trahtman, A.N.: Notable trends concerning the synchronization of graphs and automata. CTW06, El. Notes in Discrete Math. 25, 173–175 (2006)

Online Maximum k-Coverage*

Giorgio Ausiello[1], Nicolas Boria[2], Aristotelis Giannakos[2], Giorgio Lucarelli[2], and Vangelis Th. Paschos[2,3]

[1] Dip. di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza"
ausiello@dis.uniroma1.it

[2] LAMSADE, CNRS UMR 7243 and Université Paris-Dauphine
{boria,giannako,lucarelli,paschos}@lamsade.dauphine.fr

[3] Institut Universitaire de France

Abstract. We study an online model for the maximum k-vertex-coverage problem, where given a graph $G = (V, E)$ and an integer k, we ask for a subset $A \subseteq V$, such that $|A| = k$ and the number of edges covered by A is maximized. In our model, at each step i, a new vertex v_i is revealed, and we have to decide whether we will keep it or discard it. At any time of the process, only k vertices can be kept in memory; if at some point the current solution already contains k vertices, any inclusion of a new vertex in the solution must entail the definite deletion of another vertex of the current solution (a vertex not kept when revealed is definitely deleted). We propose algorithms for several natural classes of graphs (mainly regular and bipartite), improving on an easy $\frac{1}{2}$-competitive ratio. We next settle a set-version of the problem, called maximum k-(set)-coverage problem. For this problem we present an algorithm that improves upon former results for the same model for small and moderate values of k.

1 Introduction

In the *maximum k-vertex-coverage* (MkVC) problem we are given a graph $G = (V, E)$ ($|V| = n$, $|E| = m$) and an integer k, and we ask for a subset $A \subseteq V$, such that $|A| = k$ and the number of edges covered by A is maximized. The MkVC problem is NP-hard, since otherwise the optimal solution for the vertex cover problem could be found in polynomial time: for each k, $1 \leq k \leq n$, run the algorithm for the MkVC problem and stop when all elements are covered.

In this paper we consider the following online model for this problem: at each step i, a new vertex v_i with its adjacent edges is revealed, and we have to decide whether we will include v_i in the solution or discard it. At any time of the process, only k vertices can be kept in memory, so if at some point the current solution already contains k vertices, any inclusion of any new vertex in the solution must be compensated with the definite deletion of one vertex of the current solution. Of course, a vertex that is not kept when it is revealed is also definitely deleted. To our knowledge, no online model for the MkVC problem has been studied until now.

* Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

A generalization of the MkVC problem is the *maximum k-(set)-coverage* (denoted by MkC) problem, where given a universe of elements $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$, a collection of subsets of $\mathcal{E}$, $S = \{S_1, S_2, \ldots, S_n\}$, and an integer $k \leq n$, we ask for a subcollection $A = \{A_1, A_2, \ldots, A_{|A|}\} \subseteq S$, such that $|A| = k$ and the number of elements of E covered by A is maximized. The online model for the MkC problem is the same as for the MkVC.

Clearly, the MkVC problem is a special case of the MkC problem where: (i) each element belongs to exactly two sets and (ii) the intersection of any two sets of S has size at most one, since multiple edges are not permitted.

The weighted generalization of the MkC problem, denoted by WEIGHTED MkC, has been also studied in the literature. In this problem, each element $e_i \in \mathcal{E}$ has a non-negative weight $w(e_i)$, and the goal is to maximize the total weight of the elements covered by k sets.

The analogous online model for WEIGHTED MkC problem, where at each step i a set $S_i \in S$ together with its elements is revealed and only k such sets can be kept in memory, has been studied in [1], where an algorithm of competitive ratio $\frac{1}{4}$ is given. The authors in their so called *set-streaming model* assume that the universe of the instance is known *a priori*. Nevertheless, they do not use this information in the proposed algorithm.

In the classic offline setting, the MkC problem is known to be non approximable within a factor $1 - \frac{1}{e}$ [2]. On the other hand, even for the weighted version of the problem, an approximation algorithm of ratio $1 - \left(1 - \frac{1}{k}\right)^k$ is known [3]. This ratio tends to $1 - \frac{1}{e}$ as k increases, closing in this way the approximability question for the problem.

In [4] the inverse problem (i.e., the hitting set version of MkC), also called *maximum coverage problem*, has been studied: given a universe of elements $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$, a collection of subsets of $\mathcal{E}$, $S = \{S_1, S_2, \ldots, S_n\}$, a non-negative weight $w(S_i)$ for each $S_i \in S$, and an integer k, a set $B \subseteq \mathcal{E}$ is sought, such that $|B| = k$ and the total weight of the sets in S that intersect with B is maximized. It is easy to see that this version is equivalent to the WEIGHTED MkC modulo the interchange of the roles between set-system and universe of elements. An algorithm of approximation ratio $1 - \left(1 - \frac{1}{p}\right)^p$ is presented in [4] for this problem, where p is the cardinality of the largest set in S. In the case where each set has cardinality equal to two then this problem coincides with the MkVC problem; hence a $\frac{3}{4}$ approximation ratio is implied by the algorithm in [4]. Several improvements for some restricted cases of the MkVC problem are presented in [5,6]. The MkVC problem is NP-complete in bipartite graphs (by a reduction from the *densest k-subgraph* problem [7]). Moreover, it is polynomial in trees by an application of the dynamic programming algorithm for the quadratic $0 - 1$ knapsack problem presented in [8] using appropriate weights.

In this paper we study the online model described above for both the MkVC and the MkC problems. In Section 2, we prove several negative results on the competitiveness of any algorithm for the model handled for both problems. In Section 3, we present algorithms for regular graphs, regular bipartite graphs, trees and chains, achieving non-trivial competitive ratios, improving upon an easy $\frac{1}{2}$ competitiveness result holding for any graph. Finally, in Section 4 the k-(set)-coverage problem is handled. For this problem, we present an algorithm that improves upon former results for the same model for small and moderate values of k.

The following notations will be used in the sequel. They are based upon the definition of the MkVC problem and are easily extendable to the MkC problem.

For any $A \subseteq V$, we denote by $E(A)$ the set of edges covered by A and by $m(A) = |E(A)|$ the number of these edges. Let $SOL = m(A)$ be the number of edges covered by our algorithms. Moreover, we denote by $A^* \subseteq V$ an optimal subset of vertices and by $OPT = m(A^*)$ the number of edges covered by an optimal solution. The maximum degree (or the degree when it is regular) of the input graph $G = (V, E)$ is denoted by Δ. Dealing with MkC, Δ denotes the cardinality of a set of maximum size, that is, $\Delta = \max\{|S_i| : 1 \leq i \leq n\}$. For a subset $A \subseteq V$ and a vertex $v_i \in A$, we call *public* the edges incident to v_i and to another vertex in A and *private* the edges of v_i that are covered just by v_i in A. Finally, as it is common in the online setting, the quality of an algorithm is measured by means of the so-called *competitive ratio* representing the ratio of the value of the solution computed by the algorithm over the optimal value of the whole instance, i.e., the value of an optimal (offline) solution of the final instance.

Detailed proofs of the results are given in [9].

2 Negative Results

In this section we give negative results for the online maximum k-vertex-coverage problem and their corresponding adaptations for the maximum k-coverage problem. We start with a negative result for the case where we don't allow any "swaps", i.e., where the replacement of a vertex or set that belongs to the current solution by the newly revealed vertex or set is not permitted.

Proposition 1. *Any deterministic online algorithm that does not allow swaps cannot achieve a competitive ratio better than* $O\left(\frac{1}{(n-1)^{1/(k+1)}}\right)$, *for the* M$k$VC *problem, and better than* $O\left(\frac{1}{m^{1/(k+1)}}\right)$, *for the* M$k$C *problem.*

The next negative result for the MkVC problem fits the model addressed in the paper (swaps are allowed).

Proposition 2. *Any deterministic online algorithm cannot achieve a competitive ratio better than* $\frac{2k}{3k-2} \simeq \frac{2}{3}$ *for the* MkVC *problem.*

Proof. Assume that $2k-1$ vertices, $v_1^1, v_2^1, \ldots, v_{2k-1}^1$, of degree one and $2k-1$ vertices, $v_1^2, v_2^2, \ldots, v_{2k-1}^2$, of degree two are released, such that $(v_i^1, v_i^2) \in E$, $1 \leq i \leq 2k-1$, and that the algorithm selects $k' \leq k$ of them. Wlog, let $v_1^2, v_2^2, \ldots, v_{k'}^2$ be the vertices selected by the algorithm. Next the vertex v_3 of degree k' is released, where $(v_i^2, v_3) \in E$, $1 \leq i \leq k'$. The solution of the algorithm at this time is $2k'$, while the inclusion or not of v_3 does not play any role for this value. Finally, $2k-1-k'$ vertices, $v^3_{k'+1}, v^3_{k'+2}, \ldots, v^3_{2k-1}$, of degree one are released, such that $(v^2_i, v^3_i) \in E$, $k'+1 \leq i \leq 2k-1$. In this last phase, the algorithm can increase its solution by at most $k-k'$ more edges. Hence, the final solution of the algorithm is at most $k+k'$. The optimum solution consists of the vertices $v^2_{k+1}, v^2_{k+2}, \ldots, v^2_{2k-1}, v_3$, and hence is of cardinality $2(k-1)+k'$. In all, $\frac{SOL}{OPT} = \frac{k+k'}{2(k-1)+k'} \leq \frac{2k}{3k-2}$. $\square$

An analogous result can be proved for the MkC problem. Recall that for the offline version of the MkC problem an $1-\frac{1}{e} \simeq 0.63$-inapproximability result is known [2].

Proposition 3. *Any deterministic online algorithm cannot achieve a competitive ratio better than* $\frac{k+2\sqrt{k}+1}{2k+2\sqrt{k}+1} \simeq \frac{1}{2}$ *for the* MkC *problem even in the case where all sets have the same cardinality.*

3 Maximum k-Vertex-Coverage

In this section we deal with the online maximum k-vertex-coverage problem. Note, first, that there exists an easy $\frac{1}{2}$-competitive ratio for this problem. In fact, consider selecting k vertices of largest degrees. In an optimum solution all the edges are, at best, covered once, while in the solution created by this greedy algorithm, all the edges are, at worst, covered twice. Since the algorithm selects the largest degrees of the graph, the $\frac{1}{2}$-competitive ratio is immediately concluded.

Proposition 4. *There is a* $\frac{1}{2}$*-competitive ratio for the online* MkVC *problem.*

In the rest of this section we improve the $\frac{1}{2}$-competitive ratio for several classes of graphs. But first, we give an easy upper bound for the number of elements covered by any solution that will be used later. Its proof is straightforward.

Proposition 5. $OPT \leq k\Delta$.

3.1 Regular Graphs

The following preliminary result that will be used later holds for any algorithm for the MkVC problem in regular graphs.

Proposition 6. *Any deterministic online algorithm achieves a* $\frac{k}{n}$*-competitive ratio for the* MkVC *problem on regular graphs.*

Let us note that the result of Proposition 6 for the MkVC problem also holds for general graphs in the offline setting [6].

We now present an algorithm for the MkVC problem in regular graphs. Our algorithm depends on a parameter x which indicates the improvement on the current solution that a new vertex should entail, in order to be selected for inclusion in the solution. In other words, we replace a vertex of the current solution by the released one, only if the solution increases by at least $\lceil \frac{\Delta}{x} \rceil$ edges.

As we will see in what follows, the best value for x is $x = \frac{n+2k+\sqrt{4k^2+n^2}}{2n}$, leading to the following theorem.

```
Algorithm mkvc-R($x$)
1: $A = \emptyset$; $B = \emptyset$;
2: for each released vertex $v$ do
3:    if $|A| < k$ then
4:       $A = A \cup \{v\}$;
5:       if $|E(\{v\}) \setminus E(B)| \geq \lceil \frac{\Delta}{x} \rceil$ then
6:          $B = B \cup \{v\}$;
7:    else if $|B| < k$ and $|E(\{v\}) \setminus E(B)| \geq \lceil \frac{\Delta}{x} \rceil$ then
8:       Select a vertex $u \in A \setminus B$;
9:       $A = A \cup \{v\} \setminus \{u\}$; $B = B \cup \{v\}$;
10: return $A$;
```

Theorem 1. Algorithm mkvc-R *achieves* 0.55*-competitive ratio.*

Proof (Sketch). Note that $B \subseteq A$ consists of the vertices that improve the solution by at least $\lceil \frac{\Delta}{x} \rceil$; b denotes the number of these vertices, i.e., $b = |B|$. We denote by y_1 the number of edges with one endpoint in B and the other in $V \setminus B$, and by y_2 the number of edges with both endpoints in B. By definition:

$$SOL \geq y_1 + y_2 = b\Delta - y_2 = \frac{b\Delta - y_1}{2} + y_1 = \frac{b\Delta + y_1}{2} \tag{1}$$

We shall handle two cases, depending on the value of b with respect to k.

If $b < k$ then each vertex $v \in V \setminus B$ is not selected by Algorithm mkvc-R(x) to be in B because it is adjacent to at most $\lceil \frac{\Delta}{x} \rceil - 1$ vertices of $V \setminus B$. Thus, there are at least $\Delta - \lceil \frac{\Delta}{x} \rceil + 1$ edges that connect v with vertices in B. Summing up for all the vertices in $V \setminus B$, it holds that $y_1 \geq (n-b)\left(\Delta - \lceil \frac{\Delta}{x} \rceil + 1\right)$, and considering also (1) we get:

$$SOL \geq (n-b)\left(\Delta - \left\lceil \frac{\Delta}{x} \right\rceil + 1\right) + y_2 \tag{2}$$

$$SOL \geq \frac{b\Delta + (n-b)\left(\Delta - \lceil \frac{\Delta}{x} \rceil + 1\right)}{2} \tag{3}$$

Using the upper bound for the optimum provided by Proposition 5 and expressions (2) and (3), respectively, we get the following ratios:

$$\frac{SOL}{OPT} \geq \frac{(n-b)\left(\Delta - \lceil \frac{\Delta}{x} \rceil + 1\right) + y_2}{k\Delta} \geq \frac{n(x-1) - b(x-1)}{kx} \tag{4}$$

$$\frac{SOL}{OPT} \geq \frac{\frac{b\Delta + (n-b)\left(\Delta - \lceil \frac{\Delta}{x} \rceil + 1\right)}{2}}{k\Delta} \geq \frac{n(x-1) + b}{2kx} \tag{5}$$

Observe that the righthand side of (4) decreases with b while that of (5) increases; thus, the worst case occurs when righthand sides of them are equal, that is $\frac{n(x-1)-b(x-1)}{kx} = \frac{n(x-1)+b}{2kx} \Leftrightarrow b = \frac{n(x-1)}{2x-1}$ and hence:

$$\frac{SOL}{OPT} \geq \frac{n(x-1) + \frac{n(x-1)}{2x-1}}{2kx} = \frac{n(x-1)}{k(2x-1)} \tag{6}$$

If $b = k$, then trivially it holds that:

$$\frac{SOL}{OPT} \geq \frac{k\left\lceil\frac{\Delta}{x}\right\rceil}{k\Delta} \geq \frac{1}{x} \tag{7}$$

Note that (6) increases with x while (7) decreases; therefore, for the worst case we have $\frac{n(x-1)}{k(2x-1)} = \frac{1}{x} \Leftrightarrow x = \frac{n+2k+\sqrt{4k^2+n^2}}{2n}$. In all, it holds that:

$$\frac{SOL}{OPT} \geq \frac{2n}{n+2k+\sqrt{4k^2+n^2}} \tag{8}$$

If $k < 0.55n$, the ratio of (8) leads to $\frac{SOL}{OPT} \geq \frac{2n}{n+2(0.55n)+\sqrt{4(0.55n)^2+n^2}} = 0.55$. On the other hand, the ratio provided in Proposition 6 that holds for any algorithm, for $k > 0.55n$, gives $\frac{SOL}{OPT} \geq \frac{k}{n} \geq \frac{0.55n}{n} = 0.55$. □

Let us note that, as it can be easily derived from (8), *when $k = o(n)$ the competitive ratio of* ALGORITHM MkVC-R *is asymptotical to 1.*

3.2 Regular Bipartite Graphs

A better ratio can be achieved if we further restrict in regular bipartite graphs. A key-point of such an improvement is that the maximum independent set can be found in polynomial time in bipartite graphs (see for example [10]). In what follows in this section, we consider that the number of vertices, n, is known a priori.

Our ALGORITHM MkVC-B initializes its solution with the first k released vertices. At this point, a maximum independent set B, of size $b \leq k$, in the graph induced by these k vertices is found. The vertices of this independent set will surely appear in the final solution. For the remaining $k-b$ vertices we check if they cover at least $\frac{\frac{n\Delta}{2}-b\Delta}{\left\lceil\frac{n-b}{k-b}\right\rceil}$ edges different from those covered by the independent set B; if yes, we return the solution consisting of the b vertices of the independent set and these $k-b$ vertices. Otherwise, we wait for the next $k-b$ vertices and we repeat the test. In ALGORITHM MkVC-B, $G[A]$ denotes the subgraph of G induced by the vertex-subset A.

Theorem 2. ALGORITHM MkVC-B *achieves a* 0.6075*-competitive ratio.*

Proof (Sketch). Let us call *batch* the set of the $k-b$ vertices of $A \backslash B$ in Lines 5–10 of ALGORITHM MkVC-B.

The solution computed by this algorithm contains a maximum independent set of size b. Since the input graph is bipartite, it holds that $b \geq \frac{k}{2}$.

The number of edges of the graph uncovered by the vertices of the maximum independent set is in total $\frac{n\Delta}{2} - b\Delta$. Any of these edges is covered by vertices belonging to at least one of the $\left\lceil\frac{n-b}{k-b}\right\rceil$ batches. Hence, in average, each batch covers $\frac{\frac{n\Delta}{2}-b\Delta}{\left\lceil\frac{n-b}{k-b}\right\rceil}$ of those edges; so there exists a batch that covers at least $\frac{\frac{n\Delta}{2}-b\Delta}{\left\lceil\frac{n-b}{k-b}\right\rceil}$ of

```
ALGORITHM M$k$VC-B
1: $A = \{$the first $k$ released vertices$\}$;
2: Find a maximum independent set $B \subseteq A$ in $G[A]$; $b = |B|$;
3: for each released vertex $v$ do
4:   if $|A| = k$ then
5:     if $m(A) \geq b\Delta + \frac{\frac{n\Delta}{2} - b\Delta}{\lceil \frac{n-b}{k-b} \rceil}$ then
6:       return $A$;
7:     else
8:       $A = B$;
9:   else
10:    $A = A \cup \{v\}$
11: return $A$;
```

them. Therefore, the algorithm covers in total at least $b\Delta + \frac{\frac{n\Delta}{2} - b\Delta}{\lceil \frac{n-b}{k-b} \rceil}$ edges. Using Proposition 5, we get $\frac{SOL}{OPT} \geq \frac{b\Delta + \frac{\frac{n\Delta}{2} - b\Delta}{\lceil \frac{n-b}{k-b} \rceil}}{k\Delta} = \frac{b + \frac{\frac{n}{2} - b}{\lceil \frac{n-b}{k-b} \rceil}}{k}$ and since this quantity increases with b it holds that:

$$\frac{SOL}{OPT} \geq \frac{\frac{k}{2} + \frac{\frac{n}{2} - \frac{k}{2}}{\lceil \frac{n - \frac{k}{2}}{k - \frac{k}{2}} \rceil}}{k} = \frac{k + \frac{n-k}{\lceil \frac{2n-k}{k} \rceil}}{2k} \tag{9}$$

If $k \leq 0.6075n$, then (9) leads to $\frac{SOL}{OPT} \geq 0.6075$. Otherwise, using Proposition 6 we get the same ratio and the theorem is concluded. □

Note that by (9), ALGORITHM MkVC-B *achieves a competitive ratio asymptotical to* $\frac{3}{4}$ *when* $k = o(n)$.

3.3 Trees and Chains

In this section we give algorithms that further improve the competitive ratios for the MkVC problem in trees and chains. Dealing with trees the following result holds.

Proposition 7. *The* MkVC *problem can be solved within* $\left(1 - \frac{k-1}{\Delta^*}\right)$*-competitive ratio in trees, where* Δ^* *is the sum of the* k *largest degrees in the tree. The ratio is tight.*

Note that, if the number of vertices of degree greater than 1 is $r < k$ then our algorithm finds an optimum solution using just r vertices, since the edges that are adjacent to the leaves are covered by their other endpoints.

Furthermore, in the case where all the internal vertices of the tree have the same degree Δ, the ratio provided by Proposition 7 becomes $\left(1 - \frac{k-1}{k\Delta}\right)$. This ratio is better than the ratio proved for regular bipartite graphs in Theorem 2 for any $\Delta \geq 3$, but it is worse for $\Delta = 2$, i.e., in the case where the input graph is a chain.

An improvement for the MkVC problem in chains follows. The main idea of the algorithm is to partition the solution, A, into two disjoint parts, whose size is dynamically adjusted: the set B of vertices that contribute two edges in $E(A)$ and the set C of vertices that contribute one edge in $E(A)$.

```
ALGORITHM M$k$VC-C
 1: $A = \emptyset$; $B = \emptyset$; $C = \emptyset$; In any step $A \equiv B \cup C$;
 2: for each released vertex $v$ do
 3:    if $|B| \leq k$ and $v$ adds two new edges to the solution then
 4:       if $|A| = k$ then
 5:          Delete an arbitrary vertex from $C$;
 6:       $B = B \cup \{v\}$;
 7:    else if $|A| < k$ and $v$ adds one new edge to the solution then
 8:       $C = C \cup \{v\}$;
 9:       if the inclusion of $v$ in $A$ has as a result three consecutive vertices to appear
          in $A$ then
10:          Move $v$ from $C$ to $B$; Remove the middle vertex from $A$;
11: return $A$;
```

Proposition 8. *For the* MkVC *problem in chains,* ALGORITHM MkVC-C *returns the (offline) optimum, if* $k < \lceil \frac{n}{3} \rceil$ *or* $k \geq \lceil \frac{2n}{3} \rceil$, *and achieves a* 0.75*-competitive ratio, if* $\lceil \frac{n}{3} \rceil \leq k < \lceil \frac{2n}{3} \rceil$.

4 Maximum k-(set)-Coverage

In this section we present ALGORITHM MkC for the online maximum k-(set)-coverage problem. It initializes by selecting the first k released sets. Then, each time a new set P is released, the algorithm will update the current solution A_j only if for some suitably selected set Q from A_j, the solution obtained after having in A_j the set Q replaced by P covers at least $m(A_j)\left(\frac{k+1}{k}\right)$ elements. We prove that ALGORITHM MkC achieves competitive ratio strictly greater than $\frac{1}{4}$, tending to $\frac{1}{4}$ as k increases. Recall that the algorithm presented in [1] achieves also an $\frac{1}{4}$-competitive ratio. However, our analysis is tight and gives better results for moderately large values of k.

To analyze ALGORITHM MkC, let A_z be the solution computed after having all the sets released, i.e., $SOL = m(A_z)$. Fix also, an optimum solution A^*.

```
ALGORITHM M$k$C
 1: $j = 1$; $A_j =$ {the first $k$ released sets};
 2: for each released set $P$ do
 3:    Find the set $Q \in A_j$ that covers privately the smallest number of elements in
       $A_j$;
 4:    if $m(A_j \setminus \{Q\} \cup \{P\}) > m(A_j) + \frac{m(A_j)}{k}$ then
 5:       $j = j + 1$; $A_j = A_{j-1} \setminus \{Q\} \cup \{P\}$;
```

We distinguish the following two types of *bad events* that may happen during the execution of the algorithm upon arrival of a set P:

(a) $P \in A^*$ and ALGORITHM MkC does not select it, and
(b) $P \notin A^*$ and ALGORITHM MkC discards $Q \in A^*$ in order to insert P into its current solution.

The total number of bad events of both types is a measure of the distance between A_z and A^*; clearly, at most k such events may happen. Notice also that more than one bad event of type (a) may happen while the current solution is kept unchanged, i.e. may correspond to some value of j in the algorithm, while at most one bad event of type (b) might correspond to it; so, let $\ell = |\{j :$ some bad event of any type happens when the current solution is $A_j\}|$. Let A_{j_i}, $1 \leq i \leq \ell$, $1 \leq j_i \leq z$, be the i-th of these current solutions, and k_i, $1 \leq i \leq \ell$, be the number of events occurred with A_{j_i} being the current solution.

We will now provide an upper bound to $OPT = m(A^*)$ by some expression involving these "event-stroken" A_{j_i}s, $1 \leq i \leq \ell$. Consider that the s-th bad event corresponds to j_i, i.e., $\sum_{r=1}^{i-1} k_r < s \leq \sum_{r=1}^{i} k_r$. Let P_s be the new set that arrives while the current solution is A_{j_i} and Q_s be the set that covers privately the smallest number of elements in A_{j_i}. Let, also, $\widetilde{Q}_s \subseteq Q_s$ be the set of private elements of Q_s in A_{j_i}.

If the event is of type (a) then $P_s \in A^*$ is not selected and it covers a subset of the elements in $E(A_{j_i} \setminus \{Q_s\})$ plus its private elements, $\widetilde{P}_s \subseteq P_s$, in $E(A_{j_i} \setminus \{Q_s\} \cup \{P_s\})$. Note that it is $m(\widetilde{P}_s) \leq m(\widetilde{Q}_s) + \frac{m(A_{j_i})}{k}$, otherwise P_s would be selected by the algorithm. Moreover, $m(\widetilde{Q}_s) \leq \frac{m(A_{j_i})}{k}$, since Q_s has the smallest private part in A_{j_i}, and hence $m(\widetilde{P}_s) \leq \frac{2m(A_{j_i})}{k}$.

In all we get the following inclusion relation $E(A^*) \subseteq \bigcup_{i=1}^{\ell} E(A_{j_i}) \cup \bigcup_s \widetilde{P}_s$ with s varying on the indices of (a)-type bad events (remark that the sets of the optimum removed after a (b)-type bad event are always represented in the first term of the expression, since the union varies among all i for A_{j_i}s). This reduces trivially to $E(A^*) \subseteq E(A_{j_\ell}) \cup \bigcup_{i=2}^{\ell} \left[E(A_{j_{i-1}}) \setminus E(A_{j_i})\right] \cup \bigcup_s \widetilde{P}_s$. Thus, for the value of the optimum solution A^* we have the following bound:

$$OPT \leq m(A_{j_\ell}) + \sum_{i=2}^{\ell} m(E(A_{j_{i-1}}) \setminus E(A_{j_i})) + \sum_{i=1}^{\ell} \left(k_i \frac{2m(A_{j_i})}{k} \right)$$

Claim 1. *$m(E(A_{j_{i-1}}) \setminus E(A_{j_i})) \leq \frac{|A_{j_{i-1}} \setminus A_{j_i}|}{k} m(A_{j_{i-1}})$, $2 \leq i \leq \ell$.*

Using Claim 1 and since $m(A_{j_\ell}) > m(A_{j_i})$, $1 \leq i \leq \ell - 1$, and $\sum_{i=1}^{\ell} k_i = k$ we get:

$$OPT \leq m(A_{j_\ell}) + \sum_{i=2}^{\ell} \frac{|A_{j_{i-1}} \setminus A_{j_i}|}{k} m(A_{j_{i-1}}) + \sum_{i=1}^{\ell-1} \frac{2m(A_{j_i})}{k} + (k - \ell + 1) \frac{2m(A_{j_\ell})}{k}$$

By definition, it holds that $j_\ell \leq z$ and hence $m(A_{j_\ell}) \leq m(A_z) = SOL$. Moreover, by ALGORITHM MkC, $m(A_{j_\ell}) \geq \left(1+\frac{1}{k}\right)^{j_\ell - j_i} m(A_{j_i})$. Thus, we have:

$$\frac{SOL}{OPT} \geq \frac{1}{3 + \frac{1}{k}\sum_{i=2}^{\ell} \frac{j_i - j_{i-1}+2}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} - \frac{2(\ell-1)}{k}} \tag{10}$$

Claim 2. *For any $\ell \geq 2$, it holds that $\sum_{i=2}^{\ell} \frac{j_i - j_{i-1}+2}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} \leq \frac{g(\ell)}{\ln\left(1+\frac{1}{k}\right)}$, where $g(\ell) = \frac{\left(1+\frac{1}{k}\right)^2}{e} \cdot e^{g(\ell-1)}$ and $g(2) = \frac{\left(1+\frac{1}{k}\right)^2}{e}$.*

Using Claim 2 and (10), we get $\frac{SOL}{OPT} \geq \frac{1}{3+\frac{1}{k}\left[\frac{g(\ell)}{\ln\left(1+\frac{1}{k}\right)} - 2(\ell-1)\right]}$, where $g(\ell) = \frac{\left(1+\frac{1}{k}\right)^2}{e} \cdot e^{g(\ell-1)}$ and $g(2) = \frac{\left(1+\frac{1}{k}\right)^2}{e}$. This quantity is minimized for some $\ell = o(k)$. The ratio r achieved by ALGORITHM MkC for different values of k is shown in Table 1.

Table 1. Approximation ratio of ALGORITHM MkC

k	2	3	5	10	30	50	100	300	500	1000
r	0.333	0.324	0.314	0.300	0.282	0.275	0.268	0.261	0.258	0.256

To see that the ratio achieved by ALGORITHM MkC is always greater than $\frac{1}{4}$, consider the following expression for the ratio, slightly coarser than (10):

$$\frac{SOL}{OPT} \geq \frac{1}{3 + \frac{1}{k}\sum_{i=2}^{\ell} \frac{j_i - j_{i-1}}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} + \frac{1}{k}\sum_{i=2}^{\ell}\left(\frac{2}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} - 2\right)}$$

Note first that if $\ell = 1$ then both sums in the denominator become zero and hence we have a $\frac{1}{3}$-competitive ratio. For $\ell \geq 2$ we may proceed to the following analysis. For the first sum, by a similar argument as in Claim 2 we can prove that $\sum_{i=2}^{\ell} \frac{j_i - j_{i-1}}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} \leq \frac{g(\ell)}{\ln\left(1+\frac{1}{k}\right)}$, where $g(\ell) = \frac{1}{e} \cdot e^{g(\ell-1)}$ and $g(2) = \frac{1}{e}$. It is easy to see by simple induction that $g(\ell) \leq 1$ for any $\ell \geq 2$ and hence $\sum_{i=2}^{\ell} \frac{j_i - j_{i-1}}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} \leq \frac{1}{\ln\left(1+\frac{1}{k}\right)} \leq k$. For the second sum, we have:

$$\sum_{i=2}^{\ell}\left(\frac{2}{\left(1+\frac{1}{k}\right)^{j_\ell - j_{i-1}}} - 2\right) \leq \sum_{i=2}^{2}\left(\frac{2}{\left(1+\frac{1}{k}\right)} - 2\right) = \frac{2k}{k+1} - 2 = -\frac{2}{k+1}$$

Therefore, using these bounds to the ratio we get $\frac{SOL}{OPT} \geq \frac{1}{4} + \frac{1}{4}\frac{1}{2k(k+1)-1}$.

It is hopefully clear from the previous discussion, that the analysis of ALGORITHM MkC works as well for the WEIGHTED MkC problem, up to the assumption that $m(\cdot)$ in ALGORITHM MkC denotes the total weight of the elements rather than their number.

We conclude this section by providing a tight example for the ratio achieved by Algorithm mkc. The idea of the example strongly relies upon the proof given above, which indicates the "critical" values of ℓ and j_i, $1 \le i \le \ell$. For simplicity, we will consider a case where $k = 3$, but it is easy to extend our example for any k, by appropriately choosing values for ℓ and j_i.

For $k = 3$, one can see that the ratio of Algorithm mkc is minimized when $\ell = 2$ and $j_2 - j_1 = 1$. Hence, consider the scenario shown in Figure 1. Let

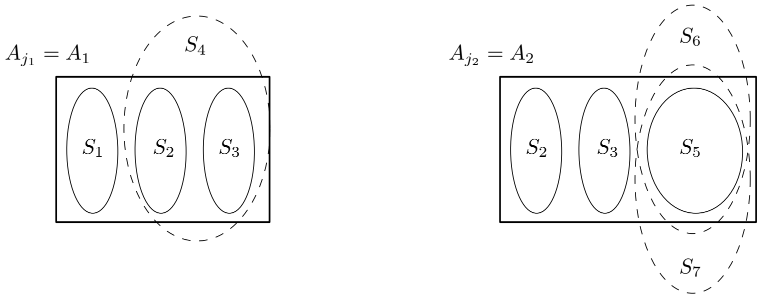


Fig. 1. A tight example for Algorithm mkc when $k = 3$

$A_1 = \{S_1, S_2, S_3\}$ be the solution after the first three sets have been released. These sets are disjoint and each one covers c elements. Next, the set S_4 appears, which covers $2c - \epsilon$ new elements plus all elements in S_2 and S_3. The algorithm does not select S_4, since it may choose S_1 as a candidate for swapping; in this case the new solution would cover $m(\{S_2, S_3, S_4\}) = 4c - \epsilon$ elements which is smaller than $m(A_1) + \frac{m(A_1)}{k} = 4c$. Then, the set S_5 is released, which is disjoint to the previous sets and covers $2c$ elements. Thus, the algorithm replaces S_1 by S_5, and the new solution is $A_2 = \{S_2, S_3, S_5\}$. Finally, S_6 and S_7 are released, each one covering the elements in S_5 plus $\frac{8c}{3} - \epsilon$ new elements. Algorithm mkc does not select any of them, since they do not satisfy the algorithm's criterion.

So, the final solution, A_2, covers $m(S_2) + m(S_3) + m(S_5) = 4c$ elements. The optimal solution consists of sets S_4, S_6 and S_7 and covers $OPT = (4c - \epsilon) + (2c + \frac{8c}{3} - \epsilon) + (\frac{8c}{3} - \epsilon) = \frac{34c}{3} - \epsilon$ elements. Therefore, the ratio achieved by Algorithm mkc is $\frac{SOL}{OPT} = \frac{4c}{\frac{34c}{3} - \epsilon} \simeq \frac{12}{34} = 0.353$.

Note that the gap between this ratio and the ratio 0.324 given in Table 1 is due to the fact that the elements of S_1 do not appear to the optimal solution. Indeed, if S_1 was included to the optimal, then $OPT = \frac{37c}{3} - \epsilon$ and $\frac{SOL}{OPT} = \frac{4c}{\frac{37c}{3} - \epsilon} \simeq \frac{12}{37} = 0.324$. This gap decreases as $k \to \infty$.

5 Conclusions

There exist several interesting questions arising from the results presented in this paper. The first of them is to improve the easy $\frac{1}{2}$-competitive ratio for m$k$vc in general graphs and the (less easy) worst-case $\frac{1}{4}$-competitive ratio in set systems. Another open question is to provide tighter upper bounds for the on-line model handled in regular graphs. We still do not see how one can improve the analysis of ALGORITHM MkC in the case of equal cardinalities, or how to tighten the upper bound of Proposition 3 in Section 2, in order to match (or to get closer to) the competitive ratio of ALGORITHM MkC. Let us note that an algorithm in the spirit of ALGORITHM MkVC-R of Section 3.1 for the case of equal-cardinality sets, only achieves ratio $\frac{1}{\sqrt{k}}$.

References

1. Saha, B., Getoor, L.: On maximum coverage in the streaming model & application to multi-topic blog-watch. In: DM 2009, pp. 697–708. SIAM, Philadelphia (2009)
2. Feige, U.: A threshold of $\ln n$ for approximating set cover. Journal of the ACM 45, 634–652 (1998)
3. Hochbaum, D.S., Pathria, A.: Analysis of the greedy approach in problems of maximum k-coverage. Naval Research Logistics 45, 615–627 (1998)
4. Ageev, A.A., Sviridenko, M.I.: Approximation algorithms for maximum coverage and max cut with given sizes of parts. In: Cornuéjols, G., Burkard, R.E., Woeginger, G.J. (eds.) IPCO 1999. LNCS, vol. 1610, pp. 17–30. Springer, Heidelberg (1999)
5. Feige, U., Langberg, M.: Approximation algorithms for maximization problems arising in graph partitioning. Journal of Algorithms 41, 174–211 (2001)
6. Han, Q., Ye, Y., Zhang, H., Zhang, J.: On approximation of max-vertex-cover. European Journal of Operational Research 143, 342–355 (2002)
7. Corneil, D.G., Perl, Y.: Clustering and domination in perfect graphs. Discrete Applied Mathematics 9, 27–39 (1984)
8. Rader Jr., D.J., Woeginger, G.J.: The quadratic 0-1 knapsack problem with series-parallel support. Operations Research Letters 30, 159–166 (2002)
9. Ausiello, G., Boria, N., Giannakos, A., Lucarelli, G., Paschos, V.T.: Online maximum k-coverage. Technical Report 299, Université Paris-Dauphine, Cahiers du LAMSADE (2010)
10. Paschos, V.T.: A survey of approximately optimal solutions to some covering and packing problems. ACM Computing Surveys 29, 171–209 (1997)

Coloring Graphs without Short Cycles and Long Induced Paths*

Petr A. Golovach, Daniël Paulusma, and Jian Song

School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, England
{petr.golovach,daniel.paulusma,jian.song}@durham.ac.uk

Abstract. The girth of a graph G is the length of a shortest cycle in G. For any fixed girth $g \geq 4$ we determine a lower bound $\ell(g)$ such that every graph with girth at least g and with no induced path on $\ell(g)$ vertices is 3-colorable. In contrast, we show the existence of an integer ℓ such that testing for 4-colorability is NP-complete for graphs with girth 4 and with no induced path on ℓ vertices.

1 Introduction

Graph coloring involves the labeling of the vertices of some given graph by k integers called colors such that no two adjacent vertices receive the same color. Due to the fact that the corresponding decision problem k-Coloring is NP-complete for any fixed $k \geq 3$, there has been considerable interest in studying its complexity when restricted to certain graph classes, see e.g. the surveys of Randerath and Schiermeyer [19] and Tuza [22]. We focus on graph classes defined by forbidden induced subgraphs. Before we summarize the known results and explain our new results, we first state the necessary terminology and notations.

1.1 Terminology

We only consider finite undirected graphs with no loops and no multiple edges. We refer to the textbook by Bondy and Murty [1] for any undefined graph terminology. Let $G = (V, E)$ be a graph. We write $G[U]$ to denote the subgraph of G *induced by* the vertices in U, i.e., the subgraph of G with vertex set U and an edge between two vertices $u, v \in U$ whenever $uv \in E$. The *length* of a path or cycle is the number of its edges. The graphs C_n and P_n denote the cycle and path on n vertices, respectively. The disjoint union of two graphs G and H is denoted $G + H$, and the disjoint union of r copies of G is denoted rG. A *linear forest* is the disjoint union of a collection of paths. Let G be a graph and $\{H_1, \ldots, H_p\}$ be a set of graphs. We say that G is $(H_1, \ldots, H_p)$*-free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$; if $p = 1$, we sometimes write H_1-free instead of (H_1)-free. If G is C_3-free, then we also say that G is *triangle-free*. The *girth* $g(G)$ of G is the length of a shortest cycle in

* This work has been supported by EPSRC (EP/G043434/1).

G. Note that G has girth at least p for some integer $p \geq 4$ if and only if G is $(C_3, \ldots, C_{p-1})$-free.

A *k-coloring* of a graph $G = (V, E)$ is a mapping $\phi : V \rightarrow \{1, \ldots, k\}$ such that $\phi(u) \neq \phi(v)$ whenever $uv \in E$. If G has a k-coloring, then G is called *k-colorable*. The *chromatic number* $\chi(G)$ of G is the smallest k such that G is k-colorable. If $\chi(G) = k$, then G is also called *k-chromatic*. Recall that the problem k-Coloring is to decide whether a graph admits an k-coloring. Here, k is *fixed*, i.e., not part of the input. The problem Vertex Coloring is to determine the chromatic number of a graph. The problem k-Precoloring Extension is to decide whether a given mapping $\phi_W : W \rightarrow \{1, \ldots, k\}$ defined on a (possibly empty) subset $W \subseteq V$ of a graph $G = (V, E)$ can be extended to an k-coloring of G.

1.2 Related Work

Král' et al. [15] completely determined the computational complexity of Vertex Coloring for graph classes characterized by one forbidden induced subgraph H. They showed that Vertex Coloring can be solved in polynomial time for H-free graphs if H is a (not necessarily proper) induced subgraph of P_4 or of $P_1 + P_3$, and that this problem stays NP-hard if H is any other graph. The computational complexity of Vertex Coloring for $\mathcal{H}$-free graphs where $\mathcal{H}$ is a family of two graphs is still open, although several partial results are known. In particular (C_3, H)-free graphs, i.e., H-free graphs with girth at least 4, are well studied. Král' et al. [15] showed that for any graph H that contains at least one cycle, Vertex Coloring is NP-complete for (C_3, H)-free graphs. Maffray and Preissmann [17] showed that Vertex Coloring is NP-complete for $(C_3, K_{1,5})$-free graphs, where $K_{1,5}$ is the 6-vertex star. Broersma et al. [6] showed that Vertex Coloring is polynomial-time solvable for $(C_3, 2P_3)$-free graphs, hereby completing a study of Dabrowski et al. [8] who considered the Vertex Coloring problem restricted to (C_3, H)-free graphs for graphs H with $|V_H| \leq 6$.

The computational complexity classification of k-Coloring for H-free graphs where k is a fixed integer and H is a fixed graph is still open as well, but the following is known. Kamiński and Lozin [14] showed that 3-Coloring is NP-complete for the class of graphs of girth at least p for any fixed $p \geq 3$, and Holyer [13] showed that 3-Coloring is NP-complete for claw-free graphs (graphs with no induced $K_{1,3}$). The first result implies that 3-Coloring is NP-complete for the class of H-free graphs if H contains a cycle. The second result implies that 3-Coloring is NP-complete for the class of H-free graphs if H is a forest that contains a vertex with degree at least 3. Hence, only the case in which H is a linear forest remains. It is known that 4-Coloring is NP-complete for P_8-free graphs [5] and that 6-Coloring is NP-complete for P_7-free graphs [4]. In contrast to these hardness results, Couturier et al. [7] generalized a result for P_5-free graphs of Hoàng et al. [12] by proving that for any fixed integers k and r, the k-Coloring problem can be solved in polynomial time for $(P_5 + rP_1)$-free graphs, whereas Randerath and Schiermeyer [18] showed that 3-Coloring can be solved in polynomial time for P_6-free graphs. Broersma et al. [5] extended the latter result by showing that 3-Coloring is polynomial-time solvable for H-free graphs if H is a linear forest with $|V_H| \leq 6$ or $H = rP_3$ for any integer r.

Also the k-Coloring problem has been studied for $\mathcal{H}$-free graphs where $\mathcal{H}$ is a family of two graphs. We refer to Randerath and Schiermeyer [19] for a detailed survey on so-called good Vizing-pairs (A, B) that satisfy the condition that every (A, B)-free graph is 3-colorable, in particular when $A = C_3$. Brandt [2] showed that every (C_3, sK_2)-free graph is $(2s-2)$-colorable for any $s \geq 3$.

1.3 Our New Results

We consider the relation between the girth of a graph and the length of a forbidden induced path for the k-Coloring problem. In Section 2 we determine, for any fixed girth $g \geq 4$, a lower bound $k(g)$ such that every $P_{k(g)}$-free graph with girth at least g is 3-colorable. This extends the result of Sumner [21] who showed that every P_5-free graph of girth at least 4 is 3-colorable in another direction than Randerath and Schiermeyer [19] who show that for all $\ell \geq 4$, every P_ℓ-free graph of girth at least 4 is $(\ell - 2)$-colorable. Our results lead to Table 1. The proofs of them are constructive, i.e., yield polynomial-time 3-coloring algorithms. As an aside, graphs with girth $g = \infty$ are forests and hence 2-colorable.g In Section 3 we show that 4-Coloring is NP-complete for (C_3, P_{164})-free graphs. The gadgets of the proofs of the aforementioned NP-completeness results for k-Coloring for P_ℓ-free graphs are not triangle-free, i.e., have girth equal to 3, and our aim was to show the existence of an integer $\ell = 164$ rather than minimizing ℓ. Our result complements the result of Kratochvíl [16] who showed that 5-Precoloring Extension is NP-complete for P_{13}-free bipartite graphs.

Table 1. 3-colorable P_ℓ-free graphs of given girth

girth	forbidden induced path
4	P_5-free [21]
5	P_7-free
6	P_{10}-free
7	P_{12}-free
$g \geq 8$	P_ℓ-free for $\ell = 2g + \lceil \frac{g-2}{4} \rceil - 3$

1.4 Future Work

A classical result of Erdös [9] tells us that for every pair of integers k and g, there exists a k-chromatic graph of girth g. However, it is not trivial to construct, for given k and g, a k-chromatic P_ℓ-free graph of girth g for ℓ as small as possible, or, for given k and ℓ, a k-chromatic P_ℓ-free graph of girth g for g as large as possible. For example, the Grötzsch graph [11] is 4-chromatic, P_6-free and of girth 4. Hence, the bound of Sumner [21] is tight. Brinkmann and Meringer [3] constructed a 4-chromatic P_{10}-free graph with girth 5. Hence, the bound in Table 1 for P_{10}-free graphs is tight with respect to the girth. We are not aware of examples of 4-chromatic graphs of girth at least 6 without long induced paths and expect that some of our bounds in Table 1 can be improved. Also, by fixing the girth and making the length of the forbidden induced path longer than the lengths in Table 1 we could obtain polynomial-time 3-coloring algorithms. Other open questions are if there exists an integer ℓ such that 3-Coloring is NP-complete for P_ℓ-free graphs, and if 3-Coloring is polynomial-time solvable for (C_3, P_7)-free graphs.

2 The Lower Bounds for 3-Colorability

We start with some additional terminology. We say that a path between two vertices u and v in G is a (u, v)*-path*. The *distance* between u and v is the length of a shortest (u, v)-path in G and is denoted $\text{dist}(u, v)$. For a vertex v and subset $U \subseteq V$ we define $\text{dist}(v, U) = \min\{\text{dist}(v, u) \mid u \in U\}$; note that $\text{dist}(v, U) = 0$ if and only if $v \in U$. We denote the *neighborhood* of a vertex u by $N(u) = \{v \mid uv \in E\}$ and its *degree* by $\deg(u) = |N(u)|$. For a subset $U \subseteq V$ and integer s, we define $N^s(U) = \{v \in V \mid \text{dist}(v, U) = s\}$ and $N^s[U] = \{v \in V \mid \text{dist}(u, v) \leq s\}$.

We make two assumptions that are valid throughout this section. First of all, we may assume that the graphs we consider are connected. Second, we may assume that they have minimum degree at least three; this follows from the observation below.

Observation. *Let G be a graph and u be a vertex of degree at most 2. Then G is 3-colorable if and only if $G - u$ is 3-colorable.*

We show the four new bounds in Table 1 in Theorems 1 – 4. Due to page restrictions we only sketch their proofs.

Theorem 1. *Every P_7-free graph of girth 5 is 3-colorable.*

Proof. Let $G = (V, E)$ be a connected P_7-free graph with minimum degree at least 3 such that $g(G) = 5$. Consider $uv \in E$ and let $U = \{u, v\}$. We first prove some useful properties of the sets $N^s(U)$:

1. $N^1(U)$ is an independent set;
2. $N^2(U)$ induces a bipartite graph;
3. $N^3(U)$ is an independent set;
4. $N^s(U) = \emptyset$ for $s \geq 4$.

Using these four properties we construct a 3-coloring of G as follows. We color the vertices u and v by the colors 1 and 2 respectively, and all the vertices of the independent set $N^1(U)$ by 3. The vertices of the bipartite graph $G[N^2(U)]$ are colored by 1 and 2. Finally, the vertices of the independent set $N^3(U)$ are colored by 3. □

Theorem 2. *Every P_{10}-free graph of girth 6 is 3-colorable.*

Proof. Let $G = (V, E)$ be a connected P_{10}-free graph with minimum degree at least 3 and girth 6. Let also $U = \{x_1, \ldots, x_6\}$ be the vertex set of a C_6 in G (vertices are enumerated in the cycle order). Denote by X_i the set of vertices of $N^1(U)$ adjacent to x_i for $i = 1, \ldots, 6$. Using the (C_3, C_4, C_5)-freeness of G, we observe the following:

1. each set X_i is independent;
2. $X_i \cap X_j = \emptyset$ for $i, j \in \{1, \ldots, 6\}$, $i \neq j$;
3. if $y_i y_j \in E$ for $y_i \in X_i$, $y_j \in X_j$ and $1 \leq i < j \leq 6$, then $j - i = 3$.

Denote by $H_1, \ldots, H_k$ components of $G[V \setminus N^1[U]]$. We need the following claim.

Claim 1. *Each graph H_j is either an isolated vertex or a star $K_{1,r}$ for some $r \geq 1$.*

We construct a 3-coloring of G. Using Properties 1–3, we color vertices x_1, x_3, x_5 and all the vertices of X_2, X_4, X_6 by the color 1, and x_2, x_4, x_6 and X_1, X_3, X_5 are colored by 2. Now we color each H_j. If H_j is an isolated vertex then this vertex is colored by the color 3. Assume that H_j is a star $K_{1,r}$ with the central vertex w and the leaves $z_1, \ldots, z_r$. If $w \notin N^2[U]$ then $z_1, \ldots, z_r$ are colored by the color 3 and w is colored by 1. Let w be adjacent to a vertex of X_i for some $i \in \{1, \ldots, 6\}$. In this case w is colored by the color 3. It remains to prove that each leaf z_s can be colored either by 1 or 2. Assume that it is not so for some z_s. Then this vertex is adjacent to two vertices in the sets $X_1, \ldots, X_6$ colored by 1 and 2 respectively. By symmetry, we assume that z_s is adjacent to $y_1 \in X_1$. Since G has no induced C_5, z_s is not adjacent to vertices X_2 and X_6, and therefore z_s is adjacent to $y_4 \in X_4$. Now w is not adjacent to vertices of X_1 and X_4. By symmetry, we can assume that $i = 2$, i.e. w is adjacent to a vertex $y_2 \in X_2$. Recall that $X_3 \neq \emptyset$ and let $y_3 \in X_3$. But then $y_2 w z_s y_1 x_1 x_6 \ldots x_3 y_3$ is an induced P_{10}. It means that each z_s is adjacent either only to vertices colored by 1 or only to vertices colored by 2 in the sets $X_1, \ldots, X_6$. In the first case it can be colored by 2 and we can use the color 1 in the second. □

Theorem 3. *Every P_{12}-free graphs of girth* 7 *is* 3*-colorable.*

Proof. Let $G = (V, E)$ be a connected graph of girth 7. Let also $U = \{x_1, \ldots, x_7\}$ be the vertex set of a C_7 in G (vertices are enumerated in the cycle order). Denote by X_i the set of vertices of $N^1(U)$ adjacent to x_i for $i = 1, \ldots, 7$. Using the absence of cycles of length $3, \ldots, 6$, we observe the following:

1. $X_i \cap X_j = \emptyset$ for $i, j \in \{1, \ldots, 7\}$, $i \neq j$;
2. $X_1 \cup \ldots \cup X_7$ is independent.

Let H be the subgraph of G induced by the set $V \setminus (\{x_1, \ldots, x_7\} \cup X_1 \cup \ldots \cup X_6)$ (notice that $X_7 \subseteq V_H$). We can prove that H is bipartite. Then we color G as follows: the vertices x_1, x_3, x_5 are colored by the color 1, x_2, x_4, x_6 by the color 2, the vertices of the set $X_1 \cup \ldots \cup X_6$ and x_7 are colored by 3, and finally all the vertices of the bipartite graph H are colored by the colors 1 and 2. □

Theorem 4. *Every P_k-free graph with $k = 2g + \lceil \frac{g(G)-2}{4} \rceil - 3$ and $g \geq 8$ is* 3*-colorable.*

Proof. Let $G = (V, E)$ be a connected graph of girth $g \geq 8$. Let also $U = \{x_1, \ldots, x_g\}$ be the vertex set of a C_g in G (vertices are enumerated in the cycle order). Let $s = \lceil \frac{g(G)-2}{4} \rceil - 1$. We need the following properties of the sets $N^t(U)$:

1. for any $t \in \{1, \dots, s\}$, $N^t(U)$ is independent;
2. for any $t \in \{1, \dots, s\}$, each vertex $x \in N^t(U)$ is adjacent to the unique vertex in $N^{t-1}(U)$.

We distinguish two cases. We first consider the case $g = 9$ and then the case $g \neq 9$.

Case 1. $g = 9$. Let $X \subseteq N^1(U)$ be the set of vertices adjacent to x_9 and let $Y \subseteq N^2(U)$ be the set of vertices adjacent to the vertices of X. Since G has no cycles of length less than 9, we observe additionally to the properties 1 and 2 that

3. Y is independent;
4. vertices of Y are not adjacent to the vertices of $N^1(U) \setminus X$.

Let H be the subgraph of G induced by the set $V \setminus (N^1[U] \cup Y)$. We can prove that H is bipartite. Then we color G as follows: the vertices x_1, x_3, x_5, x_7 are colored by the color 1 and the vertices x_2, x_4, x_6, x_8 are colored by 2, the vertex x_9 and the vertices of the independent set $N^1(U) \setminus X$ are colored by 3, X and Y are colored by 1 and 3 respectively, and finally the vertices of the bipartite graph H are colored by the colors 1 and 2.

Case 2. $g \neq 9$. Let H be the subgraph of G induced by the set $V \setminus N^s[U]$. We prove that H is bipartite as in Case 1. Then we color G. If g is even then a 3-coloring is constructed as follows: the vertices x_{2i-1} are colored by color 1 and the vertices x_{2i} are colored by 2 for $1 \leq i \leq g/2$, the vertices of the independent sets $N^1(U), \dots, N^s(U)$ are colored by 1 and 3, and finally the vertices of the bipartite graph H are colored by colors 1 and 2 if $N^s(U)$ was colored by 3, and by colors 2 and 3 otherwise. Now suppose that g is odd. Then $g \geq 11$. Let $X \subseteq N^1(U)$ be the set of vertices adjacent to x_g. By Property 2, the vertices of X are not adjacent to the vertices $x_1, \dots, x_{g-1}$. Because $g \geq 11$, we have $s \geq 2$. We color the vertices x_{2i-1} by the color 1 and the vertices x_{2i} are colored by 2 for $1 \leq i \leq \lfloor g/2 \rfloor$, the vertex x_g and the vertices of the independent set $N^1(U) \setminus X$ are colored by 3, the vertices of X are colored by 1. Then the vertices of the independent sets $N^2(U), \dots, N^s(U)$ are colored by 2 and 3, and finally the vertices of the bipartite graph H are colored by colors 1 and 3 if $N^s(U)$ was colored by 2, and by colors 1 and 2 otherwise. □

3 The NP-Completeness Result for 4-Colorability

In this section we show that 4-Coloring is NP-complete for (C_3, P_{164})-free graphs. As the problem is clearly in NP, we are left to prove NP-hardness. The NP-hardness reductions for k-Coloring for P_ℓ-free graphs are based on the presence of triangles in the gadgets. Hence, the main task is to design a triangle-free gadget that can replace a number of edges of a graph G in order to make G triangle-free. We present this gadget and its properties in Section 3.1. We show how to incorporate it in our final gadget in Section 3.2, where we present our NP-hardness reduction, which is from the NP-complete problem Not-All-Equal-3-Satisfiability (cf. [10]).

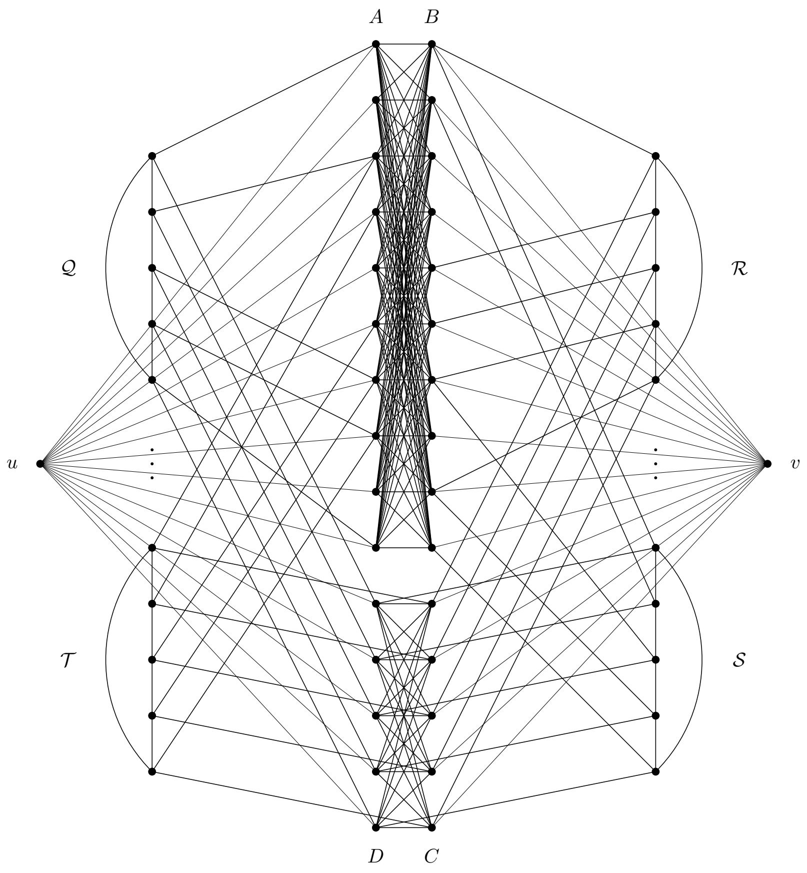


Fig. 1. The graph F; only one Q-cycle, R-cycle, S-cycle and one T-cycle are displayed

3.1 The Edge-Replacing Gadget

We define four independent sets A, B, C and D where $|V(A)| = |V(B)| = 10$ and $|V(C)| = |V(D)| = 5$. We add an edge between every vertex in A and every vertex in B. We also add an edge between every vertex in C and every vertex in D. This leads to two vertex-disjoint complete bipartite graphs with partition classes A, B and C, D, respectively.

For every subset $A_i \subseteq A$ of 5 vertices, we create two cycles Q_i and T_i, each on 5 new vertices. We say that Q_i is a Q-cycle and that T_i is a T-cycle. We add 5 edges between the vertices of Q_i and A_i in such a way that these edges form a matching. We do the same for T_i and A_i. We also add 5 matching edges between the vertices of Q_i and D, and do the same for T_i and C. We let $\mathcal{Q}$ and $\mathcal{T}$ denote the set of all $\binom{10}{5}$ Q-cycles and T-cycles, respectively. Similarly, we define two sets $\mathcal{R}$ and $\mathcal{S}$ of $\binom{10}{5}$ R-cycles and S-cycles, respectively. Here, each R-cycle and each S-cycle correspond to exactly one subset $B_i \subseteq B$ of 5 vertices. For each such B_i there are matchings between its vertices and the vertices in its R-cycle and S-cycle, respectively. There is also a matching between the vertices of each R-cycle and C, and between the vertices of each S-cycle and D. Finally, we add a new vertex u adjacent to every vertex of $A \cup D$, and a new vertex v adjacent to every vertex of $B \cup C$. The resulting graph called F is C_3-free; see Figure 1.

Lemmas 1 and 2 state some useful properties of F that we will use later on. The proof of Lemma 1 has been omitted.

Lemma 1. *Let P be an induced path in F. Then the following statements hold:*

(i) If P starts in u and ends in v, then $|V_P| \leq 8$.
(ii) If P starts in u and contains v as an internal vertex, then $|V_P| \leq 9$.
(iii) If P starts in u and does not contain v, then $|V_P| \leq 45$.
(iv) If P does not contain u or v as end-vertices, then $|V_P| \leq 90$.

Lemma 2. *The graph F is 4-colorable. Moreover, u and v are not colored alike in every 4-coloring of F.*

Proof. We first show that F is 4-colorable. We choose a vertex $c \in C$ and a vertex $d \in D$, which we give color 1 and 2, respectively. We give each vertex of $(A \cup D) \setminus \{d\}$ color 3 and each vertex of $(B \cup C) \setminus \{c\}$ color 4. We give u color 1 and v color 3. Consider a Q-cycle. We give its vertex adjacent to d color 1 and color its other four vertices by colors 2 and 4. Consider a T-cycle. We give its vertex adjacent to c color 4 and color its other four vertices by colors 1 and 2. By symmetry, we can also give the vertices of every R-cycle and S-cycle an appropriate color such that in the end we have obtained a 4-coloring of F.

We now prove the second statement. Let ϕ be a 4-coloring of F. First suppose that $|\phi(C)| \geq 2$ and $|\phi(D)| \geq 2$. Because C and D are partition classes of a complete bipartite graph, we then may without loss of generality assume that $\phi(C) = \{1,4\}$ and $\phi(D) = \{2,3\}$. This means that u can only get a color from $\{2,3\}$ and v can only get a color from $\{1,4\}$. Hence, u and v are not colored alike.

In the remaining case, we may assume without loss of generality that $|\phi(D)| = 1$. If $|\phi(A)| = 4$, then we cannot color a vertex in B. Hence, $|\phi(A)| \leq 3$. If $|\phi(A)| = 3$, then every vertex of $B \cup \{u\}$ receives the same color. Because v is adjacent to the vertices of B, this means that v must receive a different color.

Suppose that $|\phi(A)| \leq 2$. Then A contains a subset A_i of five vertices that are colored alike, say with color 3. We observe that u does not get color 3. Consider the Q-cycle corresponding to A_i. Its five vertices can neither be colored with color 3 nor with the color in $\phi(D)$. Because this cycle needs at least three colors, this means that $\phi(D) = \phi(A) = \{3\}$. Because every vertex of A is adjacent to every vertex of B, color 3 is not used on B, so $|\phi(B)| \leq 3$. Suppose that $|\phi(B)| \leq 2$. Then B contains a subset B_j of five vertices that are colored alike, say with color 4. We consider the S-cycle corresponding to B_j. Because every vertex of this cycle is not only adjacent to a vertex of B_j with color 4 but also adjacent to a vertex of D with color 3, we find that only colors 1 and 2 are available to color its five vertices. This is not possible. Hence, $|\phi(B)| = 3$, so

$\phi(B) = \{1, 2, 4\}$. This means that v must receive color 3, whereas we already deduced that u does not get color 3. This completes the proof of Lemma 2. □

3.2 Using the Edge-Replacing Gadget

In this section we present our reduction for showing that 4-Coloring is NP-complete for the class of (C_3, P_{164})-free graphs. This reduction is from the Not-All-Equal 3-Satisfiability problem with positive literals only. This problem is NP-complete [20] and is defined as follows. Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of logical variables, and a set $C = \{C_1, C_2, \ldots, C_m\}$ of three-literal clauses over X in which all literals are positive, does there exist a truth assignment for X such that each clause contains at least one true literal and at least one false literal?

We consider an arbitrary instance I of Not-All-Equal 3-Satisfiability that has variables $\{x_1, x_2, \ldots, x_n\}$ and clauses $\{C_1, C_2, \ldots, C_m\}$. From I we first construct the graph G from our previous paper [5]. We then explain how to incorporate the gadget F from Section 3.1. This will yield a graph G'. In Lemma 3 we will show that G' is (C_3, P_{164})-free. In Lemma 4 we will show that G' is 4-colorable if and only if I has a satisfying truth assignment in which each clause contains at least one true literal and at least one false literal.

Here is the construction that defines the graph G.

- For each clause C_j we introduce a gadget with vertex set

$$\{a_{j,1}, a_{j,2}, a_{j,3}, b_{j,1}, b_{j,2}, c_{j,1}, c_{j,2}, c_{j,3}, d_{j,1}, d_{j,2}\}$$

  and edge set

$$\{a_{j,1}c_{j,1}, a_{j,2}c_{j,2}, a_{j,3}c_{j,3}, b_{j,1}c_{j,1}, c_{j,1}d_{j,1}, d_{j,1}c_{j,2}, c_{j,2}d_{j,2}, d_{j,2}c_{j,3}, c_{j,3}b_{j,2}, b_{j,2}b_{j,1}\},$$

  and a disjoint gadget called the *copy* that has vertex set

$$\{a'_{j,1}, a'_{j,2}, a'_{j,3}, b'_{j,1}, b'_{j,2}, c'_{j,1}, c'_{j,2}, c'_{j,3}, d'_{j,1}, d'_{j,2}\}$$

  and edge set

$$\{a'_{j,1}c'_{j,1}, a'_{j,2}c'_{j,2}, a'_{j,3}c'_{j,3}, b'_{j,1}c'_{j,1}, c'_{j,1}d'_{j,1}, d'_{j,1}c'_{j,2}, c'_{j,2}d'_{j,2}, d'_{j,2}c'_{j,3}, c'_{j,3}b'_{j,2}, b'_{j,2}b'_{j,1}\}.$$

  We say that all these vertices (so, including the vertices in the copy) are of a-type, b-type, c-type and d-type, respectively.

- Every variable x_i is represented by a vertex in G, and we say that these vertices are of x-type.
- For every clause C_j we fix an arbitrary order of its variables $x_{i_1}, x_{i_2}, x_{i_3}$ and add edges $c_{j,h}x_{i_h}$ and $c'_{j,h}x_{i_h}$ for $h = 1, 2, 3$.
- We add an edge between every x-type vertex and every b-type vertex. We also add an edge between every x-type vertex and every d-type vertex.
- We add an edge between every a-type vertex and every b-type vertex. We also add an edge between every a-type vertex and every d-type vertex.

In Figure 2 we illustrate an example in which C_j is a clause with ordered variables $x_{i_1}, x_{i_2}, x_{i_3}$. The thick edges indicate the connection between the variables vertices and the c-type vertices of the two copies of the clause gadget. The dashed thick edges indicate the connections between the a-type and c-type vertices of the two copies of the clause gadget. We omitted the indices from the labels of the clause gadget vertices to increase the visibility.

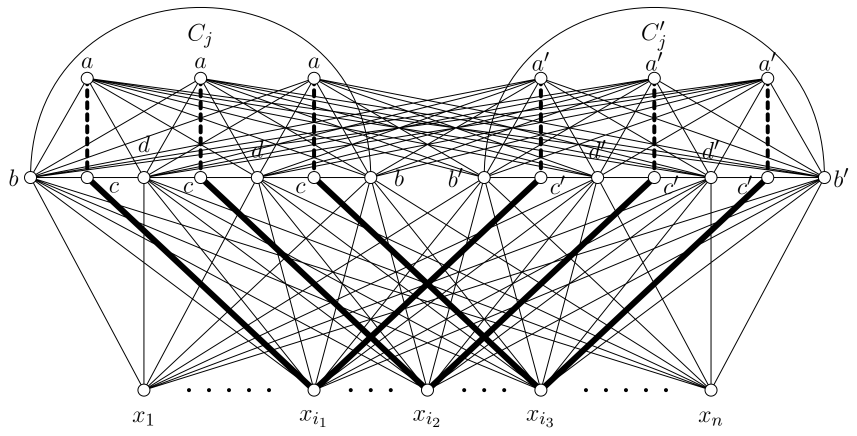


Fig. 2. The graph G for the clause $C_j = \{x_{i_1}, x_{i_2}, x_{i_3}\}$

Before we show how to obtain the graph G', we introduce the following terminology. Let H be some graph. An F*-identification* of an edge $st \in E_H$ is the following operation. We remove the edge st from H but keep the vertices s and t. We take a copy of F an remove u and v from it. We then add an edge between s and $N_F(u)$ and an edge between t and $N_F(v)$. Note that by symmetry we could reverse the role of u and v in this operation.

In order to obtain G' from G we first apply consecutive F-identifications on all edges between a-type and c-type vertices, on all edges between c-type and x-type vertices and on all edges between two b-type vertices. We take a complete graph on four new vertices $r_1, \ldots r_4$ called r-type vertices, and apply consecutive F-identifications on each edge between them. This leads to a graph K. We connect K to the modified graph G by adding an edge between every $a_{i,j}$ and every vertex in $\{r_2, r_3, r_4\}$ and an edge between every $a'_{i,j}$ and every vertex in $\{r_1, r_3, r_4\}$. This completes the construction of G'.

The proof of Lemma 3 has been omitted.

Lemma 3. *The graph G' is (C_3, P_{164})-free (but not P_{163}-free).*

Lemma 4. *The graph G' is 4-colorable if and only if I has a satisfying truth assignment in which each clause contains at least one true literal and at least one false literal.*

Proof. We need the following claim from [5].
Claim 1. The graph G has a 4-coloring in which every $a_{j,h}$ has color 1 and every $a'_{j,h}$ has color 2 if and only if I has a truth assignment in which each clause contains at least one true and at least one false literal.

Suppose that G' is 4-colorable. By Lemma 2, the vertices of the edges on which we applied F-identifications do not have the same color. This means that G is 4-colorable. It also means that the vertices $r_1, \ldots, r_4$ are not colored alike. We may assume without loss of generality that r_i gets color i for $i = 1, \ldots, 4$. Then, by construction, every $a_{j,h}$ has color 1 and every $a'_{j,h}$ has color 2 in G', and consequently in G. By Claim 1, I has a truth assignment in which each clause contains at least one true and at least one false literal.

Suppose that I has a truth assignment in which each clause contains at least one true and at least one false literal. By Claim 1, G has a 4-coloring in which every $a_{j,h}$ has color 1 and every $a'_{j,h}$ has color 2. By Lemma 2 we can extend the 4-coloring of G to a 4-coloring of G'. □

The main result of this section follows directly from Lemmas 3 and 4 after recalling that 4-Coloring is in NP and that Not-All-Equal-3-Satisfiability is NP-complete and observing that the construction of G' can be carried out in polynomial time.

Theorem 5. *The* 4-Coloring *problem is* NP*-complete even for* (C_3, P_{164})*-free graphs.*

References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory, vol. 244. Springer Graduate Texts in Mathematics (2008)
2. Brandt, S.: Triangle-free graphs and forbidden subgraphs. Discrete Applied Mathematics 120, 25–33 (2002)
3. Brinkmann, G., Meringer, M.: The Smallest 4-Regular 4-Chromatic Graphs with Girth 5. Graph Theory Notes of New York 32, 40–41 (1997)
4. Broersma, H., Fomin, F.V., Golovach, P.A., Paulusma, D.: Three Complexity Results on Coloring P_k-Free Graphs. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 95–104. Springer, Heidelberg (2009)
5. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Updating the complexity status of coloring graphs without a fixed induced linear forest, manuscript, http://www.dur.ac.uk/daniel.paulusma/Papers/Submitted/updating.pdf
6. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time, manuscript, http://www.dur.ac.uk/daniel.paulusma/Papers/Submitted/2p3.pdf
7. Couturier, J.F., Golovach, P.A., Kratsch, D., Paulusma, D.: List coloring in the absence of a linear forest. In: Proceedings of WG 2011. LNCS (to appear, 2011)
8. Dabrowski, K., Lozin, V., Raman, R., Ries, B.: Colouring Vertices of Triangle-Free Graphs. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 184–195. Springer, Heidelberg (2010)
9. Erdös, P.: Graph Theory and Probability, Canad. J. Math 11, 34–38 (1959)

10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
11. Grötzsch, H.: Zur Theorie der diskreten Gebilde. VII. Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel, Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg. Math.-Nat. Reihe 8, 109–120 (1958)
12. Hoàng, C.T., Kamiński, M., Lozin, V., Sawada, J., Shu, X.: Deciding k-colorability of P_5-free graphs in polynomial time. Algorithmica 57, 74–81 (2010)
13. Holyer, I.: The NP-completeness of edge-coloring. SIAM J. Comput. 10, 718–720 (1981)
14. Kamiński, M., Lozin, V.V.: Coloring edges and vertices of graphs without short or long cycles. Contributions to Discrete Math. 2, 61–66 (2007)
15. Král, D., Kratochvíl, J.: Complexity of coloring graphs without forbidden induced subgraphs. In: Brandstädt, A., Van Le, B. (eds.) WG 2001. LNCS, vol. 2204, pp. 254–262. Springer, Heidelberg (2001)
16. Kratochvíl, J.: Precoloring extension with fixed color bound. Acta Math. Univ. Comen. 62, 139–153 (1993)
17. Maffray, F., Preissmann, M.: On the NP-completeness of the k-colorability problem for triangle-free graphs. Discrete Math. 162, 313–317 (1996)
18. Randerath, B., Schiermeyer, I.: 3-Colorability $\in$ P for P_6-free graphs. Discrete Appl. Math. 136, 299–313 (2004)
19. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. Graphs Combin 20, 1–40 (2004)
20. Schaefer, T.J.: The complexity of satisfiability problems. In: Conference Record of the Tenth Annual ACM Symposium on Theory of Computing, San Diego, Calif, pp. 216–226. ACM, New York (1978)
21. Sumner, D.P.: Subtrees of a graph and the chromatic number. In: Chartrand, G. (ed.) 4th Int. Conf. Kalamazoo/Mich The Theory and Applications of Graphs, pp. 557–576. Wiley, New York (1980)
22. Tuza, Z.: Graph colorings with local restrictions - a survey, Discuss. Math. Graph Theory 17, 161–228 (1997)

Succinct Algebraic Branching Programs Characterizing Non-uniform Complexity Classes

Guillaume Malod

Équipe de Logique Mathématique, Institut de Mathématiques de Jussieu,
Université Paris Diderot — Paris 7, France
`malod@logique.jussieu.fr`

Abstract. We study characterizations of algebraic complexity classes by branching programs of possibly exponential size, using a succinctness condition to replace the usual one based on uniformity. We obtain characterizations of VPSPACE, the class corresponding to computations in polynomial space, and observe that algebraic polynomial space can be seen as constant algebraic space with auxiliary polynomial space Boolean computations. We also obtain the first examples of natural complete polynomials for VPSPACE, in particular showing that polynomials like the determinant, the permanent or the Hamiltonian become VPSPACE-complete when the matrix is succinctly encoded. Using the same techniques we also characterize VNP. In general, we argue that characterizations by branching programs apply to different classes, both Boolean and algebraic, with or without uniformity, and thus provide a general and uniform technique in these different settings.

1 Introduction

This work stems from two observations in Boolean complexity. The first concerns circuit characterizations of uniform complexity classes. Consider a uniform sequence of circuits (C_n) of possibly exponential size. In this case a Turing machine cannot produce an encoding of the circuit in polynomial time, so the uniformity condition employed is often based on the *direct connection language* of the circuit [17] and requires that there exists a Turing machine which can decide the nature of a gate and if a gate is an argument of another, in time logarithmic in the size of the circuit. We observe that this uniformity has two consequences. One is that different circuits for different input sizes have a common "structure", since they are defined by the same Turing machine. The other consequence is that each circuit C_n, even though it is of exponential size, has a compact representation. We would like to separate these two aspects, namely the uniformity of the computational objects for different input sizes on the one hand and the compactness of the representation of the computational object for each given input size on the other. In particular we wish to study circuit characterizations in the non-uniform case. Note that circuit characterizations have also been studied in the non-uniform case, for example in [4], which quotes Meyer and Stockmeyer suggesting "making the Turing machines nonuniform by giving them oracles". By contrast, we wish to focus directly on going from one type of compact computational object for a given input size to another. In the words of Poizat [16], we wish to avoid "introducing uniformity with such complicated combinatorial contraptions as Turing machines, only to artificially destroy it later".[1]

The second observation is the strong relationship between Boolean space complexity and accessibility questions in directed graphs (see Chap. 4 of [1]). There is a strikingly simple reduction in the case of computations done by a Turing machine in polynomial space: the number of possible configurations is simply exponential and can thus be seen as vertices in a graph where the existence of an edge can be decided by a small circuit built from the original Turing machine; the original computation of the machine then reduces to deciding the existence of a path in the graph. This argument depends on the Turing model and the fact that the cells on the tapes contain symbols chosen from a finite alphabet. We are interested in studying whether a similar result holds when the computations are algebraic. Note that the graph for polynomial space computations mentioned above is often built for each given input x but it can naturally be defined once for each fixed input size. It can therefore be seen in a way which is consistent with our first remarks, namely as a computational object for a given input size with a compact representation: the circuit deciding the existence of an edge.

We show that the uniformity and the notion of configurations provided by a Turing machine are not necessary and that similar results can be obtained in the non-uniform algebraic case. We consider weighted acyclic directed graphs with the sole property that there exists a small circuit which computes the weight of an edge given its vertices. This graph can be seen as an *algebraic branching program*, which we call *succinct*. After introducing basic definitions in Sect. 2, we show in Sect. 3 that VPSPACE can be characterized as containing the sequences of polynomials computed by such succinct algebraic branching programs defined by a sequence of circuits of polynomial size. This similarity strengthens the idea that the class VPSPACE is a good analog of PSPACE. We use this result to get examples of complete polynomials for VPSPACE, in particular showing that polynomials like the determinant, the permanent or the Hamiltonian become VPSPACE-complete when the matrix is succinctly encoded. In Sect. 4 we show that VPSPACE can be characterized by succinct layered algebraic branching programs of width 3, from which we deduce other characterizations in terms of succinct circuits. Section 5 applies similar ideas to VNP, an algebraic class similar to NP or $\sharp$P, to get a characterization by succinct layered algebraic branching programs of polynomial length. Finally, Sect. 6 applies our results to Boolean classes and relates them to similar works, illustrating the fact that our techniques provide a uniform argument valid in different settings. Due to lack of space, proofs are omitted.

[1] "Introduire de l'uniformité par un dispositif combinatoire aussi compliqué qu'une machine de Turing pour ensuite la détruire par un artifice".

2 Preliminaries

We will express most of our results in the setting defined by Valiant [18,19] for algebraic computations, focusing on sequences of polynomials computed by arithmetic circuits. We briefly recall here the necessary definitions, more details can be found in [10,5,15].

An *arithmetic circuit* is a finite simple directed acyclic graph with vertices of in-degree 0, called *input gates* and labeled by a constant or a variable, and vertices of in-degree at least 2 labeled by $+$ or $\times$. Vertices of out-degree 0 are called *output gates*. Unless specified, we will consider circuits with computation gates of in-degree 2. Circuits where addition gates can have arbitrary in-degree are called *semi-unbounded fan-in* circuits, while circuits where both types of computation gates can have arbitrary in-degree are called *unbounded fan-in* circuits. The polynomial computed by an arithmetic circuit can be defined in an obvious way by induction on the gates. If S is a subset of the gates of a circuit C, the *sub-circuit of C at S* is the graph induced by the set of all gates of C connected to a gate in S by a directed path. We will call this circuit C_S.

A circuit is a *term* if all its gates have out-degree at most 1. A circuit is *weakly skew* if for every multiplication gate $\alpha = \beta_1 \times \cdots \times \beta_k$, at most one of the edges (β_i, α) is not a bridge in the circuit, i.e. for all but at most one of the argument gates, removing the edge disconnects the circuit. A circuit is *skew* if for every multiplication gate at most one of its arguments is not an input gate. A circuit is *multiplicatively disjoint* if for every multiplication gate all of its argument sub-circuits are pairwise disjoint.

The *size* of a circuit is its number of gates. The *depth* of a circuit is the greatest length of a path from an input gate to an output gate. If the gates of a circuit are partitioned in levels so that a computation gate can only receive an arrow from a gate at the previous level or from an input gate, we say that the circuit is *layered* and define the *width* of the circuit as the maximum number of gates at a level. The *degree* of an arithmetic circuit is defined inductively: 1 for an input gate; the max of the degrees of the arguments for an addition gate ; the sum of the degrees of the arguments for a multiplication gate.

An *algebraic branching program* is a tuple (G, s, t) where G is a weighted simple directed acyclic graph and s and t are vertices in G. The *size* of a branching program is the number of its vertices. The *weight* of a path in a branching program is the product of the weights of its edges. The polynomial computed by a branching program G is the sum of the weights of all the paths from s to t in G. The *length* of a branching program is the greatest length of a path. If we can partition the vertices of a branching program in levels so that there are only edges between vertices in successive levels, we say that the branching program is *layered* and define the width as the greatest number of vertices in a level.

We will also consider arithmetic circuits which also contain gates of in-degree 1 called *projection gates*. A projection gate is labeled with a variable x and a value $\epsilon \in \{0, 1\}$; if it receives an arrow from a gate computing a polynomial $f(x, \bar{y})$ then it computes the polynomial $f(\epsilon, \bar{y})$. Circuits with projection gates were introduced in [2] to study Boolean complexity classes. The equivalent *summation* *gates* were used in [16] to study polynomial space computations in Valiant's theory.

The different kinds of circuits above let us classify sequences of polynomials into the following classes.

- $\mathrm{VP_e}$ is the class of sequences of polynomials computed by arithmetic terms of polynomial size or equivalently by constant-width branching programs of polynomial length (a width of 3 is sufficient, as shown in see [3]).
- $\mathrm{VP_{ws}}$ is the class of sequences of polynomials computed by (weakly) skew circuits of polynomial size or equivalently by branching programs of polynomial width and length.
- VP is the class of sequences of polynomials computed by multiplicatively disjoint circuits of polynomial size.
- A sequence of polynomials $(f_n(\bar{x}))$ belongs to VNP if there exists a sequence $(g_n(\bar{x},\bar{y}))$ in VP such that $f_n(\bar{x}) = \sum_\epsilon g_n(\bar{x},\bar{\epsilon})$, where the sum is over all Boolean tuples $\bar{\epsilon}$ of appropriate length. It can also be defined via multiplicatively disjoint circuits with projections of polynomial size (see [16]).
- VPSPACE is the class of sequences of polynomials computed by circuits with projections of polynomial size (see [16] for this characterization, the class was originally defined in [12] via polynomials of possibly exponential degree and coefficient-size, whose coefficient function can be computed in bits in PSPACE/poly).

In the above definitions, it is usual to consider circuits which can use arbitrary constants from a given ring, or constant-free versions, which can only use the constant -1. In any case all the constructions done here will be purely structural and as such will work whether we allow arbitrary constants or not.

3 VPSPACE and Succinct Algebraic Branching Programs

As explained in the introduction, we take from PSPACE the idea of the configuration graph with the sole condition that there exists a small circuit testing the existence of an edge. We adapt it here to algebraic computations.

Definition 1. *A* succinct algebraic branching program *is a tuple* $(B,\bar{s},\bar{t})$ *such that:*

- *B is an arithmetic circuit with one output gate and inputs $\bar{u},\bar{v},\bar{a}$ where $|\bar{u}| = |\bar{v}| = |\bar{s}| = |\bar{t}| = r$, $\bar{a}$ is a tuple of variables and constants, and $\bar{s}$ and $\bar{t}$ are tuples in $\{0,1\}^r$,*
- *the weighted directed graph G_B, whose vertices are all the tuples $\bar{u} \in \{0,1\}^r$ and where the weight of the edge from $\bar{u}$ to $\bar{v}$ is the polynomial computed by the output of B, is simple and acyclic.*

The polynomial computed by $(B,\bar{s},\bar{t})$ is the sum of the weights of all the paths in G_B from vertex $\bar{s}$ to vertex $\bar{t}$. The complexity *of $(B,\bar{s},\bar{t})$ is the size of B. If the polynomial computed by the output of B is a variable or a constant for all tuples $\bar{u},\bar{v}$ in $\{0,1\}^r$ we say that the branching program is* simple.

We will show that succinct algebraic branching programs of polynomial complexity characterize VPSPACE. As mentioned in the definition, a succinct algebraic branching program can define a graph whose weights are polynomials rather than just constants or variables. A consequence of our results will be that for polynomials in VPSPACE we can always assume that the succinct algebraic branching program is simple. In fact, we could have defined the circuit B as doing a purely Boolean computation indicating if the weight of an edge is 0, 1 or one of the inputs $\bar{a}$.

First we show that sequences in VPSPACE have succinct algebraic branching programs. We will always start from the definition of VPSPACE via circuits with projections stated in Sect. 2.

Lemma 1. *If a polynomial can be computed by an arithmetic circuit with projections of size s, then it can be be computed by a simple succinct algebraic branching program $(B, \bar{0}, \bar{1})$ of complexity $O(s^3)$.*

For the converse we will use matrix powers as an intermediate step . The size of the matrix we will consider (and the powers) may exponential, so that we also need to define succinct matrices.

Definition 2. *A* succinct matrix *is an arithmetic circuit B with inputs $\bar{u}, \bar{v}, \bar{a}$ such that $|\bar{u}| = |\bar{v}| = r$ and $\bar{a}$ is a tuple of variables and constants. The matrix defined has lines and columns indexed by all tuples in $\{0,1\}^r$. The output gate of B on inputs $\bar{u}, \bar{v} \in \{0,1\}^r$ is the coefficient $(\bar{u}, \bar{v})$ of the matrix. The* complexity *of the matrix is the size of B.*

Lemma 2. *Let M be a succinct matrix of complexity s and N an integer, then there exists a circuit with projections of size $O(s \log N)$ which, given Boolean tuples $\bar{u}$ and $\bar{v}$, computes the coefficient $(\bar{u}, \bar{v})$ of M^N.*

Both results together give us our first characterization of VPSPACE.

Theorem 1. *A sequence of polynomials belongs to* VPSPACE *iff it can be computed by a sequence of succinct algebraic branching programs of polynomial complexity (and possibly exponential size).*

One possible question with this definition of succinct algebraic branching programs and succinct matrices is whether restricting the power of the defining circuit changes the class. Looking carefully at the proof of Lemma 1 shows that the circuit built is weakly skew, so that we get an equivalent definition if we impose this restriction. We now consider circuits which instead of projection gates have two new kinds of gates of in-degree 1:

– a *production gate* is labeled by π_z, where z is a variable; if it receives an edge from a gate computing a polynomial $f(z)$, it computes the polynomial $f(0)f(1)$,
– a *summation gate* is labeled by σ_z, where z is a variable; if it receives an edge from a gate computing a polynomial $f(z)$, it computes the polynomial $f(0) + f(1)$.

Circuits with one or both of these types of gates are easily seen to be equivalent to circuits with projections [16]. Using these gates, the proof of Lemma 2 also gives us a normal form for sequences in VPSPACE, similar to the completeness of true quantified Boolean formulas for PSPACE.

Corollary 1. *A sequence of polynomials* (f_n) *belongs to* VPSPACE *iff there exists a polynomial* q *and a sequence* (C_n) *of arithmetic circuits (without projections) of polynomial size such that each polynomial* f_n *can be expressed as* $O^1_{w_1} \cdots O^{q(n)}_{w_q(n)} C_n$*, where* $O^i_{w_i}$ *is a summation or a production gate.*

Note also that a slight modification of the proof of Lemma 2 gives a simple proof of the fact that the class VPSPACE is closed for coefficient functions, as shown in [16].

We have characterized polynomials in VPSPACE by succinct algebraic branching programs. We could also have chosen exponential powers of succinct matrices. We will now use this aspect to get the first examples of natural complete polynomials for VPSPACE. We first need to define a generic succinct matrix of size n. This will be done by considering a circuit $G_n(\bar{u}, \bar{v}, \bar{x}, \bar{\alpha})$, where all the tuples $\bar{u}, \bar{v}, \bar{x}$ have length n, with the property that for any circuit $C(\bar{u}, \bar{v}, \bar{x})$ of size at most n^2 we can give Boolean values to the extra variables $\bar{\alpha}$ so that G_n computes the same polynomial as C. This can be done with a construction similar to the one given in [14], itself inspired by [5] (Sect. 5.6), so that the size of G_n is polynomial in n. The *generic succinct matrix* of size n is then the matrix $\mathcal{M}_n$ defined by G_n. The *succinct matrix power polynomial* P_n is the coefficient $(\bar{0}, \bar{1})$ of the matrix $\mathcal{M}_n^{2^n}$.

Corollary 2. *The sequence* (P_n) *is* VPSPACE*-complete.*

This result can be seen as a "scaled-up" version of the completeness of matrix powers for the class $\mathrm{VP_{ws}}$ given in [15]. We can also consider the generic succinct matrix as a succinct encoding of instances, in the spirit of [9,21]. Without succinctness, the determinant is $\mathrm{VP_{ws}}$-complete, while the permanent is VNP-complete (in characteristic different from 2) and the Hamiltonian is VNP-complete. Using techniques given in [15] and [14] which show that these polynomials can express any polynomial computed by a branching program, we can show that the complexity of the succinct versions all jump to VPSPACE and get more examples of natural complete polynomials.

Corollary 3. *The succinct determinant, the succinct permanent and the succinct Hamiltonian are* VPSPACE*-complete.*

4 VPSPACE and Succinct Layered Algebraic Branching Programs

The succinct algebraic branching program of Lemma 1 is produced by inductively applying series or parallel operations. It may therefore be exponentially long and exponentially wide. In this section we show how the construction can be modified to get a branching program of constant width. Although the width of a branching program can be defined in general, it will be more convenient to work with the common notion of layered branching programs, for which we introduce a succinct variant.

Definition 3. *A* succinct layered algebraic branching program *is a tuple* $(B, \bar{s}, \bar{t})$*, such that:*

- *B is an arithmetic circuit with one output gate and inputs $\bar{m}, \bar{u}, \bar{v}, \bar{a}$ where $|\bar{u}| = |\bar{v}| = r$, $\bar{a}$ is a tuple of variables and constants, and $\bar{s}$ and $\bar{t}$ are tuples in $\{0,1\}^r$,*
- *B defines a layered weighted direct acyclic graph G_B: $B(\bar{m}, \bar{u}, \bar{v}, \bar{a})$ is the value of the edge from vertex $\bar{u}$ of level $\bar{m}$ to vertex $\bar{v}$ of level $\bar{m}+1$.*[2]

The polynomial computed is the sum of the weights of all the paths from vertex $\bar{s}$ of level $\bar{0}$ to vertex $\bar{t}$ of level $\bar{1}$. The complexity *of $(B, \bar{s}, \bar{t})$ is the size of B. If the polynomial computed by the output of B is a variable or a constant for all tuples $\bar{u}, \bar{v}$ in $\{0,1\}^r$ we say that the branching program is* simple*.*

Note that each level in the branching program can be seen as a matrix, so that we could have equivalently called such a branching program a *succinct iterated matrix multiplication*. Depending on the case we will use one terminology or the other. This notion again characterizes VPSPACE, the main part being the following lemma, which relies on Ben-Or and Cleve's result [3] showing that arithmetic computations can be seen as products of matrices of size 3.

Lemma 3. *If a polynomial can be computed by a circuit with projections of size s, then it can be computed by a simple succinct layered algebraic branching program $(B, \bar{0}, \bar{1})$ of width* 3 *and complexity $O(s^3)$.*

We thus get another characterization of algebraic space, similar to the results obtained in [11,8] in the Boolean uniform case.

Theorem 2. *A sequence of polynomials belongs to* VPSPACE *iff it can be computed by a sequence of succinct layered algebraic branching programs of width* 3*, possibly exponential length and polynomial complexity.*

We will now see what happens when we look at an iterated matrix multiplication and compute it as a circuit. Matrix product is associative, therefore we can compute it in different ways. A first way is to compute it sequentially, i.e. we see a product $M_1 \cdots M_k$ as $\Big(\big((M_1M_2)\cdots M_{k-1}\big)M_k\Big)$. Note that if we build a circuit for this product, the multiplications are all skew. Since the class of the determinant, $\mathrm{VP}_{\mathrm{ws}}$, can also be characterized by matrix products, this is one way of proving that the determinant can be computed by skew circuits. Another way is to compute the product as a balanced tree: we see $M_1 \cdots M_k$ as $(M_1 \cdots M_{k/2})(M_{k/2+1} \cdots M_k)$ and proceed recursively for each part. This gives

[2] $\bar{m}+1$ is the Boolean tuple corresponding to the integer obtained when incrementing the integer coded by $\bar{m}$.

a tree of matrix products of depth logarithmic in the number of matrices. Each matrix product in this tree can then be replaced by a small constant-depth circuit if we allow addition gates to have unbounded in-degree. Applying this to $\mathrm{VP}_{\mathrm{ws}}$ and the determinant, this is a very simple way of getting a parallel computation, a special case of the parallelization result of [20]. Thus, seeing a computation as a matrix product lets us easily deduce two different computational properties for a class.

We will now apply this idea to *succinct* iterated matrix multiplications. As above, we will compute these products via circuits. Since our matrix products involve an exponential number of matrices, the resulting circuits may also be of exponential size. We therefore need a notion of succinct circuits. This has been done in the Boolean case in [22] and with uniformity in the algebraic case in [13].

Definition 4. *A* succinct arithmetic circuit *is a triple* $(E(\bar{u},\bar{v}), L(\bar{u}), \bar{a})$*, where* E *and* L *are circuits and* $|\bar{u}| = |\bar{v}| = r$ *and the directed graph over all* $\bar{u} \in \{0,1\}^r$*, with an edge from* $\bar{u}$ *to* $\bar{v}$ *iff* $E(\bar{u},\bar{v})$ *is non-zero, is simple and acyclic. If* $|\bar{a}| = q$*,* $L(\bar{u})$ *has outputs gates* $\alpha_1, \ldots, \alpha_q, \alpha_+, \alpha_\times$ *which take Boolean values for any Boolean tuple* $\bar{u}$*, exactly one of them being non-zero, indicating the label of the gate* $\bar{u}$*. The polynomial computed by a succinct circuit is defined as it is for arithmetic circuits. The complexity of a succinct circuit is* $|E| + |L|$*.*

The circuits E and L in the definition above now perform a purely Boolean computation and can thus be Boolean circuits or arithmetic circuits. Note that to simplify notations we have not defined an order on the arguments of a computation gate: the operations we consider are commutative, and we define our circuits, whether succinct or not, as *simple* graphs, so that there can only be one edge from a gate to another. This also allows us to easily consider circuits with gates of unbounded in-degree.

Setting aside uniformity, which we have replaced by succinctness, we consider the result of Mahajan and Rao [13] characterizing VPSPACE by circuits of polynomial width and exponential depth. Computing sequentially the iterated matrix multiplication of Theorem 2, we show that we can actually go to circuits which have constant width and which are skew.

Corollary 4. *A sequence of polynomials belongs to* VPSPACE *iff it can be computed by a sequence of succinct skew circuits of constant width, exponential depth and polynomial complexity.*

This does not mean that for our model of computations polynomial space is in fact constant space: the arithmetic circuit which results from the above theorem does not take into account the space used by the defining Boolean computation. But it means that a polynomial space algebraic computation only needs a constant number of "algebraic" cells to store partial computations and a polynomial number of Boolean cells for auxiliary computations. Note that this result could also be shown from the definition of VPSPACE as containing polynomials with PSPACE coefficients, but had apparently not been observed before. This is related to the notion of serializable languages as defined in [6].

Computing iterated matrix multiplications in parallel yields a characterization of VPSPACE via succinct circuits of polynomial depth. This was shown using uniformity in [12]. We show that it holds without uniformity and also for circuits of unbounded fan-in.

Corollary 5. *A sequence of polynomials belongs to* VPSPACE *iff it can be computed by a sequence of succinct arithmetic terms of polynomial depth and complexity, iff it can be computed by a sequence of succinct unbounded fan-in arithmetic circuits of polynomial depth and complexity, iff it can be computed by a sequence of succinct skew circuits of polynomial complexity and exponential size.*

Finally, as in Sect. 3, we can consider complete problems. Using the result from [3], the polynomial associated to iterated multiplications of a polynomial number of matrices of constant size is easily seen to be $\mathrm{VP_e}$-complete, i.e. it captures one of the smallest classes in Valiant's framework. However, the complexity of the succinct version, which could be defined similarly to the examples of Corollaries 2 and 3, once again jumps all the way to VPSPACE.

Corollary 6. *The succinct constant-size iterated matrix multiplication polynomial is* VPSPACE*-complete.*

5 Branching Programs for VNP

In Valiant's framework, the classes $\mathrm{VP_e}$ and $\mathrm{VP_{ws}}$ already have a characterization via branching programs. In the previous sections we have given branching program characterizations for VPSPACE. It is not difficult to obtain a characterization for VNP.

Lemma 4. *If $f(\bar{x})$ is a polynomial which can be written as $\sum_{\bar{\epsilon}} g(\bar{x}, \bar{\epsilon})$, where $g(\bar{x}, \bar{y})$ is computed by an arithmetic term of size s, then f can be computed by a succinct layered algebraic branching program of width exponential in s and of length and complexity polynomial in s.*

Lemma 5. *If $f(\bar{x})$ is a polynomial which can be computed by a succinct iterated product of m matrices of complexity s, then there exists a polynomial $g(\bar{x}, \bar{y})$ of size and degree polynomial in ms such that $f(\bar{x}) = \sum_{\bar{\epsilon}} g(\bar{x}, \bar{\epsilon})$.*

Theorem 3. *A sequence of polynomials belongs to* VNP *iff it can be computed by a sequence of succinct layered algebraic branching programs of polynomial length and complexity (and possibly exponential width).*

As we did in Corollary 2, we could use this characterization to define a new VNP-complete polynomial. We could again use the two ways of computing a product of of matrices to get characterizations of VNP similar to ones given for NP in [22]. However, the fact that a sequence of polynomials in VNP can be computed by the kinds of circuits below is easily seen from the definition of the class and its equality with the class $\mathrm{VNP_e}$ (see for example the proof in [15]). The main point here is that the uniformity condition used in [15] to prove a similar charcterization for a uniform version of VNP can be replaced by succinctness to get the analogous result for the original non-uniform version of the class.

Corollary 7. *Let (f_n) be a sequence of polynomials. The following are equivalent:*

1. *(f_n) belongs to* VNP*,*
2. *(f_n) can be computed by a sequence of succinct arithmetic terms of polynomial depth and complexity and exponential size,*
3. *(f_n) can be computed by a sequence of succinct circuits of polynomial depth, degree and complexity, and exponential size,*
4. *(f_n) can be computed by a sequence of succinct semi-unbounded fan-in circuits of logarithmic depth, polynomial complexity and exponential size.*

6 Uniformity and Boolean Classes

We have used succinctness as a way to ensure that computational objects of potentially exponential size could be compactly represented, in order to characterize non-uniform complexity classes. Of course, we can add uniformity back, as all the transformations we have done can easily be seen to be computable in polynomial time by a Turing machine. Thus our results also apply to uniform versions of Valiant's complexity classes. Although the algebraic setting was the main focus of our work, we can also apply these techniques to Boolean classes. Because the uniform versions are more common we write the characterizations in this setting: uniformity based on the direct connection language replaces the succinctness requirement. We could also write similar characterizations for the non-uniform versions of these classes. In our mind, the main new characterization is with layered branching programs or equivalently iterated matrix multiplications: as we have seen, computing the product sequentially or in parallel implies characterizations with skew circuits and with parallelized semi-unbounded fan-in circuits such as those first proved in [22], which were shown using uniformity and configurations of Turing machines.

Consider Boolean circuits which operate on variables and their negations. If we interpret $+$ as $\vee$ and $\times$ as $\wedge$ and consider input gates labeled by literals, then NP is Uniform-VNP. We can thus use our characterizations of VNP to get a new characterization of NP. We will use a Boolean transposition of the defintion of algebraic branching programs: a *DAG-program* is "a layered acyclic graph G with edges labelled by constants (0 or 1) or literals, and with two special vertices s and t. It accepts an input x if there is an $s \rightsquigarrow t$ path where each edge is labelled by a true literal or the constant 1" [7].[3]

Corollary 8. *A language belongs to* NP *iff it has uniform DAG-programs of exponential width and polynomial length.*

We can also use the characterization of VNP for $\sharp$P. First note that a function f belongs to $\sharp$P iff it can be computed by a uniform sequence of multiplicatively disjoint arithmetic circuits with projections whose inputs are labeled by variables x or by $1 - x$ (this was shown first in [2] with a different notion to control multiplications). Theorem 3 then implies the following.

[3] These are also called "branching programs" in [7] but in the Boolean setting this clashes with a related but different notion.

Corollary 9. *A function from* $\{0,1\}^*$ *to* $\mathbb{N}$ *belongs to* $\sharp$P *iff it can be computed by uniform algebraic branching programs of exponential width and polynomial length, where the weight of an edge can be* 0, 1, x_i *or* $1-x_i$ *for any variable* x_i.

Finally we can characterize PSPACE. First note that a language L belongs to PSPACE iff it can be decided by a uniform sequence of Boolean circuits with projections with input gates labeled by literals (see [2] or [16]). The following characterization is already known [6,11,8], we repeat it to emphasize that our techniques work both in the Boolean and the algebraic case.

Corollary 10. *A language belongs to* PSPACE *iff it has uniform algebraic branching programs of width* 3 *and exponential length, where the weight of an edge can be* 0, 1, -1, x_i *or* $1-x_i$ *for any variable* x_i.

7 Conclusion

Adding our results to known characterizations of Valiant's complexity classes by branching programs, we get the following landscape: $\mathrm{VP_e}$ corresponds to branching programs of width 3 and polynomial length; $\mathrm{VP_{ws}}$ to polynomial width and length; VNP to exponential width and polynomial length; VPSPACE to width either 3 or polynomial and exponential length. Only one class is missing: VP. This is frustrating since VP is the natural class of feasible computations. However it is not a complete surprise, as VP is also the class for which we do not have any examples of natural complete polynomial sequences. We believe that our results, which place all of Valiant's classes in the context of branching programs except VP, help explain this special behaviour. Indeed it would be difficult to fit VP in this context. It is not hard to show that VP has branching programs of quasi-polynomial width and length, but a branching program of quasi-polynomial length can compute a polynomial of quasi-polynomial degree. Restricting the degree to being polynomially bounded is a semantic condition which would make it hard to use the resulting characterization to find complete polynomials, as opposed to the syntactic characterizations of the other classes.

More generally, whether for algebraic or Boolean classes, with or without uniformity, we think that these results further emphasize the importance of branching programs as a very simple computation model which can be seen behind the characterizations and properties of several classes.

Acknowledgements. The author thanks Stefan Mengel and Hervé Fournier for their helpful comments.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, New York (2009)
2. Babai, L., Fortnow, L.: Arithmetization: A New Method in Structural Complexity Theory. Comput. Complexity 1(1), 41–66 (1991)

3. Ben-Or, M., Cleve, R.: Computing Algebraic Formulas Using a Constant Number of Registers. SIAM J. Comput. 21(1), 54–58 (1992)
4. Borodin, A.: On relating time and space to size and depth. SIAM Journal on Computing 6(4), 733–744 (1977), http://link.aip.org/link/?SMJ/6/733/1
5. Bürgisser, P.: Completeness and Reduction in Algebraic Complexity Theory, Algorithms and Computation in Mathematics, vol. 7. Springer, Berlin (2000)
6. Cai, J.-y., Furst, M.L.: Pspace survives constant-width bottlenecks. Int. J. Found. Comput. Sci. 2(1), 67–76 (1991)
7. Datta, S., Mahajan, M., Rao, B.V.R., Thomas, M., Vollmer, H.: Counting Classes and the Fine Structure between NC^1 and L. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 306–317. Springer, Heidelberg (2010)
8. Galota, M., Vollmer, H.: Functions computable in polynomial space. Information and Computation 198(1), 56 (2005)
9. Galperin, H., Wigderson, A.: Succinct representations of graphs. Information and Control 56(3), 183 (1983)
10. von zur Gathen, J.: Feasible Arithmetic Computations: Valiant's Hypothesis. J. Symb. Comput. 4(2), 137–172 (1987)
11. Hertrampf, U., Lautemann, C., Schwentick, T., Vollmer, H., Wagner, K.W.: On the power of polynomial time bit-reductions (extended abstract). In: Structure in Complexity Theory Conference, pp. 200–207 (1993)
12. Koiran, P., Perifel, S.: Vpspace and a transfer theorem over the reals. Computational Complexity 18(4), 551–575 (2009)
13. Mahajan, M., Rao, B.V.R.: Small-space analogues of valiant's classes. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 250–261. Springer, Heidelberg (2009)
14. Malod, G.: The complexity of polynomials and their coefficient functions. In: IEEE Conference on Computational Complexity, pp. 193–204. IEEE Computer Society, Los Alamitos (2007)
15. Malod, G., Portier, N.: Characterizing valiant's algebraic complexity classes. J. Complex. 24(1), 16–38 (2008)
16. Poizat, B.: A la recherche de la definition de la complexite d'espace pour le calcul des polynomes a la maniere de valiant. J. Symb. Log. 73(4), 1179–1201 (2008)
17. Ruzzo, W.L.: On uniform circuit complexity. Journal of Computer and System Sciences 22(3), 365–383 (1981)
18. Valiant, L.G.: Completeness Classes in Algebra. In: STOC 1979: Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, pp. 249–261. ACM Press, New York (1979)
19. Valiant, L.G.: Reducibility by Algebraic Projections. In: Logic and Algorithmic: an International Symposium held in honor of Ernst Specker. Monographies de l'Enseignement Mathémathique, vol. 30, pp. 365–380 (1982)
20. Valiant, L.G., Skyum, S., Berkowitz, S., Rackoff, C.: Fast Parallel Computation of Polynomials Using Few Processors. SIAM J. Comput. 12(4), 641–644 (1983)
21. Veith, H.: Succinct representation, leaf languages, and projection reductions, Information and Computation 142(2), 207 (1998)
22. Venkateswaran, H.: Circuit Definitions of Nondeterministic Complexity Classes. SIAM J. Comput. 21(4), 655–670 (1992)

LIFO-Search on Digraphs: A Searching Game for Cycle-Rank[⋆]

Paul Hunter[⋆⋆]

Department of Computer Science, University of Oxford
paul.hunter@cs.ox.ac.uk

Abstract. We consider the extension of the last-in-first-out graph searching game of Giannopoulou and Thilikos to digraphs. We show that all common variations of the game require the same number of searchers, and the minimal number of searchers required is one more than the cycle-rank of the digraph. We also obtain a tight duality theorem, giving a precise min-max characterization of obstructions for cycle-rank.

1 Introduction

Graph searching games are increasingly becoming a popular way to characterize, and even define, practical graph parameters. There are many advantages to a characterization by graph searching games: it provides a useful intuition which can assist in constructing more general or more specific parameters; it gives insights into relations with other, similarly characterized parameters; and it is particularly useful from an algorithmic perspective as many parameters associated with such games are both structurally robust and efficiently computable.

One of the most common graph searching games is the node-search game. In this game several searchers and one fugitive occupy vertices of the graph and make simultaneous moves. The (omniscient) fugitive moves along searcher-free paths of arbitrary length whereas the searchers' movements are not constrained by the topology of the graph. The goal of the game is to minimize the number of searchers required to capture the fugitive by cornering him in some part of the graph and placing a searcher on the same vertex. This game has been extensively studied [5] and several important graph parameters such as treewidth [18], pathwidth [12], and tree-depth [15] can be characterized by natural variants of this game. One variation frequently used, indeed the one which separates treewidth and pathwidth, is whether the location of the fugitive is known or unknown to the searchers. Another common variation is whether the searchers use a monotone or a non-monotone searching strategy. Monotone search strategies lead to algorithmically useful decompositions, whereas non-monotone strategies are more robust under graph operations and hence reflect structural properties, so

[⋆] This work resulted from discussions during Dagstuhl Seminar 11071 on Graph Searching, Theory and Applications.

[⋆⋆] Supported by Model Checking Real-Time Systems project, EPSRC grant ref. BLRQEK.

showing that monotone strategies require no more searchers than non-monotone strategies is an important and common question in the area. Whilst node-search games on undirected graphs tend to enjoy monotonicity [4,18,14], on digraphs the situation is much less clear [2,1,13].

Node-search games naturally extend to digraphs, however, in the translation another variation arises depending on how one views the constraints on the movement of the fugitive. One interpretation is that in the undirected case the fugitive moves along paths, so the natural translation would be to have the fugitive move along directed paths. Another view is that the fugitive moves to some other vertex in the same connected component, and here the natural translation would be to have the fugitive move within the same strongly connected component. Both interpretations have been studied in the literature, the former giving characterizations of parameters such as DAG-width [3,16] and directed pathwidth [2] and the latter giving a characterization of directed treewidth [11].

In [9], Giannopoulou and Thilikos define a variant of the node-search game in which only the most recently placed searchers may be removed; that is, the searchers must move in a last-in-first-out (LIFO) manner. They show that the visibility of the fugitive is not relevant to the minimum number of searchers required, the game is monotone, and that it characterizes tree-depth. In this paper we consider the extension of this game to digraphs.

We generalize the results of Giannopoulou and Thilikos by showing that the minimum number of searchers required to capture a fugitive on a digraph with a LIFO-search is independent of:

– Whether the fugitive is invisible or visible,
– Whether the searchers use a monotone or non-monotone search, and
– Whether the fugitive is restricted to moving in searcher-free strongly connected sets or along searcher-free directed paths.

This result is somewhat surprising: in the standard node-search game these options give rise to quite different parameters [2,3,13].

We show that on digraphs the LIFO-search game also characterizes a pre-existing measure, cycle-rank – a generalization of tree-depth to digraphs (though as the definition of cycle-rank predates tree-depth by several decades, it is perhaps more correct to say that tree-depth is an analogue of cycle-rank on undirected graphs). The cycle-rank of a digraph is an important parameter relating digraph complexity to other areas such as regular language complexity and asymmetric matrix factorization. It was defined by Eggan [6], where it was shown to be a critical parameter for determining the star-height of regular languages, and interest in it as an important digraph parameter, especially from an algorithmic perspective, has recently been rekindled by the success of tree-depth [7,10,8].

It is well known that tree-depth can also be characterized by a node-search game where a visible fugitive plays against searchers that are only placed and never moved [8]. In that paper, Ganian et al. considered one extension of this game to digraphs. Here we consider the other natural extension, where the visible fugitive moves in strongly connected sets, and show that it also characterizes cycle-rank.

Our final result uses these graph searching characterizations to define a dual parameter that characterizes structural obstructions for cycle-rank. We consider two obstructions, motivated by the shelters of [9] and the havens of [11], that define simplified strategies for the fugitive. The game characterization then implies that these structural features are necessarily present when the cycle-rank of a graph is large. By showing that such strategies are also sufficient for the fugitive, we obtain a rare instance of an exact min-max theorem relating digraph parameters.

The results of this paper can be summarized with the following characterizations of cycle-rank.

Main Theorem. *Let G be a digraph, and k a positive integer. The following are equivalent:*

(i) *G has cycle-rank $\leq k-1$,*
(ii) *On G, k searchers can capture a fugitive with a LIFO-search strategy,*
(iii) *On G, k searchers can capture a visible fugitive restricted to moving in strongly connected sets with a searcher-stationary search strategy,*
(iv) *G has no LIFO-haven of order $> k$, and*
(v) *G has no strong shelter of thickness $> k$.*

The paper is organised as follows. In Section 2 we recall the definitions and notation that we use throughout the paper. In Section 3 we define the LIFO-search and searcher-stationary games and show that they characterize cycle-rank. In Section 4 we prove the min-max theorem for cycle-rank, and in Section 5 we conclude with a discussion on further research and open problems.

2 Preliminaries

All (di)graphs in this paper are finite, simple, directed and without self-loops, although the results readily extend to multigraphs with self-loops. For simplicity, we also assume that all digraphs contain at least one vertex unless explicitly mentioned. We use standard notation and terminology, in particular $V(G)$ and $E(G)$ denote the sets of vertices and edges respectively of a digraph G and between digraphs, $\subseteq$ denotes the subgraph relation. We will often interchange an induced subgraph with the set of vertices which defines it, in particular strongly connected sets of vertices are sets of vertices that induce a strongly connected subgraph, and we will often view strongly connected components as sets of vertices. Given a digraph G and a set of vertices $X \subseteq V(G)$, we use $G \setminus X$ to denote the subgraph of G induced by $V(G) \setminus X$. An *initial component* of a digraph G is a strongly connected component C with no edges from $G \setminus C$ to C. $H \subseteq G$ is *successor-closed* if there are no edges in G from H to $G \setminus H$.

Given a finite set V, we use V^* to denote the set of finite words over V, and $V^{<k}$ to denote the set of words over V of length $< k$. We use ϵ to denote the empty word and $\cdot$ or juxtaposition to denote concatenation. For $X, Y \in V^*$ we write $X \preceq Y$ if X is a prefix of Y, that is if there exists a word $Z \in V^*$ such that $Y = X \cdot Z$. For $X = a_1 a_2 \cdots a_n \in V^*$, we use $|X|$ to denote the length of X, and

$\{\!|X|\!\}$ to denote the set $\{a_1, a_2, \ldots, a_n\}$. Given two sets A and B we use $A\Delta B$ to denote their symmetric difference, that is $A\Delta B = (A \cup B) \setminus (A \cap B)$. Given a set $\mathcal{S} \subseteq \mathcal{P}(V)$ of subsets of V, a *$\subseteq$-chain* is a subset $\{X_1, \ldots, X_n\} \subseteq \mathcal{S}$ such that $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_n$. If there is no $Y \in \mathcal{S}$ such that $Y \subset X_1$, $X_i \subset Y \subset X_{i+1}$ for some i, or $X_n \subset Y$, then $\{X_1, \ldots, X_n\}$ is a *maximal $\subseteq$-chain.*

The *cycle-rank* of a digraph G, $\mathrm{cr}(G)$, is defined as follows:

– If G is acyclic then $\mathrm{cr}(G) = 0$.
– If G is strongly connected then $\mathrm{cr}(G) = 1 + \min_{v \in V(G)} \mathrm{cr}(G \setminus \{v\})$.
– Otherwise $\mathrm{cr}(G) = \max_H \mathrm{cr}(H)$ where the maximum is taken over all strongly connected components H of G.

3 Searching Games for Cycle-Rank

We begin by formally defining the LIFO-search game, and its variants, for digraphs. Each variation of the LIFO-search game gives rise to a digraph parameter corresponding to the minimum number of searchers required to capture the fugitive under the given restrictions. The main result of this section is that for any digraph all these parameters are equal. Furthermore, we show they are all equal to one more than the cycle-rank of the digraph.

3.1 LIFO-Search for Digraphs

In summary, for the graph searching game in which we are interested the fugitive can run along searcher-free directed paths of any length, the searchers can move to any vertex in the graph, and the fugitive moves whilst the searchers are relocating. The only restriction we place on the searchers is that only the most recently placed searchers may be removed. If a searcher is placed on the fugitive then he is captured and the searchers win, otherwise the fugitive wins. The goal is to determine the minimum number of searchers required to capture the fugitive. For simplicity we assume that each searcher move consists of either placing or removing one searcher and observe that this does not affect the minimum number of searchers required to capture the fugitive. The variants we are primarily interested in are whether the searchers use a monotone or a non-monotone strategy, whether the fugitive is visible or invisible, and whether or not the fugitive must stay within the same strongly connected component when he is moving. As our fundamental definitions are dependent on these latter two options, we define four *game variants*: `i`, `isc`, `v`, `vsc`, corresponding to the visibility of the fugitive and whether he is constrained to moving within strongly connected components, and parameterize our definitions by these variants.

Let us fix a digraph G. A position in a LIFO-search on G is a pair (X, R) where $X \in V(G)^*$ and R is a (possibly empty) induced subgraph of $G \setminus \{\!|X|\!\}$. Intuitively X represents the position and ordered placement of the searchers and R represents the part of G that the fugitive can reach (in the visible case) or the set of vertices where he might possibly be located (in the invisible case). We say a position (X, R) is an *`i`-position* if R is successor-closed; an *`isc`-position* if it is a union of strongly connected components of $G \setminus \{\!|X|\!\}$; a *$\mathtt{v}$-position* if R is successor-closed and has a unique initial component; and a *$\mathtt{vsc}$-position* if R is a strongly connected component of $G \setminus \{\!|X|\!\}$.

To reflect how the game transitions to a new position during a round of the game we say, for $\mathbf{gv} \in \{\mathtt{i}, \mathtt{isc}, \mathtt{v}, \mathtt{vsc}\}$, a $\mathbf{gv}$-position (X', R') is a $\mathbf{gv}$*-successor* of (X, R) if either $X \preceq X'$ or $X' \preceq X$, with $|\{\!|X|\!\} \Delta \{\!|X'|\!\}| = 1$, and

– (for $\mathbf{gv} \in \{\mathtt{i}, \mathtt{v}\}$) For every $v' \in V(R')$ there is a $v \in V(R)$ and a directed path in $G \setminus (\{\!|X|\!\} \cap \{\!|X'|\!\})$ from v to v', or
– (for $\mathbf{gv} \in \{\mathtt{isc}, \mathtt{vsc}\}$) For every $v' \in V(R')$ there is a $v \in V(R)$ such that v and v' are contained in the same strongly connected component of $G \setminus (\{\!|X|\!\} \cap \{\!|X'|\!\})$.

Ideally we would like to assume games start from (ϵ, G), however in the visible variants of the game this might not be a legitimate position. Thus, for $\mathbf{gv} \in \{\mathtt{v}, \mathtt{vsc}\}$, if (ϵ, G) is not a $\mathbf{gv}$-position we include it as a special case, and set as its $\mathbf{gv}$-successors all $\mathbf{gv}$-positions of the form (ϵ, R). We observe that in all variants, the successor relation is monotone in the sense that if (X, R) and (X, S) are positions with $S \subseteq R$ and (X', S') is a successor of (X, S), then there is a successor (X', R') of (X, R) with $S' \subseteq R'$.

For $\mathbf{gv} \in \{\mathtt{i}, \mathtt{isc}, \mathtt{v}, \mathtt{vsc}\}$, a *(*$\mathbf{gv}$*-LIFO-)search* in a digraph G from $\mathbf{gv}$-position (X, R) is a (finite or infinite) sequence of $\mathbf{gv}$-positions $(X, R) = (X_0, R_0)$, $(X_1, R_1), \ldots$ where for all $i \geq 0$, (X_{i+1}, R_{i+1}) is a $\mathbf{gv}$-successor of (X_i, R_i). A LIFO-search is *complete* if either $R_n = \emptyset$ for some n, or it is infinite. We observe that if $R_n = \emptyset$, then $R_{n'} = \emptyset$ for all $n' \geq n$.

We say a complete LIFO-search is *winning for the searchers* if $R_n = \emptyset$ for some n, otherwise it is winning for the fugitive. A complete LIFO-search from (ϵ, G) is *monotone* if $R_{i+1} \subseteq R_i$ for all i; it is *searcher-stationary* if $X_i \preceq X_{i+1}$ for all i where $R_i \neq \emptyset$; and it *uses at most k searchers* if $|X_i| \leq k$ for all i.

Whilst a complete LIFO-search from (ϵ, G) describes a single run of the game, we are more interested in the cases where one of the players (particularly the searchers) can always force a win, no matter what the other player chooses to do. For this, we introduce the notion of a strategy. For $\mathbf{gv} \in \{\mathtt{i}, \mathtt{isc}, \mathtt{v}, \mathtt{vsc}\}$, a *(searcher)* $\mathbf{gv}$*-strategy* is a (partial[1]) function σ from the set of all $\mathbf{gv}$-positions to $V(G)^*$ such that for all (X, R), $\sigma(X, R)$ is the first component of a $\mathbf{gv}$-successor of (X, R); so with the possible exception of $(X, R) = (\epsilon, G)$, either $\sigma(X, R) \preceq X$ or $X \preceq \sigma(X, R)$. A $\mathbf{gv}$-LIFO-search $(X_0, R_0), (X_1, R_1), \ldots$ is *consistent* with a $\mathbf{gv}$-strategy σ if $X_{i+1} = \sigma(X_i, R_i)$ for all $i \geq 0$. A strategy σ is winning from (X, R) if all complete LIFO-searches from (X, R) consistent with σ are winning for the searchers. Likewise, a strategy is monotone (searcher-stationary, uses at most k searchers) if all consistent complete LIFO-searches from (ϵ, G) are monotone (searcher-stationary, use at most k searchers respectively). We say k searchers can capture a fugitive on G in the $\mathbf{gv}$-game with a (monotone)

[1] A strategy need only be defined for all positions (X, R) that can be reached from (ϵ, G) in a LIFO-search consistent with the strategy. However, as this definition is somewhat circular, we assume strategies are total.

LIFO-search strategy if there is a (monotone) **gv**-strategy that uses at most k searchers and is winning from (ϵ, G).

For **gv** $\in \{\mathtt{i}, \mathtt{isc}, \mathtt{v}, \mathtt{vsc}\}$, we define the (monotone) **gv**-LIFO-search number of G, $\mathrm{LIFO}^{\mathbf{gv}}(G)$ $(\mathrm{LIFO}^{\mathbf{mgv}}(G))$, as the minimum k for which there is a (monotone) winning **gv**-strategy that uses at most k searchers. We also define the visible, strongly connected, searcher-stationary search number of G, $\mathrm{SS}^{\mathtt{vsc}}(G)$ as the minimum k for which there is a searcher-stationary winning vsc-strategy that uses at most k searchers.

In Section 4 we will also consider fugitive **gv**-strategies: a partial function ρ from $V(G)^* \times \mathcal{P}(G) \times V(G)^*$ to induced subgraphs of G, defined for (X, R, X') if (X, R) is a **gv**-position and X' is the first component of a **gv**-successor of (X, R). A LIFO-search $(X_0, R_0), (X_1, R_1), \ldots$ is *consistent* with a fugitive **gv**-strategy ρ if $R_{i+1} = \rho(X_i, R_i, X_{i+1})$ for all $i \geq 0$, and a fugitive strategy is winning if all consistent complete LIFO-searches are winning for the fugitive. In this section, a strategy will always refer to a searcher strategy.

3.2 Relating the Digraph Searching Parameters

We observe that in all game variants, a strategy that is winning from (X, R) can be used to define a strategy that is winning from (X, R') for any $R' \subseteq R$: the searchers can play as if the fugitive is located in the larger space; and from the monotonicity of the successor relation, the assumption that the actual set of locations of the fugitive is a subset of the assumed set of locations remains invariant. One consequence is that a winning strategy on G defines a winning strategy on any subgraph of G, so the search numbers we have defined are monotone with respect to the subgraph relation.

Proposition 1. *Let G be a digraph and G' a subgraph of G. Then:*

- *$SS^{vsc}(G') \leq SS^{vsc}(G)$, and*
- *$LIFO^{\mathbf{gv}}(G') \leq LIFO^{\mathbf{gv}}(G)$ for **gv** $\in \{i, isc, v, vsc, mi, misc, mv, mvsc\}$.*

Another consequence is that a winning strategy in the invisible fugitive variant defines a winning strategy when the fugitive is visible; and a winning strategy when the fugitive is not constrained to moving within strongly connected components defines a winning strategy when he is. This corresponds to our intuition of the fugitive being more (or less) restricted. Also, in all game variants, a monotone winning strategy is clearly a winning strategy, and because a searcher-stationary LIFO-search is monotone, a winning searcher-stationary strategy is a monotone winning strategy. These observations yield several inequalities between the search numbers defined above. For example $\mathrm{LIFO}^{\mathtt{vsc}}(G) \leq \mathrm{LIFO}^{\mathtt{mi}}(G)$ as any winning monotone i-strategy is also a winning vsc-strategy. The full set of these relationships is shown in a Hasse diagram in Figure 1, with the larger measures towards the top.

The main result of this section is that all these digraph parameters are equal to one more than cycle-rank.

$$\begin{tikzcd}
 & \mathrm{LIFO}^{\mathtt{mi}}(G) \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & & \\
\mathrm{LIFO}^{\mathtt{i}}(G) \arrow[d, dash] \arrow[dr, dash] & \mathrm{LIFO}^{\mathtt{mv}}(G) \arrow[dl, dash] \arrow[dr, dash] & \mathrm{LIFO}^{\mathtt{misc}}(G) \arrow[dl, dash] \arrow[d, dash] & \mathrm{SS}^{\mathtt{vsc}}(G) \arrow[dl, dash] \\
\mathrm{LIFO}^{\mathtt{v}}(G) \arrow[dr, dash] & \mathrm{LIFO}^{\mathtt{isc}}(G) \arrow[d, dash] & \mathrm{LIFO}^{\mathtt{mvsc}}(G) \arrow[dl, dash] & \\
 & \mathrm{LIFO}^{\mathtt{vsc}}(G) & &
\end{tikzcd}$$

Fig. 1. Trivial relations between digraph searching parameters

Theorem 1. *For any digraph G:*

$$\begin{aligned}1 + cr(G) &= LIFO^{\mathtt{mi}}(G) = LIFO^{\mathtt{i}}(G) = LIFO^{\mathtt{misc}}(G) = LIFO^{\mathtt{isc}}(G)\\ &= LIFO^{\mathtt{mv}}(G) = LIFO^{\mathtt{v}}(G) = LIFO^{\mathtt{mvsc}}(G) = LIFO^{\mathtt{vsc}}(G)\\ &= SS^{\mathtt{vsc}}(G).\end{aligned}$$

Proof. From the above observations, to prove Theorem 1 it is sufficient to prove the following three inequalities:

(1) $\mathrm{LIFO}^{\mathtt{vsc}}(G) \geq \mathrm{SS}^{\mathtt{vsc}}(G)$,
(2) $\mathrm{SS}^{\mathtt{vsc}}(G) \geq 1 + \mathrm{cr}(G)$, and
(3) $1 + \mathrm{cr}(G) \geq \mathrm{LIFO}^{\mathtt{mi}}(G)$.

These are established with the following series of lemmas.

Lemma 1. *For any digraph G, $LIFO^{\mathtt{vsc}}(G) \geq SS^{\mathtt{vsc}}(G)$.*

Proof. We show that if a `vsc`-strategy is not searcher-stationary then it is not a winning strategy from (ϵ, G). The result then follows as this implies every winning `vsc`-strategy is searcher-stationary. Let σ be a `vsc`-strategy, and suppose $(X_0, R_0), (X_1, R_1), \dots$ is a complete `vsc`-LIFO-search from $(X_0, R_0) = (\epsilon, G)$ consistent with σ which is not searcher-stationary. Let j be the least index such that $X_j \succeq X_{j+1}$ and $R_j \neq \emptyset$. As $X_0 = \epsilon$, there exists $i < j$ such that $X_i = X_{j+1}$. By the minimality of j, and the assumption that we only place or remove one searcher in each round, $i = j - 1$. As $X_{j-1} \preceq X_j$, $R_j \subseteq R_{j-1}$, and as $X_{j+1} \preceq X_j$, $R_j \subseteq R_{j+1}$. As $R_j \neq \emptyset$, it follows that R_{j-1} and R_{j+1} are the same strongly connected component of $G \setminus \{\!|X_{j-1}|\!\}$. Thus (X_{j-1}, R_{j-1}) is a `vsc`-successor of (X_j, R_j). As $\sigma(X_j, R_j) = X_{j+1} = X_{j-1}$, it follows that $(X_0, R_0), (X_1, R_1), \dots (X_{j-1}, R_{j-1}), (X_j, R_j), (X_{j-1}, R_{j-1}), (X_j, R_j), \dots$ is an infinite, and hence complete, `vsc`-LIFO-search (from (ϵ, G)) consistent with σ. As $R_i \neq \emptyset$ for all $i \geq 0$, the LIFO-search is not winning for the searchers. Thus σ is not a winning strategy.

Lemma 2. *For any digraph G, $SS^{\mathtt{vsc}}(G) \geq 1 + cr(G)$.*

Proof. We prove this by induction on $|V(G)|$.

If $|V(G)| = 1$, then $\mathrm{SS}^{\mathtt{vsc}}(G) = 1 = 1 + \mathrm{cr}(G)$.

Now suppose $\mathrm{SS}^{\mathtt{vsc}}(G') \geq 1 + \mathrm{cr}(G')$ for all digraphs G' with $|V(G')| < |V(G)|$. We first consider the case when G is not strongly connected. From Proposition 1, $\mathrm{SS}^{\mathtt{vsc}}(G) \geq \max_H \mathrm{SS}^{\mathtt{vsc}}(H)$ where the maximum is taken over all strongly connected components H of G. As G is not strongly connected, $|V(H)| < |V(G)|$ for all strongly connected components H of G. Therefore, by the induction hypothesis

$$\begin{aligned}\mathrm{SS}^{\mathtt{vsc}}(G) &\geq \max_H \mathrm{SS}^{\mathtt{vsc}}(H)\\ &\geq \max_H (1 + \mathrm{cr}(H))\\ &= 1 + \mathrm{cr}(G).\end{aligned}$$

Now suppose G is strongly connected. Let σ be a winning searcher-stationary $\mathtt{vsc}$-strategy which uses $\mathrm{SS}^{\mathtt{vsc}}(G)$ searchers. As (ϵ, G) is a legitimate $\mathtt{vsc}$-position, if (X, R) is a $\mathtt{vsc}$-successor of (ϵ, G) then $|X| = 1$. Thus $|\sigma(\epsilon, G)| = 1$. Let $\sigma(\epsilon, G) = v_0$. As σ is a searcher-stationary strategy which uses the minimal number of searchers, it follows that $\mathrm{SS}^{\mathtt{vsc}}(G \setminus \{v_0\}) = \mathrm{SS}^{\mathtt{vsc}}(G) - 1$. Thus, by the induction hypothesis,

$$\begin{aligned}\mathrm{SS}^{\mathtt{vsc}}(G) &= \mathrm{SS}^{\mathtt{vsc}}(G \setminus \{v_0\}) + 1\\ &\geq (1 + \mathrm{cr}(G \setminus \{v_0\})) + 1\\ &\geq (1 + \min_{v \in V(G)} \mathrm{cr}(G \setminus \{v\})) + 1\\ &= 1 + \mathrm{cr}(G).\end{aligned}$$

Lemma 3. *For any digraph G, $1 + cr(G) \geq LIFO^{\mathtt{mi}}(G)$.*

Proof. We also prove this by induction on $|V(G)|$.

If $|V(G)| = 1$, then $1 + \mathrm{cr}(G) = 1 = \mathrm{LIFO}^{\mathtt{mi}}(G)$.

Now suppose $1 + \mathrm{cr}(G') \geq \mathrm{LIFO}^{\mathtt{mi}}(G')$ for all digraphs G' with $|V(G')| < |V(G)|$. First we consider the case when G is not strongly connected. As $|V(H)| < |V(G)|$ for each strongly connected component H, by the inductive hypothesis, there is a monotone $\mathtt{i}$-strategy, σ_H, which captures a fugitive using at most $1 + \mathrm{cr}(H)$ searchers. From the definition of cycle-rank, for each strongly connected component H of G, $\mathrm{cr}(G) \geq \mathrm{cr}(H)$, thus σ_H uses at most $1 + \mathrm{cr}(G)$ searchers. We define a monotone $\mathtt{i}$-strategy which captures a fugitive on G with at most $1 + \mathrm{cr}(G)$ searchers as follows. Intuitively, we search the strongly connected components of G in topological order using the monotone strategies σ_H. More precisely, let $H_1, H_2, \ldots, H_n$ be an ordering of the strongly connected components of G such that if there is an edge from H_i to H_j then $i < j$. We define σ as follows.

– $\sigma(\epsilon, G) = \sigma_{H_1}(\epsilon, H_1)$,

– For $1 \leq i$, if $\{\!|X|\!\} \subseteq H_i$ and $R = R' \cup \bigcup_{j=i+1}^{n} H_j$ where $\emptyset \neq R' \subseteq H_i$, $\sigma(X,R) = \sigma_{H_i}(X,R')$,
– For $1 \leq i < n$, if $\emptyset \neq \{\!|X|\!\} \subseteq H_i$ and $R = \bigcup_{j=i+1}^{n} H_j$ then $\sigma(X,R) = X'$ where X' is the maximal proper prefix of X.

From the definition of `i`-successors and the ordering of the strongly connected components if $(X_0, R_0), \ldots (X_n, R_n)$ is an `i`-LIFO-search on G where $\{\!|X_n|\!\} \subseteq H_i$ and $\bigcup_{j>i} H_j \subseteq R_{n-1} \subseteq \bigcup_{j\geq i} H_j$, then $\bigcup_{j>i} H_j \subseteq R_n \subseteq \bigcup_{j\geq i} H_j$. It follows (by induction on the length of a LIFO-search) that every LIFO-search from (ϵ, G) consistent with σ can be divided into a sequence of LIFO-searches $\lambda_1, \lambda_2, \ldots, \lambda_n$, where λ_i can be viewed as a LIFO-search consistent with σ_{H_i} with $\bigcup_{j>i} H_j$ added to the second component of every position. Thus if each σ_{H_i} is monotone, winning and uses at most $1 + \mathrm{cr}(G)$ searchers, then σ is also monotone, winning and uses at most $1 + \mathrm{cr}(G)$ searchers.

Now suppose G is strongly connected. Let v_0 be the vertex which minimizes $f(v) = \mathrm{cr}(G \setminus \{v\})$. Let $G' = G \setminus \{v_0\}$, so $\mathrm{cr}(G) = 1 + \mathrm{cr}(G')$. By the induction hypothesis, there exists a winning monotone `i`-strategy σ' which uses at most $1 + \mathrm{cr}(G')$ searchers to capture a fugitive on G'. We define an `i`-strategy σ on G which uses at most $2 + \mathrm{cr}(G') = 1 + \mathrm{cr}(G)$ searchers as follows. Initially, place (and keep) a searcher on v_0, then play the strategy σ' on $G \setminus \{v_0\}$. More precisely, $\sigma(\epsilon, G) = v_0$ and $\sigma(v_0 X, R) = v_0 \cdot \sigma'(X, R)$. Clearly any LIFO-search consistent with σ can be viewed as a LIFO-search consistent with σ' prepended with the position (ϵ, G) and where the first component of every position is prepended with v_0. Thus if σ' is monotone, then σ is monotone, and if σ' is winning then σ is winning. Thus σ is a monotone winning `i`-strategy which uses at most $1 + \mathrm{cr}(G)$ searchers.

3.3 Relation with Other Graph Parameters

With a characterization of cycle-rank in terms of several graph searching games we can compare it with other digraph measures defined by similar games. In particular, the directed pathwidth of a digraph, $\mathrm{dpw}(G)$, which can be characterized by an invisble-fugitive graph searching game [2], and the DAG-depth, $\mathrm{dd}(G)$ which can be characterized by a visible-fugitive, searcher-stationary searching game [8]. Whilst the relationships we present here are known [10,8], using the game characterizations we obtain a more simple and more intuitive proof.

Corollary 1. *For any digraph G, $dpw(G) \leq cr(G) \leq dd(G) - 1$.*

4 Obstructions for Cycle-Rank

In this section we consider the dual parameter arising from considering the graph searching games from the fugitive's perspective. We show that it can be characterized by two types of structural features, akin to the havens and brambles used to dually characterize treewidth [18]. The first of these is the natural generalization of a shelter from [9], a structural obstruction shown to be dual to tree-depth.

Definition 1. *A* strong shelter *of a digraph G is a collection $\mathcal{S}$ of non-empty strongly connected sets of vertices such that for any $S \in \mathcal{S}$*

$$\bigcap\{S' : S' \in M_{\mathcal{S}}(S)\} = \emptyset,$$

where $M_{\mathcal{S}}(S)$ is the $\subseteq$-maximal elements of $\{S' \in \mathcal{S} : S' \subset S\}$. The thickness *of a shelter $\mathcal{S}$ is the minimal length of a maximal $\subseteq$-chain.*

The second structural obstruction we consider is motivated by the definition of a haven in [11], a structural feature dual to directed treewidth.

Definition 2. *A* LIFO-haven of order *k is a function h from $V(G)^{<k}$ to induced subgraphs of G such that:*

(H1) $h(X)$ is a non-empty strongly connected component of $G \setminus \{\!|X|\!\}$, and
(H2) If $X \preceq Y$ and $|Y| < k$ then $h(Y) \subseteq h(X)$.

Whilst Adler [1] has shown that the havens of [11] do not give an exact min-max characterization of directed treewidth and Safari [17] has shown that directed versions of havens and brambles give rise to distinct parameters, we show that LIFO-havens and strong shelters both give a tight min-max characterization of cycle-rank.

Theorem 2 (Min-max theorem for cycle-rank). *Let G be a digraph and k a positive integer. The following are equivalent:*

(i) G has cycle-rank $< k$,
(ii) G has no LIFO-haven of order $> k$, and
(iii) G has no strong shelter of thickness $> k$.

Proof. (i) $\Rightarrow$ (ii). Assume that it is not the case that G has no LIFO-haven of order $> k$, that is, G has a LIFO-haven h of order $k+1$. We show the fugitive has a winning strategy against k searchers, so by Theorem 1, $\mathrm{cr}(G) \geq k$. Define a $\mathtt{vsc}$-strategy ρ for the fugitive (against k searchers) by defining $\rho(X, R, X') = h(X')$ for all suitable triples (X, R, X'). From (H1), $(X', \rho(X, R, X'))$ is a valid $\mathtt{vsc}$-position. Furthermore, (H2) implies that if (X, R) is a $\mathtt{vsc}$-position such that $R = h(X)$, then $(X', \rho(X, R, X'))$ is a $\mathtt{vsc}$-successor of (X, R), so ρ is a $\mathtt{vsc}$-strategy (defined for all LIFO-searches that use at most k searchers). Also, if $(X_0, R_0), (X_1, R_1) \ldots$ is a complete LIFO-search consistent with ρ then $R_i = h(X_i)$ for all $i > 0$. As $h(X) \neq \emptyset$ when $|X| \leq k$, it follows that all consistent complete LIFO-searches that use at most k searchers are winning for the fugitive. Thus ρ is a winning strategy for the fugitive, so $\mathrm{LIFO}^{\mathtt{vsc}}(G) > k$. By Theorem 1, $\mathrm{cr}(G) \geq k$.

(ii) $\Rightarrow$ (iii). We show that a strong shelter $\mathcal{S}$ of thickness k can be used to define a haven of order k. For each $X \in V(G)^{<k}$ we define $S_X \in \mathcal{S}$ inductively as follows. For $X = \epsilon$, let S_ϵ be any $\subseteq$-maximal element of $\mathcal{S}$. Note that $\{S \in \mathcal{S} : S \subset S_\epsilon\}$ is a strong shelter of thickness $k-1$. Now suppose $X = X'v$, $S_{X'}$ is defined, $S_{X'} \cap \{\!|X'|\!\} = \emptyset$, and $\mathcal{S}_{X'} = \{S \in \mathcal{S} : S \subset S_{X'}\}$ is a strong shelter

of thickness $k - 1 - |X'|$. From the definition of a strong shelter, there exists a $\subseteq$-maximal element of $\mathcal{S}_{X'}$ that does not contain v, as otherwise $v \in S$ for all $S \in M_{\mathcal{S}}(S_{X'})$. Let S_X be that element. As $S_{X'} \cap \{\!|X'|\!\} = \emptyset$ and $v \notin S_X$, it follows that $S_X \cap \{\!|X|\!\} = \emptyset$. Further, $\{S \in \mathcal{S} : S \subset S_X\}$ is a strong shelter of thickness $(k - 1 - |X'|) - 1 = k - 1 - |X|$, satisfying the assumptions necessary for the next stage of the induction. Now for all $X \in V(G)^{<k}$, S_X is a non-empty strongly connected set such that $S_X \cap \{\!|X|\!\} = \emptyset$. Thus there is a unique strongly connected component of $G \setminus \{\!|X|\!\}$ that contains S_X. Defining $h(X)$ to be that component we see that h satisfies (H1). For (H2), from the definition of S_X, if $X \preceq Y$ and $|Y| < k$, then $S_X \supseteq S_Y$, so $h(X) \supseteq h(Y)$. Therefore h is a haven of order k.

(iii) $\Rightarrow$ (i). Again, we prove the contrapositive, using a proof similar to [9]. Suppose $\mathrm{cr}(G) \geq k$. Let G' be a strongly connected component of G which has cycle-rank $\geq k$. We prove by induction on k that G', and hence G, has a strong shelter of thickness $k + 1$. Every digraph with $|V(G)| \geq 1$ has a strong shelter of thickness 1: take $\mathcal{S} = \{\{v\}\}$ for some $v \in V(G)$. Thus for $k = 0$, the result is trivial. Now suppose for $k' < k$ every digraph of cycle-rank $\geq k'$ contains a strong shelter of thickness $k' + 1$. For $v \in V(G')$, let $G'_v = G' \setminus \{v\}$. From the definition of cycle-rank, $\mathrm{cr}(G'_v) \geq k - 1$ for all $v \in V(G')$. Thus, by the induction hypothesis, G'_v contains a strong shelter, $\mathcal{S}_v$, of thickness $(k - 1) + 1$. As $v \notin S$ for all $S \in \mathcal{S}_v$, it follows that $\mathcal{S} = \{G'\} \cup \bigcup_{v \in V(G')} \mathcal{S}_v$ is a strong shelter. As $\mathcal{S}_v$ has thickness k for all $v \in V(G')$, $\mathcal{S}$ has thickness $k + 1$.

5 Conclusions and Further Work

Combining Theorems 1 and 2 gives our main result:

Main Theorem. *Let G be a digraph, and k a positive integer. The following are equivalent:*

(i) G has cycle-rank $\leq k - 1$,
(ii) On G, k searchers can capture a fugitive with a LIFO-search strategy,
(iii) On G, k searchers can capture a visible fugitive restricted to moving in strongly connected sets with a searcher-stationary search strategy,
(iv) G has no LIFO-haven of order $> k$, and
(v) G has no strong shelter of thickness $> k$.

This multiple characterization of cycle-rank gives a new perspective on the measure which can be useful for further investigation. For example, whilst it is known that computing the cycle-rank is NP-complete [10], the characterization in terms of a graph searching game with a visible fugitive automatically implies that for any fixed k, deciding if a digraph has cycle-rank k is decidable in polynomial time. From a parameterized complexity perspective, techniques based on separators have shown measures such as directed treewidth are fixed-parameter tractable. Whether the visible, strongly connected game characterizations of cycle-rank can improve the known complexity from XP to FPT is part of ongoing research.

References

1. Adler, I.: Directed tree-width examples. Journal of Combinatorial Theory, Series B 97(5), 718–725 (2007)
2. Barát, J.: Directed path-width and monotonicity in digraph searching. Graphs and Combinatorics 22(2), 161–172 (2006)
3. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: DAG-width and parity games. In: Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science, pp. 524–536 (2006)
4. Bienstock, D., Seymour, P.D.: Monotonicity in graph searching. Journal of Algorithms 12, 239–245 (1991)
5. Dendris, N.D., Kirousis, L.M., Thilikos, D.M.: Fugitive-search games on graphs and related parameters. Theoretical Computer Science 172(1-2), 233–254 (1997)
6. Eggan, L.C.: Transition graphs and the star-height of regular events. Michigan Mathematical Journal 10(4), 385–397 (1963)
7. Eisenstat, S., Liu, J.: The theory of elimination trees for sparse unsymmetric matrices. SIAM Journal of Matrix Analysis & Applications 26(3), 686–705 (2005)
8. Ganian, R., Hliněný, P., Kneis, J., Langer, A., Obdržálek, J., Rossmanith, P.: On digraph width measures in parameterized algorithmics. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 185–197. Springer, Heidelberg (2009)
9. Giannapolou, A., Thilikos, D.: A min-max theorem for LIFO-search. Presented at the 4th Workshop on Graph Searching, Theory and Applications, GRASTA 2011 (2011)
10. Gruber, H.: Digraph Complexity Measures and Applications in Formal Language Theory. In: 4th Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2008), pp. 60–67 (2008)
11. Johnson, T., Robertson, N., Seymour, P.D., Thomas, R.: Directed tree-width. Journal of Combinatorial Theory (Series B) 82(1), 138–154 (2001)
12. Kirousis, L., Papadimitriou, C.: Searching and pebbling. Theoretical Computer Science 47(3), 205–218 (1986)
13. Kreutzer, S., Ordyniak, S.: Digraph decompositions and monotonicity in digraph searching. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 336–347. Springer, Heidelberg (2008)
14. LaPaugh, A.S.: Recontamination does not help to search a graph. Journal of the ACM 40(2), 224–245 (1993)
15. Nesetril, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. European Journal of Combinatorics 27(6), 1022–1041 (2006)
16. Obdržálek, J.: DAG-width: Connectivity measure for directed graphs. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 814–821 (2006)
17. Safari, M.A.: D-width: A more natural measure for directed tree width. In: Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science, pp. 745–756 (2005)
18. Seymour, P.D., Thomas, R.: Graph searching, and a min-max theorem for tree-width. Journal of Combinatorial Theory (Series B) 58, 22–33 (1993)

Polynomial Kernels for Proper Interval Completion and Related Problems[⋆]

Stéphane Bessy and Anthony Perez

LIRMM, Université Montpellier II, France

Abstract. Given a graph $G = (V, E)$ and a positive integer k, the Proper Interval Completion problem asks whether there exists a set F of at most k pairs of $(V \times V) \setminus E$ such that the graph $H = (V, E \cup F)$ is a proper interval graph. The Proper Interval Completion problem finds applications in molecular biology and genomic research [11]. First announced by Kaplan, Tarjan and Shamir in FOCS '94, this problem is known to be FPT [11], but no polynomial kernel was known to exist. We settle this question by proving that Proper Interval Completion admits a kernel with $O(k^5)$ vertices. Moreover, we prove that a related problem, the so-called Bipartite Chain Deletion problem admits a kernel with $O(k^2)$ vertices, completing a previous result of Guo [10].

Introduction

The aim of a graph modification problem is to transform a given graph in order to get a certain property Π satisfied. Several types of transformations can be considered: mainly *vertex deletion* problems, *edge deletion, addition or edition* problems. The optimization version of such problems consists in finding a *minimum* set of edges (or vertices) whose modification makes the graph satisfy the given property Π. Graph modification problems cover a broad range of NP-Complete problems and have been extensively studied in the literature [14,17]. Well-known examples include the Vertex Cover [18], Feedback Vertex Set [20], or Cluster Editing [5] problems. These problems find applications in various domains, such as computational biology [11], image processing [17] or relational databases [19]. A natural approach to deal with such problems is to measure their difficulty with respect to some *parameter* k, such as the *number of allowed modifications*. *Parameterized complexity* provides a useful theoretical framework to that aim [7,15]. A problem *parameterized* by some integer k is said to be *fixed-parameter tractable* (FPT for short) whenever it can be solved in time $f(k) \cdot n^c$ for some constant $c > 0$. As we previously mentioned, a natural parameterization for graph modification problems thereby consists in the number of allowed transformations. As one of the most powerful technique to design FPT algorithms, *kernelization algorithms* have been extensively studied in the last decade (see [2] for a recent survey). A *kernelization algorithm*

⋆ Research supported by the AGAPE project (ANR-09-BLAN-0159).

is a polynomial-time algorithm (called *reduction rules*) that given an instance (I, k) of a parameterized problem P computes an instance (I', k') of P such that *(i)* (I, k) is a YES-instance if and only if (I', k') is a YES-instance and *(ii)* $|I'| \leq h(k)$ for some computable function $h()$ and $k' \leq k$. The instance (I', k') is called the *kernel* of P. We say that (I', k') is a *polynomial kernel* if the function $h()$ is a polynomial. It is well-known that a parameterized (decidable) problem is FPT if and only if it has a kernelization algorithm [15]. But this equivalence only yields kernels of super-polynomial size. To design efficient FPT algorithms, a kernel of small size - polynomial (or even linear) in k - is highly desirable [16]. However, recent results give evidence that not every parameterized problem admits a polynomial kernel, unless $NP \subseteq coNP/poly$ [3]. On the positive side, notable kernelization results include a less-than-$2k$ vertex-kernel for VERTEX COVER [18], a $4k^2$ vertex-kernel for FEEDBACK VERTEX SET [20] and a $2k$ vertex-kernel for CLUSTER EDITING [5]. We follow this line of research with respect to graph modification problems. It has been shown that a graph modification problem is FPT whenever Π is hereditary and can be characterized by a finite set of forbidden induced subgraphs [4]. However, recent results proved that several graph modification problems do not admit a polynomial kernel even for such properties Π [9,12]. In this paper, we are in particular interested in *completion* problems, where the only allowed operation is to add edges to the input graph. We consider the property Π as being the class of *proper interval graphs*. This class is a well-studied class of graphs, and several characterizations are known to exist [13,23]. In particular, there exists an *infinite* set of forbidden induced subgraphs that characterizes proper interval graphs [23]. More formally, we consider the following problem:

PROPER INTERVAL COMPLETION:
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Output: A set F of at most k pairs of $(V \times V) \setminus E$ such that the graph $H = (V, E \cup F)$ is a proper interval graph.

This problem finds applications in molecular biology and genomic research [11]. It is known to be NP-Complete for a long time [8], but FPT due to a result of Kaplan, Tarjan and Shamir in FOCS '94 [11]. [1] Nevertheless, it was not known whether this problem admits a polynomial kernel.

Our results. First, we prove that the PROPER INTERVAL COMPLETION problem admits a kernel with $O(k^5)$ vertices. To that aim, we identify *nice* parts of the graph that induce proper interval graphs and can hence be safely reduced. Then, we apply our techniques to the so-called BIPARTITE CHAIN DELETION problem, obtaining a kernel with $O(k^2)$ vertices.

[1] Notice also that the *vertex deletion* of the problem is FPT [21].

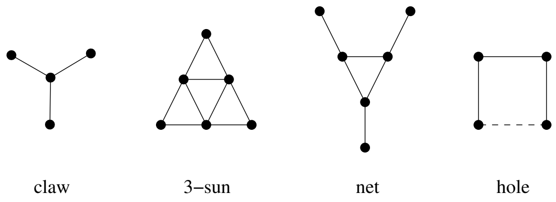


Fig. 1. The forbidden induced subgraphs of proper interval graphs. A *hole* is an induced cycle of length at least 4.

1 Preliminaries

We consider simple, loopless, undirected graphs $G = (V, E)$ where $V(G)$ denotes the vertex set of G and $E(G)$ its edge set[2]. Given a vertex $v \in V$, we use $N_G(v)$ to denote the *open neighborhood* of v and $N_G[v] = N_G(v) \cup \{v\}$ for its *closed neighborhood.* Two vertices u and v are *true twins* if $N[u] = N[v]$. If u and v are not true twins but $uv \in E$, we say that a vertex of $N[u] \triangle N[v]$ *distinguishes* u and v. Given a subset of vertices $S \subseteq V$, $N_S(v)$ denotes the set $N_G(v) \cap S$ and $N_G(S)$ denotes the set $\{N_G(s) \setminus S : s \in S\}$. Moreover, $G[S]$ denotes the subgraph *induced* by S, i.e. $G[S] = (S, E_S)$ where $E_S = \{uv \in E \ : \ u, v \in S\}$. A *join* in a graph $G = (V, E)$ is a bipartition (X, Y) of G and an order $x_1, \ldots, x_{|X|}$ on X such that for all $i = 1, \ldots, |X| - 1$, $N_Y(x_i) \subseteq N_Y(x_{i+1})$. A graph is an *interval graph* if it admits a representation on the real line such that: (i) the vertices of G are in bijection with intervals of the real line and (ii) $uv \in E$ if and only if $I_u \cap I_v \neq \emptyset$, where I_u and I_v denote the intervals associated to u and v, respectively. Such a graph is said to admit an *interval representation.* A graph is a *proper interval graph* if it admits an interval representation such that $I_u \not\subset I_v$ for every $u, v \in V$. In other words, no interval strictly contains another interval. We will make use of the two following characterizations of proper interval graphs to design our kernelization algorithm.

Theorem 1 (Forbidden subgraphs [23]). *A graph is a proper interval graph if and only if it does not contain any* $\{hole, claw, net, 3\text{-}sun\}$ *as an induced subgraph.*

The claw graph is the bipartite graph $K_{1,3}$. Denoting the bipartition by $(\{c\}, \{l_1, l_2, l_3\})$, we call c the *center* and $\{l_1, l_2, l_3\}$ the *leaves* of the claw.

Theorem 2 (Umbrella property [13]). *A graph is a proper interval graph if and only if its vertices admit an ordering σ (called* umbrella ordering*) satisfying the following property: given $v_i v_j \in E$ with $i < j$ then $v_i v_l, v_l v_j \in E$ for every $i < l < j$.*

In the following, we associate an umbrella ordering σ_G to any proper interval graph $G = (V, E)$. Remark that in an umbrella ordering σ_G of a graph G,

[2] In all our notations, we do not mention the graph G whenever the context is clear.

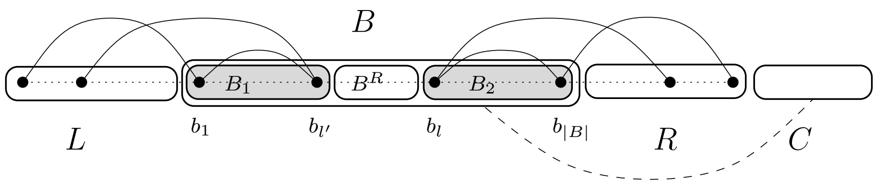


Fig. 2. A 2-branch of a graph $G = (V, E)$. The vertices of B are ordered according to the umbrella ordering σ_B. [3]

every maximal set of true twins of G is consecutive, and that σ_G is unique up to permutation of true twins of G. We remark also that for any edge uv with $u <_{\sigma_G} v$, the set $\{w \in V \ : \ u \leq_{\sigma_G} w \leq_{\sigma_G} v\}$ is a clique of G, and for every i with $1 \leq i < l$, $(\{v_1, \dots, v_i\}, \{v_{i+1}, \dots, v_n\})$ is a join of G. According to this ordering, we say that an edge uv is *extremal* if there does not exist any edge $u'v'$ different of uv such that $u' \leq_{\sigma_G} u$ and $v \leq_{\sigma_G} v'$.

Let $G = (V, E)$ be an instance of PROPER INTERVAL COMPLETION. A *completion* of G is a set $F \subseteq (V \times V) \setminus E$ such that the graph $H = (V, E \cup F)$ is a proper interval graph. In a slight abuse of notation, we use $G + F$ to denote the graph H. A *k-completion* of G is a completion such that $|F| \leq k$, and an *optimal completion* F is such that $|F|$ is minimum. We say that $G = (V, E)$ is a *positive* instance of PROPER INTERVAL COMPLETION whenever it admits a k-completion.

We now give the main definitions of this Section. The branches that we will define correspond to some parts of the graph that already behave like proper interval graphs. They are the parts of the graph that we will reduce in order to obtain a kernelization algorithm.

Definition 1 (Branch). *Let $B \subseteq V$. We say that B is a branch if the following properties hold (see Figure 2):*

(i) The graph $G[B]$ is a connected proper interval graph admitting an umbrella ordering $\sigma_B = b_1, \dots, b_{|B|}$ and,

(ii) The vertex set $V \setminus B$ can be partitioned into sets L, R and C with:
   - *no edges between B and C,*
   - *every vertex in L (resp. R) has a neighbor in B,*
   - *no edges between $\{b_1, \dots, b_{l-1}\}$ and R where b_l is the neighbor of $b_{|B|}$ with minimal index in σ_B,*
   - *no edges between $\{b_{l'+1}, \dots, b_{|B|}\}$ and L where $b_{l'}$ is the neighbor of b_1 with maximal index in σ_B and,*
   - *$N_R(b_i) \subseteq N_R(b_{i+1})$ for every $l \leq i < |B|$ and $N_L(b_{i+1}) \subseteq N_L(b_i)$ for every $1 \leq i < l'$.*

[3] In all the figures, (non-)edges between blocks stand for all the possible (non-)edges between the vertices that lie in these blocks, and the vertices within a gray box form a clique of the graph.

In the case where $L = \emptyset$ (or $R = \emptyset$), we say that B is a *1-branch*, otherwise we say that B is a *2-branch*. Morever, we denote by B_1 (resp. B_2) the set of vertices $\{v \in V : b_1 \leq_{\sigma_B} v \leq_{\sigma_B} b_{l'}\}$ (resp. $\{v \in V : b_l \leq_{\sigma_B} v \leq_{\sigma_B} b_{|B|}\}$). We call B_1 and B_2 the *attachment cliques* of B, and use B^R to denote $B \setminus (B_1 \cup B_2)$.

In both cases, in a 1- or 2-branch, whenever the proper interval graph $G[B]$ is a *clique*, we say that B is a *K-join*. Observe that, in a 1- or 2-branch B, for any extremal edge uv in σ_B, the set of vertices $\{w \in V : u \leq_{\sigma_B} w \leq_{\sigma_B} v\}$ defines a K-join. In particular, this means that a branch can be decomposed into a sequence of K-joins. Observe however that the decomposition is not unique and we will precise in Section 2.1, when we will reduce the size of 2-branches, how to fix a decomposition. Finally, we say that a K-join is *clean* whenever its vertices are not contained in any claw or 4-cycle. Remark that a subset of a K-join (resp. clean K-join) is also a K-join (resp. clean K-join).

2 Kernel for Proper Interval Completion

2.1 Reduction Rules

Basic rules. We say that a rule is *safe* if when it is applied to an instance (G, k) of the problem, (G, k) admits a k-completion iff the instance (G', k') reduced by the rule admits a k'-completion.

The first reduction rule gets rid of connected components that are already proper interval graphs. This rule is trivially safe and can be applied in $O(n+m)$ time using any recognition algorithm for proper interval graphs [6].

Rule 1 (Connected components). *Remove any connected component of G that is a proper interval graph.*

The following reduction rule can be applied since proper interval graphs are closed under true twin addition and induced subgraphs. For a class of graphs satisfying these two properties, we know that this rule is safe [1] (roughly speaking, we edit all the large set of true twins in the same way).

Rule 2 (True twins [1]). *Let T be a set of true twins in G such that $|T| > k$. Remove $|T| - (k+1)$ arbitrary vertices from T.*

We also use the classical *sunflower* rule, allowing to identify a set of edges that must be added in any optimal completion.

Rule 3 (Sunflower). *Let $\mathcal{S} = \{C_1, \ldots, C_m\}$, $m > k$ be a set of claws having two leaves u, v in common but distinct third leaves. Add uv to F and decrease k by* 1.
Let $\mathcal{S} = \{C_1, \ldots, C_m\}$, $m > k$ be a set of distinct 4*-cycles having a non-edge uv in common. Add uv to F and decrease k by* 1.

Lemma 1. *Rule 3 is safe and can be carried out in polynomial time.*

Extracting a clean K-join from a K-join. Now, we want to reduce the size of the 'simplest' branches, namely the K-joins. More precisely, later, we will bound the number of vertices in a clean K-join (whose vertices are not contain in any claw or 4-cycle), and so, we first indicate how to extract a clean K-join from a K-join.

Lemma 2. *Let $G = (V, E)$ be a positive instance of* Proper Interval Completion *on which Rule 2 and Rule 3 have been applied and B be a K-join of G. There are at most $k^3 + 4k^2 + 5k + 1$ vertices of B that belong to a claw or a 4-cycle.*

Since any subset of a K-join forms a K-join, Lemma 2 implies that it is possible to remove a set of at most $k^3 + 4k^2 + 5k + 1$ vertices from any K-join to obtain a clean K-join.

Bounding the size of the K-joins. Now, we set a rule that will bound the number of vertices in a clean K-join, once applied. Although quite technical to prove, this rule is the core tool of our process of kernelization.

Rule 4 (K-join). *Let B be a clean K-join of size at least $2k+2$. Let B_L be the* $k+1$ first vertices *of B, B_R be its* $k+1$ last vertices *and $M = B \setminus (B_R \cup B_L)$. Remove the set of vertices M from G.*

Lemma 3. *Rule 4 is safe.*

Sketch of the proof. Let $G' = G \setminus M$. Observe that the restriction to G' of any k-completion of G is a k-completion of G', since proper interval graphs are closed under induced subgraphs. So, let F be a k-completion for G'. We denote by $H = G' + F$ the resulting proper interval graph and by σ_H an umbrella ordering of H. We prove that we can insert the vertices of M into σ_H and modify it if necessary, to obtain an umbrella ordering for G without adding any edge (in fact, some edges of F might even be deleted during the process). This will imply that G admits a k-completion as well. To see this, we need the following structural description of G. We denote by N the set $\cap_{b \in B} N_G(b) \setminus B$, and abusively still denote by L (resp. R) the set $L \setminus N$ (resp. $R \setminus N$).

Claim 1. *The sets L and R are cliques of G.*

Observation 3. *By definition of a K-join, given any vertex $r \in R$, if $N_B(r) \cap B_L \neq \emptyset$ holds then $M \subseteq N_B(r)$. Similarly, given any vertex $l \in L$, if $N_B(l) \cap B_R \neq \emptyset$ holds then $M \subseteq N_B(l)$.*

We use these facts to prove that an umbrella ordering can be obtained for G by inserting the vertices of M into σ_H. Let b_f and b_l be respectively the first and last vertex of $B \setminus M$ appearing in σ_H. We let B_H denote the set $\{u \in V(H) : b_f \leq_{\sigma_H} u \leq_{\sigma_H} b_l\}$. Observe that B_H is a clique in H since $b_f b_l \in E(G)$ and that $B \setminus M \subseteq B_H$. Now, we modify σ_H by ordering the true twins in H according to their neighborhood in M: if x and y are true twins in H, are consecutive in σ_H, verify $x <_{\sigma_H} y <_{\sigma_H} b_f$ and $N_M(y) \subset N_M(x)$, then we exchange x and y

in σ_H. This process stops when the considered true twins are ordered following the join between $\{u \in V(H) : u <_{\sigma_H} b_f\}$ and M. We proceed similarly on the right of B_H. The obtained order is clearly an umbrella ordering too (in fact, we just re-labeled some vertices in σ_H), and we abusively still denote it by σ_H.

Claim 2. *The set $B_H \cup \{m\}$ is a clique of G for any $m \in M$, and consequently $B_H \cup M$ is a clique of G.*

Claim 3. *Let m be any vertex of M and σ'_H be the ordering obtained from σ_H by removing B_H and inserting m to the position of B_H. The ordering σ'_H respects the umbrella property.*

Claim 4. *Let $m \in M$. Then m can be added to the graph H while preserving an umbrella ordering.*

In order to prove Claim 4 we do not use the fact that the vertices of H do not belong to M. It follows that we can iteratively insert the vertices of M into σ_H, preserving an umbrella ordering at each step. □

The following observation results from the application of Rule 4 and from Section 2.1.

Observation 4. *Let $G = (V, E)$ be a positive instance of* Proper Interval Completion *reduced under Rules 2 to 4. Any K-join of G has size at most $k^3 + 4k^2 + 7k + 3$.*

Cutting the 1-branches. We now turn our attention to branches of a graph $G = (V, E)$, proving how they can be reduced.

Rule 5 (1-branch). *Let B be a 1-branch such that $|B^R| \geq 2k + 1$. Remove $B^R \setminus B_f$ from G, where B_f denotes the $2k + 1$ last vertices of B^R.*

Lemma 4. *Rule 5 is safe.*

The following property of a reduced graph will be used to bound the size of our kernel.

Observation 5. *Let $G = (V, E)$ be a positive instance of* Proper Interval Completion *reduced under Rules 2 to 5. The 1-branches of G contain at most $k^3 + 4k^2 + 9k + 4$ vertices.*

Cutting the 2-branches. To obtain a rule reducing the 2-branches, we need to introduce a particular decomposition of 2-branches into K-joins. Let B be a 2-branch with an umbrella ordering $\sigma_B = b_1, \ldots, b_{|B|}$. As usual, we denote by $B_1 = b_1, \ldots, b_{l'}$ its first attachment clique and by $B_2 = b_l, \ldots, b_{|B|}$ its second. The reversal of the permutation σ_B gives a second possibility to fix B_1 and B_2. We fix one of these possibilities and define $\mathcal{B}$, the *K-join decomposition* of B. The K-joins of $\mathcal{B}$ are defined by $B'_i = b_{l_{i-1}+1}, \ldots, b_{l_i}$ where b_{l_i} is the neighbor of $b_{l_{i-1}+1}$ with maximal index. The first K-join of $\mathcal{B}$ is B_1 (so, $l_0 = 0$ and $l_1 = l'$), and once B'_{i-1} is defined, we set B'_i: if $b_{l_{i-1}+1} \in B_2$, then we set

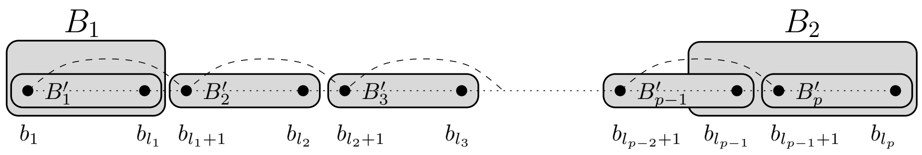


Fig. 3. The K-join decomposition

$B'_i = b_{l_{i-1}+1}, \ldots, b_{|B|}$, otherwise, we choose $B'_i = b_{l_{i-1}+1}, \ldots, b_{l_i}$ (see Figure 3). Now, we can prove the next lemma, which bounds the number of K-joins in the K-join decomposition of a 2-branch providing that some connectivity assumption holds.

Lemma 5. *Let $G = (V, E)$ be an instance of* Proper Interval Completion *and B be a 2-branch containing $p \geq (k+4)$ K-joins in its K-join decomposition. If the attachment cliques of B belong to the same connected component of $G[V \setminus B^R]$, then there is no k-completion for G.*

In the case where $G[V \setminus B^R]$ is not connected, the given 2-branch can be reduced using the following rule.

Rule 6 (2-branch). *Let G be a connected graph and B be a 2-branch such that $G[V \setminus B^R]$ is not connected. Assume that $|B^R| \geq 4(k+1)$ and let B'_1 be the $2k+1$ vertices after B_1 and B'_2 the $2k+1$ vertices before B_2. Remove $B \setminus (B_1 \cup B'_1 \cup B'_2 \cup B_2)$ from G.*

Lemma 6. *Rule 6 is safe.*

Observation 6. *Let $G = (V, E)$ be a positive instance of* Proper Interval Completion *reduced under Rules 2 to 6. The 2-branches of G contain at most $(k+3)(k^3 + 4k^2 + 5k + 1)$ vertices.*

2.2 Detecting the Branches

We now turn our attention to the complexity needed to compute reduction rules 4 to 6. Mainly, we indicate how to obtain the maximum branches in order to reduce them. The detection of a branch is straightforward except for the attachment cliques, where several choices are possible. Nevertheless, this can be dealt with, and we obtain the following results.

Lemma 7. *Let $G = (V, E)$ be a graph and x a vertex of G. In time $O(n^2)$, it is possible to detect a maximum 1-branch of G containing x as first vertex.*

Lemma 8. *Let $G = (V, E)$ be a graph and x and y two adjacent vertices of G. It is possible to compute in cubic time a maximum (in cardinality) K-join that admits x and y as ends.*

Now, for every edge xy of G, we compute a maximum K-join that contains x and y as ends and a reference to all the vertices that this K-join contains. This computation takes a $O(n^3m)$ time and gives, for every vertex, some maximum K-joins that contain this vertex. Finally, we can detect the 2-branches B with a set B^R non empty. Observe that this is enough for our purpose since we want to detect 2-branches of size at least $(k+3)(k^3+4k^2+5k+1)$ and the attachment cliques contain at most $2(k^3+4k^2+7k+3)$ vertices.

Lemma 9. *Let $G=(V,E)$ be a graph, x a vertex of G and B' a given maximal K-join that contains x. There is a quadratic time algorithm to decide if there exists a 2-branch B of G which contains x as a vertex of B^R, and if it exists, to find a maximum 2-branch with this property.*

Altogether, using a $O(n^4)$ brute force detection to localize all the 4-cycles and the claws, we obtain the following result.

Lemma 10. *Given a graph $G=(V,E)$, the reduction rules 4 to 6 can be carried out in polynomial time, namely in time $O(n^3m)$.*

2.3 Kernelization Algorithm

We are now ready to the state the main result of this Section. The kernelization algorithm consists of an exhaustive application of Rules 1 to 6.

Theorem 7. *The* Proper Interval Completion *problem admits a kernel with $O(k^5)$ vertices.*

Sketch of the proof. Let $G=(V,E)$ be a positive instance of Proper Interval Completion reduced under Rule 1 to 6, and F be a k-completion for G. Let $H=G+F$ be the corresponding proper interval graph and σ_H be its umbrella ordering. By definition, there are at most $2k$ vertices of H incident to edges of F. Let A be the set of such vertices. Using structural properties of the umbrella ordering σ_H, one can see that the set of vertices lying between two consecutive vertices of A in σ_H define either 1- or 2- branches. The claim bound on the number of vertices then follow from Observations 5 and 6. $\square$

3 A Special Case: Bi-clique Chain Completion

Bipartite chain graphs are defined as bipartite graphs whose parts are connected by a join. Equivalently, they are known to be the graphs that do not admit any $\{2K_2, C_5, K_3\}$ as an induced subgraph [24], a $2K_2$ being the graph $(\{u,u',v,v'\},\{(u,u'),(v,v')\})$. In [10], Guo proved that the so-called Bipartite Chain Deletion With Fixed Bipartition problem, where one is given a *bipartite* graph $G=(V,E)$ and seeks a subset of E of size at most k whose deletion from E leads to a bipartite chain graph, admits a kernel with $O(k^2)$ vertices. We define *bi-clique chain graphs* to be the graphs formed by two disjoint cliques linked by a join. They correspond to interval graphs which can be covered with two cliques. Since the complement of a bipartite chain graph is a bi-clique chain graph, this result also holds for the Bi-clique Chain Completion With Fixed Bi-clique Partition problem. Using similar techniques than in Section 2, we prove that when the bipartition is not fixed, both problems admit a quadratic-vertex kernel.

Theorem 8. *The* Bi-clique Chain Completion *and* Bipartite Chain Deletion *problems admit kernels with $O(k^2)$ vertices.*

4 Conclusion

In this paper we prove that the Proper Interval Completion problem admits a kernel with $O(k^5)$ vertices. One natural question arises from our results: does the Interval Completion problem admit a polynomial kernel? Observe that this problem is known to be FPT not for long [22]. Finally, we proved that the Bi-clique Chain Completion problem admits a kernel with $O(k^2)$ vertices, which completes a result of Guo [10]. In all cases, a natural question is thus whether these bounds can be improved?

References

1. Bessy, S., Paul, C., Perez, A.: Polynomial kernels for 3-leaf power graph modification problems. Discrete Applied Mathematics 158(16), 1732–1744 (2010)
2. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, Springer, Heidelberg (2009)
3. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci 75(8), 423–434 (2009)
4. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. 58(4), 171–176 (1996)
5. Chen, J., Meng, J.: A $2k$ kernel for the cluster editing problem. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 459–468. Springer, Heidelberg (2010)
6. Corneil, D.G.: A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. Discrete Appl. Math. 138, 371–379 (2004)
7. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, Heidelberg (1999)
8. Golumbic, M.C., Kaplan, H., Shamir, R.: On the complexity of DNA physical mapping. ADVAM: Advances in Applied Mathematics 15 (1994)
9. Guillemot, S., Paul, C., Perez, A.: On the (non-)existence of polynomial kernels for P_l-free edge modification problems. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 147–157. Springer, Heidelberg (2010)
10. Guo, J.: Problem kernels for NP-complete edge deletion problems: Split and related graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 915–926. Springer, Heidelberg (2007)
11. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping. In: FOCS, pp. 780–791 (1994)

12. Kratsch, S., Wahlström, M.: Two edge modification problems without polynomial kernels. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 264–275. Springer, Heidelberg (2009)
13. Looges, P.J., Olariu, S.: Optimal greedy algorithms for indifference graphs. Computers & Mathematics with Applications 25(7), 15–25 (1993)
14. Mancini, F.: Graph modification problems related to graph classes. PhD thesis, University of Bergen, Norway (2008)
15. Niedermeier, R.: Invitation to fixed parameter algorithms. Oxford Lectures Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2006)
16. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Inf. Process. Lett 73(3-4), 125–129 (2000)
17. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Applied Mathematics 144(1-2), 173–182 (2004)
18. Soleimanfallah, A., Yeo, A.: A kernel of order 2k-c for vertex cover. Discrete Mathematics 311(10-11), 892–895 (2011)
19. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. 13(3), 566–579 (1984)
20. Thomassé, S.: A $4k^2$ kernel for feedback vertex set. ACM Transactions on Algorithms 6(2) (2010)
21. Villanger, Y.: Proper interval vertex deletion. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 228–238. Springer, Heidelberg (2010)
22. Villanger, Y., Heggernes, P., Paul, C., Telle, J.A.: Interval completion is fixed parameter tractable. SIAM J. Comput. 38(5), 2007–2020 (2009)
23. Wegner, G.: Eigenschaften der nerven homologische-einfactor familien in $\mathbb{R}^n$. PhD thesis, Universität Göttigen, Göttingen, Germany (1967)
24. Yannakakis, M.: Computing the minimum fill-in is NP-Complete. SIAM J. Alg. and Discr. Meth. 2(1), 77–79 (1981)

Parameterized Complexity of Vertex Deletion into Perfect Graph Classes*

Pinar Heggernes[1], Pim van 't Hof[1], Bart M.P. Jansen[2], Stefan Kratsch[2], and Yngve Villanger[1]

[1] University of Bergen, P.O. Box 7803, N-5020 Bergen, Norway
{pinar.heggernes,pim.vanthof,yngve.villanger}@ii.uib.no
[2] Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{bart,kratsch}@cs.uu.nl

Abstract. Vertex deletion problems are at the heart of parameterized complexity. For a graph class $\mathcal{F}$, the $\mathcal{F}$-DELETION problem takes as input a graph G and an integer k. The question is whether it is possible to delete at most k vertices from G such that the resulting graph belongs to $\mathcal{F}$. Whether PERFECT DELETION is fixed-parameter tractable, and whether CHORDAL DELETION admits a polynomial kernel, when parameterized by k, have been stated as open questions in previous work. We show that PERFECT DELETION (k) and WEAKLY CHORDAL DELETION (k) are $W[2]$-hard. In search of positive results, we study restricted variants such that the deleted vertices must be taken from a specified set X, which we parameterize by $|X|$. We show that for PERFECT DELETION and WEAKLY CHORDAL DELETION, although this restriction immediately ensures fixed parameter tractability, it is not enough to yield polynomial kernels, unless NP $\subseteq$ coNP/poly. On the positive side, for CHORDAL DELETION, the restriction enables us to obtain a kernel with $\mathcal{O}(|X|^4)$ vertices.

1 Introduction

The minimum number of vertices to delete from a given graph so that the resulting graph is a member of a graph class $\mathcal{F}$ is a way of measuring how close the input graph is to being in $\mathcal{F}$. Vertex deletion problems correspond to some of the most fundamental NP-complete graph problems [14]; if we take $\mathcal{F}$ to be the class of complete graphs, we simply get the CLIQUE problem, and if we take $\mathcal{F}$ to be the class of edgeless graphs, we get the VERTEX COVER problem. Vertex deletion problems have received much attention also for more general graph classes $\mathcal{F}$; e.g., the number of vertex deletions needed to make a graph acyclic, corresponding to the FEEDBACK VERTEX SET problem, has applications in deadlock recovery. In addition, seemingly unrelated problems are easier to solve on graphs which are close to being a member of some simple graph class.

* Supported by the Netherlands Organization for Scientific Research, project "KERNELS: Combinatorial Analysis of Data Reduction", and by the Research Council of Norway, project "SCOPE: Exploiting Structure to Cope with Hard Problems".

For example, Graph Isomorphism [18] can be solved efficiently on "almost" forests. Since all vertex deletion problems for non-trivial, polynomial-time recognizable, hereditary graph classes are NP-complete [19], many of these problems have been studied with respect to parameterized complexity [12].

Parameterized complexity associates with every instance a non-negative integer k, called the *parameter*. A parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is *Fixed Parameter Tractable (FPT)* if there is an algorithm which decides whether $(x, k) \in Q$ in time $f(k)\ |x|^{\mathcal{O}(1)}$ for some computable function f. There is a hierarchy of intractable parameterized problem classes above FPT, the main ones being: $\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P] \subseteq \text{XP}$. An important subfield of parameterized complexity is *kernelization* [17], a formalization of data reduction. For a parameterized problem Q, if there is an algorithm which transforms an instance (x, k) in time $(|x| + k)^{\mathcal{O}(1)}$ into an *equivalent* instance (x', k'), with the guarantee that $(x, k) \in Q \Leftrightarrow (x', k') \in Q$ and $|x'|, k' \leq g(k)$ for some computable function g, then we say that Q admits a *kernel*. In fact, a parameterized problem is FPT if and only if it is decidable and admits a kernel [12]; unfortunately the guaranteed size is typically exponential. If $g \in k^{\mathcal{O}(1)}$ then the obtained kernel is a *polynomial kernel*, which is a highly desirable property. Whether or not an FPT problem admits a polynomial kernel has received considerable attention recently, especially after the establishment of methods for proving non-existence of polynomial kernels, up to some complexity theoretical assumptions [6,3,4]. Vertex deletion problems are most commonly defined and parameterized as follows.

$\mathcal{F}$-Deletion (k)
Input: A graph G and an integer k.
Parameter: k.
Question: Is there a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ is a member of graph class $\mathcal{F}$?

Well known FPT problems of this type include Vertex Cover [9] (i.e., Independent Deletion (k)), Feedback Vertex Set [25] (i.e., Forest Deletion (k)), Odd Cycle Transversal [24] (i.e., Bipartite Deletion (k)), Chordal Deletion (k) [21] and Planar Deletion (k) [22]. When $\mathcal{F}$ is characterized by a finite set of forbidden *induced subgraphs*, then $\mathcal{F}$-Deletion (k) is FPT [8], and admits a polynomial kernel through its correspondence to d-Hitting Set (k) [1]. If $\mathcal{F}$ is characterized by a finite set of forbidden *minors*, then $\mathcal{F}$-Deletion (k) is FPT with an $\mathcal{O}(n^3)$ time algorithm for every fixed value of k, obtained via the Graph Minors machinery [13]. A recent result of Fomin et al. shows that if this set of forbidden minors contains an "onion graph", then the problem additionally admits a polynomial kernel [13].

For some time it was unknown whether there were vertex deletion problems, for hereditary graph classes that can be recognized in polynomial time, that were *not* FPT by the natural parameterization. This was settled by Lokshtanov, who showed that Wheel-free Deletion (k) is $W[2]$-hard [20]. He posed the FPT-status of Perfect Deletion (k) as an open question.

We show that PERFECT DELETION (k) and WEAKLY CHORDAL DELETION (k) are $W[2]$-hard. We then turn our focus to kernelization. The CHORDAL DELETION (k) problem was studied by Marx [21], who gave an involved FPT algorithm which combines branching with an irrelevant-vertex reduction step and Courcelle's Theorem for bounded treewidth. He posed as an open question whether CHORDAL DELETION (k) admits a polynomial kernel. Even an *exponential* simple kernel would yield a simple FPT algorithm for CHORDAL DELETION (k) through exploring the kernel by brute force, and hence finding a *polynomial* kernel seems to be a formidable task. Observing that access to an approximate solution is often helpful in designing kernels [23,5], and that no good approximation algorithms for CHORDAL DELETION (k) are known, one might consider the effect of supplying a constant-factor approximation to the kernelization algorithm. This does not help much; an exponential-size kernel which has access to a constant-factor approximation would immediately yield a new algorithm to solve CHORDAL DELETION COMPRESSION (k) [16], solving the general version as well. The following restricted problem variant is more amenable to analysis and provides further insight:

RESTRICTED $\mathcal{F}$-DELETION ($|X|$)
Input: A graph G, a set of vertices $X \subseteq V(G)$ such that $G - X$ is a member of class $\mathcal{F}$, and an integer k.
Parameter: $|X|$.
Question: Is there a set $S \subseteq X$ of size at most k such that graph $G - S$ is a member of class $\mathcal{F}$?

In this *restricted* variant of vertex deletion problems, a set X of candidates for deletion is given in the input, and we are only allowed to delete vertices from X. The parameter measures the number of candidate vertices, and with this parameterization the problem becomes trivially FPT. However, we show that RESTRICTED PERFECT DELETION ($|X|$) and RESTRICTED WEAKLY CHORDAL DELETION ($|X|$) do not admit polynomial kernels unless NP $\subseteq$ coNP/poly, and the same holds for RESTRICTED WHEEL-FREE DELETION ($|X|$) [20]. In contrast to these hardness results, we show that RESTRICTED CHORDAL DELETION ($|X|$) admits a kernel with $\mathcal{O}(|X|^4)$ vertices. We hope that this forms a first step towards a polynomial kernel for CHORDAL DELETION (k).

Finally, the study of perfect graphs and their subclasses is well established, with several books (e.g., [7,15]) and thousands of papers. The interest in the field is boosted by the recent proof of the Perfect Graph Theorem by Chudnovsky et al. [10], after being a conjecture by Berge [2] for over 40 years. Chordal graphs are a subset of weakly chordal graphs that are a subset of perfect graphs.

2 Preliminaries

If G is a graph then $V(G)$ and $E(G)$ denote the vertex and edge set, respectively. We only consider finite, simple, and undirected graphs. For a vertex $v \in V(G)$, the set of vertices adjacent to v is called the *(open) neighborhood* of v, and is denoted by $N_G(v)$. The *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. For a vertex set $S \subseteq V(G)$, the neighborhood of S is $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$. The subgraph of G induced by S is denoted by $G[S]$. The graph $G[V(G) \setminus S]$ is denoted by $G - S$. The *contraction* of an edge $uv \in E(G)$ deletes u and v from G and replaces them with a new vertex whose neighborhood is $N_G(\{u, v\})$. The resulting graph is denoted by G/uv. For a finite set X, $\binom{X}{a}$ denotes the collection of all subsets of X of size a.

A *walk* W from vertex v_1 to vertex v_r in a graph G is a sequence of vertices $(v_1, v_2, \ldots, v_r)$ such that $v_i v_{i+1} \in E(G)$ for $1 \leq i \leq r - 1$. The vertices $\{v_1, v_r\}$ are the *endpoints* of the walk, whereas $\{v_2, \ldots, v_{r-1}\}$ are the *interior* vertices of the walk. If W is a walk then we use $V(W)$ to denote the set of its vertices. For a walk $W' = (x_1, \ldots, x_t)$, we will also use the notation (v_i, W', v_j) to denote the walk $(v_i, x_1, \ldots, x_t, v_j)$, assuming that $v_i x_1, x_t v_j \in E(G)$. A *chord* of a walk is an edge between two vertices which are not successive on the walk. A walk without chords is an *induced* walk. A walk is a *path* if all its vertices are distinct, and it is a *cycle* if all interior vertices are distinct and the endpoints coincide.

We denote by P_n and C_n an induced path and an induced cycle on n vertices, respectively. A *hole* in a graph is an induced subgraph isomorphic to C_t for $t \geq 5$. An *anti-hole* is the edge-complement of a hole. A hole or anti-hole is *odd* if it contains an odd number of vertices. A graph is *chordal* if it does not contain C_t, for $t \geq 4$, as an induced subgraph; an equivalent condition is that all cycles of length at least four have a chord. A graph is *perfect* if, for each of its induced subgraphs, the chromatic number equals the size of the largest clique. As conjectured a long time ago [2], and proved recently [10], a graph is perfect if and only if it does not contain any odd hole or odd anti-hole as an induced subgraph. A graph is *weakly chordal* if it does not contain any hole or anti-hole as an induced subgraph [15]. A *cograph* is a graph which does not contain P_4 as an induced subgraph. Weakly chordal graphs, chordal graphs, and cographs are all perfect. In addition, chordal graphs and cographs are both weakly chordal, but they are not related to each other with respect to inclusion. It is an easy observation that each of these graph classes is closed under vertex deletions, i.e., *hereditary*. Furthermore, they can all be recognized in polynomial time. In this extended abstract, some of the proofs are omitted due to page restrictions.

3 Hardness of Perfect and Weakly Chordal Deletion

In this section we prove several hardness results for deleting vertices to obtain a perfect or weakly chordal graph. As the main result we will show that PERFECT DELETION (k) is $W[2]$-hard, by a reduction from HITTING SET (k). We will then argue that the same reduction yields several interesting results as a corollary.

HITTING SET (k)
Input: A finite set U of size n, a family $\mathcal{H}$ of subsets of U, and an integer k.
Parameter: k.
Question: Is there a set $Y \subseteq U$ of size at most k that has a nonempty intersection with each set of $\mathcal{H}$?

Theorem 1. Perfect Deletion (k) *is* $W[2]$*-hard.*

Proof. We give a parameterized reduction from the $W[2]$-complete Hitting Set (k) problem [12]. Let $(U, \mathcal{H}, k)$ be an instance of Hitting Set (k). We assume, without loss of generality, that $|H| \geq 2$ for every $H \in \mathcal{H}$, since sets of size 1 can easily be eliminated in polynomial time. We construct an equivalent instance (G, k) of Perfect Deletion (k), by building a graph G as follows:

- Create an independent set X on $|U|$ vertices; $X = \{v_u \mid u \in U\}$.
- For each set $H = \{u_1, \ldots, u_t\} \in \mathcal{H}$, where $t \geq 2$ by assumption, do as follows:
  - Add $|H| + 1$ new vertices $h_1, \ldots, h_{t+1}$ to G. The set $\mathcal{G}_H = \{h_1, \ldots, h_{t+1}\}$ is called the *set gadget* for H.
  - Add the edges $\{h_1, v_{u_1}\}, \{v_{u_1}, h_2\}, \{h_2, v_{u_2}\}, \ldots, \{v_{u_t}, h_{t+1}\}, \{h_{t+1}, h_1\}$, creating an odd chordless cycle $(h_1, v_{u_1}, h_2, v_{u_2}, \ldots, v_{u_t}, h_{t+1})$ of length at least 5.
- Take the *join* of the set gadgets by adding all edges between vertices of different set gadgets: for each set $H \in \mathcal{H}$, make all vertices of $\mathcal{G}_H$ adjacent to all vertices of $\mathcal{G}_{H'}$ for every $H' \neq H$.

This concludes the description of the graph G. To prove the equivalence of the instances $(U, \mathcal{H}, k)$ and (G, k), we first formulate some claims on the structure of G.

Claim 1. *The graph* $G - X$ *is a cograph and therefore perfect.*

Claim 2. *Any hole in* G *intersects* X *and exactly one set gadget* $\mathcal{G}_H$*.*

Proof. Since $G - X$ is a cograph by Claim 1 and therefore contains no hole, any hole in G contains at least one vertex of X. Moreover, since X is an independent set, the graph $G[X]$ contains no holes. This implies that any hole must intersect at least one set gadget. Since any three vertices from three different set gadgets induce a triangle K_3, no hole contains vertices from more than two different set gadgets. Now assume for contradiction that G contains a hole D which contains vertices from exactly two set gadgets. Since we have taken the join of the set gadgets, graph $G[V(D) \setminus X]$ is connected and must hence be an induced path. This in turn implies that D contains at most one vertex of X, since vertices of the independent set X cannot be consecutive vertices of D. However, since holes have at least 5 vertices, this implies that D contains at least 4 consecutive vertices that belong to $G - X$. Since these vertices induce a path on at least 4 vertices, this contradicts the fact that $G - X$ is a cograph by Claim 1.

Claim 3. *Any anti-hole in* G *has length* 5 *and is therefore a hole of length* 5.

Proof. Recall that an anti-hole in G is a hole in the edge-complement $\overline{G}$ of G. Suppose $\overline{G}$ contains a hole D. Observe that every set gadget induces a complete graph minus one edge in $\overline{G}$, and that the graphs induced by the set gadgets are exactly the connected components of $\overline{G} - X$. Since a complete graph minus one edge does not contain a hole, D contains at least one vertex of X. The set X is a clique in $\overline{G}$, so D contains at most two vertices of X, which must be consecutive vertices of D. This implies that the graph $\overline{G}[V(D) \setminus X]$ is connected, and is therefore contained in one connected component of $\overline{G} - X$; let $\mathcal{G}_H$ be the set gadget that induces this connected component. Since $G[\mathcal{G}_H]$ is a complete graph minus one edge in $\overline{G}$, D contains at most three vertices of $\mathcal{G}_H$. This means that D has length at most 5. Since every hole has length at least 5 by definition, we conclude that any hole in $\overline{G}$, and consequently any anti-hole in G, has length exactly 5. The claim follows from the fact that C_5 is self-complementary.

Claim 4. *If $S \subseteq V(G)$ such that $G - S$ is perfect, then there is a set $S' \subseteq X$ with $|S'| \leq |S|$ such that $G - S'$ is perfect.*

Proof. Let $S \subseteq V(G)$ be a set which intersects all odd holes and odd anti-holes in G, and assume $S \nsubseteq X$. Consider a vertex $w \in S \setminus X$, which must belong to some set gadget $\mathcal{G}_H$. By construction, w has exactly two neighbors in $G[X \cup \mathcal{G}_H]$, and at least one of these is contained in X; let v_u be such a vertex in $N_G(w) \cap X$. Let D be a hole in G containing w. As a result of Claim 2, D is contained in $G[X \cup \mathcal{G}_H]$. Since w has only two neighbors in $G[X \cup \mathcal{G}_H]$, D also contains vertex v_u. Hence every hole that contains w also contains v_u. The same holds for every anti-hole containing w, since every anti-hole in G is a hole by Claim 3. This proves that for $S'' = (S \setminus \{w\}) \cup \{v_u\}$ the graph $G - S''$ is also perfect. By repeating this argument we obtain the desired set $S' \subseteq X$.

We are now set to prove correctness of our reduction. First assume that (G, k) is a yes-instance of Perfect Deletion (k), and let $S \subseteq V(G)$ be a set of at most k vertices such that $G - S$ is perfect. By Claim 4, we may assume that $S \subseteq X$. Let $Y \subseteq U$ contain all $u \in U$ for which the corresponding vertex v_u is contained in S. By construction of G, for every set $H = \{u_1, \ldots, u_t\} \in \mathcal{H}$, the set $\{v_{u_1}, \ldots, v_{u_t}\} \cup \mathcal{G}_H$ induces an odd hole in G. Since $S \subseteq X$ intersects all odd holes, S contains at least one vertex of $v_{u_1}, \ldots, v_{u_t}$, which shows that Y hits set H. Hence Y is a hitting set for $\mathcal{H}$ of the requested size, which means that $(U, \mathcal{H}, k)$ is a yes-instance of Hitting Set (k).

For the reverse direction, assume that $(U, \mathcal{H}, k)$ is a yes-instance of Hitting Set (k), and let $Y \subseteq U$ be a set of at most k vertices that intersects every set in $\mathcal{H}$. We let $S \subseteq X \subseteq V(G)$ contain all vertices v_u for which $u \in Y$. Clearly, $|S| \leq k$; we show that $G - S$ is perfect. By Claim 3, it suffices to check that S contains at least one vertex of every odd hole in G. Let D be an odd hole in G. By Claim 2, D contains vertices of X and vertices of exactly one set gadget $\mathcal{G}_H$, for some $H = \{u_1, \ldots, u_t\} \in \mathcal{H}$. Recall that the vertices of $\{v_{u_1}, \ldots, v_{u_t}\} \cup \mathcal{G}_H$ induce an odd hole in $G[X \cup \mathcal{G}_H]$, and it follows from the construction of G that $G[X \cup \mathcal{G}_H]$ does not contain any other hole. Thus, D is an odd hole constructed due to some set $H \in \mathcal{H}$. Since $Y \cap H \neq \emptyset$, this implies that $S \cap V(D) \neq \emptyset$ as well. This shows that the two instances are equivalent. □

From the construction used in the proof of Theorem 1 it may be verified that all the holes and anti-holes in G are odd: Claim 3 shows that every anti-hole in G is an odd hole, and using Claim 2 the chordless cycles of G can be seen to coincide with the odd holes used to represent the sets of $\mathcal{H}$. Hence $G - S$ is perfect if and only if $G - S$ is weakly chordal, and we immediately get the following corollary.

Corollary 1. WEAKLY CHORDAL DELETION (k) *is* $W[2]$*-hard.*

For another implication of our reduction, consider the problem HITTING SET (n), where the parameter is the number of elements n. Dom et al. [11] showed that this problem does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. Claim 4 shows that we may demand the deletion set S of our constructed instance to be a subset of X, without changing the answer. Since the reduction of Theorem 1 can be performed in polynomial time, we can also interpret it as a polynomial parameter transformation from an instance $(U, \mathcal{H}, k)$ of HITTING SET (n) with parameter $n = |U|$ to an instance (G, X, k) of RESTRICTED PERFECT DELETION $(|X|)$ with parameter $|X| = |U|$ (and also to RESTRICTED WEAKLY CHORDAL DELETION $(|X|)$). By standard techniques for kernelization lower bounds ([11]), this polynomial parameter transformation yields the following results.

Corollary 2. *Neither* RESTRICTED PERFECT DELETION $(|X|)$ *nor* RESTRICTED WEAKLY CHORDAL DELETION $(|X|)$ *admits a polynomial kernel, unless NP* $\subseteq$ *coNP/poly.*

4 Polynomial Kernel for Restricted Chordal Deletion

In this section, we prove that RESTRICTED CHORDAL DELETION $(|X|)$ admits a kernel with at most $2|X|^4 + |X|^3 + |X|^2 + |X|$ vertices. To simplify the reduction procedure, we first work on an *annotated* version of the problem. The annotated problem is equivalent to our problem when the set of annotated vertices is empty.

ANNOTATED RESTRICTED CHORDAL DELETION $(|X|)$
Input: A graph G, a set of vertices $X \subseteq V(G)$ such that $G - X$ is chordal, a set of *critical pairs* $C \subseteq \binom{X}{2}$, and an integer k.
Parameter: $|X|$.
Question: Is there a set $S \subseteq X$ of size at most k such that $G - S$ is chordal, and S contains at least one vertex of each pair $\{u, v\} \in C$?

A set S as described above is called a *valid solution*. For ease of notation, we will use F to denote the chordal graph $G - X$. We now present four reduction rules that will constitute a kernelization algorithm for ANNOTATED RESTRICTED CHORDAL DELETION $(|X|)$. Each reduction rule takes as input an instance of this problem, and if the rule is applicable, it outputs an equivalent reduced instance. We apply the reduction rules in the given order; whenever we apply a reduction rule to an instance, we assume that none of the previous reduction rules can be applied on that instance. For all the rules and proofs below, let (G, X, C, k) be an instance of ANNOTATED RESTRICTED CHORDAL DELETION $(|X|)$. The correctness of some of our rules relies on the following result.

Proposition 1 ([21]). *Let G be a graph containing three vertices u, v, w, such that $u, w \in N_G(v)$ and $uw \notin E(G)$. If there is a walk W from u to w none of* *whose interior vertices is in $N_G[v]$, then there is a chordless cycle in G containing $\{u, v, w\}$ and a non-empty subset of $V(W)$.*

Rule 1. *If there is a vertex $v \in X$ such that $G[\{v\} \cup V(F)]$ is not chordal, then reduce to the instance $(G - \{v\}, X \setminus \{v\}, C', k - 1)$, where C' is obtained from C by deleting all pairs which contain v.*

Rule 2. *If there are two vertices $u, v \in X$ with $\{u, v\} \notin C$, such that $G[\{u, v\} \cup V(F)]$ is not chordal, then reduce to the instance $(G, X, C \cup \{\{u, v\}\}, k)$.*

It is easy to see that Rules 1 and 2 are safe.

Rule 3. *If there is an edge $uv \in E(F)$ such that $N_G(u) \cap X = N_G(v) \cap X$, then reduce to the instance $(G/uv, X, C, k)$.*

Lemma 1. *Rule 3 is safe.*

Proof. Assume that Rule 1 and Rule 2 are not applicable, whereas Rule 3 is applicable, on (G, X, C, k). Suppose (G, X, C, k) is a yes-instance, and let S be a valid solution. Since X does not contain u or v, we have $S \cap \{u, v\} = \emptyset$. Observe that the class of chordal graphs is closed under contracting edges. Hence, since $G - S$ is chordal, $(G - S)/uv = G/uv - S$ is also chordal. Consequently, S is a valid solution for $(G/uv, X, C, k)$, which is thus a yes-instance.

For the reverse direction, suppose that $(G/uv, X, C, k)$ is a yes-instance, and let S be a valid solution. We will show that S is also a valid solution for (G, X, C, k). For contradiction, assume that $G - S$ contains an induced cycle D of length at least 4. Since $G - X$ is chordal, D contains at least one vertex of $X \setminus S$. In fact, since Rule 1 and Rule 2 cannot be applied and S contains at least one vertex of each critical pair, D contains at least three vertices $x, y, z \in X \setminus S$. If D contains neither u nor v, then D was also present in $G/uv - S$, contradicting the assumption that $G/uv - S$ is chordal. If D contains both u and v, then D is an induced cycle on at least 5 vertices in $G - S$, containing u, v, x, y, z. This means that $G/uv - S$ contains an induced cycle on at least 4 vertices, contradicting the assumption that $G/uv - S$ is chordal.

Consider finally the case where D contains either u or v, say u. At most two of the vertices x, y, z are adjacent to u, as D has no chords. Assume without loss of generality that $y \notin N_G(u)$, which implies that $y \notin N_G(v)$ since $y \in X$ and $N_G(u) \cap X = N_G(v) \cap X$. Let a, b be the predecessor and successor of vertex y on D, from which it follows that $ab \notin E(G)$. Since D is an induced cycle, it contains a walk from a to b none of whose interior vertices belongs to $N_G[y]$. If we substitute the occurrence of u on this walk by the vertex resulting from the contraction of u and v, we obtain a walk from a to b in G/uv none of whose interior vertices belongs to $N_{G/uv}[y]$, and none of the vertices of this walk are contained in S. By Proposition 1 this shows that $G/uv - S$ contains a chordless cycle of length at least 4, contradicting the assumption that $G/uv - S$ is chordal. We conclude that $G - S$ is chordal, and consequently that (G, X, C, k) is a yes-instance. □

With these first three rules, we are able to bound the length of any induced path in F.

Lemma 2. *If (G, X, C, k) is a reduced instance with respect to Rules 1-3, and P is an induced path in F, then P contains at most $2|X| + 1$ vertices.*

Proof. Suppose F contains an induced path $P = (p_1, \ldots, p_t)$. We say that an edge p_ip_{i+1} of P is *promoted* by a vertex $x \in X$ if $x \in N(p_i) \setminus N(p_{i+1})$ or $x \in N(p_{i+1}) \setminus N(p_i)$. For any two consecutive vertices p_i and p_{i+1} of P, we have that $N(p_i) \cap X \neq N(p_{i+1}) \cap X$, since Rule 3 cannot be applied. This means in particular that there is a vertex $x \in X$ such that $x \in N(p_i) \setminus N(p_{i+1})$ or $x \in N(p_{i+1}) \setminus N(p_i)$, for each i between 1 and $t-1$. Consequently, every edge of P is promoted by some vertex of X. Moreover, if a vertex $x \in X$ is adjacent to two vertices p_i and p_j of P with $i < j$, then x is also adjacent to each of the vertices $p_{i+1}, \ldots, p_{j-1}$, as otherwise $G[\{x\} \cup V(F)]$ would not be chordal, and Rule 1 would have been applicable. Thus, each vertex of X can promote at most two edges of P. Since every edge of P is promoted by some vertex of X, it follows that P contains at most $2|X|$ edges, and hence at most $2|X| + 1$ vertices. □

We now give the final reduction rule that will provide the polynomial bound on the size of a kernel.

Rule 4. *Repeat the following for each ordered triple (u, v, w) of distinct vertices in X: if there is an induced path P between u and w whose internal vertices are all in $F - N_G(v)$, then mark all the internal vertices of P. Let Y be the set of vertices of F that were not marked during this procedure. Reduce to the instance $(G - Y, X, C, k)$.*

Lemma 3. *Rule 4 is safe.*

Proof. Assume that Rules 1-3 cannot be applied, whereas Rule 4 can be applied on (G, X, C, k). Suppose (G, X, C, k) is a yes-instance, and let S be a valid solution. Since the class of chordal graphs is closed under taking induced subgraphs, and $G - S$ is chordal, we see that $G - Y - S$ is chordal. It follows that S is also a valid solution for $(G - Y, X, C, k)$.

For the reverse direction, suppose that $(G - Y, X, C, k)$ is a yes-instance, and let S be a valid solution for this instance. We will show that S is a valid solution for (G, X, C, k) as well. Assume for contradiction that it is not. This means that $G - Y - S$ is chordal, whereas $G - S$ has an induced cycle D of length at least 4. As $G - X$ is chordal and Rules 1-2 cannot be applied, D contains at least three vertices $x, y, z \in X \setminus S$. The subgraph $F' = F[V(D) \setminus X]$ is a disjoint union of induced paths. If each of the vertices of F' was marked during the application of Rule 4, then D is also an induced cycle in $G - Y - S$, which contradicts the assumption that $G - Y - S$ is chordal.

Suppose there is a path P in F' which contains an unmarked vertex. Let $x, z \in X$ be the two neighbors on D of the endpoints of P. Note that x and z belong to $X \setminus S$, and they are distinct since D contains at least three vertices of $X \setminus S$. Let $y \notin \{x, z\}$ be a third vertex of D which belongs to $X \setminus S$. Since D has no chords, the path (x, P, z) is an induced path from x to z whose interior vertices are contained in $F - N_G[y]$. Hence, when we tested the triple (x, y, z) in Rule 4, we found a path P' from x to z, not containing any neighbor of y, all whose vertices we marked. Let a and b be the predecessor and successor of y on D, which implies $ab \notin E(G)$. The cycle D contains a walk from a to b none of whose interior vertices belongs to $N_G[y]$. If we substitute the occurrence of P on this walk by P', we obtain a new walk W' from a to b, and since P' does not contain any vertex of $N_G[y]$, it follows that none of the interior vertices of W' belong to $N_G[y]$. By Proposition 1 this implies that G contains a chordless cycle D' whose vertices are a subset of $V(W') \cup \{y\}$. By construction, the new walk does not contain any vertex of S, so the chordless cycle exists in $G - S$. Since W' contains none of the unmarked vertices of P, the cycle D' contains strictly fewer of the unmarked vertices than the original cycle D. Consequently, after repeating this procedure at most $|D|$ times, we find an induced cycle D'' in $G - S$ containing no unmarked vertex of F, and at least four vertices: x, y, z and a marked vertex of F. Since all the vertices of $V(D'') \cap V(F)$ are marked, D'' is also an induced cycle in $G - Y - S$, which contradicts the assumption that $G - Y - S$ is chordal. □

We are now ready to state the kernel result on the annotated problem.

Theorem 2. Annotated Restricted Chordal Deletion $(|X|)$ *admits a kernel with at most* $2|X|^4 + |X|^3 + |X|$ *vertices.*

Proof. Rules 1-3 can trivially be applied in polynomial time. When we apply Rule 4, we need to test $\mathcal{O}(|X|^3)$ triples. For each triple (u, v, w), determining whether there is a path P from u to w whose internal vertices are contained in $F - N_G(v)$ can be done by simply trying to find a shortest path from u to w in the subgraph of G induced by u, w and the vertices of $F - N_G(v)$. Hence this rule can also be applied in polynomial time.

Let (G, X, C, k) be an instance that is reduced with respect to Rules 1-4. Observe that $G[F]$ can be covered by $|X|^3$ induced paths, and by Lemma 2, each such path contains at most $2|X| + 1$ vertices. Consequently, $|V(F)| \leq 2|X|^4 + |X|^3$. Since $V(G) = V(F) \cup X$, the result follows. □

Finally, the main result of this section is given in Theorem 3 below. Given an instance (G, X, k) of Restricted Chordal Deletion $(|X|)$, we immediately get an equivalent instance $(G, X, \emptyset, k)$ of Annotated Restricted Chordal Deletion $(|X|)$. From the latter, we can obtain an equivalent reduced instance (G', X, C, k), where G' has at most $2|X|^4 + |X|^3 + |X|$ vertices, by Theorem 2. In this instance, C is most likely not empty. The next theorem shows that we can turn this instance to an equivalent reduced instance of Restricted Chordal Deletion $(|X|)$ of slightly larger size. This is done by adding two new vertices to create a chordless cycle of length 4, for each pair of vertices in C.

Theorem 3. Restricted Chordal Deletion $(|X|)$ *admits a kernel with at most* $2|X|^4 + |X|^3 + |X|^2 + |X|$ *vertices.*

5 Conclusion

The reduction in the proof of Theorem 1 shows that it is possible to construct a small set of vertices X which models the universe of a Hitting Set instance, and that for every subset $X' \subseteq X$ we can add some vertices to create a hole in the graph which intersects X exactly in X', without creating other holes. This makes it possible to reduce HITTING SET (k) to PERFECT DELETION (k), while also giving a reduction from HITTING SET (n) to RESTRICTED PERFECT DELETION ($|X|$). Our positive result for RESTRICTED CHORDAL DELETION ($|X|$) shows that chordless cycles, the forbidden structures for chordal graphs, do not have the same modeling power.

It will be very interesting to settle the kernelization complexity of CHORDAL DELETION (k). We observe that this problem admits a linear-vertex kernel on planar and bounded-genus graphs: this follows from the meta-theorem by Bodlaender et al. [6], since the problem can be formulated in MSOL, has finite integer index, and is quasi-compact because chordal planar graphs have constant treewidth. Concerning structural parameterizations, it is not hard to prove that CHORDAL DELETION admits a polynomial kernel when parameterized by the size of a minimum vertex cover. A slightly more involved construction shows that the same good news holds for the parameterization by the size of a minimum feedback vertex set. As an intermediate step in obtaining a polynomial kernel for CHORDAL DELETION (k), one might consider CHORDAL DELETION parameterized by vertex deletion distance to an interval graph.

We conclude with some open questions regarding graph modification problems. It would be interesting to determine the FPT status of PERFECT EDGE DELETION/COMPLETION. Since the class of perfect graphs is closed under taking the complement, the deletion and completion problems are equally hard. The question of INTERVAL VERTEX DELETION has been open for some time. Maybe a study of AT-FREE VERTEX DELETION might shed some insight into this problem. Besides *computing* these modification sets, *using* them as parameters to other problems is also an interesting area which has not been thoroughly explored. For example, what is the status of FEEDBACK VERTEX SET parameterized by vertex deletion distance to a chordal graph, or INDUCED LONG PATH with the same parameter? These parameterizations have the potential to "beat treewidth", since a graph of large treewidth can nevertheless be close to chordal.

Acknowledgments. We would like to thank Dániel Marx for an insightful discussion on CHORDAL DELETION.

References

1. Abu-Khzam, F.N.: A kernelization algorithm for d-Hitting set. J. Comput. Syst. Sci. 76(7), 524–531 (2010)
2. Berge, C.: Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 10, 114 (1961)
3. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels (Extended abstract). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 563–574. Springer, Heidelberg (2008)

4. Bodlaender, H.L., Thomassé, S., Yeo, A.: Kernel bounds for disjoint cycles and disjoint paths. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 635–646. Springer, Heidelberg (2009)
5. Bodlaender, H.L., van Dijk, T.C.: A cubic kernel for feedback vertex set and loop cutset. Theory Comput. Syst. 46(3), 566–597 (2010)
6. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M. (Meta) Kernelization. In: Proc. 50th FOCS, pp. 629–638 (2009)
7. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: a survey. Society for Industrial and Applied Mathematics, Philadelphia (1999)
8. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. 58(4), 171–176 (1996)
9. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. 411, 3736–3756 (2010)
10. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. Annals of Mathematics 164, 51–229 (2006)
11. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through colors and iDs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
12. Downey, R., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
13. Fomin, F.V., Lokshtanov, D., Misra, N., Philip, G., Saurabh, S.: Hitting forbidden minors: Approximation and kernelization. In: Proc. 28th STACS, pp. 189–200 (2011)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Co., New York (1978)
15. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, Annals of Discrete Mathematics, vol. 57. North-Holland, Amsterdam (2004)
16. Guo, J., Moser, H., Niedermeier, R.: Iterative compression for exactly solving NP-hard minimization problems. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 65–80. Springer, Heidelberg (2009)
17. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
18. Kratsch, S., Schweitzer, P.: Isomorphism for graphs of bounded feedback vertex set number. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 81–92. Springer, Heidelberg (2010)
19. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. J. Comput. Syst. Sci. 20(2), 219–230 (1980)
20. Lokshtanov, D.: Wheel-free deletion is $W[2]$-hard. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 141–147. Springer, Heidelberg (2008)
21. Marx, D.: Chordal deletion is fixed-parameter tractable. Algorithmica 57(4), 747–768 (2010)
22. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 292–303. Springer, Heidelberg (2007)
23. Moser, H.: A problem kernelization for graph packing. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 401–412. Springer, Heidelberg (2009)
24. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. 32(4), 299–301 (2004)
25. Thomassé, S.: A quadratic kernel for feedback vertex set. ACM Transactions on Algorithms 6(2) (2010)

Constructive Dimension and Hausdorff Dimension: The Case of Exact Dimension

Ludwig Staiger

Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
von-Seckendorff-Platz 1, D-06099 Halle, Germany
staiger@informatik.uni-halle.de

Abstract. The present paper generalises results by Lutz and Ryabko. We prove a martingale characterisation of exact Hausdorff dimension. On this base we introduce the notion of exact constructive dimension of (sets of) infinite strings.

Furthermore, we generalise Ryabko's result on the Hausdorff dimension of the set of strings having asymptotic Kolmogorov complexity $\leq \alpha$ to the case of exact dimension.

The paper addresses a problem from Algorithmic Information Theory. In his papers [8,9] Lutz came up with an effectivisation of Hausdorff dimension, called constructive dimension. Constructive dimension characterises the algorithmic complexity of (sets of) infinite strings as real numbers. It turned out to be equivalent to asymptotic Kolmogorov complexity (cf. [20]) and is related to the concept of partial randomness of infinite strings [22,1]. However, the results of Reimann and Stephan [14] show, unlike the case of random infinite strings, different notions of Kolmogorov complexity (cf. [23,24]) yield different notions of partial randomness.

To distinguish these types of partial randomness requires a refinement of the complexity scale of (sets of) infinite strings. The present paper shows that an effectivisation of Hausdorff's original concept of dimension [7], referred to as exact Hausdorff dimension in [10,6,11], is possible and leads, similarly to the case of "usual" dimensions (cf. [15,16,18,19,8,9]), to close connections between exact Hausdorff dimension and exact constructive dimension. In contrast to the "usual" constructive or Hausdorff dimension an exact dimension of a string or a set of strings is a real function, referred to as gauge function [10,6,11]. This makes it more difficult to specify uniquely 'the' exact Hausdorff dimension of set of strings.

After introducing some notation, in Section 2, we present Hausdorff's original approach [7], give a definition of what is an exact Hausdorff dimension of a set and generalise the martingale characterisation of Hausdorff dimension [8,9].

In Section 3, using Levin's and Schnorr's (cf. [25,17]) optimal left computable super-martingale, we obtain in a natural way a definition of exact constructive dimension. Here we also derive the particularly interesting fact that the exact dimension of an infinite string ξ can be identified with Levin's [25] universal left computable continuous semi-measure $\mathbf{M}$ restricted to the set of finite prefixes of ξ.

It is well-known (cf. [23,24]) that Levin's semi-measure $\mathbf{M}$ yields the a priori complexity KA, a particular kind of Kolmogorov complexity. In the fourth section we generalise Ryabko's result that the set of infinite strings having asymptotic Kolmogorov complexity $\leq \alpha$ has Hausdorff dimension α and obtain, for the special case of the a priori complexity KA and for a large class of gauge functions, a similar coincidence in the case of exact dimensions.

Finally, in Section 5, we apply our results to the family of functions of the logarithmic scale, which was also considered by Hausdorff [7]. Here we give evidence that, unlike the case of asymptotic Kolmogorov complexity, the results involving exact dimensions depend on the kind of complexity (cf. [23,24]) we use. We show, in particular, that an analogous coincidence as proved in Section 4 does not hold for plain Kolmogorov complexity.

1 Notation and Preliminaries

In this section we introduce the notation used throughout the paper. By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the set of natural numbers and by $\mathbb{Q}$ the set of rational numbers. Let X be an alphabet of cardinality $|X| = r \geq 2$. By X^* we denote the set of finite words on X, including the *empty word* e, and X^ω is the set of infinite strings (ω-words) over X.

For $w \in X^*$ and $\eta \in X^* \cup X^\omega$ let $w \cdot \eta$ be their *concatenation*. This concatenation product extends in an obvious way to subsets $W \subseteq X^*$ and $B \subseteq X^* \cup X^\omega$.

We denote by $|w|$ the *length* of the word $w \in X^*$ and $\mathbf{pref}(B)$ is the set of all finite prefixes of strings in $B \subseteq X^* \cup X^\omega$. We shall abbreviate $w \in \mathbf{pref}(\eta)$ ($\eta \in X^* \cup X^\omega$) by $w \sqsubseteq \eta$, and $\eta[0..n]$ is the n-length prefix of η provided $|\eta| \geq n$. A language $W \subseteq X^*$ is referred to as *prefix-free* if $w \sqsubseteq v$ and $w, v \in W$ imply $w = v$. If $W \subseteq X^*$ then $\mathrm{Min}_\sqsubseteq W := \{w : w \in W \wedge \forall v (v \in W \rightarrow v \not\sqsubset w)\}$ is the (prefix-free) set of minimal w.r.t. $\sqsubseteq$ elements of W.

A *super-martingale* is a function $\mathcal{V} : X^* \rightarrow [0, \infty)$ which satisfies $\mathcal{V}(e) \leq 1$ and the super-martingale inequality

$$r \cdot \mathcal{V}(w) \geq \textstyle\sum_{x \in X} \mathcal{V}(wx) \text{ for all } w \in X^* . \tag{1}$$

If Eq. (1) is satisfied with equality $\mathcal{V}$ is called a martingale. Closely related with (super-)martingales are continuous (or cylindrical) (semi-)measures $\mu : X^* \rightarrow [0, 1]$ where $\mu(e) \leq 1$ and $\mu(w) \geq \sum_{x \in X} \mu(wx)$ for all $w \in X^*$.

Indeed, if $\mathcal{V}$ is a super-martingale then $\mu(w) := r^{-|w|} \cdot \mathcal{V}(w)$ is a continuous (semi-)measure, and vice versa. It should be mentioned that for any continuous semi-measure μ and every prefix-free subset $W \subseteq X^*$ the inequality $\sum_{w \in W} \mu(w) \leq 1$ holds. This proves also the corresponding super-martingale inequality for prefix-free sets $W \subseteq X^*$:

$$\mathcal{V}(e) \geq \textstyle\sum_{w \in W} r^{-|w|} \cdot \mathcal{V}(w) \tag{2}$$

For a computable domain $\mathcal{D}$, such as $\mathbb{N}$, $\mathbb{Q}$ or X^*, we refer to a function $f : \mathcal{D} \to \mathbb{R}$ as *left computable* (or *approximable from below*) provided the set $\{(d, q) : d \in \mathcal{D} \wedge q \in \mathbb{Q} \wedge q < f(d)\}$ is computably enumerable. Accordingly, a function $f : \mathcal{D} \to \mathbb{R}$ is called *right computable* (or *approximable from above*) if the set $\{(d, q) : d \in \mathcal{D} \wedge q \in \mathbb{Q} \wedge q > f(d)\}$ is computably enumerable, and f is *computable* if f is right and left computable.

If we refer to a function $f : \mathcal{D} \to \mathbb{Q}$ as computable we usually mean that it maps the domain $\mathcal{D}$ to the domain $\mathbb{Q}$, that is, it returns the exact value $f(d) \in \mathbb{Q}$.

2 Hausdorff's Approach

A function $h : (0, \infty) \to (0, \infty)$ is referred to as a *gauge function* provided h is positive, right continuous and non-decreasing. The h-dimensional outer measure of a set $F \subseteq X^\omega$ on the space X^ω is given by

$$\mathcal{H}^h(F) := \lim_{n\to\infty} \inf\Big\{ \sum_{v \in V} h(r^{-|v|}) : V \subseteq X^* \wedge F \subseteq V \cdot X^\omega \wedge \min_{v \in V} |v| \geq n \Big\} . \quad (3)$$

If $\lim_{t\to 0} h(t) > 0$ then $\mathcal{H}^h(F) < \infty$ if and only if F is finite.

The usual α-dimensional Hausdorff measure $\mathcal{H}^\alpha$ is defined by the family of gauge functions $h_\alpha(t) = t^\alpha$, that is, $\mathcal{H}^\alpha = \mathcal{H}^{h_\alpha}$. Here $h_0(t) = t^0$ defines the counting measure on X^ω.

In this case it is possible to define the (usual) Hausdorff dimension of a set $F \subseteq X^\omega$ as

$$\dim_{\mathrm{H}} F := \sup\{\alpha : \alpha = 0 \vee \mathcal{H}^\alpha(F) = \infty\} = \inf\{\alpha : \alpha \geq 0 \wedge \mathcal{H}^\alpha(F) = 0\} . \quad (4)$$

As we see from Eq. (3) for our purposes the behaviour of gauge function is of interest only in a small vicinity of 0. Moreover, in many cases we are not interested in the exact value of $\mathcal{H}^h(F)$ when $0 < \mathcal{H}^h(F) < \infty$. Thus we can often make use of scaling a gauge function and altering it in a range $(\varepsilon, 1]$ apart from 0.

The following properties of gauge functions h and the related measure $\mathcal{H}^h$ are proved in the standard way (see e.g. [4,5]).

Property 1. Let h, h' be gauge functions.

1. If $c_1 \cdot h(r^{-n}) \leq h'(r^{-n}) \leq c_2 \cdot h(r^{-n})$ for some $c_1, c_2,\ 0 < c_1 \leq c_2$, then $c_1 \cdot \mathcal{H}^h(F) \leq \mathcal{H}^{h'}(F) \leq c_2 \cdot \mathcal{H}^h(F)$.
2. If $\lim\limits_{n\to\infty} \frac{h(r^{-n})}{h'(r^{-n})} = 0$ then $\mathcal{H}^{h'}(F) < \infty$ implies $\mathcal{H}^h(F) = 0$, and $\mathcal{H}^h(F) > 0$ implies $\mathcal{H}^{h'}(F) = \infty$.

Here the first property could be called equivalence of gauge functions. In fact, if h and h' are equivalent in the sense of Property 1 then for all $F \subseteq X^\omega$ the measures $\mathcal{H}^h(F)$ and $\mathcal{H}^{h'}(F)$ are both zero, finite or infinite. In the same way the second property gives an pre-order of gauge functions. The pre-order is denoted by $\prec$ where $h' \prec h$ is an abbreviation for $\lim\limits_{n\to\infty} \frac{h(r^{-n})}{h'(r^{-n})} = 0$, that is, $h(r^{-n})$ tends faster to 0 than $h'(r^{-n})$ as n tends to infinity.

By analogy to the change-over-point $\dim_{\mathrm{H}} F$ (see Eq. (4)) for $\mathcal{H}^{\alpha}(F)$ the partial pre-order $\prec$ yields a suitable notion of Hausdorff dimension in the range of arbitrary gauge functions.

Definition 1. We refer to a gauge function h as *exact Hausdorff dimension function* for $F \subseteq X^{\omega}$ provided

$$\mathcal{H}^{h'}(F) = \begin{cases} \infty\,, & \text{if } h' \prec h \text{ , and} \\ 0\,, & \text{if } h \prec h' \,. \end{cases}$$

Remark that, since $\prec$ is not a total ordering, nothing is said about the measure $\mathcal{H}^{h'}(F)$ for functions h' which are equivalent or not comparable to h. Hausdorff called a function h *dimension* of F provided $0 < \mathcal{H}^{h}(F) < \infty$. This case is covered by our definition and Property 1.

One easily observes that $h_0(t) := t$ yields $\mathcal{H}^{h_0}(F) \leq 1$, thus $\mathcal{H}^{h'}(F) = 0$ for all $h',\ h_0 \prec h'$. Therefore, we can always assume that a gauge function satisfies $h(t) > t^2,\ t \in (0,1)$.

2.1 Exact Hausdorff Dimension and Martingales

In this section we show a generalisation of Lutz's theorem to arbitrary gauge functions. To obtain a transparent notation we do not use Lutz's s-gale notation but instead we follow Schnorr's approach of combining martingales with order functions. For a discussion of both approaches see Section 13.2 of [3].

Let, for a super-martingale $\mathcal{V} : X^* \to [0,\infty)$, a gauge function h and a value $c \in (0,\infty]$ be $S_{c,h}[\mathcal{V}] := \left\{\xi : \xi \in X^{\omega} \wedge \limsup_{n\to\infty} \frac{\mathcal{V}(\xi[0..n])}{r^n \cdot h(r^{-n})} \geq c\right\}$. In particular, $S_{\infty,h}[\mathcal{V}]$ is the set of all ω-words on which the super-martingale $\mathcal{V}$ is successful w.r.t. the order function $f(n) = r^n \cdot h(r^{-n})$ in the sense of Schnorr [17].

Now we can prove the analogue to Lutz's theorem. In view of Property 1 we split the assertion into two parts.

Lemma 1. *Let $F \subseteq X^{\omega}$ and h, h' be gauge functions such that $h \prec h'$ and $\mathcal{H}^{h}(F) < \infty$. Then $F \subseteq S_{\infty,h'}[\mathcal{V}]$ for some martingale $\mathcal{V}$.*

Proof. First we follow the lines of the proof of Theorem 13.2.3 in [3] and show the assertion for $\mathcal{H}^{h}(F) = 0$. Thus there are prefix-free subsets $U_i \subseteq X^*$ such that $F \subseteq \bigcap_{i\in\mathbb{N}} U_i \cdot X^{\omega}$ and $\sum_{u\in U_i} h(r^{-|u|}) \leq 2^{-i}$.

Define $\mathcal{V}_i(w) := \begin{cases} r^{|w|} \cdot \sum_{wu\in U_i} h(r^{-|wu|}), & \text{if } w \in \mathbf{pref}(U_i) \setminus U_i \text{ , and} \\ \sup\{r^{|v|} \cdot h(r^{-|v|}) : v \sqsubseteq w \wedge v \in U_i\}, & \text{otherwise}^{1}. \end{cases}$

In order to prove that $\mathcal{V}_i$ is a martingale we consider three cases:

$w \in \mathbf{pref}(U_i) \setminus U_i$**:** Since then $U_i \cap w \cdot X^* = \bigcup_{x\in X} U_i \cap wx \cdot X^*$, we have $\mathcal{V}_i(w) = r^{|w|} \cdot \sum_{wu\in U_i} h(r^{-|wu|}) = r^{-1} \cdot \sum_{x\in X} r^{|wx|} \sum_{wxu\in U_i} h(r^{-|wxu|}) = r^{-1} \cdot \sum_{x\in X} \mathcal{V}_i(wx)$.

[1] This yields $\mathcal{V}_i(w) = 0$ for $w \notin \mathbf{pref}(U_i) \cup U_i \cdot X^*$.

$w \in U_i \cdot X^*$ **:** Let $w \in v \cdot X^*$ where $v \in U_i$. Then $\mathcal{V}_i(w) = \mathcal{V}_i(wx) = r^{|v|} \cdot h(r^{-|v|})$ whence $\mathcal{V}_i(w) = r^{-1} \cdot \sum_{x \in X} \mathcal{V}_i(wx)$.
$w \notin \mathbf{pref}(U_i) \cup U_i \cdot X^*$ **:** Here $\mathcal{V}_i(w) = \mathcal{V}_i(wx) = 0$.

Now, set $\mathcal{V}(w) := \sum_{i \in \mathbb{N}} \mathcal{V}_i(w)$.

Then, for $\xi \in \bigcap_{i \in \mathbb{N}} U_i \cdot X^\omega$ there are $n_i \in \mathbb{N}$ such that $\xi[0..n_i] \in U_i$ and we obtain $\frac{\mathcal{V}(\xi[0..n_i])}{r^{n_i} \cdot h'(r^{-n_i})} \geq \frac{\mathcal{V}_i(\xi[0..n_i])}{r^{n_i} \cdot h'(r^{-n_i})} = \frac{h(r^{-n_i})}{h'(r^{-n_i})}$ which tends to infinity as i tends to infinity.

Now let $\mathcal{H}^h(F) < \infty$. Then $h \prec \sqrt{h \cdot h'} \prec h'$. Thus $\mathcal{H}^{\sqrt{h \cdot h'}}(F) = 0$ and we can apply the first part of the proof to the functions $\sqrt{h \cdot h'}$ and h'. □

The next lemma is in some sense a converse to Lemma 1.

Lemma 2. *Let h be a gauge function, $c \in (0, \infty]$ and $\mathcal{V}$ be a super-martingale. Then $\mathcal{H}^h(S_{c,h}[\mathcal{V}]) \leq \frac{\mathcal{V}(e)}{c}$.*

Proof. It suffices to prove the assertion for $c < \infty$.

Define $V_k := \{w : w \in X^* \wedge |w| \geq k \wedge \frac{\mathcal{V}(w)}{r^{|w|} \cdot h(r^{-|w|})} \geq c - 2^{-k}\}$ and set $U_k := \mathrm{Min}_{\sqsubseteq} V_k$. Then $S_{c,h}[\mathcal{V}] \subseteq \bigcap_{k \in \mathbb{N}} U_k \cdot X^\omega$.

Now $\sum\limits_{w \in U_k} h(r^{-|w|}) \leq \sum\limits_{w \in U_k} h(r^{-|w|}) \cdot \frac{\mathcal{V}(w)}{r^{|w|} \cdot h(r^{-|w|})} \cdot \frac{1}{c - 2^{-k}} = \frac{1}{c - 2^{-k}} \cdot \sum\limits_{w \in U_k} \frac{\mathcal{V}(w)}{r^{|w|}} \leq \frac{\mathcal{V}(e)}{c - 2^{-k}}$ (cf. Eq. (2)). Thus $\mathcal{H}^h(\bigcap_{k \in \mathbb{N}} U_k \cdot X^\omega) \leq \frac{\mathcal{V}(e)}{c}$. □

Lemmata 1 and 2 yield the following martingale characterisation of exact Hausdorff dimension functions.

Theorem 1. *Let $F \subseteq X^\omega$. Then a gauge function h is an exact Hausdorff dimension function for F if and only if*

1. *for all gauge functions h' with $h \prec h'$ there is a super-martingale $\mathcal{V}$ such that $F \subseteq S_{\infty,h'}[\mathcal{V}]$, and*
2. *for all gauge functions h'' with $h'' \prec h$ and all super-martingales $\mathcal{V}$ it holds $F \not\subseteq S_{\infty,h''}[\mathcal{V}]$.*

Proof. Assume h to be exact for F and $h \prec h'$. Then $h \prec \sqrt{h \cdot h'} \prec h'$. Thus $\mathcal{H}^{\sqrt{h \cdot h'}}(F) = 0$ and applying Lemma 1 to $\sqrt{h \cdot h'}$ and h' yields a super-martingale $\mathcal{V}$ such that $F \subseteq S_{\infty,h'}[\mathcal{V}]$.

If $h'' \prec h$ then $\mathcal{H}^{h''}(F) = \infty$ and according to Lemma 2 $F \not\subseteq S_{\infty,h''}[\mathcal{V}]$ for all super-martingales $\mathcal{V}$.

Conversely, let Conditions 1 and 2 be satisfied. Let $h \prec h'$, and let $\mathcal{V}$ be a super-martingale such that $F \subseteq S_{\infty,h'}[\mathcal{V}]$. Now Lemma 2 shows $\mathcal{H}^{h'}(F) \leq \mathcal{H}^{h'}(S_{\infty,h'}[\mathcal{V}]) = 0$.

Finally, suppose $h'' \prec h$ and $\mathcal{H}^{h''}(F) < \infty$. Then $\mathcal{H}^{\sqrt{h \cdot h''}}(F) = 0$ and Lemma 1 shows that there is a super-martingale $\mathcal{V}$ such that $F \subseteq S_{\infty,\sqrt{h \cdot h''}}[\mathcal{V}]$. This contradicts Condition 2. □

Lemmata 1 and 2 also show that we can likewise formulate Theorem 1 for martingales instead of super-martingales.

3 Constructive Dimension: The Exact Case

The constructive dimension is a variant of dimension defined analogously to Theorem 1 using only left computable super-martingales. For the usual family of gauge functions $h_\alpha(t) = t^\alpha$ it was introduced by Lutz [8] and resulted, similarly to $\dim_{\mathrm{H}}$ in a real number assigned to a subset $F \subseteq X^\omega$. In the case of left computable super-martingales the situation turned out to be simpler because the results of Levin [25] and Schnorr [17] show that there is an optimal left computable super-martingale $\mathcal{U}$, that is, every other left computable super-martingale $\mathcal{V}$ satisfies $\mathcal{V}(w) \leq c_{\mathcal{V}} \cdot \mathcal{U}(w)$ for all $w \in X^*$ and some constant $c_{\mathcal{V}} > 0$ not depending on w. Thus we may define

Definition 2. *Let $F \subseteq X^\omega$. We refer to $h : \mathbb{R} \to \mathbb{R}$ as an* exact constructive dimension function *for F provided $F \subseteq S_{\infty,h'}[\mathcal{U}]$ for all $h', h \prec h'$ and $F \not\subseteq S_{\infty,h''}[\mathcal{U}]$ for all $h'', h'' \prec h$.*

Originally, Levin showed that there is an optimal left computable continuous semi-measure $\mathbf{M}$ on X^*.

Thus we might use $\mathcal{U}_{\mathbf{M}}$ with $\mathcal{U}_{\mathbf{M}}(w) := r^{|w|} \cdot \mathbf{M}(w)$ as our optimal left computable super-martingale. The proof of the next theorem makes use of this fact and of the inequality $\mathbf{M}(w) \geq \mathbf{M}(w \cdot v)$.

Theorem 2. *The function h_ξ defined by $h_\xi(r^{-n}) := \mathbf{M}(\xi[0..n])$ is an exact constructive dimension function for the set $\{\xi\}$.*

Closely related to Levin's optimal left computable semi-measure is the *a priori entropy* (or *complexity*) $\mathrm{KA} : X^* \to \mathbb{N}$ defined by

$$\mathrm{KA}(w) := \lfloor -\log_r \mathbf{M}(w) \rfloor \tag{5}$$

First we mention the following bound from [12].

Theorem 3. *Let $F \subseteq X^\omega$, h be a gauge function and $\mathcal{H}^h(F) > 0$.*

Then for every $c > 0$ with $\mathcal{H}^h(F) > c \cdot \mathbf{M}(e)$ there is a $\xi \in F$ such that $\mathrm{KA}(\xi[0..n]) \geq_{\mathrm{ae}} -\log_r h(r^{-n}) - \log_r c$.

This lower bound on the maximum complexity of an infinite string in F yields a set-theoretic lower bound on the success sets $S_{c,h}[\mathcal{U}]$ of $\mathcal{U}$.

Theorem 4. *Let $-\infty < c < \infty$ and let h be a gauge function. Then there is a $c' > 0$ such that*

$$\{\xi : \exists^\infty n(\mathrm{KA}(\xi[0..n]) \leq \log_r h(r^{-n}) + c)\} \subseteq S_{c',h}[\mathcal{U}].$$

Proof. If ξ has infinitely many prefixes such that $\mathrm{KA}(\xi[0..n]) \leq -\log_r h(r^{-n}) + c$ then, since $\mathcal{U}(w) \geq c'' \cdot r^n \cdot \mathbf{M}(w)$ for a suitable $c'' > 0$, we obtain in view of Eq. (5) $\limsup_{n\to\infty} \frac{\mathcal{U}(\xi[0..n])}{r^n \cdot h(r^{-n})} \geq \limsup_{n\to\infty} \frac{c'' \cdot r^n \cdot \mathbf{M}(\xi[0..n])}{r^n \cdot h(r^{-n})} \geq c'' \cdot r^{-c-1}$. □

Corollary 1. *Let h, h' be gauge functions such that $h \prec h'$ and $c \in \mathbb{R}$. Then*

1. *$\{\xi : \mathrm{KA}(\xi[0..n]) \leq_{\mathrm{io}} \log_r h(r^{-n}) + c\} \subseteq S_{\infty,h'}[\mathcal{U}]$, and*
2. *$\mathcal{H}^{h'}\left(\{\xi : \mathrm{KA}(\xi[0..n]) \leq_{\mathrm{io}} -\log_r h(r^{-n}) + c\}\right) = 0$.*

4 Complexity

In this section we are going to show that, analogously to Ryabko's and Lutz's results for the "usual" dimension the bound given in Corollary 1 is tight for a large class of (computable) gauge functions. To this end we prove that certain sets of infinite strings diluted according to a gauge function h have positive Hausdorff measure $\mathcal{H}^h$.

4.1 A Generalised Dilution Principle

We are going to show that for a large family of gauge functions, a set of finite positive measures can be constructed. Our construction is a generalisation of Hausdorff's 1918 construction. Instead of his method of cutting out middle thirds in the unit interval we use the idea of dilution functions as presented in [21]. In fact dilution appears much earlier (see e.g. [2,18,9]).

We consider *prefix-monotone* mappings, that is, mappings $\varphi : X^* \to X^*$ satisfying $\varphi(w) \sqsubseteq \varphi(v)$ whenever $w \sqsubseteq v$. We call a function $g : \mathbb{N} \to \mathbb{N}$ a *modulus function* for φ provided $|\varphi(w)| = g(|w|)$ for all $w \in X^*$. This, in particular, implies that $|\varphi(w)| = |\varphi(v)|$ for $|w| = |v|$ when φ has a modulus function.

Every prefix-monotone mapping $\varphi : X^* \to X^*$ defines as a limit a partial mapping $\overline{\varphi} :\subseteq X^\omega \to X^\omega$ in the following way: $\mathbf{pref}(\overline{\varphi}(\xi)) = \mathbf{pref}(\varphi(\mathbf{pref}(\xi)))$ whenever $\varphi(\mathbf{pref}(\xi))$ is an infinite set, and $\overline{\varphi}(\xi)$ is undefined when $\varphi(\mathbf{pref}(\xi))$ is finite.

If, for some strictly increasing function $g : \mathbb{N} \to \mathbb{N}$, the mapping φ satisfies the conditions $|\varphi(w)| = g(|w|)$ and for every $v \in \mathbf{pref}(\varphi(X^*))$ there are $w_v \in X^*$ and $x_v \in X$ such that

$$\varphi(w_v) \sqsubset v \sqsubseteq \varphi(w_v \cdot x_v) \wedge \forall y \big(y \in X \wedge y \neq x_v \to v \not\sqsubseteq \varphi(w_v \cdot y)\big) \tag{6}$$

then we call φ a *dilution function* with modulus g. If φ is a dilution function then $\overline{\varphi}$ is a one-to-one mapping.

For the image $\overline{\varphi}(X^\omega)$ we obtain the following bounds on its Hausdorff measure.

Theorem 5. *Let $g : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function, φ a corresponding dilution function and $h : (0, \infty) \to (0, \infty)$ be a gauge function. Then*

1. $\mathcal{H}^h(\overline{\varphi}(X^\omega)) \leq \liminf_{n\to\infty} \frac{h(r^{-g(n)})}{r^{-n}}$
2. *If $c \cdot r^{-n} \leq_{\text{ae}} h(r^{-g(n)})$ then $c \leq \mathcal{H}^h(\overline{\varphi}(X^\omega))$.*

Proof. The first assertion follows from $\overline{\varphi}(X^\omega) \subseteq \bigcup_{|w|=n} \varphi(w) \cdot X^\omega$ and $|\varphi(w)| = g(|w|)$.

The second assertion is obvious for $\mathcal{H}^h(\overline{\varphi}(X^\omega)) = \infty$. Let $\mathcal{H}^h(\overline{\varphi}(X^\omega)) < \infty$, $\varepsilon > 0$, and $V \cdot X^\omega \supseteq \overline{\varphi}(X^\omega)$ such that $\sum_{v\in V} h(r^{-|v|}) \leq \mathcal{H}^h(\overline{\varphi}(X^\omega)) + \varepsilon$. The set $W_V := \{w_v \cdot x_v : v \in V \wedge \varphi(w_v) \sqsubset v \sqsubseteq \varphi(w_v \cdot x_v)\}$ (see Eq. (6)) is prefix-free and it holds $W_V \cdot X^\omega \supseteq X^\omega$. Thus W_V is finite and $\sum_{w\in W_V} r^{-|w|} = 1$.

Assume now $\min\{|v| : v \in V\}$ large enough such that $c \cdot r^{-|v|} \leq_{\mathrm{ae}} h(r^{-|v|})$ for all $v \in V$.

Then $\sum_{v\in V} h(r^{-|v|}) \geq \sum_{wx\in W_V} h(r^{-|\varphi(wx)|}) = \sum_{wx\in W_V} h(r^{-g(|wx|)})$
$\geq \sum_{wx\in W_V} c \cdot r^{-|wx|} = c\,.$

As $\varepsilon > 0$ is arbitrary, the assertion follows. □

Corollary 2. *If* $c \cdot r^{-n} \leq_{\mathrm{ae}} h(r^{-g(n)}) \leq c' \cdot r^{-n}$ *then* $c \leq \mathcal{H}^h(\overline{\varphi}(X^\omega)) \leq c'$.

In connection with Theorem 5 and Corollary 2 it is of interest which gauge functions allow for a construction of a set of positive finite measure via dilution. Hausdorff's cutting out was demonstrated for upwardly convex[2] gauge functions. We consider the slightly more general case of functions fulfilling the following.

Lemma 3. *If a gauge function* h *is upwardly convex on some interval* $(0, \varepsilon)$ *and* $\lim_{t\to 0} h(t) = 0$ *then there is an* $n_0 \in \mathbb{N}$ *such that for all* $n \geq n_0$ *there is an* $m \in \mathbb{N}$ *satisfying*

$$r^{-n} < h(r^{-m}) \leq r^{-n+1}\,. \tag{7}$$

In particular, Eq. (7) implies that the gauge function h does not tend faster to 0 than the identity function $\mathrm{id} : \mathbb{R} \to \mathbb{R}$.

Proof. If h is monotone, upwardly convex on $(0, \varepsilon)$ and $h(0) = 0$ then, in particular, $h(\gamma) \geq \gamma \cdot h(\gamma')/\gamma'$ whenever $0 \leq \gamma \leq \gamma' \leq \varepsilon$. Let $n \in \mathbb{N}$ and let $m \in \mathbb{N}$ be the largest number such that $r^{-n} < h(r^{-m})$. Then $h(r^{-m-1}) \leq r^{-n} < h(r^{-m}) \leq r \cdot h(r^{-m-1}) \leq r^{-n+1}$. □

Remark 1. Using the scaling factor $c = r^{n_0}$, that is, considering $c \cdot h$ instead of h and taking $h'(t) = \min\{c \cdot h(t), r\}$ one can always assume that $n_0 = 0$ and $h'(1) > 1$. Defining then $g(n) := \max\{m : m \in \mathbb{N} \wedge r^{-n} < h(r^{-m})\}$ we obtain via Property 1 and Corollary 2 that for every gauge function h fulfilling Eq. (7) there is a subset F_h of X^ω having finite and positive $\mathcal{H}^h$-measure.

4.2 Computable Gauge Functions

The aim of this section is to show that the modulus function g and thus the dilution function φ can be chosen computable if the gauge function h fulfilling Eq. (7) is computable.

Lemma 4. *Let* $h : \mathbb{Q} \to \mathbb{R}$ *be a computable gauge function satisfying the conditions that* $1 < h(1) < r$ *and for every* $n \in \mathbb{N}$ *there is an* $m \in \mathbb{N}$ *such that* $r^{-n} < h(r^{-m}) \leq r^{-n+1}$. *Then there is a computable strictly increasing function* $g : \mathbb{N} \to \mathbb{N}$ *such that* $r^{-n-1} < h(r^{-g(n)}) < r^{-n+1}$.

Proof. We define g inductively. To this end we compute for every $n \geq 1$ a closed interval I_n such that $h(r^{-g(n)}) \in I_n \subset (r^{-n}, \min I_{n-1})$.

[2] A function $f : \mathbb{R} \to \mathbb{R}$ is called *upwardly convex* if $f(a + t(b - a)) \geq f(a) + t(f(b) - f(a))$ for all $t \in [0, 1]$.

We start with $g(0) := 0$ and $I_{-1} = [r, r+1]$ and estimate I_0 as an sufficiently small approximating interval of $h(r^{-g(0)}) > 1$ satisfying $I_0 \subseteq (1, r)$.

Assume now that for n the value $g(n)$ and the interval I_n satisfying $h(r^{-g(n)}) \in I_n \subset (r^{-n}, \min I_{n-1})$ are computed.

We search for an m and an approximating interval $I(m)$, $h(r^{-m}) \in I(m)$, such that $I(m) \subset (r^{-n-1}, \min I_n)$. Since $\liminf_{m\to\infty} h(r^{-m}) = 0$ and $\exists m(r^{-n-1} < h(r^{-m}) \leq r^{-n}) < \min I_n$ this search will eventually be successful. Define $g(n+1)$ as the first such m found by our procedure and set $I_n := I(m)$.

Finally, the monotonicity of h implies $g(n+1) > g(n)$. □

With Corollary 2 we obtain the following.

Corollary 3. *Under the hypotheses of Lemma 4 there is a computable dilution function $\varphi : X^* \to X^*$ such that $r^{-1} \leq \mathcal{H}^h(\overline{\varphi}(X^\omega)) \leq r$.*

4.3 Complexity of Diluted Infinite Strings

In the final part of this section we show that, for a large class of computable gauge functions, the set $\{\xi : \mathrm{KA}(\xi[0..n]) \leq_{\mathrm{io}} -\log_r h(r^{-n})+c\}$ (see Corollary 1) has the function h as an exact dimension function, that is, a converse to Corollary 1.2.

We use the following estimate on the a priori complexity of a diluted string from [21].

Theorem 6. *Let $\varphi : X^* \to X^*$ be a one-to-one prefix-monotone recursive function satisfying Eq (6) with strictly increasing modulus function g. Then*
$$\big|\mathrm{KA}\big(\overline{\varphi}(\xi)[0..g(n)]\big) - \mathrm{KA}\big(\xi[0..n]\big)\big| \leq O(1) \text{ for all } \xi \in X^\omega .$$

This auxiliary result yields that certain sets of non-complex strings have non-null h-dimensional Hausdorff measure.

Theorem 7. *If $h : \mathbb{Q} \to \mathbb{R}$ is a computable gauge function satisfying Eq. (7) then there is a $c \in \mathbb{N}$ such that*
$$\mathcal{H}^h(\{\zeta : \mathrm{KA}(\zeta[0..\ell]) \leq_{\mathrm{ae}} -\log_r h(r^{-\ell}) + c\}) > 0.$$

Proof. From the gauge function h we construct a computable dilution function φ with modulus function g such that $r^{-(l+k+1)} < g(r^{-g(l)}) < r^{-(l+k-1)}$ for a suitable constant k (cf. Lemma 4 and Remark 1). Then, according to Corollary 3, $\mathcal{H}^h(\overline{\varphi}(X^\omega)) > 0$.

Using Theorem 6 we obtain $\mathrm{KA}\big(\overline{\varphi}(\xi)[0..g(l)]\big) \leq \mathrm{KA}\big(\xi[0..l]\big) + c_1 \leq l + c_2$ for suitable constants $c_1, c_2 \in \mathbb{N}$. Let $n \in \mathbb{N}$ satisfy $g(l) < n \leq g(n+1)$. Then $\mathrm{KA}\big(\overline{\varphi}(\xi)[0..n]\big) \leq \mathrm{KA}\big(\overline{\varphi}(\xi)[0..g(l+1)]\big) \leq l + 1 + c_2$.

Now from $l + k - 1 < -\log_r h(r^{-g(l)}) \leq -\log_r h(r^{-n})$ we obtain the assertion $\mathrm{KA}\big(\overline{\varphi}(\xi)[0..n]\big) \leq -\log_r h(r^{-n}) + k + c_2$. □

Now Corollary 1.2 and Theorem 7 the following analogue to Ryabko's [15] result.

Lemma 5. *If $h : \mathbb{Q} \to \mathbb{R}$ is a computable gauge function satisfying Eq. (7) then there is a $c \in \mathbb{N}$ such that h is an exact Hausdorff dimension for the sets $\{\xi : \mathrm{KA}(\xi[0..n]) \leq_{\mathrm{io}} -\log_r h(r^{-n}) + c\}$ and $\{\zeta : \mathrm{KA}(\zeta[0..\ell]) \leq_{\mathrm{ae}} -\log_r h(r^{-\ell}) + c\}$.*

5 Functions of the Logarithmic Scale

The final part of this paper is devoted to a generalisation of the "usual" dimensions using Hausdorff's family of functions of the logarithmic scale. This family is, similarly to the family $h_\alpha(t) = t^\alpha$, also linearly ordered and, thus, allows for more specific versions of Corollary 1.2 and Theorem 7.

A function of the form where the first non-zero exponent satisfies $p_i > 0$

$$h_{(p_0,\ldots,p_k)}(t) = t^{p_0} \cdot \prod_{i=1}^{k} \left(\log^i \tfrac{1}{t}\right)^{p_i} \tag{8}$$

is referred to as a *function of the logarithmic scale* (see [7]). Here we have the convention that $\log^i t = \max\{\underbrace{\log_r \ldots \log_r}_{i \text{ times}} t\ , 1\}$.

One observes that the lexicographic order on the tuples $(p_0, \ldots, p_k)$ yields an order of the functions $h_{(p_0,\ldots,p_k)}$ in the sense that $(p_0, \ldots, p_k) >_{\mathrm{lex}} (q_0, \ldots, q_k)$ if and only if $h_{(q_0,\ldots,q_k)}(t) \prec h_{(p_0,\ldots,p_k)}(t)$.

This gives rise to a generalisation of the "usual" Hausdorff dimension as follows.

$$\begin{aligned} \dim_{\mathrm{H}}^{(k)} F &:= \sup\{(p_0, \ldots, p_k) : \mathcal{H}^{h_{(p_0,\ldots,p_k)}}(F) = \infty\} \\ &= \inf\{(p_0, \ldots, p_k) : \mathcal{H}^{h_{(p_0,\ldots,p_k)}}(F) = 0\} \end{aligned} \tag{9}$$

When taking supremum or infimum we admit also values $-\infty$ and ∞ although we did not define the corresponding functions of the logarithmic scale. E.g. $\dim_{\mathrm{H}}^{(1)} F = (0, \infty)$ means that $\mathcal{H}^{h_{(0,\gamma)}}(F) = \infty$ but $\mathcal{H}^{h_{(\alpha,-\gamma)}}(F) = 0$ for all $\gamma \in (0, \infty)$ and all $\alpha > 0$.

The following theorems generalise Ryabko's [15] result on the "usual" Hausdorff dimension (case $k = 0$) of the set of strings having asymptotic Kolmogorov complexity $\leq p_0$.

Let $h_{(p_0,\ldots,p_k)}$ be a function of the logarithmic scale. We define its first logarithmic truncation as $\beta_h(t) := -\log_r h_{(p_0,\ldots,p_{k-1})}$. Observe that $\beta_h(r^{-n}) = p_0 \cdot n + \sum_{i=1}^{k-1} p_i \cdot \log^i n$ and $-\log h_{(p_0,\ldots,p_k)}(r^{-n}) = \beta_h(r^{-n}) + p_k \cdot \log^k n$, for sufficiently large $n \in \mathbb{N}$.

Then from Corollary 1.2 we obtain the following result.

Theorem 8 ([13]). *Let $k > 0$, $(p_0, \ldots, p_k)$ be a $(k+1)$-tuple and $h_{(p_0,\ldots,p_k)}$ be a function of the logarithmic scale. Then*

$$\dim_{\mathrm{H}}^{(k)} \left\{\xi : \xi \in X^\omega \wedge \liminf_{n\to\infty} \frac{\mathrm{KA}(\xi[0..n]) - \beta_h(2^{-n})}{\log^k n} < p_k\right\} \leq (p_0, \ldots, p_k)\,.$$

Proof. From $\liminf_{n\to\infty} \frac{\mathrm{KA}(\xi[0..n]) - \beta_h(2^{-n})}{\log^k n} < p_k$ follows $\mathrm{KA}(\xi[0..n]) \leq \beta_h(2^{-n}) + p'_k \cdot \log^k n + O(1)$ for some $p'_k < p_k$. Thus $h_{(p_0,\ldots,p'_k)} \prec h_{(p_0,\ldots,p_k)}$ and the assertion follows from Corollary 1.2. $\square$

Using Theorem 7 we obtain a partial converse to Theorem 8 slightly refining Satz 4.11 of [13].

Theorem 9. *Let $k > 0$, $(p_0, \ldots, p_k)$ be a $(k+1)$-tuple where $p_0 > 0$ and $p_0, \ldots, p_{k-1}$ are computable numbers. Then for the function $h_{(p_0,\ldots,p_k)}$ it holds*

$$\dim_{\mathrm{H}}^{(k)} \left\{ \xi : \xi \in X^\omega \wedge \limsup_{n\to\infty} \frac{\mathrm{KA}(\xi[0..n]) - \beta_h(2^{-n})}{\log^k n} \le p_k \right\} = (p_0, \ldots, p_k) \,.$$

Proof. Let $p'_k < p_k$ be a computable number. Then $h_{(p_0,\ldots,p'_k)}$ is a computable gauge function, $h_{(p_0,\ldots,p'_k)} \prec h_{(p_0,\ldots,p_k)}$ and $\mathcal{H}^h(\{\xi : \mathrm{KA}(\xi[0..n]) \le -\log_r h(r^{-n}) + c_h\}) > 0$ for $h = h_{(p_0,\ldots,p'_k)}$ and some constant c_h. Moreover $\mathrm{KA}(\xi[0..n]) \le -\log_r h(r^{-n}) + c_h$ implies $\limsup_{n\to\infty} \frac{\mathrm{KA}(\xi[0..n]) - \beta_h(2^{-n})}{\log^k n} \le p_k$. Thus $\dim_{\mathrm{H}}^{(k)} \Big\{ \xi : \xi \in X^\omega \wedge \limsup_{n\to\infty} \frac{\mathrm{KA}(\xi[0..n]) - \beta_h(2^{-n})}{\log^k n} \le p_k \Big\} \ge (p_0, \ldots, p'_k)$.

As p'_k can be made arbitrarily close to p_k the assertion follows. □

Ryabko's [15] theorem is independent of the kind of complexity we use. The following example shows that, already in case $k = 1$, Theorem 8 does not hold for plain Kolmogorov complexity KS (cf. [23,24,3]).

Example 1. It is known that $\mathrm{KS}(\xi[0..n]) \le n - \log_r n + O(1)$ for all $\xi \in X^\omega$ (cf. [3, Corollary 3.11.3]). Thus every $\xi \in X^\omega$ satisfies $\liminf_{n\to\infty} \frac{\mathrm{KS}(\xi[0..n]) - n}{\log_r n} < -\frac{1}{2}$. Consequently,

$\dim_{\mathrm{H}}^{(1)} \{ \xi : \xi \in X^\omega \wedge \liminf_{n\to\infty} \frac{\mathrm{KS}(\xi[0..n]) - n}{\log_{|X|} n} < -\frac{1}{2} \} = (1, 0) >_{\mathrm{lex}} (1, -\frac{1}{2})$.

It would be desirable to prove Theorem 7 for arbitrary gauge functions or Theorem 9 for arbitrary $(k+1)$-tuples. One obstacle is that, in contrast to the case of real number dimension where the computable numbers are dense in the reals, already the computable pairs (p_0, p_1) are not dense in the above mentioned lexicographical order of pairs. This can be verified by the following fact.

Remark 2. Let $p_0 \in (0,1)$. If $r^{-p_0 \cdot n} \le h(r^{-n}) \le n \cdot r^{-p_0 \cdot n}$ for a computable function $h : \mathbb{Q} \to \mathbb{R}$ and sufficiently large $n \in \mathbb{N}$ then p_0 is a computable real. Thus, if p_0 is not a computable number, the interval between $h_{(p_0,0)}$ and $h_{(p_0,1)}$ does not contain a computable gauge function.

References

1. Calude, C.S., Staiger, L., Terwijn, S.A.: On partial randomness. Ann. Pure Appl. Logic 138(1-3), 20–30 (2006)
2. Daley, R.P.: The extent and density of sequences within the minimal-program complexity hierarchies. J. Comput. System Sci. 9, 151–163 (1974)
3. Downey, R.G., Hirschfeldt, D.R.: Algorithmic Randomness and Complexity. In: Theory and Applications of Computability. Springer, New York (2010)
4. Edgar, G.: Measure, topology, and fractal geometry, 2nd edn. Undergraduate Texts in Mathematics. Springer, New York (2008)
5. Falconer, K.: Fractal geometry. John Wiley & Sons Ltd., Chichester (1990)
6. Graf, S., Mauldin, R.D., Williams, S.C.: The exact Hausdorff dimension in random recursive constructions. Mem. Amer. Math. Soc. 71(381), x+121 (1988)

7. Hausdorff, F.: Dimension und äußeres Maß. Math. Ann. 79(1-2), 157–179 (1918)
8. Lutz, J.H.: Gales and the constructive dimension of individual sequences. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 902–913. Springer, Heidelberg (2000)
9. Lutz, J.H.: The dimensions of individual strings and sequences. Inform. and Comput. 187(1), 49–79 (2003)
10. Mauldin, R.D., Graf, S., Williams, S.C.: Exact Hausdorff dimension in random recursive constructions. Proc. Nat. Acad. Sci. U.S.A. 84(12), 3959–3961 (1987)
11. Mauldin, R.D., McLinden, A.P.: Random closed sets viewed as random recursions. Arch. Math. Logic 48(3-4), 257–263 (2009)
12. Mielke, J.: Refined bounds on Kolmogorov complexity for ω-languages. Electr. Notes Theor. Comput. Sci. 221, 181–189 (2008)
13. Mielke, J.: Verfeinerung der Hausdorff-Dimension und Komplexität von ω-Sprachen. PhD thesis, Martin-Luther-Universität Halle-Wittenberg (2010)
14. Reimann, J., Stephan, F.: Hierarchies of randomness tests. In: Mathematical logic in Asia, pp. 215–232. World Sci. Publ., Hackensack (2006)
15. Ya, B., Ryabko: Coding of combinatorial sources and Hausdorff dimension. Dokl. Akad. Nauk SSSR 277(5), 1066–1070 (1984)
16. Ya, B., Ryabko: Noise-free coding of combinatorial sources, Hausdorff dimension and Kolmogorov complexity. Problemy Peredachi Informatsii 22(3), 16–26 (1986)
17. Schnorr, C.-P.: Zufälligkeit und Wahrscheinlichkeit. Eine algorithmische Begründung der Wahrscheinlichkeitstheorie. Lecture Notes in Mathematics, vol. 218. Springer, Berlin (1971)
18. Staiger, L.: Kolmogorov complexity and Hausdorff dimension. Inform. and Comput. 103(2), 159–194 (1993)
19. Staiger, L.: A tight upper bound on Kolmogorov complexity and uniformly optimal prediction. Theory Comput. Syst. 31(3), 215–229 (1998)
20. Staiger, L.: Constructive dimension equals Kolmogorov complexity. Inform. Process. Lett. 93(3), 149–153 (2005)
21. Staiger, L.: On oscillation-free ε-random sequences. Electr. Notes Theor. Comput. Sci. 221, 287–297 (2008)
22. Tadaki, K.: A generalization of Chaitin's halting probability Ω and halting self-similar sets. Hokkaido Math. J. 31(1), 219–253 (2002)
23. Uspensky, V.A.: Complexity and entropy: an introduction to the theory of Kolmogorov complexity. In: Watanabe, O. (ed.) Kolmogorov complexity and computational complexity. EATCS Monogr. Theoret. Comput. Sci, pp. 85–102. Springer, Berlin (1992)
24. Uspensky, V.A., Shen, A.: Relations between varieties of Kolmogorov complexities. Math. Systems Theory 29(3), 271–292 (1996)
25. Zvonkin, A.K., Levin, L.A.: The complexity of finite objects and the basing of the concepts of information and randomness on the theory of algorithms. Uspehi Mat. Nauk 25(6(156)), 85–127 (1970)

Dag Realizations of Directed Degree Sequences[⋆]

Annabell Berger and Matthias Müller-Hannemann

Department of Computer Science,
Martin-Luther-Universität Halle-Wittenberg
{berger,muellerh}@informatik.uni-halle.de

Abstract. We consider the following graph realization problem. Given a sequence $S := \binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ with $a_i, b_i \in \mathbb{Z}_0^+$, does there exist an acyclic digraph (a dag, no parallel arcs allowed) $G = (V, A)$ with labeled vertex set $V := \{v_1, \ldots, v_n\}$ such that for all $v_i \in V$ indegree and outdegree of v_i match exactly the given numbers a_i and b_i, respectively? The complexity status of this problem is open, while a generalization, the f-factor dag problem can be shown to be NP-complete. In this paper, we prove that an important class of sequences, the so-called opposed sequences, admit an $O(n+m)$ realization algorithm, where n and $m = \sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ denote the number of vertices and arcs, respectively. For an opposed sequence it is possible to order all non-source and non-sink tuples such that $a_i \leq a_{i+1}$ and $b_i \geq b_{i+1}$. Our second contribution is a realization algorithm for general sequences which significantly improves upon a naive exponential-time algorithm. We also investigate a special and fast realization strategy "lexmax", which fails in general, but succeeds in more than 97% of all sequences with 9 tuples.

1 Introduction

The problem. The realization of graphs and digraphs with prescribed degree sequences has attracted researchers for several decades, with classical contributions by Havel, Hakimi, Erdös, Gallai, Ryser, Fulkerson, Chen, Kleitman and Wang [1,2,3,4,5,6,7,8]. We here consider the following problem.

Problem 1 (dag realization problem). *Given is a finite sequence $S := \binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ with $a_i, b_i \in \mathbb{Z}_0^+$. Does there exist an acyclic digraph (without parallel arcs) $G = (V, A)$ with the labeled vertex set $V := \{v_1, \ldots, v_n\}$ such that we have indegree $d_G^-(v_i) = a_i$ and outdegree $d_G^+(v_i) = b_i$ for all $v_i \in V$?*

If the answer is "yes", we call sequence S *dag sequence* and the acyclic digraph G (a so-called "dag") a *dag realization*. Unless explicitly stated, we assume that a sequence does not contain any *zero tuples* $\binom{0}{0}$. Moreover, we will tacitly assume that $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$, as this is obviously a necessary condition for any realization to exist, since the number of ingoing arcs must equal the number of

⋆ This work was supported by the DFG Focus Program Algorithm Engineering, grant MU 1482/4-2.

outgoing arcs. Furthermore, we denote tuples $\binom{a_i}{b_i}$ with $a_i > 0$ and $b_i = 0$ as *sink tuples*, those with $a_i = 0$ and $b_i > 0$ as *source tuples*, and the remaining ones with $a_i > 0$ and $b_i > 0$ as *stream tuples*.

To the best of our knowledge there is no previous work dealing with this problem. On the other hand, there is a relaxed version of this problem omitting the condition of acyclicity. This problem is called *digraph realization problem*. Sequence S is then called *digraph sequence* and the corresponding digraph G a *digraph realization*. There are many beautiful, classical results – a long story about repeated reinventions of similar results under completely different names – handling of solving this problem in two different ways. The first approach uses recursive algorithms to construct digraph realizations. The second one gives a complete characterization of digraph sequences. Hence, it is possible to check a polynomial number of inequalities (in the size of the number of tuples in a sequence) which leads to the correct decision of the realizability of a sequence. However, these views come from another analogous problem — the *graph realization problem* which asks whether a given *undirected sequence* $S := (d_1), \ldots, (d_n)$ with $d_i \in \mathbb{Z}_0^+$ possesses a realization as a graph. The characterization approach was found by Erdös and Gallai [3] and a realization algorithm was introduced by Havel [1] and Hakimi [2]. All their ideas and algorithms can be used for the digraph realization problem with only slight modifications. An interesting point of view is to consider the graph realization problem as a special case of the digraph realization problem. We consider the *symmetric digraph realization problem*, where we only allow digraph realizations containing with an arc (u, v) also arc (v, u). Clearly, for such sequences $\binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ we need the necessary conditions (1) $a_i = b_i$ for all indices i and (2) $\sum_{i \in \mathbb{N}_n} a_i = \sum_{i \in \mathbb{N}_n} b_i$ is even. To solve this problem one can consider a sequence $S := \binom{a_1}{a_1}, \ldots, \binom{a_n}{a_n}$ with the additional condition that a digraph should contain a maximum number of directed 2-cycles. Each of these directed 2-cycles $C := (u, v, u)$ can be associated with one undirected arc $\{u, v\}$. Interestingly, there exists a symmetric digraph realization for S if and only if there is a digraph realization [7]. Hence, the undirected sequence $S_u := (a_1), \ldots, (a_n)$ (with $\sum_{i \in \mathbb{N}_n} a_i$ is even) is a graph sequence if and only if sequence $S := \binom{a_1}{a_1}, \ldots, \binom{a_n}{a_n}$ is a digraph sequence. It follows that it is possible to derive all characterizations of Erdös and Gallai for graph sequences from the characterizations of digraphs. In this paper we restrict ourselves to realization algorithms. For more details consider our paper about characterizations of dag sequences [9]. Up to now it remains an open problem whether the dag realization problem can be solved in polynomial time. We solved this problem for a special class of sequences — the so-called *opposed sequences* — which are realizable in polynomial time. However, the general problem is indeed unsolved and it is clear that the problem is somewhat more complicated than the digraph realization problem.

Only for sparse graphs with $m \leq |V| - 1$ arcs, we have a characterization, because then we can construct a digraph realization such that the underlying undirected graph has no cycles, i.e. is a forest. However, the case $m \geq |V|$ remains unclear. For *source-sink-sequences*, i.e. sequences only containing source and sink tuples, testing for dag realizability is equivalent to the digraph realization problem, since each digraph realization of a source-sink sequence is automatically a dag, as each vertex either has only incoming or only outgoing arcs. On the other hand, a generalization of our problem – the *f-factor dag problem* – is NP-complete.

Problem 2 (f-factor dag problem). *Given is a sequence $S := \binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ and a digraph $G = (V, A)$ with a labeled vertex set $V := \{v_1, \ldots, v_n\}$. Does there exist an acyclic subdigraph $H = (V, A')$ of G with $d^-_H(v_i) = a_i$ and $d^+_H(v_i) = b_i$ for all $v_i \in V$?*

Our dag realization problem is a special case of the f-factor dag problem, namely if G is the complete digraph. If we relax the f-factor dag problem to the f-factor problem by omitting the property of acyclicity, then we get a polynomial time algorithm, see [10]. However, let us consider sequence $S := \binom{0}{1}, \binom{1}{1}, \ldots, \binom{1}{1}, \binom{1}{0}$. It possesses only one unique (up to isomorphic ones) dag realization, namely a Hamiltonian path. Hence, a polynomial time algorithm solving our f-factor dag realization problem, could find in the same way a directed Hamiltonian path on digraph G.

Theorem 1 (f-factor dag problem). *The f-factor dag problem is NP-complete.*

Related work. There exist four different publications characterizing digraph sequences. Gale [4], Ryser [5], Fulkerson [6] and Chen [7] all contributed to a complete characterization of digraph realizations. Here, we focus on the construction of a dag realization and give an overview about the classical results. In 1973, Kleitman and Wang [8] proposed two different algorithms to construct a digraph realization for a given digraph sequence. These algorithms are extensions of the undirected version of Havel [1] and Hakimi [2] for the graph realization problem.

Kleitman and Wang did not have their main focus on the digraph realization problem. Indeed, they gave the proof for an extension of the so-called *k-factor conjecture* of Kundu [11]. To solve a directed version of this problem, they developed two different algorithms for realizing digraph sequences [8]. We summarize their results and formulate their theorems with a slight, but easy extension for the reverse implication.

Theorem 2 (digraph realization with arbitrary tuple choice [8]) *Let $S = \binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ be a sequence and $\binom{a_i}{b_i}$ a tuple with $b_i > 0$. Let M_i be the set of all stream tuples and sink tuples of S without tuple $\binom{a_i}{b_i}$. Furthermore, let $\binom{a_{l_1}}{b_{l_1}}, \ldots, \binom{a_{l_{|M_i|}}}{b_{l_{|M_i|}}}$ be a decreasing lexicographical sorting by the first component of tuples in M_i. Sequence S is a digraph sequence if and only if the following conditions are true:*

1. $|M_i| \geq b_i$ and
2. sequence $S^i := \binom{a_1^i}{b_1^i}, \ldots, \binom{a_n^i}{b_n^i}$ with

$$b_j^i := \begin{cases} b_j & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad a_j^i := \begin{cases} a_j - 1 & \text{if } i \in \{l_1, \ldots, l_{b_i}\} \\ a_j & \text{otherwise} \end{cases}$$

is a digraph sequence.

This theorem is the analogous version of the algorithm of Havel and Hakimi. For a digraph sequence an algorithm constructs a digraph $G = (V, A)$ using a repeated application of this theorem on the current sequence S^i. Hence, it is possible to construct in each step the arcs (v_i, v_{l_j}) with $j \in \{1, \ldots, b_i\}$ or if conditions 1. or 2. are not fulfilled, we can conclude that sequence S is not a digraph sequence. Kleitman and Wang also give a second algorithm for constructing realizations. Here, the choice of a tuple $\binom{a_i}{b_i}$ is not arbitrary but one does not need a lexicographical sorting of the candidate set M.

Theorem 3 (digraph realization with a special tuple choice [8]) *Let $S = \binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ be a sequence and $\binom{a_i}{b_i}$ a largest stream tuple or sink tuple with respect to a lexicographical ordering by the second component. Furthermore, we define the set M consisting all stream tuples and sink tuples of S without tuple $\binom{a_i}{b_i}$. Consider a decreasing sorting of tuples $\binom{a_{l_1}}{b_{l_1}}, \ldots, \binom{a_{l_{|M|}}}{b_{l_{|M|}}}$ in M such that we have $i < j$ if and only if $a_{l_i} \geq a_{l_j}$. Sequence S is a digraph sequence if and only if:*

1. $|M| \geq b_i$ and
2. sequence $S' := \binom{a'_1}{b'_1}, \ldots, \binom{a'_n}{b'_n}$ with

$$b'_j := \begin{cases} b_j & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad a'_j := \begin{cases} a_j - 1 & \text{if } j \in \{l_1, \ldots, l_{b_i}\} \\ a_j & \text{otherwise} \end{cases}$$

is a digraph sequence.

Note that this theorem can be used to realize source-sink sequences. We would like to point out that the algorithms of Kleitman and Wang are relatively unknown until today. So it is not surprising that these ideas have been reinvented several times.

Our contribution. While the complexity status of the general dag realization remains open, we here show that the problem can be solved in linear time (in the size of the generated graph) for an important class of sequences, the so-called *opposed sequences.* For an opposed sequence it is possible to order all non-source and non-sink tuples such that $a_i \leq a_{i+1}$ and $b_i \geq b_{i+1}$. For opposed sequences, we introduce a new kind of order on the tuples, which we call opposed sorting, and show that each opposed sorting is a topological sorting for at least one dag realization. Systematic experiments with all dag sequences with up to 9 tuples show that opposed sequences appear quite frequently, depending on the graph density. In particular, for sequences with low and high density, there is a good chance that a sequence is opposed.

We also give an algorithm for general sequences which significantly improves upon a straightforward exponential time algorithm, but cannot be shown to run in polynomial time either. The key insight for our general algorithm is the following observation. For each realizable sequence there is a subset of "candidate" stream tuples containing at least one element by which we can reduce the sequence to a smaller and still realizable sequence. This subset of candidates is often small, in fact for opposed sequences it is exactly one uniquely determined element.

We disprove our own conjecture that a certain "lexmax strategy", which selects from the above mentioned "candidate" set always the lexicographically maximum element, solves the dag realization in polynomial time by giving an explicit counterexample. However, we observe that this strategy is successful in many cases, in particular it solves more than 97% of all dag sequences with 9 tuples for all number of arcs m. We also give a further example showing that no strategy which selects a fixed candidate from the candidate set can yield a correct algorithm. Finally, we obtain the nice structural insight that each dag sequence has a dag realization with a special kind of topological sorting derived from a partial order — the opposed relation.

Overview. The remainder of the paper is organized as follows. In Section 2, we present our results for opposed sequences. Afterwards, in Section 3, we introduce our general dag realization algorithm and a lexmax strategy, and discuss their limits. All proofs have been omitted due to page limitations. A full version of the paper is available as a technical report [12].

2 Opposed Sequences

We now turn towards the dag realization problem and define a new ordering $\leq_{opp} \subset \mathbb{Z}^2 \times \mathbb{Z}^2$.

Definition 1 (opposed relation). *Given are* $c_1 := \binom{a_1}{b_1} \in \mathbb{Z}^2$ *and* $c_2 := \binom{a_2}{b_2} \in \mathbb{Z}^2$. *We define:* $c_1 \leq_{opp} c_2 \Leftrightarrow (a_1 \leq a_2 \wedge b_1 \geq b_2)$.

The opposed relation is reflexive, transitive and antisymmetric and therefore a partial order. On the other hand it is not possible to compare all tuples c_1 and c_2. Hence, the opposed order is not a total order. In the following, we consider a special class of sequences which can easily be handled with respect to our dag realization problem. We call a sequence S *opposed sequence*, if it is possible to sort its stream tuples in such a way, that $a_i \leq a_{i+1}$ and $b_i \geq b_{i+1}$ is valid for stream tuples with indices i and $i+1$. In this case, we have the property $\binom{a_i}{b_i} \leq_{opp} \binom{a_{i+1}}{b_{i+1}}$ for all stream tuples. At the beginning of the sequence we insert all source tuples such that the b_i build a decreasing sequence and at the end of sequence S we put all sink tuples in increasing ordering with respect to the corresponding a_i. The notion *opposed sequence* still describes a sequence, where it is possible to compare all stream tuples among each other and to put them in a "chain". Indeed, this is not always possible, because the opposed order is not a total order. With such a labelling we call a sequence *increasing opposed sequence.* If we label an increasing opposed sequence in its opposite direction we call this sequence *decreasing opposed sequence.* Obviously, it is possible that a stream tuple is not comparable with a source tuple or a sink tuple. However, these sequences turn out to be "good-natured". Next we follow the classic approach of solving this problem with an inductive construction of a realization. It turns out that we need a completely new approach to save classic ideas. We give some interesting insights, for example we found that an increasing opposed sorting of an opposed dag sequence is a topological sorting of at least one dag realization, too.

2.1 Realization of Opposed Sequences

The following theorem is the basis of a realization algorithm. It is similar to an inductive, greedy-like algorithm as in the classic approach of Havel and Hakimi [1,2]. On the other hand, it differs completely in its details from this algorithm. In each step, we choose the smallest stream tuple with respect to the opposed relation and connect it with largest sources — namely with one arc for each source. After this step, this stream tuple will be a new source. If this step is not possible we are sure that sequence S is not a dag sequence.

Theorem 4 (Opposed Sequences). *Let S be an increasing opposed sequence which is not a source-sink-sequence. Furthermore, let tuple $\binom{a_{i_{min}}}{b_{i_{min}}}$ be the smallest stream tuple with respect to the opposed ordering. Then sequence S is a dag sequence if and only if there exist at least $a_{i_{min}}$ source tuples in S and if* $S' := \binom{0}{b_1 - 1}, \ldots, \binom{0}{b_{a_{i_{min}}} - 1}, \binom{0}{b_{a_{i_{min}}+1}}, \ldots, \binom{0}{b_{i_{min}-1}}, \binom{0}{b_{i_{min}}}, \binom{a_{i_{min}+1}}{b_{i_{min}+1}}, \ldots, \binom{a_n}{b_n}$ *is a dag sequence.*

Proof. (Sketch) Let S be a dag sequence and G an arbitrary dag realization. Using a series of transformations we have to show that G can be modified to a dag realization G^{***} such that vertex $v_{i_{min}}$ only possesses the maximum sources in its incoming neighborhood set. After removing all incoming arcs of vertex $v_{i_{min}}$ we obtain a dag G' with vertex degree sequence S'. □

This theorem reduces an opposed dag sequence until we get a source-sink-sequence S'. By applying Theorem 2 for digraph realizations, we can realize sequence S'. Algorithm 1 combines the insights of both theorems and constructs a dag realization for any opposed sequence which is realizable.

Using bucket sort, the reordering of a given opposed sequence such that it is in increasing opposed order requires $O(m + n)$ time. Algorithm 1 can also be implemented to run in time $O(m + n)$ using a "bucket" technique. The idea is to maintain the source and sink tuples in buckets (realized as linked lists), one bucket for each outdegree and indegree, respectively, thus with at most $2n - 2$ buckets. Non-empty buckets are ordered decreasingly, and each bucket has a pointer to its two neighbors in this order.

Moreover, we maintain three pointers, one to the non-empty bucket corresponding to the source with the largest outdegree, one to the sink with largest

Algorithm 1. Dag Realization Algorithm for Opposed Sequences

Input: An opposed sequence S in increasing, opposed order.
Output: A dag realization $G = (V, A)$ of S or the answer that S is not a dag sequence.

```
1: initialize $realizable \leftarrow TRUE$;
2: initialize $A \leftarrow \emptyset$;
3: while (the set of stream tuples in $S$ is not empty) and ($realizable == TRUE$) do
4:     //application of Theorem 4
5:     choose the stream tuple $\binom{a_j}{b_j}$ with smallest index $j$;
6:     if the number of sources in $S$ is smaller than $a_j$ then
7:         $realizable \leftarrow FALSE$;
8:     else
9:         set $b_i \leftarrow b_i - 1$ for the $a_j$ largest sources $\binom{0}{b_{l_1}}, \dots, \binom{0}{b_{l_{a_j}}}$;
10:        set $a_j \leftarrow 0$;
11:        set $A \leftarrow A \cup \{(v_{l_1}, v_j), (v_{l_2}, v_j), \dots, (v_{l_{a_j}}, v_j)\}$;
12:        delete $\binom{0}{0}$-tuples in $S$;
13:    end if
14: end while
15: while (the set of source tuples in $S$ is not empty) and ($realizable == TRUE$) do
16:     //realization of a source-sink-sequence.
17:     choose a largest source tuple $\binom{0}{b_j}$;
18:     if the number of sinks in $S$ is smaller than $b_j$ then
19:         $realizable \leftarrow FALSE$;
20:     else
21:         set $a_i \leftarrow a_i - 1$ for the $b_j$ largest sinks $\binom{a_{k_1}}{0}, \dots, \binom{a_{k_{b_j}}}{0}$;
22:         set $A \leftarrow A \cup \{(v_j, v_{k_1}), (v_j, v_{k_2}), \dots, (v_j, v_{k_{b_j}})\}$;
23:         delete $\binom{0}{0}$-tuples in $S$;
24:     end if
25: end while
```

indegree, and one to the source with the smallest outdegree. The stream tuples are kept in a list, sorted according to the increasing opposed order. Thus line 5 can be done in $O(1)$. With our data structure, it is easy to execute line 9 in $O(a_j)$ time, that is, to select the a_j largest source tuples, to decrease their outdegree by one and to update our data structure (which means in particular to shift tuples from their current bucket to the next lower bucket or to delete them). In line 10, the selected stream tuple with index j becomes a new source and is inserted into our bucketing data structure. This can also be done in $O(a_j)$ time. Likewise, in line 17 choosing the largest source tuple can be done in $O(1)$ and line 21 in $O(b_j)$. In total, this yields $O\left(\sum_{i=1}^{n}(a_i + b_i)\right) = O(m + n)$ time.

We point out, that there is the possibility to generalize Theorem 4 and to formulate an algorithm handling the realization of all sequences. The problem is, that the opposed order is not a total order. Therefore, there does not exist a unique minimal stream tuple in each sequence. Hence, we get a realization algorithm but destroy the polynomial running time in general. More details are given in Section 3. Theorem 4 leads to a further property connected with topological orderings. Let us consider a well-known algorithm determining a topological ordering of a dag. The iterative deletion of one current source results in a labeling of vertices representing a possible topological sorting for a dag. Applying our theorem we can conclude the following: Each stream tuple $\binom{a_{i+i_{min}}}{b_{i+i_{min}}}$ with $i \in \{0, \ldots, n_s - 1\}$ (n_s denotes the number of stream tuples) is minimum stream tuple in the ith iteration of the while statement in line 3. Clearly, tuple $\binom{a_{i+i_{min}}}{b_{i+i_{min}}}$ is in the $(i+1)$th iteration of this loop a new source $\binom{0}{b_{i+i_{min}}}$ (under the restriction that the sequence is not yet a source-sink-sequence). That means, if we consider a dag realization G which was constructed by Algorithm 1, then we could remove sources following the increasing opposed sorting. This yields the following corollary.

Corollary 1 (topological sorting). *An increasing opposed sorting of an opposed dag sequence S is a topological sorting of at least one dag realization of S.*

This new view enables us to see the dag realization problem as a well-known classical problem. If we consider the complete acyclic digraph with a vertex labeling with respect to its topological sorting, we have to find a labeled subdigraph with sequence S in following the numbering of the increasing opposed sorting. This problem is nothing else but an f-factor problem on a bipartite graph after an easy reduction via Tutte [10]. This leads to a polynomial-time algorithm. On the other hand, we have a necessary criterion for determining the realizability of opposed sequences.

Corollary 2 (necessary criterion for the realizability of opposed sequences). *Let S be an increasing opposed dag sequence. Denote the number of source tuples in S by q and the number of sink tuples by s. Then it follows $a_i \leq \min\{n-s, i-1\}$ and $b_i \leq \min\{n-q, n-i\}$ for all $i \in \mathbb{N}_n$.*

3 General Dag Realization

3.1 The Main Result

In Section 2, we introduced a realization algorithm for opposed sequences. Its underlying ideas can be extended to the general case – the realization of arbitrary dag sequences. This new algorithm does not necessary possess a polynomial running time. On the other hand many dag sequences are realizable in polynomial running time using the following Theorem 5 and a further "strategy". We call a sequence $S = \binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}$ with q source tuples and s sink tuples *canonically sorted*, if and only if the first q tuples in this labelling are decreasingly sorted source tuples (with respect to the b_i) and the last s tuples are increasingly sorted sink tuples (with respect to the a_i.)

Theorem 5 (characterization of dag realizability). *Let S be a canonically sorted sequence containing $k > 0$ source tuples. Furthermore, we assume that S is not a source-sink-sequence. We define the set $V_{min} := \left\{ \binom{a_i}{b_i} \mid \binom{a_i}{b_i}\right.$* is a stream tuple, $a_i \leq k$ and there is no stream tuple $\binom{a_j}{b_j} <_{opp} \binom{a_i}{b_i}\Big\}$. *$S$ is a dag sequence if and only if $V_{min} \neq \emptyset$ and there exists an element $\binom{a_{i_{min}}}{b_{i_{min}}} \in V_{min}$ such that $S' := \binom{0}{b_1 - 1}, \ldots, \binom{0}{b_{a_{i_{min}}} - 1}, \binom{0}{b_{a_{i_{min}}+1}}, \ldots, \binom{0}{b_{i_{min}-1}}, \binom{0}{b_{i_{min}}}, \binom{a_{i_{min}+1}}{b_{i_{min}+1}}, \ldots, \binom{a_n}{b_n}$ is a dag sequence.*

This theorem gives us further interesting insights. Consider the relation $R_{top} \subseteq V \times V$, where we define: $vR_{top}w$ if and only if there exists no directed path starting in w and ending in v. For each dag sequence we can find a dag realization with a topological ordering such that we have for each pair of tuples with $v_i R_{top} v_j$ either $\binom{a_i}{b_i} \leq_{opp} \binom{a_j}{b_j}$ or they are not comparable but it never follows $\binom{a_j}{b_j} <_{opp} \binom{a_i}{b_i}$. Note, that this is not a general property for acyclic digraphs. Consider for example digraph $G = (V, A)$ with $V := \{v_1 \ldots v_7\}$ and $A = \{(v_1, v_3), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_4, v_6), (v_4, v_7)\}$. The corresponding stream tuples of $v_3 R_{top} v_4$ are $\binom{a_3}{b_3}$ and $\binom{a_4}{b_4}$ with $\binom{a_4}{b_4} = \binom{1}{3} <_{opp} \binom{a_3}{b_3} = \binom{2}{1}$.

Corollary 3. *Let S be a dag sequence. Then there exists a dag realization $G = (V, A)$ with a topological ordering $v_{l_1}, \ldots, v_{l_{n_s}}$ of all n_s vertices corresponding to stream tuples, such that we cannot find $\binom{a_{l_j}}{b_{l_j}} <_{opp} \binom{a_{l_i}}{b_{l_i}}$ for all $l_i < l_j$.*

Theorem 5 proves the existence of a dag realization which contains a vertex corresponding to an element of V_{min} only possessing maximum sources as incoming neighborhood set. We define a largest possible subset $V'_{min} \subseteq V_{min}$ with the property that all elements in V'_{min} are pairwise distinct (w.l.o.g. we can restrict our candidate set to V'_{min}). For each $\binom{a_{i_j}}{b_{i_j}} \in V'_{min}$ we construct the reduced sequence S'_j, set $S_j := S'_j$ and apply our Theorem 5 repeatedly until S'_j is a source-sink-sequence or we find out that S'_j is not a dag sequence, because V_{min} is empty.

Theorem 5 ensures the possibility for reducing a dag sequence into a source-sink-sequence. The latter can be realized by using Theorem 2. The whole algorithm is summarized in Algorithm 2.

The bottleneck of this approach is the size of set V'_{min}. We only find stream tuples in this set which are not comparable with respect to the opposed relation. In an opposed dag sequence we have $|V'_{min}| = 1$, if sequence S is not a source-sink-sequence, because in this case there exists a unique minimum stream tuple (up to isomorphic ones). Hence, Theorem 4 is a special case of our Theorem 5. However, there are many sequences which are not opposed but Theorem 5 still yields a polynomial decision time. Consider for example dag sequence $S := \binom{0}{3}, \binom{0}{3}, \binom{2}{2}, \binom{3}{3}, \binom{3}{0}, \binom{2}{0}, \binom{1}{0}$ which is not an opposed sequence, because stream tuples $\binom{2}{2}$ and $\binom{3}{3}$ are not comparable with respect to the opposed ordering. However, we have $|V'_{min}| = |\{\binom{2}{2}\}| = 1$ and so we reduce S to $S' = \binom{0}{2}, \binom{0}{2}, \binom{0}{2}, \binom{3}{3}, \binom{3}{0}, \binom{2}{0}, \binom{1}{0}$, leading to the realizable source-sink-sequence $\binom{0}{1}, \binom{0}{1}, \binom{0}{1}, \binom{0}{3}, \binom{3}{0}, \binom{2}{0}, \binom{1}{0}$.

Algorithm 2. DagRealization(sequence S)

```
Input: A canonical sorted sequence $S$.
Output: A Boolean flag indicating whether $S$ is realizable.
 1: if $S$ is not a source-sink-sequence then
 2:    Count the number of sources in $S$ and determine set $V'_{min}$.
 3:    for all $v_j \in V'_{min}$ do
 4:       Create a working copy $S'$ of $S$ with tuples $\binom{a'_i}{b'_i} = \binom{a_i}{b_i}$;
 5:       Set $b'_i \leftarrow b'_i - 1$ for $a'_j$ largest sources $\binom{0}{b'_i}$.
 6:       Set $a'_j \leftarrow 0$.
 7:       Delete $\binom{0}{0}$-tuples.
 8:       if DagRealization($S'$) then
 9:          return TRUE;
10:       end if
11:    end for
12:    return FALSE;
13: else
14:    while the set of source tuples in $S$ is not empty do
15:       //Realization of a source-sink-sequence.
16:       choose a largest source tuple $\binom{0}{b_j}$.
17:       if number of sinks in $S$ is smaller than $b_j$ then
18:          return FALSE;
19:       end if;
20:       Set $a_i \leftarrow a_i - 1$ for $b_j$ largest sinks $\binom{a_i}{0}$.
21:       Delete $\binom{0}{0}$-tuples.
22:    end while
23:    return TRUE;
24: end if
```

At the beginning of our work, we conjectured that the choice of the lexicographical largest tuple from V_{min} would solve our problem. We call this approach *lexmax strategy.* Consider our smallest counterexample – dag sequence

$$S = \binom{0}{3}, \binom{0}{1}, \binom{1}{1}, \binom{1}{2}, \binom{2}{3}, \binom{4}{4}, \binom{3}{0}, \binom{2}{0}, \binom{1}{0}$$

– which cannot be realized with this strategy. The set $V_{min} = \{\binom{1}{2}, \binom{2}{3}\}$ contains two elements. If we choose the lexicographical larger tuple than we do not get a dag sequence S'. More interesting is the insight, that there is *no general strategy* for choosing a correct element in V_{min} *without the consideration of all sinks.* To see this, consider the following example.

Example 1. We consider the sequences $S_1 := \binom{0}{5}, \binom{0}{5}, \binom{0}{5}, \binom{0}{2}, \binom{0}{2}, \binom{5}{5}, \binom{5}{5}, \binom{2}{2}, \binom{2}{2}, \binom{9}{0}, \binom{6}{0}, \binom{2}{0}, \binom{1}{0}, \binom{1}{0}$ and $S_2 := \binom{0}{5}, \binom{0}{5}, \binom{0}{5}, \binom{0}{2}, \binom{0}{2}, \binom{5}{5}, \binom{5}{5}, \binom{2}{2}, \binom{2}{2}, \binom{7}{0}, \binom{6}{0}, \binom{6}{0}$, only differing in their sink tuples. The sets $V_{min}^{S_1} = \{\binom{2}{2}, \binom{5}{5}\}$ and $V_{min}^{S_2} = \{\binom{2}{2}, \binom{5}{5}\}$ are identical. For realizing sequence S_1 we first choose element $\binom{2}{2}$ and for sequence S_2 we only have the possibility to choose tuple $\binom{5}{5}$. More precisely, we select one

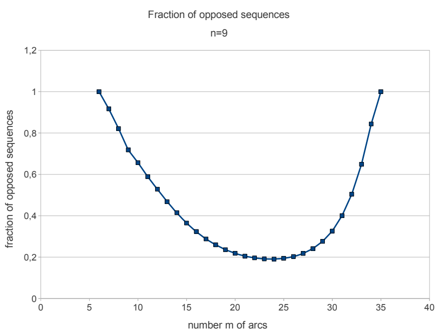


Fig. 1. The fraction of opposed sequences in the set of all non-trivial dag sequences with 9 tuples and different densities $m \in \{5, \ldots, 35\}$

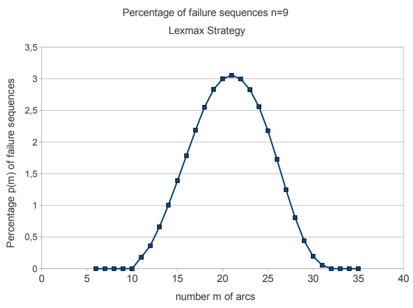


Fig. 2. The percentage of non-trivial dag sequences with $n = 9$ and different densities $m \in \{5, \ldots, 35\}$ where the lexmax strategy failed

after another the tuples $\binom{2}{2}, \binom{5}{5}, \binom{2}{2}, \binom{5}{5}$ to construct a dag realization of S_1,while for S_2 we have to choose $\binom{5}{5}, \binom{5}{5}, \binom{2}{2}, \binom{2}{2}$ in this order. In both cases, these orders cannot be changed — already not for the choice of the first tuple.

Obviously, an efficient correct algorithm has to consider the kind of sinks, otherwise it is not possible to choose the right element in V_{min}. Unfortunately, we cannot give such an approach. However, in a series of systematic experiments for all dag sequences with $7, 8, 9$ tuples, we observed that our lexmax strategy determines a dag realization for a large fraction of all dag sequences.

3.2 Discussion of Experimental Results

In this subsection we briefly discuss the significance and limitations of our results. To this end, we consider two questions:

1. Does there exist a large fraction of opposed sequences in the set of all sequences or does it turn out to be a rare class of sequences?
2. How large is the fraction of sequences which are realizable by using the lexmax strategy?

In a first step we generated the set of all non-isomorphic dag sequences with $7, 8$ and 9 tuples. We restricted this set by ignoring all “trivial sequences” (i.e., source-sink-sequences and sequences with only one stream tuple), since these can always be decided in $O(n + m)$ time. We implemented a version of Algorithm 2 where we replaced line 3 by the lexmax strategy: “**for** vertex v_j with the lexicographically largest $\binom{a_j}{b_j} \in V'_{min}$ **do**”. Note, that it was not possible to enumerate dag sequences systematically for larger instances because the number of them increases exponentially fast and becomes already huge for $n = 10$. In experiment (1) we investigated the connection between the fraction of opposed sequences (with respect to all non-trivial sequences) and the density (number of arcs) of sequences with a constant number of tuples. Our observation is that the fraction of opposed sequences is strongly connected to their density, see Figure 1. We observe that sequences with a middle density have the smallest fractions of opposed sequences and can be regarded as "more complicated". On the other hand, we see that opposed sequences are a relevant class of sequences because they are not at all rare. In experiment (2), we analyzed the lexmax strategy. In Figure 2, one can observe that the lexmax strategy leads to a dag realization for at least 97% of all (non-trivial) dag sequences. Furthermore, we observe a strong connection between the density of a sequence and the realizability with the aid of the lexmax strategy. We see again that sequences with a middle density are the most difficult ones. Finally, we would like to point out that a number of straightforward reduction rules can be used to simplify given sequences. Combined with the lexmax strategy, these reductions significantly increase the success rate of our approach.

Acknowledgement. The authors wish to thank Winfried Hochstättler for many stimulating discussions on this fascinating problem.

References

1. Havel, V.: A remark on the existence of finite graphs. Casopis Pest. Math. 80, 477–480 (1955)
2. Hakimi, S.: On the realizability of a set of integers as degrees of the vertices of a simple graph. J. SIAM Appl. Math. 10, 496–506 (1962)
3. Hakimi, S.: Graphs with prescribed degree of vertices (hungarian). Mat. Lapok 11, 264–274 (1960)
4. Gale, D.: A theorem on flows in networks. Pacific J. Math. 7, 1073–1082 (1957)
5. Ryser, H.: Combinatorial properties of matrices of zeros and ones. Canad J. Math. 9, 371–377 (1957)
6. Fulkerson, D.: Zero-one matrices with zero trace. Pacific J. Math. 10, 831–836 (1960)
7. Chen, W.K.: On the realization of a (p,s)-digraph with prescribed degrees. Journal of the Franklin Institute 281, 406–422 (1966)
8. Kleitman, D.J., Wang, D.L.: Algorithms for constructing graphs and digraphs with given valences and factors. Discrete Mathematics 6, 79–88 (1973)
9. Berger, A., Müller-Hannemann, M.: Dag characterizations of directed degree sequences. Technical Report 2011/6, Martin-Luther-Universität Halle-Wittenberg, Department of Computer Science (2011)
10. Tutte, W.T.: Graph factors. Combinatorica 1, 79–97 (1981)
11. Kundu, S.: The k-factor conjecture is true. Discrete Mathematics 6, 367–376 (1973)
12. Berger, A., Müller-Hannemann, M.: Dag realisations of directed degree sequences. Technical Report 2011/5, Martin-Luther-Universität Halle-Wittenberg, Department of Computer Science (2011)

A Coinductive Calculus for Asynchronous Side-Effecting Processes

Sergey Goncharov and Lutz Schröder

Safe and Secure Cognitive Systems, DFKI GmbH, Bremen

Abstract. We present an abstract framework for concurrent processes in which atomic steps have generic side effects, handled according to the principle of monadic encapsulation of effects. Processes in this framework are potentially infinite resumptions, modelled using final coalgebras over the monadic base. As a calculus for such processes, we introduce a concurrent extension of Moggi's monadic metalanguage of effects. We establish soundness and completeness of a natural equational axiomatisation of this calculus. Moreover, we identify a corecursion scheme that is explicitly definable over the base language and provides flexible expressive means for the definition of new operators on processes, such as parallel composition. As a worked example, we prove the safety of a generic mutual exclusion scheme using a verification logic built on top of the equational calculus.

1 Introduction

Imperative programming languages work with many different side effects, such as I/O, state, exceptions, and others, which all moreover come with strong variations in detail. This variety is unified by the principle of monadic encapsulation of side-effects [21], which not only underlies extensive work in semantics (e.g. [17]) but, following Wadler [33], forms the basis for the treatment of side-effects in functional languages such as Haskell [24] and F# [30]. Monads do offer support for concurrent programming, in particular through variants of the *resumption monad transformer* [5, 13], which lifts resumptions in the style of Hennessy and Plotkin [15] to the monadic level, and which has moreover been used in information flow security [14], semantics of functional logic programming languages such as Curry [31], modelling underspecification of compilers, e.g. for ANSI C [23], and to model the semantics of the π-calculus [11]. However, there is to date no concurrent correspondent to Moggi's *computational meta-language* [21], which underlies Haskell's do-notation; this language is essentially limited to linear sequential monadic programs, and does not include native support for concurrency.

The objective of the present work is therefore to develop an extension of the computational meta-language that can serve as a minimal common basis for generic concurrent programming and semantics. We define an abstract *meta-calculus for monadic processes* that is based on the resumption monad transformer, and hence generic over the base effects inherent in individual process steps. Resumptions allow for interleaving concurrency in a setting with *asynchronous* communication, e.g. through shared variables (synchronous communication would seem to require additional infrastructure in the underlying monad). We work with infinite resumptions, which brings tools from coalgebra into play, in particular corecursion and coinduction [27]. We present a complete equational axiomatization of our calculus which includes a simple loop construct (in coalgebraic terms, coiteration) and then derive a powerful *corecursion schema* for the definition of process combinators. It has a fully syntactic justification, i.e. one can explicitly construct a solution to a corecursive equation system by means of the basic term language. Even semantically, our corecursion scheme does not seem follow from previous corecursion results (e.g. [1, 32, 19]), as it permits prefixing corecursive calls with monadic sequential composition.

We exemplify our corecursion scheme with the definition of a number of basic imperative programming and process-algebraic primitives including parallel composition, and present a worked example, in which we outline a safety proof for a monadic version of Dekker's mutual exclusion algorithm, i.e. a concurrent algorithm with generic side-effects. To this end, we employ a more high-level verification logic that we develop on top of the basic equational calculus.

Further related work. There is extensive work on axiomatic perspectives on effectful iteration and recursion, including traced pre-monoidal categories [2], complete iterative algebras [19], Kleene monads [12], and recursive monadic binding [9]. The abstract notion of resumption goes back at least to Hennessy and Plotkin [15]. A framework where infinite resumptions of a somewhat different type than considered here form the morphisms of a category of processes, which is thus enriched over coalgebras for a certain functor, has been introduced by Krstic et al. [18], but no metalanguage is provided for such processes. A metalanguage that essentially adds least fixed points, i.e. *inductive* data types as opposed to *coinductive* process types as used in the present work, to Moggi's base language has been defined by Filinski [10]; reasoning principles in this framework are necessarily of a rather different flavour. A resumption monad without a base effect, the *delay monad*, has been studied by Capretta [4] with a view to capturing general recursion. Our variant of the resumption monad transformer belongs to the class of *parametrized monads*, for which a form of corecursive scheme has been established [32], which however does not seem to imply the one introduced here.

2 Computational Monads and Resumptions

We briefly recall the basic concepts of the monadic representation of side-effects, and then present the specific semantic framework required for the present work. Intuitively, a monad associates to each type A a type TA of computations with results in A; a function with side effects that takes inputs of type A and returns values of type B is, then, just a (pure) function of type $A \to TB$. One of the equivalent ways to define a monad over a category $\mathbf{C}$ is by giving a *Kleisli triple* $(T, \eta, \_^\star)$ (usually referred to just as T) where $T : \operatorname{Ob} \mathbf{C} \to \operatorname{Ob} \mathbf{C}$ is a function, η is a family of morphisms $\eta_A : A \to TA$ called *unit*, and $\_^\star$ assigns to each morphism $f : A \to TB$ a morphism $f^\star : TA \to TB$ such that $\eta_A^\star = \mathrm{id}_{TA}$, $\quad f^\star \circ \eta_A = f$, $\quad$ and $\quad g^\star \circ f^\star = (g^\star \circ f)^\star$. Thus, η_A converts values of type A into side-effect free computations, and $\_^\star$ supports the sequential composition $g^\star f$ of programs $f : A \to TB$ and $g : B \to TC$. A monad over a Cartesian category is *strong* if it is equipped with a natural transformation

$\tau_{A,B} : A \times TB \to T(A \times B)$ called *strength*, subject to certain coherence conditions. The strength serves to propagate context through computations [21]. Since we are interested in concurrency, we require additional structure for non-determinism:

Definition 1 (Strong semi-additive monads). In the above notation, a strong monad T is *semi-additive* if there exist natural transformations $\delta : 1 \to T$ and $\varpi : T \times T \to T$ making every TA an internal bounded join-semilattice object so that $\_^\star$ and τ respect the join-semilattice structure in the following sense:

$$\begin{aligned} f^\star\delta &= \delta, & f^\star\varpi &= \varpi\langle f^\star, f^\star\rangle, \\ \tau(f \times \delta) &= \delta, & \tau(f \times \varpi) &= \varpi\langle \tau(f \times \pi_1), \tau(f \times \pi_2)\rangle. \end{aligned}$$

The above definition forces the nondeterministic choice to be an *algebraic operation* in sense of Plotkin and Power [25]. Hence, the semilattice structure distributes over binding from the left (but not necessarily from the right) as reflected in our calculus in Section 3.

Example 2. [21] The core examples of strong semi-additive monads are the finite powerset monad $\mathcal{P}_\omega$, or, in the domain-theoretic setting, various powerdomain constructions. Moreover, the powerset monad $\mathcal{P}$ and more generally, the *quantale monad* [16] $\lambda X.Q^X$ for a quantale [26] Q are strong semi-additive monads. Further examples of semi-additive monads can be obtained from basic ones by combining them with other effects, e.g. by applying suitable monad transformers. In particular, the following monad transformers (which produce a new monad Q, given a monad T) preserve semi-additivity over any base category with sufficient structure:

1. Exceptions: $QA = T(A + E)$,
2. States: $QA = S \to T(A \times S)$,
3. I/O: $QA = \mu X. T(A + I \to (X \times O))$,
4. Continuations: $QA = (A \to TK) \to TK$.

E.g., the non-deterministic state monad $TX = S \to P(S \times X)$, is a strong semi-additive monad both over $\mathbf{Set}$ (with P denoting any variant of powerset) and over any reasonable category of domains (with P denoting a powerdomain construction with deadlock).

To model processes which are composed of atomic steps to be thought of as pieces of imperative code with generic side-effects, we use a variant of the *resumption monad transformer* [5]: Assuming that for every $X \in \mathrm{Ob}\,(\mathbf{C})$ the endofunctor $T(Id + X) : \mathbf{C} \to \mathbf{C}$ possesses a final coalgebra, which we denote by $\nu\gamma.\,T(\gamma + X)$, we define a new monad R by

$$RX = \nu\gamma.\,T(\gamma + X)$$

— R exists, e.g., if the base category is locally presentable and T is accessible [34], a basic example being $TX = S \to P(S \times X)$ where P is finite powerset or a powerdomain. Intuitively, a *resumption*, i.e. a computation in RX, takes an atomic step in T and then returns either a value in X or a further computation in RX, possibly continuing in this way indefinitely. Using a final coalgebra semantics amounts to identifying processes up to coalgebraic behavioural equivalence, which generalizes strong bisimilarity.

3 A Calculus for Side-Effecting Processes

As originally observed by Moggi [21], strong monads support a *computational metalanguage*, i.e. essentially a generic sequential imperative programming language. Here we introduce a concurrent version of the metalanguage, the *concurrent metalanguage*, based semantically on the resumption monad transformer; its distinctive features are support for non-deterministic choice and resumptions, in particular an infinite loop construct.

The concurrent metalanguage is parametrised over a countable signature Σ including a set of atomic types W, from which the type system is generated by the grammar

$$A ::= W \mid 1 \mid A \times A \mid A + A \mid TA \mid T_\nu A$$

— that is, we support sums and products, leaving functional types for future work. Base effects are represented by T, and resumptions by T_ν.

Moreover, Σ includes (typed) function symbols $f : A \to B$, where A and B are types. The terms of the language, also referred to as *programs*, and their types are then determined by the rules shown in Fig. 1. The dotted line separates operators for sequential non-determinism from the process operators; obvious rules for variables, function application, and Cartesian structure (unit element $\star$, pairing $\langle \_, \_ \rangle$, projections fst, snd) are omitted. Besides the standard term language for sums and products and the bind and return operators do and ret of the computational metalanguage, the concurrent metalanguage includes operations $\varnothing$ and $+$ representing deadlock and non-deterministic choice, respectively, as well as two specific constructs for resumptions: Given a resumption $p : T_\nu A$, $\mathsf{out}(p) : T(T_\nu A + A)$ executes the first step of p, returning either the remaining resumption or a final value. Moreover, for $q : T(A + B)$ depending on $x : A$, $\mathsf{init}\, x := p\ \mathsf{unfold}\{q\} : T_\nu B$ iterates q indefinitely or until it outputs a value in B; values $x : A$ are fed through the loop, starting with the initial value $p : A$. Judgements $\Gamma \rhd t : A$ read 'term t has type A in context Γ', where a *context* is a list $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ of typed variables. Programs whose type is of the form $T_\nu A$ are called *processes*. The notions of free and bound variables are defined in a standard way, as well as a notion of capture-avoiding substitution.

The semantics of the concurrent metalanguage is defined over ME_ν-*models*, referred to just as *models* below. A model is based on a distributive category [6] $\mathbf{C}$, i.e. a category with binary sums and finite products such that the canonical map $A \times B + A \times C \to A \times (B + C)$ is an isomorphism, with inverse $\mathit{dist} : A \times (B + C) \to A \times B + A \times C$. (E.g., Cartesian closed categories are distributive.) Moreover, it specifies a strong semi-additive monad T on $\mathbf{C}$ such that for every $A \in \mathrm{Ob}\,(\mathbf{C})$ the functor $T(Id{+}A)$ possesses a final coalgebra denoted $RA = \nu\gamma.\, T(\gamma + A)$, thus defining a functor R (*resumptions*).

A model interprets base types as objects of $\mathbf{C}$. The interpretation $[\![A]\!]$ of types A is then defined by standard clauses for 1, $A \times B$, and $A + B$ and $[\![TA]\!] = T[\![A]\!]$, $[\![T_\nu A]\!] = R[\![A]\!]$. For $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ we put $[\![\Gamma]\!] = [\![A_1]\!] \times \cdots \times [\![A_n]\!]$. Moreover, a model interprets function symbols $f : A \to B$ as morphisms $[\![f]\!] : [\![A]\!] \to [\![B]\!]$, which induces an interpretation $[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$ of programs $\Gamma \rhd t : A$ given by the usual clauses for variables, function application, pairing, projections, injections, and $\star$. The operations $+$ and $\varnothing$ are interpreted by the bounded join semilattice operations ϖ and δ of T, respectively. For the monad operations and the case operator, we have

$$\textbf{(case)}\ \frac{\varGamma \rhd s : A+B \quad \varGamma, x : A \rhd t : C \quad \varGamma, y : B \rhd u : C}{\varGamma \rhd \mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto t;\ \mathsf{inr}\, y \mapsto u : C} \qquad \textbf{(nil)}\ \frac{}{\varGamma \rhd \varnothing : TA}$$

$$\textbf{(inl)}\ \frac{\varGamma \rhd t : A}{\varGamma \rhd \mathsf{inl}\, t : A+B} \qquad \textbf{(inr)}\ \frac{\varGamma \rhd t : B}{\varGamma \rhd \mathsf{inr}\, t : A+B} \qquad \textbf{(ret)}\ \frac{\varGamma \rhd t : A}{\varGamma \rhd \mathsf{ret}\, t : TA}$$

$$\textbf{(do)}\ \frac{\varGamma \rhd p : TA \quad \varGamma, x : A \rhd q : TB}{\varGamma \rhd \mathsf{do}\ x \leftarrow p; q : TB} \qquad \textbf{(plus)}\ \frac{\varGamma \rhd p + q : TA}{\varGamma \rhd p : TA \quad \varGamma \rhd q : TA}$$

$$\textbf{(out)}\ \frac{\varGamma \rhd p : T_\nu A}{\varGamma \rhd \mathsf{out}(p) : T(T_\nu A + A)} \qquad \textbf{(unf)}\ \frac{\varGamma \rhd p : A \quad \varGamma, x : A \rhd q : T(A+B)}{\varGamma \rhd \mathsf{init}\, x := p\ \mathsf{unfold}\, \{q\} : T_\nu B}$$

Fig. 1. Typing rules for the concurrent metalanguage (excerpt)

- $[\![\varGamma \rhd \mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto t;\ \mathsf{inr}\, y \mapsto u : C]\!] =$
  $\big[[\![\varGamma, x : A \rhd t : C]\!], [\![\varGamma, y : B \rhd u : C]\!]\big] \circ \mathit{dist} \circ \big\langle \mathrm{id}, [\![\varGamma \rhd s : A + B]\!]\big\rangle,$
- $[\![\varGamma \rhd \mathsf{do}\ x \leftarrow p; q : TB]\!] = [\![\varGamma, x : A \rhd q : TB]\!]^\star \circ \tau_{[\![\varGamma]\!],[\![A]\!]} \circ \langle \mathrm{id}, [\![\varGamma \rhd p : TA]\!]\rangle,$
- $[\![\varGamma \rhd \mathsf{ret}\, t : TA]\!] = \eta_A \circ [\![\varGamma \rhd t : A]\!],$

where as usual $\langle f, g\rangle : A \to B \times C$ denotes pairing of morphisms $f : A \to B$, $g : A \to C$, and $[f, g] : A + B \to C$ denotes copairing of $f : A \to C$ and $g : B \to C$.

It remains to interpret out, which is just the final coalgebra structure of RA, and the loop construct $\mathsf{init}\, x := p\ \mathsf{unfold}\, \{q\}$ which captures coiteration. Formally, let $\alpha_A : RA \to T(RA + A)$ be the final coalgebra structure, and for a coalgebra $f : X \to T(X + A)$, let $\boldsymbol{\varGamma}_f : X \to RA$ be the unique coalgebra morphism. Then we put

$$[\![\varGamma \rhd \mathsf{out}(p) : T(T_\nu A + A)]\!] = \alpha_{[\![A]\!]} \circ [\![\varGamma \rhd p : T_\nu A]\!]$$
$$[\![\varGamma \rhd \mathsf{init}\, x := p\ \mathsf{unfold}\, \{q\} : T_\nu B]\!] = R\pi_2 \circ \boldsymbol{\varGamma}_f \circ \langle \mathrm{id}, [\![\varGamma \rhd p : A]\!]\rangle$$

where $f = T(\mathit{dist}) \circ \tau\langle \pi_1, g\rangle$ with $g = [\![\varGamma, x : A \rhd q : T(A+B)]\!]$. Thus, $\boldsymbol{\varGamma}_f$ is uniquely determined by the commutative diagram

$$\begin{tikzcd}
{[\![\varGamma]\!] \times [\![A]\!]} \arrow[r, "T(\mathit{dist})\circ\tau\langle\pi_1,g\rangle"] \arrow[d, "\boldsymbol{\varGamma}_f"'] & {T([\![\varGamma]\!] \times [\![A]\!] + [\![\varGamma]\!] \times [\![B]\!])} \arrow[d, "T(\boldsymbol{\varGamma}_f+\mathrm{id})"] \\
{R([\![\varGamma]\!] \times [\![B]\!])} \arrow[r, "\alpha_{[\![\varGamma]\!]\times[\![B]\!]}"] & {T(R([\![\varGamma]\!] \times [\![B]\!]) + [\![\varGamma]\!] \times [\![B]\!]).}
\end{tikzcd}$$

A model is said to *satisfy* a well-typed equation $\varGamma \rhd t = s$ if $[\![\varGamma \rhd t : A]\!] = [\![\varGamma \rhd s : A]\!]$.

As suggestive abbreviations for use in process definitions, we write cont for (ret inl) and stop for (ret inr). Moreover, we write $(\mathsf{next}\, p\ \mathsf{is}\ \mathsf{rest}\, x \mapsto q;\ \mathsf{done}\, y \mapsto r)$ for $(\mathsf{do}\ z \leftarrow p; \mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}\, x \mapsto q;\ \mathsf{inr}\, y \mapsto r)$. We also define a converse $\mathsf{tuo} : T(T_\nu A + A) \to T_\nu A$ to out by

$$\mathsf{tuo}(p) \ = \ \mathsf{init}\, q := p\ \mathsf{unfold}\, \{\mathsf{next}\ q\ \mathsf{is}\ \mathsf{rest}\, y \mapsto \mathsf{cont}(\mathsf{out}(y));\ \mathsf{done}\, x \mapsto \mathsf{stop}\, x\}.$$

(case_inl) $\mathsf{case}\ \mathsf{inl}\,p\ \mathsf{of}\ \mathsf{inl}\,x \mapsto q;\ \mathsf{inr}\,y \mapsto r = q[p/x]$ **(fst)** $\mathsf{fst}\langle p, q\rangle = p$

(case_inr) $\mathsf{case}\ \mathsf{inr}\,p\ \mathsf{of}\ \mathsf{inl}\,x \mapsto q;\ \mathsf{inr}\,y \mapsto r = r[p/y]$ **(snd)** $\mathsf{snd}\langle p, q\rangle = q$

(case_id) $\mathsf{case}\ p\ \mathsf{of}\ \mathsf{inl}\,x \mapsto \mathsf{inl}\,x;\ \mathsf{inr}\,y \mapsto \mathsf{inr}\,y = p$ **(pair)** $\langle \mathsf{fst}\,p, \mathsf{snd}\,p\rangle = p$

(case_sub)
$$\begin{aligned}&\mathsf{case}\ p\ \mathsf{of}\ \mathsf{inl}\,x \mapsto t[q/z];\ \mathsf{inr}\,y \mapsto t[r/z]\\ &= t[\mathsf{case}\ p\ \mathsf{of}\ \mathsf{inl}\,x \mapsto q;\ \mathsf{inr}\,y \mapsto r/z]\end{aligned} \qquad (x, y \notin \mathrm{Vars}(r))$$

($\star$) $p : 1 = \star$ **(unit$_1$)** $\mathsf{do}\ x \leftarrow p; \mathsf{ret}\,x = p$ **(unit$_2$)** $\mathsf{do}\ x \leftarrow \mathsf{ret}\,a; p = p[a/x]$

(assoc) $\mathsf{do}\ x \leftarrow (\mathsf{do}\ y \leftarrow p; q); r = \mathsf{do}\ x \leftarrow p; y \leftarrow q; r$ $\quad (y \notin \mathrm{Vars}(r))$

(nil) $p + \varnothing = p$ **(comm)** $p + q = q + p$ **(idem)** $p + p = p$

(assoc_plus) $p + (q + r) = (p + q) + r$ **(dist_nil)** $\mathsf{do}\ x \leftarrow \varnothing; r = \varnothing$

(dist_plus) $\mathsf{do}\ x \leftarrow (p + q); r = \mathsf{do}\ x \leftarrow p; r + \mathsf{do}\ x \leftarrow q; r$

(coiter)
$$\frac{\mathsf{out}(p) = \mathsf{next}\ q\ \mathsf{is}\ \mathsf{rest}\,x \mapsto \mathsf{cont}\,p;\ \mathsf{done}\,y \mapsto \mathsf{stop}\,y}{p[y/x] = \mathsf{init}\,x := y\ \mathsf{unfold}\,\{q\}}$$

Fig. 2. Axiomatization of the concurrent metalanguage

An axiomatization ME_ν of the concurrent metalanguage is given in Fig. 2 (where we omit the standard equational logic ingredients including the obvious congruence rules). Apart from the standard axioms for products and coproducts, ME_ν contains three well-known monad laws, axioms for semi-additivity (middle section) and a novel (bidirectional) rule **(coiter)** for effectful coiteration.

Theorem 3. ME_ν *is sound and strongly complete over* ME_ν*-models.*

(Recall that *strong completeness* refers to completeness for logical consequence from possibly infinite sets of axioms.)

A core result on the concurrent metalanguage is an expressive corecursion scheme supported by the given simple axiomatisation. Its formulation is inspired by the treatment of guarded recursive equation systems in ACP [3]. It requires n-ary coproducts $A_1 + \dots + A_n$ with coproduct injections $\mathsf{inj}_i^n : A_i \to A_1 + \ldots + A_n$ and a correspondingly generalized case construct; all this can clearly be encoded in ME_ν.

Theorem and Definition 4 (Mutual corecursion). *Let $f_i : A_i \to T_\nu B_i$, $i = 1, \dots, k$ be fresh function symbols. A* guarded corecursive scheme *is a system of equations*

$$\mathsf{out}(f_i(x)) = \mathsf{do}\ z \leftarrow p_i; \mathsf{case}\ z\ \mathsf{of}\ \mathsf{inj}_1^{n_i}\,x_1 \mapsto p_1^i;\ \ldots\ ;\mathsf{inj}_{n_i}^{n_i}\,x_{n_i} \mapsto p_{n_i}^i$$

for $i = 1, \dots, k$ such that for every i, $p_i : T(C_{i1} + \dots + C_{in_i})$ does not contain any f_j, and for every i, j p_j^i either does not contain any f_m or is of the form $p_j^i \equiv \mathsf{cont}\, f_m(x_j)$

for some m. Such a guarded corecursive scheme uniquely defines $f_1, \dots, f_k$ (as morphisms in the model), and the solutions f_i are expressible as programs in $\mathsf{ME}_{\boldsymbol{\nu}}$.

Roughly, a guarded corecursive scheme defines, for each f_i, what happens when executing the first step of the resumption $f_i(x)$. The actual effect of the first step can depend on x but not on the f_m, while the new resumption is defined by corecursive calls to the f_m. As a first application of guarded corecursive schemes, we define a binding operation do_ν with the same typing as do but with T replaced by T_ν corecursively by

$$\mathsf{out}(\mathsf{do}_\nu\ x \leftarrow p; q) = \mathsf{next}\ \mathsf{out}(p)\ \mathsf{is}\ \mathsf{rest}\, x\ \mapsto \mathsf{cont}(\mathsf{do}_\nu\ x \leftarrow p; q);\ \mathsf{done}\, x \mapsto \mathsf{out}(q).$$

Similarly, we define operations ret_ν, $\varnothing_\nu$, and $+_\nu$ as analogues of ret, $\varnothing$, and $+$ by putting $\mathsf{ret}_\nu\, p = \mathsf{tuo}(\mathsf{stop}\, p)$, $\varnothing_\nu = \mathsf{tuo}(\varnothing)$, and $p +_\nu q = \mathsf{tuo}(\mathsf{out}(p) + \mathsf{out}(q))$. *These operations turn T_ν into a strong-semiadditive monad*; formally, we can derive (in $\mathsf{ME}_{\boldsymbol{\nu}}$) the top and middle sections of Fig. 2 with T replaced by T_ν (the monad laws already follow from results of Uustalu [32]).

Remark 5. Similarly as in Moggi's metalanguage [21], our models come with an arbitrary (distributive) base category $\mathbf{C}$, with $\mathbf{Set}$ and $\omega\mathbf{Cpo}$ as prominent instances, thus establishing our corecursion scheme (Theorem 4) at a high level of generality. Theorem 3 does depend on this broad notion of model. Contrastingly, the FIX-logic of Crole and Pitts [7] is designed to work with a smaller class of order-theoretic models.

4 Programming with Side-Effecting Processes

Above, we have begun to define operations on processes, namely $\varnothing_\nu$ and $+_\nu$. We next show how to define more complex operations, including parallel composition, by means of guarded corecursive schemes.

Note that over distributive categories, one can define the type of Booleans with the usual structure as $2 = 1 + 1$. We write $(\mathsf{if}\, b\, \mathsf{then}\, p\, \mathsf{else}\, q)$ as an abbreviation for $(\mathsf{case}\ b\ \mathsf{of}\ \mathsf{inl}\, x \mapsto p;\ \mathsf{inr}\, x \mapsto q)$ where b has type 2.

Sequential composition. Although T_ν is a monad, its binding operator is not quite what one would want as sequential composition of processes, as it merges the last step of the first process with the first step of the second process. We can, however, capture sequential composition (with the same typing) in the intended way by putting

$$\mathsf{seq}\ x \leftarrow p; q = \mathsf{do}_\nu\ x \leftarrow p; \mathsf{tuo}(\mathsf{cont}\, q).$$

Branching. Using the effect-free if operator defined earlier, we can define a conditional branching operator for processes $p, q : T_\nu A$ and a condition $b : T2$ by

$$\mathsf{if}_\nu\, b\, \mathsf{then}\, p\, \mathsf{else}\, q = \mathsf{tuo}\big(\mathsf{do}\ z \leftarrow b; \mathsf{if}\, z\, \mathsf{then}\, (\mathsf{cont}\, p)\ \mathsf{else}\ (\mathsf{cont}\, q)\big).$$

Looping. For terms $\Gamma \rhd p$; $\Gamma, x : A \rhd b : T2$; and $\Gamma, x : A \rhd q : T_\nu A$, we define loops

$$\Gamma \rhd \mathsf{init}\, x := p\ \mathsf{while}\ b\ \mathsf{do}\ q : T_\nu A \quad \text{and} \quad \Gamma \rhd \mathsf{init}\, x := p\ \mathsf{do}\ q\ \mathsf{until}\, b : T_\nu A$$

as follows. We generalize the until loop to a program $U^b_{x,q}(r)$ for $\Gamma \rhd r : T_\nu A$ intended to represent $\mathsf{seq}\ y \leftarrow r; \mathsf{init}\, x := y\ \mathsf{while}\ b\ \mathsf{do}\ q$ (so that $(\mathsf{init}\, x := p\ \mathsf{do}\ q\ \mathsf{until}\, b)$

$= U^b_{x,q}(q[p/x]))$ and abbreviate $W^b_{x,q}(p) = (\mathsf{init}\, x := p \;\mathsf{while}\; b \;\mathsf{do}\; q)$. We then define the four functions $W^b_{x,q}$ and $U^b_{x,q}$ (for $b : 2$) by the guarded corecursive scheme

$$\begin{aligned}\mathsf{out}(W^b_{x,q}(p)) &= \mathsf{do}\; v \leftarrow b[p/x]; \mathsf{if}\; v \;\mathsf{then}\; \mathsf{cont}(U^{\neg b}_{x,q}(q[p/x])) \;\mathsf{else}\; \mathsf{stop}(p),\\ \mathsf{out}(U^b_{x,q}(r)) &= \mathsf{next}\; \mathsf{out}(r) \;\mathsf{is}\; \mathsf{rest}\, z \mapsto \mathsf{cont}(U^b_{x,q}(z));\; \mathsf{done}\, y \mapsto \mathsf{cont}(W^{\neg b}_{x,q}(y)).\end{aligned}$$

Exceptions. As the concurrent metalanguage includes coproducts, the exception monad transformer($T^E A = T(A + E)$) [5] and the corresponding operations for raising and handling exceptions are directly expressible in $\mathsf{ME}_{\boldsymbol{\nu}}$.

Interleaving. We introduce process interleaving $\| : T_\nu A \times T_\nu B \to T_\nu(A \times B)$ by a CCS-style expansion law [20] (using an auxiliary left merge $\lfloor\!\lfloor$)

$$\begin{aligned}\mathsf{out}(p \parallel q) &= \mathsf{out}(p \lfloor\!\lfloor q) + \mathsf{do}\,\langle x, y\rangle \leftarrow \mathsf{out}(q \lfloor\!\lfloor p); \mathsf{ret}\langle y, x\rangle,\\ \mathsf{out}(p \lfloor\!\lfloor q) &= \mathsf{next}\; \mathsf{out}(p) \;\mathsf{is}\; \mathsf{rest}\, r \mapsto \mathsf{cont}(r \parallel q);\\ &\qquad\qquad\qquad\qquad \mathsf{done}\, x \mapsto \mathsf{cont}(\mathsf{do}_\nu\; y \leftarrow q; \mathsf{ret}_\nu\langle x, y\rangle).\end{aligned}$$

This is easily seen to be equivalent to the guarded corecursive scheme

$$\begin{aligned}\mathsf{out}(p \parallel q) = \;&\mathsf{do}\; u \leftarrow (p \lfloor q + p \rfloor q);\\ &\quad\mathsf{case}\; u \;\mathsf{of}\; \mathsf{inl}\langle s, t\rangle \mapsto \mathsf{cont}(s \parallel t);\; \mathsf{inr}\, r \mapsto \mathsf{cont}\, r\end{aligned}$$

where for $p : T_\nu A$, $q : T_\nu B$, $p \lfloor q : T(T_\nu A \times T_\nu B + T_\nu(A \times B))$ is defined as

$$p \lfloor q = \mathsf{next}\; \mathsf{out}(p) \;\mathsf{is}\; \mathsf{rest}\, r \mapsto \mathsf{ret}\,\mathsf{inl}\langle r, q\rangle;\; \mathsf{done}\, x \mapsto \mathsf{ret}\,\mathsf{inr}(\mathsf{do}_\nu\; y \leftarrow q; \mathsf{ret}_\nu\langle x, y\rangle)$$

and $p \rfloor q : T(T_\nu A \times T_\nu B + T_\nu(A \times B))$ is the evident dual of $p \lfloor q$.

5 Verification and Process Invariants

We now explore the potential of our formalism as a *verification* framework, extending existing monad-based program logics [28, 29] to concurrent processes. A cornerstone of these frameworks is a notion of *pure* program:

Definition 6 (Pure programs). A program $p : TA$ is *pure* if

- p is *discardable*, i.e., $\mathsf{do}\; y \leftarrow p; \mathsf{ret}\star = \mathsf{ret}\star$;
- p is *copyable*, i.e. $\mathsf{do}\; x \leftarrow p; y \leftarrow p; \mathsf{ret}\langle x, y\rangle = \mathsf{do}\; x \leftarrow p; \mathsf{ret}\langle x, x\rangle$; and
- p commutes with any other discardable and copyable program q, i.e. $(\mathsf{do}\; x \leftarrow p; y \leftarrow q; \mathsf{ret}\langle x, y\rangle) = \mathsf{do}\; y \leftarrow q; x \leftarrow p; \mathsf{ret}\langle x, y\rangle$.

Intuitively, pure programs are those that can access internal data behind the computation but cannot affect it. A typical example of a pure program is a getter method. Pure programs form a submonad P of T [28]. A *test* is a program of type $P2$ (recall that 2 denote the type of Booleans). All logical connectives extend to tests; e.g. $\neg b = (\mathsf{do}\; x \leftarrow b; \mathsf{ret}\,\neg x)$ for $b : P2$.

Given a program $p : TA$ and tests $\phi, \psi : P2$, the program $\mathsf{filter}(p, \phi, \psi) : TA$ is defined by the equation

$$\mathsf{filter}(p, \phi, \psi) = \mathsf{do}\ x \leftarrow \phi; y \leftarrow p; z \leftarrow \psi; \mathsf{if}(x \Rightarrow z)\ \mathsf{then}\ \mathsf{ret}\, y\ \mathsf{else}\ \varnothing.$$

Intuitively, filter modifies the given program p by removing those threads that satisfy the precondition ϕ but fail the postcondition ψ. This enables us to encode a Hoare triple by the equivalence

$$\{\phi\}p\{\psi\} \iff \mathsf{filter}(p, \phi, \psi) = p$$

— i.e. the Hoare triple $\{\phi\}p\{\psi\}$ is satisfied iff $\mathsf{filter}(p, \phi, \psi)$ does not remove any execution paths from p. On the other hand, filter extends to processes as follows:

$$\mathsf{filter}_\nu(p, \phi, \psi) = \mathsf{init}\, z := p\ \mathsf{unfold}\, \{\mathsf{tuo}(\mathsf{filter}(\mathsf{out}(z), \phi, \psi))\}.$$

It turns out that the above definition of Hoare triple is equivalent to a previous generic definition, which in particular enables use of existing sequential monad-based Hoare calculi [28, 29]:

Lemma 7. *For every program p and tests ϕ, ψ, $\{\phi\}p\{\psi\}$ is equivalent to the equation*

$$\mathsf{do}\ x \leftarrow \phi; y \leftarrow p; z \leftarrow \psi; \mathsf{ret}\langle x, y, z, x \Rightarrow z\rangle =$$
$$\mathsf{do}\ x \leftarrow \phi; y \leftarrow p; z \leftarrow \psi; \mathsf{ret}\langle x, y, z, \top\rangle.$$

A test ϕ is an *invariant* of process p if $\mathsf{filter}_\nu(p, \phi, \phi) = p$. We use $\mathsf{inv}(p, \phi)$ as a shorthand for this equality. Given a process $p : T_\nu A$, we define *partial execution* of p by

$$\mathsf{exec}(p) = \mathsf{tuo}(\mathsf{next}\ p\ \mathsf{is}\ \mathsf{rest}\, x \mapsto \mathsf{out}(x);\ \mathsf{done}\, x \mapsto \mathsf{stop}\, x).$$

For every p, $\mathsf{exec}(p)$ is precisely the program obtained by collapsing the first and the second steps of p into one. We denote by $\mathsf{exec}^n(p)$ the n-fold application of exec to p. This allows us to formalize satisfaction of a safety property ϕ by a process p:

'p is safe w.r.t. ψ at ϕ' iff for every n, $\{\phi\}\,\mathsf{exec}^n(p)\{\psi\}$.

Note, however, that this definition is not directly expressible in our logic, because it involves quantification over the naturals. Often this problem can be overcome by picking out a suitable process invariant.

Lemma 8. *Let ϕ, ψ, and ξ be tests such that $\phi \Rightarrow \xi$ and $\xi \Rightarrow \psi$, and let p be a process. Then* $\mathsf{inv}(p, \xi)$ *implies* $\{\phi\}\,\mathsf{exec}^n(p)\{\psi\}$ *for every n.*

6 Worked Example: Dekker's Mutual Exclusion Algorithm

We illustrate the use of our calculus by encoding Dekker's mutual exclusion algorithm. This algorithm was originally presented as an Algol program, and hence presumes some fixed imperative semantics, while we present (and verify) a version with *generic* side-effects. We introduce the following signature symbols:

$$\mathrm{set\_flag} : 2 \times 2 \to T1, \qquad \mathrm{set\_turn} : 2 \to T1,$$
$$\mathrm{get\_flag} : 2 \to P2, \qquad \mathrm{turn\_is} : 2 \to P2,$$

which can be roughly understood as interface functions accessing variables `flag1`, `flag2` and `turn`. This is justified by a suitable equational axiomatization of the above operators, which includes the following axioms (we assume $i \neq j$):

$$\begin{aligned}
\mathsf{do}\ \mathrm{set\_flag}(i,b); \mathrm{get\_flag}(i) &= \mathsf{do}\ \mathrm{set\_flag}(i,b); \mathsf{ret}\, b\\
\mathsf{do}\ \mathrm{set\_flag}(i,b); \mathrm{set\_flag}(j,c) &= \mathsf{do}\ \mathrm{set\_flag}(j,c); \mathrm{set\_flag}(i,b)\\
\mathsf{do}\ \mathrm{set\_flag}(i,b); \mathrm{set\_flag}(i,c) &= \mathrm{set\_flag}(i,c)
\end{aligned}$$

$$\begin{aligned}
\mathsf{do}\ \mathrm{set\_turn}(b); \mathrm{turn\_is}(c) &= \mathsf{do}\ \mathrm{set\_turn}(b); \mathsf{ret}(b \Leftrightarrow c)\\
\mathsf{do}\ \mathrm{set\_turn}(b); \mathrm{set\_turn}(c) &= \mathrm{set\_turn}(c).
\end{aligned}$$

(Obvious further axioms are omitted.) The crucial part of Dekker's algorithm is a (sub) program implementing *busy waiting*. In our case this is captured by the function $busy\_wait : 2 \to T1$ defined as follows:

$$\begin{aligned}
busy\_wait(i) = \mathsf{while}\ \mathrm{get\_flag}(\mathrm{flip}(i))\ \mathsf{do}\ \mathsf{if}_\nu\ &\mathrm{turn\_is}(\mathrm{flip}(i))\\
&\mathsf{then}\ \mathsf{seq}\,[\mathrm{set\_flag}(i,\bot)]; \mathsf{await}(\mathrm{turn\_is}(i))\\
&\mathsf{else}\ [\mathrm{set\_flag}(i,\top)]
\end{aligned}$$

Here, we used the following shorthands: $[p] = \mathsf{tuo}(\mathsf{stop}\, p)$ denotes the one-step process defined by $p : TA$, $\mathrm{flip} : 2 \to 2$ is the function swapping the coproduct components of $2 = 1 + 1$; $(\mathsf{while}\ b\ \mathsf{do}\ q)$ encodes $(\mathsf{init}\, x \leftarrow \star\ \mathsf{while}\ b\ \mathsf{do}\ q)$; finally, $(\mathsf{await}\, b)$ with b of type $T2$, intuitively meaning 'wait until b', is defined by the equation:

$$\mathsf{await}\, b = \mathsf{while}\ \neg b\ \mathsf{do}\ \mathsf{ret}_\nu\, \star.$$

Finally, we define a generic process accessing the critical section:

$$\begin{aligned}
proc(i,p) = \mathsf{seq}\,&[\mathrm{set\_flag}(i,\top)]; busy\_wait(i);\\
&[\mathrm{in\_cs}(i)]; p; [\mathrm{out\_cs}(i)];\\
&[\mathrm{set\_turn}(\mathrm{flip}(i))]; [\mathrm{set\_flag}(i,\bot)].
\end{aligned}$$

Here we use the functions $\mathrm{in\_cs}, \mathrm{out\_cs} : 2 \to T1$ in order to keep track of the beginning and the end of the critical section. These functions are supposed to work together with the testing function $\mathrm{cs} : 2 \to T2$ as prescribed by the axioms

$$\begin{aligned}
\mathsf{do}\ \mathrm{in\_cs}(i); \mathrm{cs}(i) &= \mathsf{do}\ \mathrm{in\_cs}(i); \mathsf{ret}\, \top,\\
\mathsf{do}\ \mathrm{out\_cs}(i); \mathrm{cs}(i) &= \mathsf{do}\ \mathrm{out\_cs}(i); \mathsf{ret}\, \bot.
\end{aligned}$$

Now the safety condition for the algorithm can be expressed by the formula

$$\forall n.\ \{\neg\mathrm{cs}(\bar{1}) \wedge \neg\mathrm{cs}(\bar{2})\}\ \mathsf{exec}^n(proc(\bar{1},p) \parallel proc(\bar{2},q))\{\neg\mathrm{cs}(\bar{1}) \vee \neg\mathrm{cs}(\bar{2})\}$$

where $\bar{1}$ and $\bar{2}$ are the canonical coproduct injections $\mathsf{inl}\,\star$ and $\mathsf{inr}\,\star$. By Lemma 8, it suffices to show that the following formula

$$\neg\mathrm{cs}(\bar{1}) \wedge \mathrm{cs}(\bar{2}) \wedge \mathrm{get\_flag}(\bar{2})\ \vee \neg\mathrm{cs}(\bar{2}) \wedge \mathrm{cs}(\bar{1}) \wedge \mathrm{get\_flag}(\bar{1})\ \vee \neg\mathrm{cs}(\bar{1}) \wedge \neg\mathrm{cs}(\bar{2})$$

is an invariant of $proc(\bar{1},p) \parallel proc(\bar{2},q)$. As can be shown by definition of parallel composition, this holds iff the same formula is an invariant of both $proc(\bar{1},p)$ and $proc(\bar{2},q)$, which in turn can be shown by coinduction in ME_ν.

7 Conclusions and Further Work

We have studied asynchronous concurrency in a framework of generic effects. To this end, we have combined the theories of computational monads and final coalgebras to obtain a framework for processes with generic side-effecting steps, the *concurrent metalanguage* ME_ν. We have presented a sound and complete equational calculus for ME_ν, and we have obtained a syntactic corecursion scheme in which corecursive functions are syntactically reducible to a basic loop construct. Within this calculus, we have given generic definitions for standard imperative constructs and a number of standard process operators, most notably parallel composition.

In future research, we intend to develop more expressive verification logics for side-effecting processes, detached from equational reasoning. Initial results of this kind have already been used in an example verification of a generic mutual exclusion scheme following Dekker's algorithm. Specifically, we have provided an encoding of generic Hoare triples and an associated proof principle for safety properties. An interesting perspective in this direction is to identify a variant of the assume/guarantee principle for side-effecting processes (cf. e.g. [8]). A further topic of investigation is to develop weak notions of process equivalence in our framework, such as testing equivalence [22].

Finally, the decidability status of ME_ν remains open. Note that in case of a positive answer, all equations between functions defined by corecursion schemes, e.g. process algebra identities, become decidable. While experience suggests that even very simple calculi that combine loop constructs with monadic effects tend to be undecidable, the corecursion axiom as a potential source of trouble seems rather modest, and no evident encoding of an undecidable problem appears to be directly applicable.

References

[1] Bartels, F.: Generalised coinduction. Math. Structures in Comp. Sci. 13(2), 321–348 (2003)
[2] Benton, N., Hyland, M.: Traced premonoidal categories. ITA 37(4), 273–299 (2003)
[3] Bergstra, J.A., Klop, J.W.: The algebra of recursively defined processes and the algebra of regular processes. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 82–94. Springer, Heidelberg (1984)
[4] Capretta, V.: General recursion via coinductive types. Logical Methods in Computer Science 1(2) (2005)
[5] Cenciarelli, P., Moggi, E.: A syntactic approach to modularity in denotational semantics. Technical report, Category Theory and Computer Science (1993)
[6] Cockett, J.R.B.: Introduction to distributive categories. Mathematical Structures in Computer Science 3(3), 277–307 (1993)
[7] Crole, R.L., Pitts, A.M.: New foundations for fixpoint computations. In: Logic in Computer Science, LICS 1990, pp. 489–497. IEEE Computer Society Press, Los Alamitos (1990)
[8] de Roever, W.P., de Boer, F.S., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge University Press, Cambridge (2001)
[9] Erkök, L., Launchbury, J.: Recursive monadic bindings. In: ICFP 2000, pp. 174–185. ACM, New York (2000)
[10] Filinski, A.: On the relations between monadic semantics. Theor. Comp. Sci. 375, 41–75 (2007)

[11] Fiore, M.P., Moggi, E., Sangiorgi, D.: A fully abstract model for the π-calculus. Inf. Comput. 179(1), 76–117 (2002)
[12] Goncharov, S.: Kleene monads. PhD thesis, Universität Bremen (2010)
[13] Harrison, W.L.: The essence of multitasking. In: Johnson, M., Vene, V. (eds.) AMAST 2006. LNCS, vol. 4019, pp. 158–172. Springer, Heidelberg (2006)
[14] Harrison, W.L., Hook, J.: Achieving information flow security through monadic control of effects. J. Computer Security 17, 599–653 (2009)
[15] Hennessy, M., Plotkin, G.D.: Full abstraction for a simple parallel programming language. In: Becvar, J. (ed.) MFCS 1979. LNCS, vol. 74, pp. 108–120. Springer, Heidelberg (1979)
[16] Jacobs, B.: From coalgebraic to monoidal traces. Electron. Notes Theor. Comput. Sci. 264(2), 125–140 (2010)
[17] Jacobs, B., Poll, E.: Coalgebras and Monads in the Semantics of Java. Theoret. Comput. Sci. 291, 329–349 (2003)
[18] Krstić, S., Launchbury, J., Pavlovi, D.P.: Categories of processes enriched in final coalgebras. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 303–317. Springer, Heidelberg (2001)
[19] Milius, S., Palm, T., Schwencke, D.: Complete iterativity for algebras with effects. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 34–48. Springer, Heidelberg (2009)
[20] Milner, R.: Communication and concurrency. Prentice-Hall, Inc, Englewood Cliffs (1989)
[21] Moggi, E.: Notions of computation and monads. Inf. Comput. 93, 55–92 (1991)
[22] Nicola, R.D., Hennessy, M.: Testing equivalence for processes. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 548–560. Springer, Heidelberg (1983)
[23] Papaspyrou, N., Macos, D.: A study of evaluation order semantics in expressions with side effects. J. Funct. Program. 10(3), 227–244 (2000)
[24] Peyton Jones, S. (ed.): Haskell 98 Language and Libraries — The Revised Report. Cambridge (2003); also: J. Funct. Programming 13, 2003.
[25] Plotkin, G., Power, J.: Semantics for algebraic operations. In: Mathematical Foundations of Programming Semantics, MFPS 2001. ENTCS, vol. 45 (2001)
[26] Rosenthal, K.I.: Quantales and their applications. Pitman Research Notes in Mathematics Series, vol. 234. Longman Scientific & Technical (1990)
[27] Rutten, J.: Universal coalgebra: A theory of systems. Theoret. Comput. Sci. 249, 3–80 (2000)
[28] Schröder, L., Mossakowski, T.: Monad-independent dynamic logic in HasCASL. J. Logic Comput. 14, 571–619 (2004)
[29] Schröder, L., Mossakowski, T.: HasCASL: Integrated higher-order specification and program development. Theor. Comput. Sci. 410(12-13), 1217–1260 (2009)
[30] Syme, D., Granicz, A., Cisternino, A.: Expert F#. Apress (2007)
[31] Tolmach, A.P., Antoy, S.: A monadic semantics for core Curry. In: WFLP 2003. ENTCS, vol. 86(3), pp. 16–34 (2003)
[32] Uustalu, T.: Generalizing substitution. ITA 37(4), 315–336 (2003)
[33] Wadler, P.: How to declare an imperative. ACM Computing Surveys 29, 240–263 (1997)
[34] Worrell, J.: Terminal sequences for accessible endofunctors. In: Coalgebraic Methods in Computer Science, CMCS 1999. ENTCS, vol. 19 (1999)

Hardness, Approximability, and Exact Algorithms for Vector Domination and Total Vector Domination in Graphs[⋆]

Ferdinando Cicalese[1], Martin Milanič[2], and Ugo Vaccaro[1]

[1] University of Salerno, Dipartimento di Informatica, 84084 Fisciano (SA), Italy
{cicalese,uv}@dia.unisa.it
[2] University of Primorska, UP FAMNIT and UP PINT, 6000 Koper, Slovenia
martin.milanic@upr.si

Abstract. We consider two graph optimization problems called vector domination and total vector domination. In vector domination one seeks a small subset S of vertices of a graph such that any vertex outside S has a prescribed number of neighbors in S. In total vector domination, the requirement is extended to all vertices of the graph. We prove that these problems cannot be approximated to within a factor of $c \log n$, for suitable constants c, unless every problem in NP is solvable in slightly super-polynomial time. We also show that two natural greedy strategies have approximation factors $O(\log \Delta(G))$, where $\Delta(G)$ is the maximum degree of the graph G. We also provide exact polynomial time algorithms for several classes of graphs. Our results extend, improve, and unify several results previously known in the literature.

1 Introduction

The concept of domination in graphs has been extensively studied, both in structural and algorithmic graph theory, because of its numerous applications to a variety of areas. Informally, a vertex of a graph is said to dominate itself and all of its neighbors. Generally, one seeks small sets that dominate the whole graph. Domination naturally arises in facility location problems, in problems involving finding sets of representatives, in monitoring communication or electrical networks, and in land surveying. The two books [8] [9] discuss the main results and applications of domination in graphs. Many variants of the basic concepts of domination have appeared in the literature. Again, we refer to [8] [9] for a survey of the area.

In this paper we provide hardness results and approximation algorithms for an interesting generalization of the basic concept of domination, firstly introduced in [7]. Here, a subset of vertices S is said to dominate a vertex v if either $v \in S$, or there are in S a *prescribed* number of neighbors of v (see below for formal definitions). Again, one seeks small subsets that dominate (in this new sense) the whole vertex set of the graph.

Main Definitions. For a graph $G = (V,E)$ and a vertex $v \in V$, denote by $N(v)$ the set of neighbors of v, by $d(v)$ the degree of v, and by $\Delta(G)$ the maximum degree of any vertex

⋆ Supported in part by "Agencija za raziskovalno dejavnost Republike Slovenije", research program P1-0285.

in G. A *dominating set* in a graph $G = (V,E)$ is a subset S of the graph's vertex set such that every vertex not in the set has a neighbor in the set. A *total dominating set* in G is a subset $S \subseteq V$ such that every vertex of the graph has a neighbor in the set: for every $v \in V$ there exists a vertex $u \in S$ such that $uv \in E$.

The *vector domination* is the following problem: Given a graph $G = (V,E)$, and a vector $(k_v : v \in V)$ such that for all $v \in V$, $k_v \in \{0,1,\dots,d(v)\}$, find a *vector dominating set* (VDS) S of minimum size, that is, a set $S \subseteq V$ minimizing $|S|$ and such that $|S \cap N(v)| \geq k_v$ for all $v \in V \backslash S$. The *total vector domination* is the problem of finding a minimum-sized *total vector dominating set*, that is, a set $S \subseteq V$ such that $|S \cap N(v)| \geq k_v$ for all $v \in V$. In the context of computer networks, a dominating set D of nodes represents a set of nodes each of which has a resource, or service capability, and each node that does not have this resource, or needs this service, can gain access to it by accessing a neighboring node. A vector dominating set could be useful in those scenarios in which it is required that nodes not having a resource must have a prescribed number of neighbors possessing said resource. This requirement could be dictated by fault-tolerance prescriptions, or by other natural needs like security or privacy, in the same spirit of [14]. Other important applications of vector dominating sets will be subsequently discussed in the paper.

Of special interest for us will be also the following special cases of vector domination: For $0 \leq q < 1$, a *q-dominating set* in G is a subset $S \subseteq V$ such that every vertex not in the set has more than a q-fraction of its neighbors in the set: for every $v \in V \setminus S$, it holds that $|N(v) \cap S| > q|N(v)|$. For $0 \leq q < 1$, a *total q-dominating set* in G is a subset $S \subseteq V$ such that every vertex has more than a q-fraction of its neighbors in the set: for every $v \in V$, it holds that $|N(v) \cap S| > q|N(v)|$. By $\gamma(G)$ ($\gamma^q(G)$, $\gamma_t(G)$, $\gamma_t^q(G)$) we denote the minimum size of a dominating (q-dominating, total dominating, total q-dominating) set in G. The problem of finding (for a fixed $0 \leq q < 1$) in a given graph a dominating (q-dominating, total q-dominating) set of minimum size will be referred to simply as the *domination* (*q-domination*, *total q-domination*). Notice that for every graph G, for all $q < 1/\Delta(G)$, the (total) q-dominating sets of G correspond to the graph's (total) dominating sets, while for all $q \geq 1 - 1/\Delta(G)$, the q-dominating sets of G correspond to the graph's vertex covers.

Clearly, the (total) q-domination corresponds to the special case of the (total) vector domination, in which $k_v = \lfloor q \cdot d(v) \rfloor + 1$ for all $v \in V$. In fact, we shall mainly use q-domination for our inapproximability results, and provide algorithmic results in terms of the more general problem of vector domination.

Our Results and Related Work. We first provide two natural greedy algorithms for vector domination and total vector domination in general graphs, having approximation factor of $H_{2\Delta(G)}$ and $H_{\Delta(G)}$, respectively. Subsequently, we prove that above results are essentially best possible, in the sense that both the q-domination and its total variant are inapproximable within an $O(\log|V(G)|)$ factor, unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$. Notice that our inapproximability result is provided for *any* fixed $0 \leq q < 1$, hence it is not subsumed by the standard domination and total domination problems (except for $q = 0$). We individuate special classes of graphs for which vector domination and total vector domination can be optimally solved in polynomial time. More specifically, we provide polynomial time algorithms for computing minimum size vector domination sets and total vector domination sets for trees (P_4-free graphs and threshold graphs are considered in the extended version of the paper [2]).

The algorithmic aspect of total vector domination in strongly chordal graphs (a superclass of trees) was studied in [6], where polynomial time algorithms for that purpose were given. However, the authors of [6] point out that their approach cannot be modified to handle the case of vector domination, and that a new approach is needed.

Strictly related to our results is also the paper [11]. The authors study the hardness of approximating minimum monopolies in graphs. Monopolies in graphs represent an important sub-area in graph theory, with many applications in distributed computing (see the survey [15]). In our language, a monopoly corresponds to a total $1/2$-dominating set, and a partial monopoly to a $1/2$-dominating set. Therefore, our inapproximability results for q-domination and total q-domination can be seen as a significant extensions of the results of [11] from the case $q = 1/2$ to arbitrary q (the paper [11] obtains better inapproximability multiplicative constants under stronger complexity assumptions). Conversely, our algorithmic results on trees extend the corresponding result of [12] from total $1/2$-domination to the much more general total vector domination problem.

The paper [5] studies the hardness of approximating k-tuple domination in graphs. In our framework, their problem is equivalent to vector domination in the special case where the input vector $(k_v : v \in V)$ has all components equal to an integer k. Therefore, our results generalize also [5].

Our findings are also relevant to the important new area of influence spread in social networks [4]. For instance, paper [17] investigates algorithmic and complexity aspects of positive influence dominating sets (PIDS) in social networks. In our language, PIDS correspond to total vector dominating sets where the vector $(k_v : v \in V)$ is such that $k_v = \lceil d(v)/2 \rceil$ for each v in the network. In [17] it is proved that PIDS is APX-hard. Our hardness results for total q-domination are more general, and also stronger since we prove non approximability within a logarithmic factor. In the same area, the paper [19] introduced the problem of identifying a minimum set of nodes that could influence a whole network within a time bound d. There, a set of nodes S influences a new node x in one step ($d = 1$) if the majority of neighbors of x is in S. The paper [19] mostly studies hardness results for the case $d = 1$. It is clear that our scenario includes that of [19] (in the case $d = 1$) and corresponds to a more extensive model of influence among nodes, similar to the one considered in [13] for a related but different problem.

2 Approximability Results

In this section, we show that vector domination and total vector domination can be approximated in polynomial time by a factor of $H_{2\Delta(G)}$ and $H_{\Delta(G)}$, respectively. (We denote by $H_k = \sum_{i=1}^{k} \frac{1}{i}$ the k-th harmonic number.) Since $H_k \leq \log k + 1$ for $k \geq 1$, the algorithms given by the theorems below provide $O(\log \Delta(G))$-approximation for vector and total vector domination, respectively. (We denote by log the natural logarithm.)

Theorem 1. *Vector domination can be approximated in polynomial time by a factor of $H_{2\Delta(G)}$.*

Proof. For a graph $G = (V,E)$ and a vector $(k_v : v \in V)$ s.t. for all $v \in V$, $k_v \in \{0,1,\ldots,d(v)\}$, we define a function $f : 2^V \to \mathbb{N}$, as follows:

$$f(S) = \sum_{v \in V} \tau_v(S)\,, \quad \text{where } \tau_v(S) = \begin{cases} \min\{|S \cap N(v)|, k_v\}, & \text{if } v \notin S; \\ k_v, & \text{if } v \in S. \end{cases} \tag{1}$$

The following properties of f can be verified: (i) f is integer-valued; (ii) $f(\emptyset) = 0$; (iii) *f is non-decreasing*; (iv) *A set $S \subseteq V$ satisfies $f(S) = f(V)$ if and only if S is a vector dominating set*; (v) *f is submodular.*

A function $f : 2^V \to \mathbb{N}$ is *submodular* if for all $S \subseteq T \subseteq V$ and for all $w \in V$, the inequality $f(T \cup \{w\}) - f(T) \le f(S \cup \{w\}) - f(S)$ holds. The only non-trivial property to show is (v), i.e, the submodularity of f. The proof is given below.

Fact 1. *The function $f : 2^V \to \mathbb{N}$, given by (1), is submodular.*

Proof. It suffices to show that all the functions $\tau_v(\cdot)$ are submodular, that is, that for all $S \subseteq T \subseteq V$ and for all $w \in V$,

$$\tau_v(T \cup \{w\}) - \tau_v(T) \le \tau_v(S \cup \{w\}) - \tau_v(S)\,. \tag{2}$$

Observe that τ_v is non-decreasing.

Suppose first that $\tau_v(T) = k_v$. Then $\tau_v(T \cup \{w\}) = k_v$ and the left-hand side of inequality (2) is equal to 0. Hence inequality (2) holds since τ_v is non-decreasing.

From now on, we assume that $\tau_v(T) < k_v$, which implies $\tau_v(T) = |T \cap N_G(v)|$. Since τ_v is non-decreasing, $\tau_v(S) < k_v$, and hence $\tau_v(S) = |S \cap N_G(v)|$. Inequality (2) simplifies to

$$\tau_v(T) - \tau_v(S) = |(T \setminus S) \cap N_G(v)| \ge \tau_v(T \cup \{w\}) - \tau_v(S \cup \{w\})\,. \tag{3}$$

We may assume that $\tau_v(T \cup \{w\}) > \tau_v(S \cup \{w\})$, since otherwise the right-hand side of (3) equals 0, and inequality (3) holds.

Therefore, $\tau_v(S \cup \{w\}) < k_v$, implying $\tau_v(S \cup \{w\}) = |(S \cup \{w\}) \cap N_G(v)|$. If also $\tau_v(T \cup \{w\}) < k_v$ then $\tau_v(T \cup \{w\}) = |(T \cup \{w\}) \cap N_G(v)|$ and equality holds in (3).

So we may assume that $\tau_v(T \cup \{w\}) = k_v$. Note that v does not belong to $T \cup \{w\}$ (since otherwise either $\tau_v(T)$ or $\tau_v(S \cup \{w\})$ would equal to k_v). Suppose that the inequality (3) fails. Then

$$|(T \setminus S) \cap N_G(v)| < k_v - |(S \cup \{w\}) \cap N_G(v)|\,,$$

which implies

$$|(T \cup \{w\}) \cap N_G(v)| < k_v\,.$$

However, together with the fact that $v \notin T \cup \{w\}$, this contradicts the assumption that $\tau_v(T \cup \{w\}) = k_v$. □

Back to the proof of Theorem 1, by (iv) we have that an optimal solution to the vector dominating set is provided by a minimum size S such that $f(S) = f(V)$. In other words, we have recast vector domination as a particular case of the well known `MINIMUM SUBMODULAR COVER` [18].

Let $\mathbb{A}$ denote the natural greedy strategy which starts with $S = \emptyset$ and iteratively adds to S the element $v \in V \setminus S$ s.t. $f(S \cup \{v\}) - f(S)$ is maximum, until $f(S) = f(V)$ is achieved. By a classical result of Wolsey [18], it follows that algorithm $\mathbb{A}$ is an H_τ-approximation algorithm for vector domination, where $\tau = \max_{y \in V} f(\{y\})$. For every $y \in V$, we have $f(\{y\}) = \sum_{v \in V \setminus \{y\}} \tau_v(\{y\}) + \tau_y(\{y\}) \leq d(y) + k_y \leq 2d(y)$. Hence $\max_{y \in V} f(\{y\}) \leq 2\Delta(G)$ yielding the desired result. □

Theorem 2. *Total vector domination can be approximated in polynomial time by a factor of* $H_{\Delta(G)}$.

Proof. The argument is analogous to the one used for Theorem 1. Given a graph $G = (V,E)$ and vector $(k_v : v \in V)$ s.t. for all $v \in V$, $k_v \in \{0,1,\ldots,d(v)\}$, we define a function $f : 2^V \to \mathbb{N}$, as follows:

$$f(S) = \sum_{v \in V} \min\{|S \cap N(v)|, k_v\}\,. \tag{4}$$

Like (1) this function f also satisfies properties (i)-(iii) defined above. A set $S \subseteq V$ satisfies $f(S) = f(V)$ if and only if S is a *total* vector dominating set. Moreover, similarly as was done in Fact 1, it can be verified that f is submodular.

Therefore, again by the results of Wolsey the natural greedy strategy provides an H_τ-approximation algorithm for total vector domination, where $\tau = \max_{y \in V} f(\{y\})$. It can be seen that $\max_{y \in V} f(\{y\}) \leq \Delta(G)$, which concludes the proof. □

3 Inapproximability Results

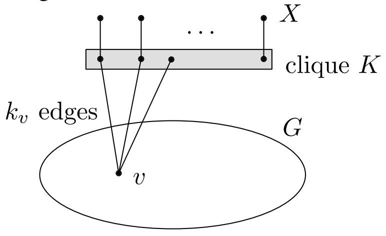
Our inapproximability results are given in terms of the q-domination problem. In fact, it turns out that both the q-domination and its total variant are inapproximable within a $\log|V(G)|$ factor as shown in Theorems 3 and 4 below. *A fortiori* the same results hold for the vector domination problem. Hence, the approximations results of the previous section are basically best possible. Due to the space constraint, we will limit ourselves to sketch the inapproximability result on q-domination (Theorem 3). We shall use the following lemma which is basically an *ad hoc* extension of the hardness of approximating domination within $c\log|V(G)|$ given in [1] (for the proof, see citeCMV).

Lemma 1. *There exists a constant* $c > 0$ *such that for every integer* $B > 0$ *there is no polynomial time algorithm approximating domination on input graphs G satisfying* $\gamma(G) \geq B\Delta(G)$ *within a factor of* $c\log|V(G)|$*, unless NP* $\subseteq$ *DTIME*$(n^{O(\log\log n)})$.

Theorem 3. *There exists a constant* $c > 0$ *such that for every* $0 \leq q < 1$ *there is no polynomial time algorithm approximating q-domination within a factor of* $c\log|V(G)|$*, unless NP* $\subseteq$ *DTIME*$(n^{O(\log\log n)})$.

Proof. Let $0 \leq q < 1$, and let $B = \lceil \frac{q}{1-q} \rceil$. Let G be a graph with $\gamma(G) \geq B\Delta(G)$ and such that $|V(G)| \geq 4B$. We transform G into a graph G' which consists of G together with a complete graph K on $k = B\Delta(G)$ vertices such that K is disjoint from G.

In addition, every vertex v from G is adjacent to precisely $k_v = \lfloor \frac{qd_G(v)}{1-q} \rfloor$ vertices in K. (This assignment is done in an arbitrary way.) Finally, every vertex in the clique K has a private neighbor outside $V(G)$. We denote by X the set of all these private neighbors (see figure below).



The graph G' in the proof of Theorem 3

Notice that $k_v = \left\lfloor \frac{qd_G(v)}{1-q} \right\rfloor \leq \left\lceil \frac{qd_G(v)}{1-q} \right\rceil \leq \left\lceil \frac{q}{1-q} \right\rceil \Delta(G) = k$. Hence it is indeed possible to assign to v precisely k_v neighbors in K. In addition, k_v is an integer satisfying $\frac{k_v}{d_G(v)+k_v} \leq q < \frac{k_v+1}{k_v+d_G(v)}$. These inequalities are instrumental to the following Claim, whose proof is in [2].

Claim: $\gamma^q(G') = \gamma(G) + k$.

Let c be the constant given by Lemma 1, and let $c' = c/4$. Let us write $n = |V(G)|$ and $n' = |V(G')|$. Note that, by the assumption $|V(G)| \geq 4B$, it follows that $n' = n + 2k = n + 2B\Delta(G) \leq 1/2n^2 + 1/2n^2 = n^2$. Suppose for a contradiction that there exists a polynomial time algorithm A which computes a q-dominating set S' for G' such that $|S'| \leq c'(\log n')\gamma^q(G')$.

Let $S = S' \cap V(G)$. It is not hard to see that S is a dominating set in G. Moreover, we can bound the size of S as follows:

$$|S| \leq |S'| \leq c'(\log n')\gamma^q(G') \leq c'(\log(n^2))(\gamma(G)+k) \leq 2c'(\log n)(2\gamma(G)),$$

where the second inequality follows by the assumption on A; the third one by $n' \leq n^2$ and $\gamma^q(G') = \gamma(G) + k$; the fourth one by $k = B\Delta(G) \leq \gamma(G)$. Finally, by the choice of c' we get $|S| \leq c(\log n)\gamma(G)$, and the conclusion follows by Lemma 1. □

By means of a slightly more involved construction, we can prove the analogous result for total q-domination.

Theorem 4. *There exists a constant $c > 0$ such that for every $0 \leq q < 1$, there is no polynomial time algorithm approximating total q-domination within a factor of $c \log |V(G)|$, unless NP $\subseteq$ DTIME$(n^{O(\log \log n)})$.*

4 A Polynomial Algorithm for Vector Domination in Trees

Since trees are strongly chordal, total vector domination is solvable in time $O(n+m)$ on trees [6,16], where $n = |V(G))|$ and $m = |E(G)|$. As mentioned in [6], their approach does not apply to the vector domination problem. In this section we describe an $O(n^2)$ algorithm that solves vector domination in trees. The algorithm is based on an efficient solution to the following problem:

CARDINALITY-CONSTRAINED PARTITION (*CCP*):

Given n ordered pairs of real numbers $(a_1,b_1),(a_2,b_2),\dots,(a_n,b_n)$ and an integer k such that $0 \le k \le n$, find a partition (I,J) of the set $\{1,\dots,n\}$ with $|I| = k$ that minimizes the sum $\sum_{i\in I} a_i + \sum_{j\in J} b_j$.

This problem admits an $O(n(k+1))$ solution by dynamic programming: For $1 \le i \le n$ and $0 \le j \le \min\{i,k\}$, let $v_{i,j}$ denote the optimum value of the *CCP* problem with the input $((a_1,b_1),\dots,(a_i,b_i);j)$. Clearly, $v_{n,k}$ is an optimal value of the above *CCP* problem. The values $v_{i,j}$ can be computed in $O(n(k+1))$ time using the following recurrences:

(1) $v_{1,0} = b_1, \quad v_{1,1} = a_1$;
(2) $v_{i,0} = v_{i-1,0} + b_i$ for all $2 \le i \le n$;
(3) $v_{i,i} = v_{i-1,i-1} + a_i$ for all $2 \le i \le k$;
(4) $v_{i,j} = \min\{v_{i-1,j-1} + a_i, v_{i-1,j} + b_i\}$ for all $2 \le i \le k$ and $1 \le j \le \min\{i,k\}$.

In what follows, we will denote by $CCP(A,k)$ the optimal value of the *CCP* problem on the input $((a_{11},a_{21}),\dots,(a_{n1},a_{n2});k)$, where $A = \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ a_{21} & \cdots & a_{n2} \end{pmatrix}$ is a $2 \times n$ matrix and $k \le n$ is a non-negative integer.

Theorem 5. *A minimum vector dominating set in a tree can be found in time $O(n^2)$.*

Proof. We claim that Algorithm 1 below computes a minimum vector dominating set for (T,k), where T is a tree. Let us root T at an arbitrary vertex r. For $v \in V(T)$, we denote by T_v the subtree of T induced by v and all its descendants. For a subgraph H of G, we denote by $k|_H$ the restriction of k to $V(H)$. The algorithm will compute, using a bottom-up traversal of the tree, the following values, for all $v \in V(T)$:

– $\gamma(v)$: the minimum size of a vector dominating set for $(T_v, k|_{T_v})$;
– $\gamma^+(v)$: the minimum size of a vector dominating set for $(T_v, k|_{T_v})$ that contains v;
– $\gamma^-(v)$: the minimum size of a vector dominating set for (T_v, k_v^-), where $k_v^- : V(T_v) \to \mathbb{Z}$ is given by
$$k_v^-(u) = \begin{cases} \max\{k(v)-1,0\}, & \text{if } u = v; \\ k(u), & \text{otherwise.} \end{cases}$$

The following proposition establishes a way to compute these values:

Proposition 1. *Let v be an internal node of T, and let $C(v)$ denote the set of children of v. Let A be the $2 \times |C(v)|$ matrix with $A_{1j} = \gamma^+(j)$ and $A_{2j} = \gamma(j)$ for all $j \in C(v)$. Then:*

(i) $\gamma^+(v) = \sum_{j\in C(v)} \gamma^-(j) + 1$.
(ii) If $k(v) > |C(v)|$ *then* $\gamma(v) = \gamma^+(v)$. *Otherwise,* $\gamma(v) = \min\{\gamma^+(v), CCP(A,k(v))\}$.
(iii) If $k(v) > |C(v)| + 1$ *then* $\gamma^-(v) = \gamma^+(v)$. *Otherwise,* $\gamma^-(v) = \min\{\gamma^+(v), CCP(A, \max\{k(v)-1,0\})\}$.

Proof. (i). For all $j \in C(v)$, let D_j denote a minimum vector dominating set for (T_j, k_j^-). Then, the set $\bigcup_j D_j \cup \{v\}$ is a vector dominating set for $(T_v, k|_{T_v})$ that contains v; hence $\gamma^+(v) \le \sum_{j\in C(v)} \gamma^-(j) + 1$. Conversely, if D is a minimum-sized vector dominating set for $(T_v, k|_{T_v})$ that contains v, then for every $j \in C(v)$, the set $D \cap V(T_j)$ is a vector

Algorithm 1. Vector domination in trees

```
Input: A tree $T = (V,E)$, a function $k : V \to \mathbb{Z}$.
Output: The minimum size of a VDS for $(T,k)$.
 1: Let $R = \{v \in V(T) \ : k(v) > d(v)\}$.
 2: Set $T$ to $T-R$ and $k$ to $k' : V(T-R) \to \mathbb{Z}$, given by $k'(v) = k(v) - |N(v) \cap R|$ for all $v \in V(T)-R$.
 3: if $T-R$ is disconnected, with components $T_1,\dots,T_p$
 4:     Solve the problem recursively on all $(T_i, k|_{T_i})$, for all $i$.
 5:     Let $\gamma_i$ denote the minimum size of a vector dominating set for $(T_i, k|_{T_i})$.
 6:     return $\sum_{i=1}^{p} \gamma_i + |R|$.
 7: Let $r \in V(T)$ and root $T$ at $r$.
 8: for all leaves $\ell$ of $T$ (other than the root) do
 9:     set $\gamma(\ell) = k(\ell)$. (since at this point $k(\ell) \in \{0,1\}$.)
10:     set $\gamma^+(\ell) = 1$.
11:     set $\gamma^-(\ell) = 0$.
12: for all internal nodes $v$ of $T$ (traversed in a bottom-up manner) do
13:     let $C(v)$ be the set of children of $v$.
14:     set $\gamma^+(v) = \sum_{j \in C(v)} \gamma^-(j) + 1$.
15:     let $A$ be the $2 \times |C(v)|$ matrix with $A_{1j} = \gamma^+(j)$ and $A_{2j} = \gamma(j)$ for all $j \in C(v)$.
16:     if $k(v) > |C(v)|$
17:         set $\gamma(v) = \gamma^+(v)$.
18:     else
19:         set $\gamma(v) = \min\{\gamma^+(v), CCP(A,k(v))\}$.
20:     if $k(v) > |C(v)| + 1$
21:         set $\gamma^-(v) = \gamma^+(v)$.
22:     else
23:         set $\gamma^-(v) = \min\{\gamma^+(v), CCP(A, \max\{k(v)-1, 0\})\}$.
24: return $\gamma(r)$.
```

dominating set for (T_j, k_j^-). Therefore, $|D \cap V(T_j)| \geq \gamma^-(j)$ and consequently $\gamma^+(v) = |D| \geq \sum_{j \in C(v)} \gamma^-(j) + 1$.

(ii). If $k(v) > |C(v)|$ then every vector dominating set for $(T_v, k|_{T_v})$ contains v, so we have $\gamma(v) = \gamma^+(v)$ in this case. Suppose now that $k(v) \leq |C(v)|$. First, we show the inequality "$\leq$". It follows from the definitions that $\gamma(v) \leq \gamma^+(v)$. Let (I,J) be an optimal solution for $CCP(A,k(v))\}$. For all $j \in C(v)$ define

$$D_j = \begin{cases} \text{a minimum VDS for } (T_j, k|_{T_j}) \text{ that contains } j, & \text{if } j \in I\,; \\ \text{a minimum VDS for } (T_j, k|_{T_j}), & \text{otherwise.} \end{cases}$$

Then, $j \in D_j$ for all $j \in I$. Therefore, since $|I| = k(v)$, the set $D := \bigcup_{j \in C(v)} D_j$ is a vector dominating set for $(T_v, k|_{T_v})$. Consequently, $\gamma(v) \leq CCP(A,k(v))$.

To see the converse inequality, suppose that $\gamma(v) < \gamma^+(v)$ (otherwise, we are done). For every minimum vector dominating set D for $(T_v, k|_{T_v})$ it holds that $v \notin D$ and also $|D \cap C(v)| \geq k(v)$. Hence, it is enough to show that

$$\gamma(v) \geq \min_{\substack{I \subseteq C(v) \\ |I| = k(v)}} \left(\sum_{i \in I} \gamma^+(i) + \sum_{i \in C(v) \setminus I} \gamma(i) \right). \tag{5}$$

Let D be a minimum vector dominating set for $(T_v, k|_{T_v})$ and let $I \subseteq D \cap C(v)$ such that $|I| = k(v)$. Then, for all $i \in I$, the set $D \cap V(T_i)$ is a minimum-sized vector dominating set for $(T_i, k|_{T_i})$ that contains i (otherwise, a smaller such set, say D'_i, could be used to produce a vector dominating set for $(T_v, k|_{T_v})$ smaller than D – namely $(D \setminus V(T_i)) \cup D'_i$). Therefore $|D \cap V(T_i)| = \gamma^+(i)$. Similarly, $|D \cap V(T_i)| = \gamma(i)$ for all $i \in C(v) \setminus I$. Summing up over all i, we get $\sum_{i \in I} \gamma^+(i) + \sum_{i \in C(v) \setminus I} \gamma(i) = \sum_{i \in C(v)} |D \cap V(T_i)| = |D| = \gamma(v)$. Inequality (5) follows.

The proof of (iii) is similar to that of (ii). □

The correctness of the procedure follows from Proposition 1. To analyze the time complexity, observe that at each leaf, a constant amount of computation is performed. The total time spent at an internal node v is proportional to $O(|C(v)|(k(v)+1)) = O(d(v)^2)$. Altogether, this results in the time complexity of $O(n^2)$. An optimal solution can be computed with a standard backtracking procedure, via a top-down traversal of the tree. □

5 Concluding Remarks

We have studied some algorithmic issues related to natural extensions of the well known concepts of domination and total domination in graphs. We have shown that the problems are approximable to within a logarithmic factor, and proved that this is essentially best possible. We have also provided an exact polynomial time algorithm for the vector domination problem in trees. (Due to lack of space, we will present our exact polynomial time algorithms for P_4-free graphs and threshold graphs in the journal version of the paper; see also [2].)

We leave it as a question for future research to determine the complexity status of the (total) vector domination problem for graphs of bounded tree-width or bounded clique-width.

References

1. Chlebík, M., Chlebíkova, J.: Approximation hardness of dominating set problems in bounded degree graphs. Information and Computation 206, 1264–1275 (2008)
2. Cicalese, F., Milanič, M., Vaccaro, U.: Hardness, approximability, and exact algorithms for vector domination and total vector domination in graphs. arXiv:1012.1529v1 [cs.DM] , http://arxiv.org/abs/1012.1529
3. Feige, U.: A threshold of $\ln n$ for approximating set cover. Journal of ACM 45, 634–652 (1998)
4. 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)
5. Klasing, R., Laforest, C.: Hardness results and approximation algorithms of k-tuple domination in graphs. Inform. Process. Lett. 89(2), 75–83 (2004)
6. Liao, C.S., Chang, G.J.: k-tuple domination in graphs. Inform. Process. Lett. 87(1), 45–50 (2003)
7. Harant, J., Prochnewski, A., Voigt, M.: On dominating sets and independent sets of graphs. Combinatorics, Probability and Computing 8, 547–553 (1999)

8. Haynes, T.W., Hedetniemi, S., Slater, P.: Fundamentals of Domination in Graphs. Marcel Dekker, New York (1998)
9. Haynes, T.W., Hedetniemi, S., Slater, P.: Domination in Graphs: Advanced Topics. Marcel Dekker, New York (1998)
10. Henning, M.A., Kazemi, A.P.: *k*-tuple total domination in graphs. Discrete Appl. Math. 158, 1006–1011 (2010)
11. Mishra, S., Radhakrishnan, J., Sivasubramanian, S.: On the hardness of approximating minimum monopoly problems. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 277–288. Springer, Heidelberg (2002)
12. Mishra, S., Rao, S.B.: Minimum monopolies in regular and trees graphs. Discrete Mathematics 306, 1586–1594 (2006)
13. Mossel, E., Roch, S.: On the submodularity of influence in social networks. In: Proc. 39th Ann. ACM Symp. on Theory of Comp., pp. 128–134. ACM, New York (2007)
14. Naor, M., Roth, P.: Optimal file sharing in distributed networks. SIAM J. on Computing 24, 158–183 (1995)
15. Peleg, D.: Local majorities, coalitions and monopolies in graphs: a review. Theoretical Computer Science 282, 231–257 (2002)
16. Uehara, R.: Linear time algorithms on chordal bipartite and strongly chordal graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 993–1004. Springer, Heidelberg (2002)
17. Wang, F., Du, H., Camacho, E., Xu, K., Lee, W., Shi, Y., Shan, S.: On positive influence dominating sets in social networks. Theoretical Computer Science 412, 265–269 (2011)
18. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica 2, 385–393 (1982)
19. Zou, F., Willson, J.K., Zhang, Z., Wu, W.: Fast information propagation in social networks. Discrete Mathematics, Algorithms and Applications 2, 1–17 (2010)

Enumeration of Minimal Dominating Sets and Variants

Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine

Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, France
`{mamadou.kante,limouzy,mary,nourine}@isima.fr`

Abstract. In this paper, we are interested in the enumeration of minimal dominating sets in graphs. A polynomial delay algorithm with polynomial space in split graphs is presented. We then introduce a notion of maximal extension (a set of edges added to the graph) that keeps invariant the set of minimal dominating sets, and show that graphs with extensions as split graphs are exactly the ones having chordal graphs as extensions. We finish by relating the enumeration of some variants of dominating sets to the enumeration of minimal transversals in hypergraphs.

1 Introduction

In many areas such as data mining, data bases, biology, social networks, etc., people are interested in enumerating a list of objects satisfying some properties [2,23]. For instance, in social networks, for marketing purposes, it can be useful to be able to enumerate the maximal communities, which corresponds in graph theory in enumerating maximal cliques. Classically, an algorithm which scans all possible solutions and outputs the desired solutions can be used. However, such a scenario cannot be used since in many cases, the size of outputs can be much smaller than the number of possible solutions. On the other side, since the size of the output can be huge compared to the size of the input, to measure the efficiency of an enumeration algorithm, the size of the input is not relevant for time complexity, contrary to classical decision problems. A natural parameter for measuring the time complexity of an enumeration algorithm is the number of outputs. Therefore, we will say that an enumeration algorithm runs in *output-polynomial* time if its running time is bounded by a polynomial depending on the number of outputs and the size of the input.

Minimum dominating set problem is a classic graph optimization NP-complete problem. However, contrary to other classic NP-complete graph optimization problems where there exist output-polynomial time algorithm for enumerating maximal (or minimal) solutions, *e.g.* maximal cliques or maximal independent sets [18,19], there is no known output-polynomial time algorithm that enumerates the set of minimal dominating sets of a graph. This paper is motivated by the quest for an output-polynomial time algorithm for the enumeration of minimal dominating sets of graphs (Dom for short). The dominating set problem is related to the well-known transversal problem in hypergraphs. Indeed, the set of minimal dominating sets of a graph is in bijection with the set of minimal transversals of its closed neighbourhood hypergraph [5]. It has been shown in [16] that the enumeration of minimal transversals in hypergraphs (Trans-Hyp for short) can be polynomially reduced to Dom. Trans-Hyp has been intensively studied in the last two decades due to its connection to several problems and particularly problems in data mining where frequently occurring patterns are of interest [3]. However, the question whether Trans-Hyp admits an output-polynomial time algorithm is still open. In fact, despite the number of papers on Trans-Hyp (see for instance these papers [3,13,14] and their bibliography section), the best known algorithm for Trans-Hyp is the one by Fredman and Khachiyan [17] which runs in time $O(n^{\log(n)})$ where n is the size of the hypergraph plus the number of minimal transversals.

Despite the link between Dom and Trans-Hyp and the importance of Dom in other areas such as building protocols in networks [24], to our knowledge, the only paper dealing with Dom is the one by Fomin et al. [15]. This paper, based on the Measure and Conquer technique from exact algorithms, gives an algorithm for Dom. However, its running time is $O(1.7159^n)$, where n is the number of vertices of the input graph. Hence, this algorithm is not an output-polynomial time one for Dom. It just informs us that the number of minimal dominating sets in a graph is upper bounded by $O(1.7159^n)$. Moreover, the algorithm does not use the fact that we deal with graphs, and uses instead the closed neighbourhood hypergraph. In this paper, we tackle Dom by restricting ourselves to some classes of graphs, as in the case of many output-polynomial time algorithms for Trans-Hyp.

Our contribution. After some preliminaries in Section 2, we recall some known output-polynomial time algorithms for Dom in Section 3. These results are of two types: those that come from meta-theorems in parameterized complexity theory and those that can be obtained from tractable cases of Trans-Hyp. In Section 4 we consider Dom in split graphs [5]. Split graphs are interesting for several reasons. In particular, it is a non trivial sub-class of chordal graphs where no output-polynomial time algorithm for Dom is known, and two important variants of Dom, namely total dominating sets and connected dominating sets coincide in split graphs. Section 5 is devoted to the extension of the result in Section 4 to other classes of graphs. For that we introduce a notion of maximal extension (a set of edges added to the graph) that keeps invariant the set of minimal dominating sets. We show that graphs that have chordal graphs as maximal extensions are exactly the one having split graphs as maximal extensions and derive a polynomial delay algorithm for chordal P_6-free graphs. We discuss in Section 6 our second goal consisting in studying the relationship between Trans-Hyp and Dom. We first show that the enumeration of the minimal total dominating sets is equivalent to Trans-Hyp and the enumeration of connected dominating sets is Trans-Hyp-Hard. Both are Trans-Hyp-complete when restricted to split graphs. Then, we show that the decision prolem associated to the enumeration of minimal dominating sets containing a set is co-NP-complete.

2 Preliminaries

If A and B are two sets, $A \backslash B$ denotes the set $\{x \in A \mid x \notin B\}$. The power-set of a set V is denoted by 2^V. The set of natural integers is denoted by $\mathbb{N}$. The size of a set A is denoted $|A|$.

We refer to [10] for our graph terminology. A graph G is a pair $(V(G), E(G))$, where $V(G)$ is the set of vertices and $E(G) \subseteq V(G) \times V(G)$ is the set of edges. A graph G is said to be *undirected* if $(x, y) \in E(G)$ implies $(y, x) \in E(G)$, hence we can write xy (equivalently yx). In this paper graphs are simple, loop-free and undirected. We let $G[X]$, called the sub-graph of G induced by $X \subseteq V_G$, the graph $(X, E(G) \cap (X \times X))$. A graph G is said *chordal* if it has no induced cycle of length greater than or equal to 4.

For a graph G, we let $N_G(x)$ be the set of neighbours of x, *i.e.* the set $\{y \mid xy \in E(G)\}$, and we let $N_G[x]$ be $N_G(x) \cup \{x\}$. For $X \subseteq V(G)$, we write $N_G[X]$ and $N_G(X)$ for respectively $\bigcup\limits_{x \in X} N_G[x]$ and $N_G[X] \setminus X$.

A *dominating set* in a graph G is a set of vertices D such that every vertex of G is either in D or is adjacent to some vertex of D. It is said *minimal* if for any $x \in D$, $D \setminus \{x\}$ is not a dominating set. The set of all minimal dominating sets of G will be denoted by $\mathcal{D}(G)$. Let D be a dominating set of G and $x \in D$. We say that x has a *private neighbour* y if $y \in N_G[x] \setminus N_G[D \setminus x]$. The set of private neighbours of x in D is denoted $P_D(x)$. The following is straightforward.

Lemma 1. *Let D be a minimal dominating set of a graph G. Then for all $x \in D$ we have $P_D(x) \neq \emptyset$.*

A *hypergraph* $\mathcal{H}$ is a pair $(V(\mathcal{H}), \mathcal{E}(\mathcal{H}))$ where $V(\mathcal{H})$ is a finite set and $\mathcal{E}(H) \subseteq 2^{V(\mathcal{H})}$. It is worth noticing that graphs are special cases of hypergraphs. By abuse of notations, we will call the elements of $\mathcal{E}(\mathcal{H})$ edges. A *transversal* (or *hitting set*) of $\mathcal{H}$ is a set $A \subseteq V$ that meets every edge of $\mathcal{E}(\mathcal{H})$. A transversal is *minimal* if it does not contain any other transversal as a subset. The set of all minimal transversals of $\mathcal{H}$ is denoted $Tr(\mathcal{H})$. The size of a hypergraph $\mathcal{H}$, denoted $||\mathcal{H}||$, is $|V(\mathcal{H})| + \sum\limits_{e \in \mathcal{E}(H)} |e|$.

Let $f : \mathbb{N} \to \mathbb{N}$. For a hypergraph $\mathcal{H}$ and $\mathscr{C} \subseteq 2^{V(\mathcal{H})}$, we say that an algorithm enumerates $\mathscr{C}$ with delay $f(||\mathcal{H}||)$ if, after a pre-processing that takes time $p(||\mathcal{H}||)$ for some polynomial p, it outputs the elements of $\mathscr{C}$ without repetitions, the delay between two outputs being bounded by $f(||\mathcal{H}||)$. If f is a polynomial, we call it a polynomial delay algorithm. We denote by Trans-Hyp, the enumeration problem of minimal transversals in hypergraphs. Similarly we denote by Dom, the enumeration problem of minimal dominating sets in graphs.

It is well-known that Dom can be polynomially reduced to Trans-Hyp as follows. For a graph G, we let $\mathcal{N}(G)$, the *closed neighbourhood hypergraph*, be $(V(G), \{N_G[x] \mid x \in V(G)\})$.

Lemma 2. [5] *Let G be a graph and $D \subseteq V(G)$. Then D is a dominating set of G if and only if D is a transversal of $\mathcal{N}(G)$.*

Let us finish these preliminaries by some constructions of graphs from hypergraphs. If $\mathcal{H}$ is a hypergraph, we let $\mathcal{I}(\mathcal{H})$, the *bipartite incidence* graph of $\mathcal{H}$, be the graph with vertex-set $V(\mathcal{H}) \cup \{y_e \mid e \in \mathcal{E}(\mathcal{H})\}$ and edge-set $\{xy_e \mid x \in V(G)$ and $x \in e\}$. $\mathcal{I}'(\mathcal{H})$, the *split incidence* graph of $\mathcal{H}$, is the graph obtained from $\mathcal{I}(\mathcal{H})$ by replacing $\mathcal{I}(\mathcal{H})[V(\mathcal{H})]$ by a clique on $V(\mathcal{H})$. Note that the neighbourhood of the vertex y_e in $\mathcal{I}(\mathcal{H})$ is exactly the set e. See Figure 1 for an example of $\mathcal{I}(\mathcal{H})$ and $\mathcal{I}'(\mathcal{H})$.

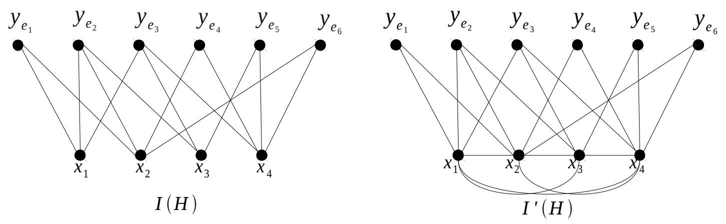


Fig. 1. An example of the bipartite incidence graph $\mathcal{I}(\mathcal{H})$ and the split incidence graph $\mathcal{I}'(\mathcal{H})$ of the hypergraph $\mathcal{H} = (\{x_1, x_2, x_3, x_4\}, \{e_1, e_2, e_3, e_4, e_5, e_6\})$ where $e_1 = \{x_1, x_2\}$, $e_2 = \{x_1, x_2, x_3\}$, $e_3 = \{x_1, x_3, x_4\}$, $e_4 = \{x_2, x_4\}$, $e_5 = \{x_3, x_4\}$, $e_6 = \{x_2, x_4\}$

3 Examples of Tractable Cases

Even if Dom is not a well-studied problem, there are some known tractable cases. We review some of them.

Let us first start with the well-known graph classes in parameterized complexity theory. *Tree-width* [20] and *clique-width* [9] are well-known complexity measures in graph theory. The first is well-known thanks to its important role in the proof of the *Graph Minor Theorem* [21] and the notorious Courcelle's meta-theorem [6]. Courcelle's meta-theorem says that every decision and optimization graph problem expressible in *monadic second-order logic* can be solved in polynomial-time in graph classes of bounded tree-width. Most of the well-known NP-complete problems are expressible in monadic second-order logic, *e.g.* 3-colorability, computing the minimum dominating set, the minimum vertex-cover, etc. Deciding if a set is a minimal dominating set is also expressible in monadic second-order logic. Clique-width is important because it not only generalizes tree-width, but it also yields a meta-theorem (see [8]) similar to the one for tree-width. Clique-width generalizes tree-width in the sense that every class of graphs of bounded tree-width has bounded clique-width, but the converse is false, complete graphs have clique-width 2 and unbounded tree-width. A natural question is whether these meta-theorems [6,8] can be extended to counting and enumeration problems. In fact, the results in [6,8] are also stated for counting problems. And, in [7], Courcelle extends them to enumeration problems. He proved that if $P(x_1, \ldots, x_m, X_1, \ldots, X_q)$ is a graph property, depending on vertices $x_1, \ldots, x_m$ and sets of vertices $X_1, \ldots, X_q$, expressible in monadic second-order logic and $\mathcal{C}$

is a class of graphs of bounded tree/clique-width then there exists an algorithm that for every graph G in $\mathcal{C}$ enumerates the set $\{(u_1, \ldots, u_m, Z_1, \ldots, Z_q) \mid G$ satisfies $P(u_1, \ldots, u_m, Z_1, \ldots, Z_q)\}$ with linear delay and uses linear space. The proof is rather involved and uses, as the other meta-theorems, machinery from logical tools. However, even if this "enumeration" meta-theorem is interesting and general, there are many natural graph classes that do not have bounded tree/clique-width, *e.g.* interval graphs, split graphs, planar graphs, etc. For some of them, one can prove that Dom is tractable by using a translation to tractable cases of Trans-Hyp. The following is an exemple.

Proposition 1. Dom *admits a polynomial delay algorithm when restricted to*

1. *Strongly chordal graphs.*
2. *Graph classes of bounded degeneracy.*

Proof. (1) If a graph G is strongly chordal, then $\mathcal{N}(G)$ is β-acyclic (see for instance [5]). By [11], Trans-Hyp admits a polynomial delay algorithm in β-acyclic hypergraphs.

(2) In [13], it is defined a notion of degeneracy for hypergraphs that extends the one on graphs. One easily verifies that if G is k-degenerate, then so is $\mathcal{N}(G)$. Since Trans-Hyp admits a polynomial delay algorithm in degenerate hypergraphs [13], we are done. □

Examples of strongly chordal graphs are directed path graphs (which include interval graphs), comparability chordal graphs, etc. Examples of degenerate graph classes are planar graphs, bounded-degree graphs, graphs of bounded genus, graphs of bounded tree-width, etc. Hence, the tractability of Dom in graph classes of tree-width k can be derived from Proposition 1(2). However, the use of Courcelle's result yields better bounds. By using [7], we have that Dom admits an algorithm with delay $f(k) \cdot n$ for some function $f : \mathbb{N} \to \mathbb{N}$, while [13] proves that Dom admits an algorithm with delay $n^{g(k)}$ for some function $g : \mathbb{N} \to \mathbb{N}$.

The notion of tree-width was extended to hypergraphs. However, there exist several notions (see for instance [1]). If we define the tree-width of a hypergraph $\mathcal{H}$ as the tree-width of its incidence graph $\mathcal{I}(\mathcal{H})$, then Trans-Hyp admits a polynomial delay algorithm in hypergraphs of bounded tree-width [13]. But, for the other notions, Trans-Hyp is hard even for classes of simple hypergraphs with *hypertree-width* 2 [12]. A natural question is whether there exists a class $\mathcal{C}$ of graphs with unbounded tree-width but such that $\mathcal{N}(\mathcal{C}) := \{\mathcal{N}(G) \mid G \in \mathcal{C}\}$ has bounded tree-width. The following proposition proves that it is not possible ($twd(G)$ denotes the tree-width of a graph G). Its proof (as most of the proofs in this extended abstract) is omitted because of space constraints.

Proposition 2. *For every graph G,*

$$\frac{twd(\mathcal{I}(\mathcal{N}(G))) - 1}{2} \leq twd(G) \leq twd(\mathcal{I}(\mathcal{N}(G))).$$

4 Dom in Split Graphs

We recall that a graph G is a *split* graph if its vertex-set can be partitioned into an independent set S and a clique C. (Here we consider S maximal.) Hence, we will denote a split graph G by the pair $(C(G) \cup S(G), E(G))$. We prove in this section that Dom in split graphs admits a linear delay algorithm that uses polynomial space. We first notice that a minimal dominating set D in a split graph G can be partitioned into a clique and an independent set, denoted respectively by $D_C = D \cap C(G)$ and $D_S = D \cap S(G)$. Lemma 3 shows that a minimal dominating set D is characterized by D_C. Note that D_S cannot characterize D_C, since several minimal dominating sets can have the same set D_S.

Lemma 3. *Let G be a split graph and D a minimal dominating set of G. Then $D_S = S \setminus N_G(D_C)$.*

Lemma 4. *Let G be a split graph and D be a minimal dominating set of G. Then for all $A \subseteq D_C$, the set $A \cup (S(G) \setminus N_G(A))$ is a minimal dominating set of G.*

Proof. Let D be a minimal dominating set of G and $A \subseteq D_C$. We show that $D' = A \cup (S(G) \setminus N_G(A))$ is a minimal dominating set. If $A = \emptyset$, then $D' = S(G)$ and since $S(G)$ is a maximal independent set, it is a minimal dominating set. Now suppose that $A \neq \emptyset$ and $x \in A$. Clearly $P_D(x) \neq \emptyset$ since D is minimal. This implies that $P_{D'}(x)$ is also not empty. Moreover, for any element $y \in D'_S$, we have $P_{D'}(y) = \{y\}$. We conclude that D' is a minimal dominating set. $\square$

A consequence of Lemmas 3 and 4 is the following.

Corollary 1. *Let G be a split graph. Then, there is a bijection between $\mathcal{D}(G)$ and the set $\mathcal{DI}(G) = \{D_C \mid D \in \mathcal{D}(G)\}$. The set $\mathcal{DI}(G)$ is moreover closed under inclusion (*i.e.*, is an independent system).*

So the generation of minimal dominating sets of a split graph is equivalent to the generation of elements in $\mathcal{DI}(G)$. In the following we give a linear delay algorithm to generate $\mathcal{DI}(G)$. Let $D, D' \in \mathcal{DI}(G)$. We say that D' covers D if $D \subseteq D'$ and $D' \setminus D$ is a singleton. We denote by $COV(D)$ the set $\{x \in C \setminus D \mid D \cup \{x\}$ covers $D\}$. If $D \in \mathcal{DI}(G)$ then y in $C \setminus D$ belongs to $COV(D)$ if each vertex in $D \cup \{y\}$ has a private neighbour. In order to enumerate $\mathcal{DI}(G)$, we call Dominant$(\emptyset, C(G))$.

Theorem 1. *Algorithm 1 generates the set $\mathcal{DI}(G)$ with $O(m + n)$ delay and uses space bounded by $O(n^2)$.*

5 Completion

In this section we introduce the notion of the maximal extension of a graph by keeping the set of minimal dominating sets invariant. The idea behind this operation is to maintain invariant the minimal edges, *w.r.t.* inclusion, in $\mathcal{N}(G)$.

Algorithm 1. Dominant(D,COV)

```
Data   : a split graph $G$
Result : $\mathcal{DI}(G)$
begin
    Output(D)
    foreach $x \in COV$ do
        COV=COV$\setminus\{x\}$
        NewCOV= $\emptyset$
        foreach  $y \in COV$ do
            if each vertex in $D \cup \{x, y\}$ has a private neighbour then
                NewCOV=NewCOV$\cup\{y\}$
        Dominant($D \cup \{x\}$,NewCOV)
end
```

Let G be a graph. A vertex $x \in V(G)$ is said to be *irredundant* if for all $y \neq x$, $N_G[y] \not\subseteq N_G[x]$, otherwise it is called *redundant*. In case of twins, we choose arbitrarily one to being irredundant, the others are so redundant. The set of irredundant (resp. redundant) vertices is denoted by $M(G)$ (resp. $RN(G)$). The *completion* graph of a graph G is the graph G_{co} with vertex set $V(G)$ and edge set $E(G) \cup \{xy \mid x, y \in RN(G),\ x \neq y\}$, i.e. G_{co} is obtained from G by replacing $G[RN(G)]$ by a clique on $RN(G)$. Note that the completion graph of a split graph G is G itself. However, the completion operation does not preserve the chordality of a graph. For instance, trees are chordal graphs but their completion graphs are not always chordal. Figure 2 gives some examples of completion graphs.

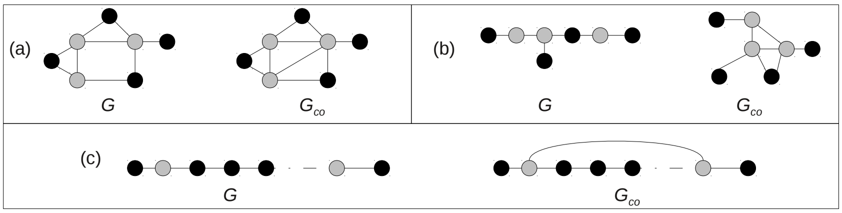


Fig. 2. (a) a non-chordal graph, its completion is a split graph (b) a chordal graph with an induced P_6, its completion is a split graph (c) a path P_n, its completion is not chordal

Lemma 5. *For any graph G, we have $\mathcal{D}(G) = \mathcal{D}(G_{co})$.*

In the following we are interested in graphs such that Dom in their completion graphs has an efficient generation algorithm. For two graphs G and H, we say that G is H-free if it does not contain H as an induced subgraph. For $k \geq 1$, we let P_k be the path on k vertices. A vertex is simplicial if the graph induced by its neighbourhood is a clique.

Lemma 6. *If G is a P_6-free chordal graph, then for all $x \in M(G)$, x is a simplicial vertex in G_{co}. Furthermore, the set $M(G)$ is an independent set in G_{co}.*

Proposition 3. *Let G be a P_6-free chordal graph. Then G_{co} is a split graph.*

Proof. From Lemma 6, it follows that $M(G)$ forms an independent set in G_{co}, and since $RN(G)$ forms a clique, we are done. □

The next theorem characterizes completion graphs that are split.

Theorem 2. *Let G be a graph. Then G_{co} is a chordal graph if and only if G_{co} is a split graph.*

6 Related Problems

In this section we discuss some variants of dominating sets and related problems. The enumeration of total dominating sets polynomially reduces to Trans-Hyp with respect to the classical Karp reduction. However, another natural variant, the enumeration of connected dominating sets is not known to have a reduction to Trans-Hyp, we show here that it is harder than Trans-Hyp. The problem in Section 6.3 is motivated by the investigation of an enumeration algorithm for dominating sets, which is inspired from an approach in [4].

6.1 Total Dominating Set

A total dominating set can be viewed as a dominating set in which the vertices do not cover themselves. A *total dominating set* of a graph G is a subset of vertices $D \subseteq V(G)$ such that for all $x \in V(G)$, $N_G(x) \cap D \neq \emptyset$; D is minimal if for all $x \in D$, $D \setminus \{x\}$ is not a total dominating set. We note $\mathcal{D}_t(G)$ the set of all minimal total dominating sets of G. For a graph G, we let $\mathcal{N}_o(G)$, the *open neighbourhood hypergraph* be $(V(G), \{N_G(x) \mid x \in V(G)\})$. We let Tds be the problem of listing $\mathcal{D}_t(G)$ for a graph G.

Proposition 4. Tds *is equivalent to* Trans-Hyp*.*

Proof. We first show that one can reduce Tds to Trans-Hyp on open neighbourhood hypergraph (the reduction was first noted in [22]). Let G be a graph. It is enough to show that $D \subseteq V(G)$ is a total dominating set in G if and only if it is a transversal of $\mathcal{N}_o(G)$. If D is a total dominating set of G, then for each $x \in V(G)$, $N_G(x) \cap D \neq \emptyset$. Therefore, D is a transversal of $\mathcal{N}_o(G)$. Conversely, if T is a transversal of $\mathcal{N}_o(G)$, then for each $x \in V(G)$, $T \cap N_G(x) \neq \emptyset$, *i.e.* T is a total dominating set of G.

We show now that Trans-Hyp can be reduced to Tds. Let $\mathcal{H}$ be a hypergraph. Assume furthermore that $\mathcal{H}$ has no dominating vertex, *i.e.*, a vertex belonging to all edges. Note that this case is not restrictive since if $x \in V(\mathcal{H})$ is a dominating vertex, then $Tr(\mathcal{H}) = \{\{x\}\} \cup Tr(\mathcal{H} \setminus \{x\})$ and consider so this reduced hypergraph. We then show that $\mathcal{D}_t(\mathcal{I}'(\mathcal{H})) = Tr(\mathcal{H})$.

(i) Let D be a minimal total dominating set of $\mathcal{I}'(\mathcal{H})$ and let $e \in \mathcal{E}(\mathcal{H})$. Then, there exists $x \in V(\mathcal{H}) \cap D$ such that $xy_e \in E(\mathcal{I}'(\mathcal{H}))$, *i.e.* $x \in e$. We now claim that $y_e \notin D$ for all $e \in \mathcal{E}(\mathcal{H})$. Otherwise, there exists $x \in e \cap D$ and since $\mathcal{I}'(\mathcal{H})[V(\mathcal{H})]$ is a clique, $D \setminus y_e$ is also a total dominating set $(D \cap V(\mathcal{H}) \geq 2)$, contradicting the minimality of D. Thus D is a transversal of $\mathcal{H}$.

(ii) Let T be a transversal of $\mathcal{H}$. Then, for all $e \in \mathcal{E}(\mathcal{H})$, $T \cap e \neq \emptyset$, *i.e.* for all $z \in V(\mathcal{I}'(\mathcal{H})) \setminus V(\mathcal{H})$ there exists $x \in T$ such that $xz \in E(\mathcal{I}'(\mathcal{H}))$. Since there is no dominating vertex, $|T| \geq 2$, and because $\mathcal{I}'(\mathcal{H})[V(\mathcal{H})]$ is a clique, for all $x \in V(\mathcal{H})$, there exists $y \in T$ such that $xy \in E(\mathcal{I}'(\mathcal{H}))$. Hence, T is a total dominating set of $\mathcal{I}'(\mathcal{H})$.

From (i) and (ii) we can conclude that $\mathcal{D}_t(\mathcal{I}'(\mathcal{H})) = Tr(\mathcal{H})$. □

Remark 1. The proof of Proposition 4 reveals that TRANS-HYP is reduced to TDS in split graphs. Hence, TDS in graphs is equivalent to TDS in split graphs.

6.2 Connected Dominating Set

A *connected dominating set* in a graph G is a subset D of $V(G)$ such that D is a dominating set of G and such that $G[D]$ is connected. A connected dominating set D is minimal, if for all $x \in D$, $D \setminus \{x\}$ is not a connected dominating set, in other words either $D \setminus \{x\}$ is not a dominating set or $G[D \setminus \{x\}]$ is not connected. We denote by $\mathcal{D}_c(G)$ the set of all minimal connected dominating sets of G. We let CDS the problem of generating $\mathcal{D}_c(G)$ for a graph G.

Proposition 5. CDS *in split graphs is equivalent to* TRANS-HYP.

Proof. Let $\mathcal{H}$ be a hypergraph. Then we show that $\mathcal{D}_c(\mathcal{I}'(\mathcal{H})) = Tr(\mathcal{H})$.

(i) Let $D \in \mathcal{D}_c(\mathcal{I}'(\mathcal{H}))$. Note firstly that $D \subseteq V(\mathcal{H})$. Indeed, suppose that there is $y_e \in D$ for some $e \in \mathcal{E}(\mathcal{H})$. Since, D must be connected, there is a neighbour z of y_e in D. Since $\{y_e \mid e \in \mathcal{E}(\mathcal{H})\}$ is an independent set, z must belong to $V(\mathcal{H})$. But since $\mathcal{I}'(\mathcal{H})[V(\mathcal{H})]$ forms a clique, $P_D(y_e) \subseteq P_D(z)$ and thus $D \setminus \{y_e\}$ is yet a connected dominating set, which contradicts the minimality of D. Now, for each $e \in \mathcal{E}(\mathcal{H})$, there exists $x \in D$ such that $xy_e \in E(\mathcal{I}'(\mathcal{H}))$, hence $D \cap e \neq \emptyset$. And so D is a transversal of $\mathcal{H}$.

(ii) Let T be a transversal of $\mathcal{H}$. Since $\mathcal{I}'(\mathcal{H})[V(\mathcal{H})]$ is a clique, T is connected and, for each $x \in V(\mathcal{H})$, there exists $y \in T$ such that $xy \in E(\mathcal{I}'(\mathcal{H}))$. Furthermore, for each $e \in \mathcal{E}(\mathcal{H})$, $T \cap e \neq \emptyset$, *i.e.* for each $y_e \in V(\mathcal{I}'(\mathcal{H})) \setminus V(\mathcal{H})$, there is $z \in T$ such that $zy_e \in E(\mathcal{I}'(\mathcal{H}))$. Hence, T is a connected dominating set of $\mathcal{I}'(\mathcal{H})$.

From (i) and (ii) we can conclude that $\mathcal{D}_c(\mathcal{I}'(\mathcal{H})) = Tr(\mathcal{H})$.

It remains to reduce CDS to TRANS-HYP. For a split graph G, we let $\mathcal{H}$ be the hypergraph $(C(G), \{N_G(x) \mid x \in S(G)\})$. It is easy to see that $G = \mathcal{I}'(\mathcal{H})$ and so from above, $\mathcal{D}_c(\mathcal{I}'(\mathcal{H})) = Tr(\mathcal{H})$. □

Remark 2. We do not currently know if CDS is equivalent to TRANS-HYP in all graphs. However, CDS in bipartite graphs is TRANS-HYP-Hard. Indeed consider, for a hypergraph $\mathcal{H}$, the graph B defined as follows: $V(B) = V(\mathcal{I}(\mathcal{H})) \cup \{x, y\}$ and $E(B) = E(\mathcal{I}(\mathcal{H})) \cup \{xy\} \cup \{xz \mid z \in V(\mathcal{H})\}$, then one can easily show that $Tr(\mathcal{H}) = \mathcal{D}_c(B)$.

We can remark that $\mathcal{D}_c(G)$ and $\mathcal{D}_t(G)$ are equal in split graphs, and they also coincide with another set $\mathcal{D}_{mc}(G) := \mathcal{D}_c(G) \cap \mathcal{D}(G)$ which is the minimal dominating sets that are connected. Note that $\mathcal{D}_{mc}(G)$ can be empty in general.

6.3 Dominating Sets Containing a Set

For a hypergraph $\mathcal{H}$ and a subset A of $V(\mathcal{H})$, we denote by $Tr(\mathcal{H}, A)$, the set of minimal transversals of $\mathcal{H}$ containing A. The problem consisting in asking whether $\mathcal{T} = Tr(\mathcal{G}, A)$, given $\mathcal{T} \subseteq 2^{V(\mathcal{H})}$, is denoted by Tcs.

Proposition 6. [4] Tcs *is co-NP-complete.*

For a graph G and a subset A of $V(G)$, we denote by $\mathcal{D}(G, A)$, the set of minimal dominating sets containing A. The problem consisting in asking whether $\mathcal{T} = \mathcal{D}(G, A)$, given $\mathcal{T} \subseteq 2^{V(G)}$, is denoted by Dcs.

Proposition 7. Dcs *is co-NP-complete.*

Proof. Dcs is in coNP. It suffices to guess a set of vertices and check in polynomial time if this set is a dominating set containing A and not in $\mathcal{T}$. So it is sufficient to show the reduction from Tcs to Dcs. Let $\mathcal{H}$ be a hypergraph and $A \subseteq V(\mathcal{H})$. We construct the graph B such that $V(B) = V(\mathcal{I}'(\mathcal{H})) \cup \{w, x, y, z\}$, and $E(B) = E(\mathcal{I}'(\mathcal{H})) \cup \{wx, xy, yz\} \cup \{\{wy_e\} \mid e \in \mathcal{E}(\mathcal{H})\}$. An example is given in Figure 3. We show that $Tr(\mathcal{H}, A)$ is in bijection with $\mathcal{D}(B, A \cup \{x, y\})$.

(i) Let $T \in Tr(\mathcal{H}, A)$ then we claim that $T' = T \cup \{x, y\} \in \mathcal{D}(B, A \cup \{x, y\})$. Indeed $A \cup \{x, y\} \subseteq T'$ and all vertices in $V(\mathcal{H})$ are covered because $\mathcal{I}'(\mathcal{H})[V(\mathcal{H})]$ forms a clique and $T \neq \emptyset$. Furthermore for all $e \in \mathcal{E}\mathcal{H}$, $e \cap T \neq \emptyset$, so $N_B[y_e] \cap T' \neq \emptyset$ and w and z are covered by T' because $\{x, y\} \subseteq T'$. We must also check that T' is minimal. Since $T \in Tr(\mathcal{H}, A)$, it is clear that we can not remove a vertex in T, and we can not remove neither x nor y otherwise, w or z would not be covered. So $T' \in \mathcal{D}(B, A \cup \{x, y\})$.

(ii) Let now $D \in \mathcal{D}(B, A \cup \{x, y\})$, we show that $D' = D \setminus \{x, y\} \in Tr(\mathcal{H}, A)$. We first claim that $D' \subseteq V(\mathcal{H})$. Actually, since D is a minimal dominating set, for all $z \in D$, $P_D(z) \neq \emptyset$. But $P_D(x) \subseteq \{w\}$ and so, if y_e, for some $e \in \mathcal{E}(\mathcal{H})$, belong to T, then $P_D(x)$ would be empty, which contradicts the minimality of D. Also, w and z cannot belong to D, otherwise $P_D(x)$ or $P_D(y)$ would be empty. Furthermore, since for all $e \in \mathcal{E}(\mathcal{H})$, y_e must be covered by some vertex in D and since $D \subseteq V(\mathcal{H})$, D' is a transversal of $\mathcal{H}$. Finally, by definition, $A \subseteq D'$, and then D' is a transversal of $\mathcal{H}$ containing A.

From (i) and (ii) we can conclude that $Tr(\mathcal{H}, A) = \{D \setminus \{x, y\} \mid D \in \mathcal{D}(B, A \cup \{x, y\})\}$ and hence Tcs is reduced to Dcs. □

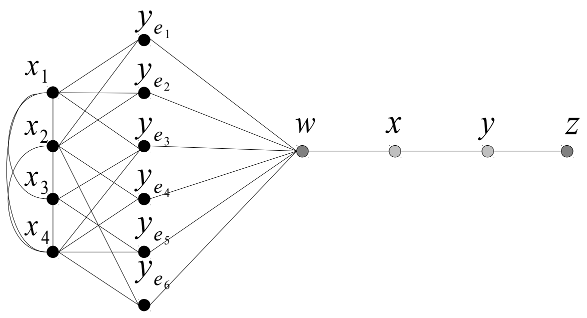


Fig. 3. An example of $B(\mathcal{H})$ where $\mathcal{H} = (\{x_1, x_2, x_3, x_4\}, \{e_1, e_2, e_3, e_4, e_5, e_6\})$ with $e_1 = \{x_1, x_2\}$, $e_2 = \{x_1, x_2\}$, $e_3 = \{x_1, x_3, x_4\}$, $e_4 = \{x_2, x_4\}$, $e_5 = \{x_3, x_4\}$, $e_6 = \{x_2, x_4\}$

Acknowledgments. We would like to thank the anonymous referees for their helpful suggestions and comments. A. Mary and L. Nourine are partially supported by the ANR (french National Research Agency) project DAG (ANR-09-DEFIS, 2009-2012). M.M. Kanté and V. Limouzy are supported by the ANR junior project DORSO (2011-2015).

References

1. Adler, I., Gottlob, G., Grohe, M.: Hypertree-Width and Related Hypergraph Invariants. European Journal of Combinatorics 28, 2167–2181 (2007)
2. Agrawal, R., Imielinski, T., Swami, A.N.: Database Mining: A Performance Perspective. IEEE Trans. Knowl. Data Eng. 5(6), 914–925 (1993)
3. Boros, E., Elbassioni, K., Khachiyan, L., Gurvich, V.: An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. Discrete Applied Mathematics 154(16), 2350–2372 (2006)
4. Boros, E., Gurvich, V., Hammer, P.L.: Dual subimplicants of positive Boolean functions. Optimization Methods and Software 10, 147–156 (1998)
5. Brandstädt, A., Bang Le, V., Spinrad, J.P.: Graph Classes a Survey. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia (1999)
6. Courcelle, B.: The Monadic Second-Order Logic of Graphs I: Recognizable Sets of Finite Graphs. Inf. Comput. 85(1), 12–75 (1990)
7. Courcelle, B.: Linear Delay Enumeration and Monadic Second-Order Logic. Discrete Applied Mathematics 157, 2675–2700 (2009)
8. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. Theory of Computing Systems 33(2), 125–150 (2000)
9. Courcelle, B., Olariu, S.: Upper Bounds to the Clique-Width of Graphs. Discrete Applied Mathematics 101(1-3), 77–114 (2000)
10. Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)
11. Eiter, T., Gottlob, G.: Identifying the Minimal Transversals of a Hypergraph and Related Problems. SIAM Journal on Computing 24(6), 1278–1304 (1995)

12. Eiter, T., Gottlob, G.: Hypergraph transversal computation and related problems in logic and AI. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 549–564. Springer, Heidelberg (2002)
13. Eiter, T., Gottlob, G., Makino, K.: New Results on Monotone Dualization and Generating Hypergraph Transversals. SIAM Journal on Computing 32(2), 514–537 (2003)
14. Eiter, T., Makino, K., Gottlob, G.: Computational aspects of monotone dualization: A brief survey. Discrete Applied Mathematics 156(11), 2035–2049 (2008)
15. Fomin, F.V., Grandoni, F., Pyatkin, A., Stepanov, A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Transactions on Algorithms 5(1) (2008)
16. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: On the Enumeration of Minimal Dominating Sets and Related Notions (2011) (manuscript)
17. Fredman, M., Khachiyan, L.: On the complexity of dualization of monotone disjunctive normal forms. Journal of Algorithms 21(3), 618–628 (1996)
18. Gély, A., Nourine, L., Sadi, B.: Enumeration aspects of maximal cliques and bicliques. Discrete Applied Mathematics 157(7), 1447–1459 (2009)
19. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Information Processing Letters 27, 119–123 (1988)
20. Robertson, N., Seymour, P.D.: Graph minors V:Excluding a Planar Graph. J. Comb. Theory, Ser. B 41, 92–114 (1986)
21. Robertson, N., Seymour, P.D.: Graph minors XX:Wagner's Conjecture. J. Comb. Theory, Ser. B 92(2), 325–357 (2004)
22. Thomassé, S., Yeo, A.: Total domination of graphs and small transversals of hypergraphs. Combinatorica 27(4), 473–487 (2007)
23. Wasserman, S., Faust, K.: Social Network Analysis. Cambridge University Press, Cambridge (1994)
24. Wu, J., Li, H.: A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks. Telecommunication Systems 18(1-3), 13–36 (2001)

Specification Patterns and Proofs for Recursion through the Store

Nathaniel Charlton and Bernhard Reus

University of Sussex, Brighton

Abstract. *Higher-order store* means that code can be stored on the mutable heap that programs manipulate, and is the basis of flexible software that can be changed or re-configured at runtime. Specifying such programs is challenging because of *recursion through the store*, where new (mutual) recursions between code are set up on the fly. This paper presents a series of formal specification patterns that capture increasingly complex uses of recursion through the store. To express the necessary specifications we extend the separation logic for higher-order store given by Schwinghammer *et al.* (CSL, 2009), adding parameter passing, and certain recursively defined families of assertions. Finally, we apply our specification patterns and rules to an example program that exploits many of the possibilities offered by higher-order store; this is the first larger case study conducted with logical techniques based on work by Schwinghammer *et al.* (CSL, 2009), and shows that they are practical.

1 Introduction and Motivation

Popular "classic" languages like ML, Java and C provide facilities for manipulating code stored on the heap at runtime. With ML one can store newly generated function values in heap cells; with Java one can load new classes at runtime and create objects of those classes on the heap. Even for C, where the code of the program is usually assumed to be immutable, programs can dynamically load and unload libraries at runtime, and use function pointers to invoke their code. Heaps that contain code in this way have been termed *higher-order store*.

This important language feature is the basis of flexible software systems that can be changed or re-configured at runtime. For example, the module mechanism of the Linux kernel allows one to load, unload and update code which extends the functionality of the kernel, without rebooting the system [9]. Examples of modules include drivers for hardware devices and filesystems, and executable interpreters that provide support for running new kinds of executables; by updating function pointers in the "syscall table", modules can at run time intercept any system call that the kernel provides. In [19,14] bugfixing and upgrading C programs without restarting them is discussed; for instance, a version of the OpenSSH server is built that can update itself while running when a new version becomes available, without disconnecting existing users.

Obtaining logics, and therefore verification methods, for such programs has been very challenging however, due to the complexity of higher-order heaps (see for example the discussion in [15]). Semantically, the denotation of such a heap is a mixed-variant recursively defined domain. The recursive nature of the heap complicates matters, because in addition to loading, updating and deallocating code, programs may "tie knots in the store" [13], i.e. create new recursions on the fly; this is known as *recursion through the store*. In fact, this knot-tying is happening whenever code on the heap is used in a recursive way, such as in the Visitor pattern [8] which involves a mutual recursion between the visitor's methods and the visited structure's methods.

To enable logical reasoning about software that uses higher-order store, the contributions of this paper are as follows.

- We present and classify patterns of formal specification for programs that recurse through the store, using recursive assertions and nested triples (Section 4) ([18] considered only a very simple form for specifications).
- We state a generic "master pattern" covering all the recursively defined specification patterns we identified in this paper, and argue that the fixed points needed to give semantics to such specifications always exist (Section 4).
- We apply the specification and proof techniques we developed to an example program which exploits many of the possibilities offered by higher-order store (Section 5). This is the first larger case study conducted with logical techniques based on [18], and shows that they are practical. Note that we use a slight extension of [18], adding call-by-value procedure parameters, inductively defined predicates and certain recursively defined families of assertions for lists. There is no space for giving proofs, but it should be pointed out that proofs have been done and we have developed a verification tool for support. We refer to this in the Conclusions (Section 6).

2 Our Running Example Program

We now present an example program which demonstrates in a simple way some of the possibilities offered by higher-order store and recursion through the store, of which it makes essential use. Our program performs evaluation of (binary) expressions represented as binary trees. A tree node is either an integer leaf or a labeled binary fork which identifies the expression operator. The distinction is effected by a label which is 0 for operators and 1 for leaves. For operations we will look at some typical examples like plus, but it is inherent to the approach that any (binary) operation can be interpreted. This flexibility is achieved by giving each tree node a pointer to the evaluation procedure to be used to evaluate it. The referenced evaluation procedure "implements" the meaning of the labeled node. This flexibility goes beyond the classic "visitor" pattern, which only works for a predefined class of node types.

Importantly the code implementing the various evaluations of different tree nodes is *not fixed* by the main program; instead, each operator evaluation procedure is implemented as a *loadable module*, which can be loaded on demand and a pointer to which can be saved in the tree nodes. This results in the data structures shown in Fig. 1.

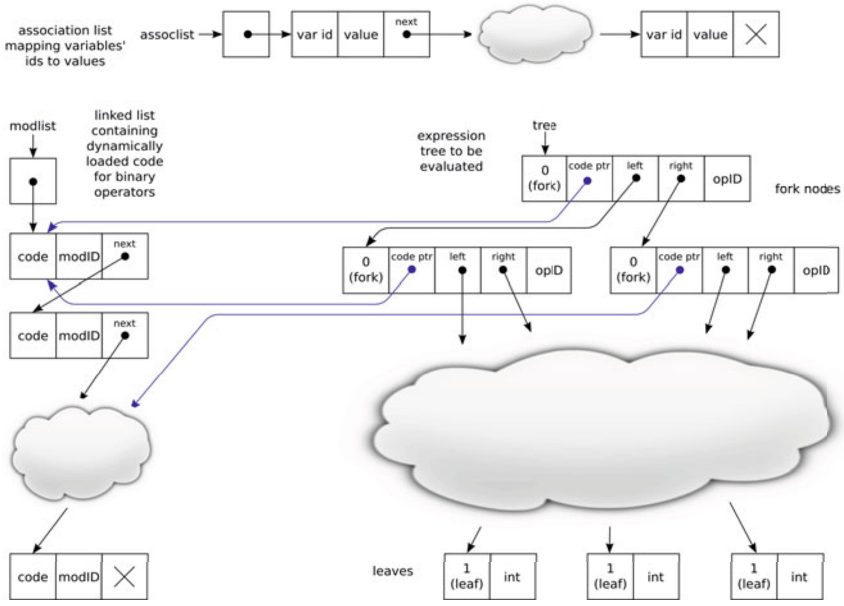

association list mapping variables' ids to values
assoclist
var id
value
next
modlist
linked list containing dynamically loaded code for binary operators
expression tree to be evaluated
tree
0 (fork)
code ptr
left
right
opID
fork nodes
code
modID
leaves
1 (leaf)
int

Fig. 1. The data structures used by our example program

The main program. Its code is given in Fig. 2. The eval $[e](\boldsymbol{e})$ statement invokes the code stored in the heap at address e, with a vector of (value) parameters $\boldsymbol{e}$. The shorthand res (explained later) simulates reference parameters. The expression '$\lambda \boldsymbol{x}.C$' denotes an *unevaluated* procedure with body C, taking formal parameters $\boldsymbol{x}$, as a value; thus $[e] :=$ '$\lambda \boldsymbol{x}.C$' is used to store procedures into the heap. As in ML, all variables in our language are immutable, so that once they are bound to a value, their values do not change. This property of the language lets us avoid side conditions on variables when studying frame rules. Our main program's code assumes the following to be in the heap:

1. The input: variable *tree* is a pointer to a binary tree as described above situated on the heap; *res* is a reference cell to store the result of the evaluation. In case the expression trees also encode variables, an association list mapping those variables to values is globally available at address *assoclist*.
2. Module-loading infrastructure: a linked list storing modules, pointed to by a globally known pointer *modlist*, and two procedures pointed to by *searchMods* and *loader*. Calling eval $[\mathit{searchMods}](\mathit{opID}, \text{res } \mathit{codeaddr})$ searches the list of loaded modules for the one implementing operator *opID*, returning its address or null (0) if it is not present. Calling eval $[\mathit{loader}](\mathit{opID}, \text{res } \mathit{codeaddr})$ always *guarantees* a module implementing operator *opID* is loaded, loading it if necessary, and returning its address.
3. A "tree visitor" procedure pointed to by *evalTree*, whose address is known to all modules and the main program. Note that this visitor does not contain

```
// constant offsets
const CodePtrO = 1
const LeftO = 2
const RightO = 3
const OpIDO = 4
const ValO = 1

[evalTree] :=
 ‘λ tree, resaddr.
   let kind = [tree] in
     if kind = 1 then
       let val = [tree + ValO] in  [resaddr] := val
     else
       let codePtr = [tree + CodePtrO] in
         eval [codePtr](tree, resaddr)
 ’ ;
eval [evalTree](tree, res res)
```

Fig. 2. Code for the “main program” part of our running example

the individual procedures for node types as in the standard pattern because we can directly store pointers to them within the nodes.

The main program first stores the procedure *evalTree* in the heap before calling it for the given input tree and result cell. For space reasons this code assumes that the tree and a suitable global *modlist* are already set up; we do not describe the initial construction of these data structures. We will, however, demonstrate that further loading can be done once the evaluator is already in action, namely from one of the procedures called by *evalTree*.

Some illustrative modules. Independently of the main program we can write the loadable modules starting with the basic ones for the evaluation of nodes that are labeled VAR, PLUS, TIMES etc. The VAR module evaluates its left subtree to an integer n, and then looks up the value of x_n, the variable with ID n, from the association list (the right subtree is ignored). Fig. 3 contains an implementation of PLUS. Note how this implementation calls back *evalTree* which in turn makes further calls to modules (either for PLUS again or for other operators): this is mutual recursion through the store.

As well as implementing arithmetic operators, the module mechanism can be used to extend the program in more dramatic ways. We can implement an operator ASSIGN, so that expression ASSIGN E_1 E_2 *updates* the association list, giving variable x_{E_1} the new value E_2, and returns, say, the variable's new value. Then we can turn our expression evaluator into an interpreter for a programming language; we can add modules implementing the usual control constructs such as sequential composition, alternation and repetition. Fig. 3 gives the implementation for WHILE. We emphasise that the WHILE operator can only be implemented because the code for each operator decides how often and when to evaluate the subexpressions; if the main program were in charge of the tree traversal (i.e. a tree fold was being used), WHILE could not be written.

We finish by examining some further modules (also in Fig. 3) which illustrate more complex uses of higher-order store. The LOAD_OVERWRITE procedure

```
PLUS: ‘$\lambda$ $tree, resaddr.$
  let $left$ = [$tree$ + LeftO] in
  let $right$ = [$tree$ + RightO] in
    eval [$evalTree$]($left$, res $leftVal$) ;
    eval [$evalTree$]($right$, res $rightVal$) ;
    [$resaddr$] := $leftVal + rightVal$’

WHILE: ‘$\lambda$ $tree, resaddr.$
  let $left$ = [$tree$ + LeftO] in
  let $right$ = [$tree$ + RightO] in
  let $b$ = new 0 in
    eval [$evalTree$]($left, b$) ;
    while [$b$] do
      (eval [$evalTree$]($right, resaddr$) ;
      eval [$evalTree$]($left, b$)) ;
    free $b$’

LOAD_OVERWRITE : ‘$\lambda$ $tree, resaddr.$
  let $opcode$ = [$tree$ + OpIDO] in
    eval [$loader$]($opcode$, res $procptraddr$) ;
    [$tree$ + CodePtrO] := $procptraddr$
    eval [$evalTree$]($tree, resaddr$)’

OSCILLATE : ‘$\lambda$ $tree, resaddr.$
  let $left$ = [$tree$ + LeftO] in
    eval [$evalTree$]($left, resaddr$) ;
    let $selfCodeptr$ = [$tree$ + CodePtrO] in
      let $oldCode$ = [$selfCodeptr$] in
      [$selfCodeptr$] :=
        ‘$\lambda$ $tree, resaddr.$
          let $right$ = [$tree$ + RightO] in
            eval [$evalTree$]($right, resaddr$) ;
            let $selfCodeptr$ = [$tree$ + CodePtrO] in
              [$selfCodeptr$] := $oldCode$ ’ ’
```

Fig. 3. Code for some modules demonstrating various uses of higher order store

first loads the code for the tree node's opID into the module list, then updates its own code pointer before calling that to evaluate the tree with the freshly loaded procedure. Note that next time the same code pointer in the tree is visited the newly loaded procedure is executed straight away and no more loading occurs. This update affects only the pointer in the tree data structure. The operator OSCILLATE chooses to evaluate the left subtree and returns its result. But it also updates itself with a version that, when evaluated, picks the right subtree for evaluation and then updates back to the original version. In this case the code in the module list itself is updated and thus *all* tree references pointing to it from the tree are affected by the update.

We have also considered *specialisation of code at runtime*. We wrote an implementation of the binomial coefficient $\binom{n}{k} := n!/k!(n-k)!$ which, when it detects that its left subtree (i.e. n) is an integer literal, calculates $n!$ (once) and generates on the fly an optimised implementation of $\binom{n}{k}$ that reuses the value.

3 The Programming and Assertion Languages

The programming language. We work with a simple imperative programming language extended with operations for stored procedures and heap manipulation, as described and used in Section 2. The syntax of the language is shown in Fig. 4, where $\boldsymbol{x}$ (resp. $\boldsymbol{e}$) represents a vector of distinct variables (resp.

$$e ::= \ 0 \mid 1 \mid -1 \mid \ldots \mid e_1 + e_2 \mid \ldots \mid x \mid \text{‘}\lambda \boldsymbol{x}.C\text{’} \qquad \Sigma ::= \ x \mid \{e\} \mid \emptyset \mid \Sigma_1 \cup \Sigma_2$$

$$\begin{aligned} C ::= \ & [e_1] := e_2 \mid \mathsf{let}\ y = [e]\ \mathsf{in}\ C \mid \mathsf{eval}\ [e](\boldsymbol{e}) \mid \mathsf{let}\ x = \mathsf{new}\ \boldsymbol{e}\ \mathsf{in}\ C \\ & \mid \mathsf{free}\ e \mid \mathsf{skip} \mid C_1; C_2 \mid \mathsf{if}\ e_1 = e_2\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \end{aligned}$$

$$\begin{aligned} P ::= \ & \text{True} \mid \text{False} \mid P_1 \vee P_2 \mid P_1 \wedge P_2 \mid P_1 \Rightarrow P_2 \mid \forall x.P \mid \exists x.P \mid e_1 = e_2 \mid e_1 \leq e_2 \\ & \mid e_1 \mapsto e_2 \mid \mathsf{emp} \mid P_1 \star P_2 \mid \text{lseg}(e_1, e_2, \Sigma) \mid \text{tree}(e_1, \Sigma) \mid \ldots \\ & \mid \{P\}\, e(\boldsymbol{e})\, \{Q\} \mid P \otimes Q \mid \Sigma_1 \subseteq \Sigma_2 \mid (\mu^k\ R_1(\boldsymbol{x}_1), \ldots, R_n(\boldsymbol{x}_n) \ .\ \mathscr{A}_1, \ldots, \mathscr{A}_n)(\boldsymbol{e}) \end{aligned}$$

Fig. 4. Syntax of expressions, commands and assertions

a vector of expressions). This language extends that in [18] by providing for the passing of value parameters. For convenience we employ two abbreviations: we allow ourselves a looping construct $\mathsf{while}\ [e]\ \mathsf{do}\ C$, which can be expressed with recursion through the store, and we write $\mathsf{eval}\ [e](\boldsymbol{e}, \mathsf{res}\ v)\ ;\ C$ as shorthand for $\mathsf{let}\ vaddr = \mathsf{new}\ 0\ \mathsf{in}\ (\mathsf{eval}\ [e](\boldsymbol{e}, vaddr)\ ;\ \mathsf{let}\ v = [vaddr]\ \mathsf{in}\ C\ ;\ \mathsf{free}\ vaddr)$.

The assertion language. The assertion language, shown in Fig. 4, follows [18], adding finite sets, more general recursive definitions and inductive predicates, and with some changes to accommodate parameter passing. The language is based on first-order intuitionistic logic augmented with the standard connectives of separation logic [17], and several further extensions:

Nested triples: Triples are assertions, so they can appear in pre- and post-conditions of triples. This *nested* use of triples is crucial because it allows one to specify stored code behaviourally, i.e. in terms of properties that it satisfies. The triple $\{P\}\, e(\boldsymbol{e})\, \{Q\}$ means that e denotes code satisfying $\{P\} \cdot \{Q\}$ when invoked with parameters $\boldsymbol{e}$. For code that does not expect any parameters, $\boldsymbol{e}$ will have length zero and we write simply $\{P\}\, e\, \{Q\}$.

Invariant extension: Intuitively invariant extension $P \otimes Q$ denotes a modification of P where all the pre- and post-conditions of triples inside P are $\star$-extended with Q. The operator $\otimes$ is from [4,18] and is not symmetric.

Inductively defined predicates: Predicates describing the linked lists and trees we use are available; their defining axioms are given in Fig. 5. The "..." in Fig. 4 indicates that any similar predicates can be added as required. $\text{lseg}(x, y, \Sigma)$ denotes a linked list segment from x to y, of nodes of three cells each, where Σ is the set of addresses of the nodes. $\text{lseg}{<}T(\cdot){>}(x, y, \Sigma)$ says additionally that the first value in each of the segment's nodes satisfies T, which is an assertion with an expression hole. $\text{tree}(t, \Sigma)$ denotes an expression tree rooted at t where the code pointers pointing *out of* the tree point to the set of addresses Σ.

(Mutually) Recursively defined assertions: Recursively defined assertions are the key to our work, because they let us reason naturally about challenging patterns of execution, such as self-updating code and recursion through the store. We use the notation $\mu^k\ R_1(\boldsymbol{x}_1), \ldots, R_n(\boldsymbol{x}_n)\ .\ P_1, \ldots, P_n$ to indicate that n predicates are defined mutually recursively with arguments $\boldsymbol{x}_i$ and bodies

$$\begin{aligned}
\mathrm{lseg}(x,y,\sigma) \quad \Leftrightarrow \quad & (x = y \wedge \sigma = \emptyset \wedge \mathsf{emp}) \\
\vee \; & (\exists nxt, \sigma' \,.\, x \mapsto \_, \_, nxt \star \mathrm{lseg}(nxt, y, \sigma') \wedge nxt \neq x \wedge \sigma = \sigma' \cup \{x\})
\end{aligned}$$

$$\mathrm{lseg}\,\langle T(\cdot)\rangle\,(x,y,\sigma) \quad \Leftrightarrow \quad \mathrm{lseg}(x,y,\sigma) \;\wedge\; \forall a.a \in \sigma \Rightarrow (a \mapsto T(\cdot) \star \mathrm{True})$$

$$\mathrm{tree}(t,\tau) \quad \Leftrightarrow \quad \mathrm{tree}_{\mathrm{fork}}(t,\tau) \;\vee\; (\exists n \,.\, t \mapsto 1, n \;\wedge\; \tau = \emptyset)$$

$$\begin{aligned}
\mathrm{tree}_{\mathrm{fork}}(t,\tau) \quad \Leftrightarrow \quad & \exists codePtr, left, right, opId, \tau', \tau''. \quad \tau = \{codePtr\} \cup \tau' \cup \tau'' \\
& \wedge\; t \mapsto 0, codePtr, left, right, opId \star \mathrm{tree}(left, \tau') \star \mathrm{tree}(right, \tau'')
\end{aligned}$$

Fig. 5. Inductively defined predicates used to specify and prove our example program

P_i, respectively, and the superscript k indicates that the k-th such predicate is selected. Note that (just to save space) we avoid introducing a syntactic category for predicates so $(\mu^k\ R_1(\boldsymbol{x}_1), \ldots, R_n(\boldsymbol{x}_n).P_1, \ldots, P_n)(\boldsymbol{e})$ is a formula but without arguments $\mu^k\ R_1(\boldsymbol{x}_1), \ldots, R_n(\boldsymbol{x}_n).P_1, \ldots, P_n$ is not a syntactically correct construct. Throughout the paper we will, however, for the sake of brevity, use abbreviations of the form $A := \mu^k\ R_1(\boldsymbol{x}_1), \ldots, R_n(\boldsymbol{x}_n).P_1, \ldots, P_n$ that are understood to be used with proper arguments in formulae. In Section 4 we will give a grammar for formulae that can be allowed in those recursive definitions since existence of fixed points is not automatic. We write $\mathscr{A}$ for an *allowed formula*, i.e. one that is of an appropriate form to ensure the existence of a solution; recursion variables R_i are only allowed to appear inside such an $\mathscr{A}$.

Each assertion describes a property of states, which consist of an (immutable) environment and a mutable heap. We use some abbreviations to improve readability: $e \in \Sigma := \{e\} \subseteq \Sigma$, $e \mapsto \_ := \exists x.\, e \mapsto x$ and $e \mapsto P[\cdot] := \exists x.\, e \mapsto x \wedge P[x]$ where $P[\cdot]$ is an assertion with an expression hole, such as $\{Q\} \cdot \{R\}$, $\cdot = e$ or $\cdot \leq e$. Additionally we have $e \mapsto e_0, \ldots, e_n := e \mapsto e_0 \star \cdots \star (e{+}n) \mapsto e_n$. The set of free variables of an expression or assertion is largely obvious, but note that $fv(\text{‘}\lambda\boldsymbol{x}.C\text{’}) := fv(C) - \boldsymbol{x}$.

4 Specification Patterns for Recursion through the Store

In this section we present a series of patterns, of increasing complexity, for specifying recursion through the store. By *pattern* we mean the shape of the specification, in particular the shape of the recursively defined assertion needed to deal with the recursion through the store.

Recursion via one or finitely many fixed pointers. The specification Φ_1 in Fig. 6 describes code that operates on a data structure D and calls itself recursively through a pointer g into the heap. Φ_2 (also in Fig. 6) describes two pieces of code on the heap that call themselves and each other recursively via two pointers g_1 and g_2. Similarly $\Phi_3, \Phi_4, \ldots$ can be formulated. (The $D, D_1, D_2, \ldots$ in Fig. 6 are metavariables, and in applications of the patterns they will be replaced by concrete formulae describing data structures.)

Fixed pointers: $\mathit{\Phi}_1 := \quad \mu^1 R \,.\, g \mapsto \forall \boldsymbol{x}.\{D_1 \star R\} \cdot (\boldsymbol{p})\{D_2 \star R\}$

$$\mathit{\Phi}_2 := \quad \mu^1 R \,.\, g_1 \mapsto \forall \boldsymbol{x}_1.\{D_1 \star R\} \cdot (\boldsymbol{p}_1)\{D_2 \star R\} \star g_2 \mapsto \forall \boldsymbol{x}_2.\{D_3 \star R\} \cdot (\boldsymbol{p}_2)\{D_4 \star R\}$$

With dynamic loader: $\mathit{\Phi}_{\text{loader}} := \quad R_{\text{loaded}}(g_1) \star R_{\text{loaded}}(g_2) \star \mathit{LoaderCode}$ where $R_{\text{loaded}} := \quad \mu^1\ R(c), \mathit{LoaderCode}\ .$
$c \mapsto \forall \boldsymbol{x}.\{D \star R(g_1) \star R(g_2) \star \mathit{LoaderCode}\} \cdot (\boldsymbol{p})\{D \star R(g_1) \star R(g_2) \star \mathit{LoaderCode}\}$,
$\mathit{loader} \mapsto \forall a, \mathit{ID}.\ \{a \mapsto \_\} \cdot (a, \mathit{ID})\ \{R(a)\}$

List of code: $\mathit{CLseg} := \quad \mu^1 R(x,y,\sigma)\ .\ \text{lseg}{<}T_0(\cdot){>}(x,y,\sigma)$ where $T_0(\cdot)$ is

$$\forall \boldsymbol{x}.\{\exists a,\sigma.\, D \star \mathit{codelist} \mapsto a \star R(a,\mathsf{null},\sigma)\} \cdot (\boldsymbol{p})\{\exists a,\sigma.\, D \star \mathit{codelist} \mapsto a \star R(a,\mathsf{null},\sigma)\}$$

Data structure with pointers: $\mathit{CLseg}' := \mu^1 R(x,y,\sigma).\ \text{lseg}{<}T_1(\cdot){>}(x,y,\sigma)$ where $T_1(\cdot)$ is

$$\forall \boldsymbol{x}\ .\ \left\{ \begin{array}{l} \exists a,\sigma,\tau.\ \tau \subseteq \sigma\ \wedge\ D(\tau) \\ \star\ \mathit{codelist} \mapsto a\ \star\ R(a,\mathsf{null},\sigma) \end{array} \right\} \cdot (\boldsymbol{p}) \left\{ \begin{array}{l} \exists a,\sigma,\tau.\ \tau \subseteq \sigma\ \wedge\ D(\tau) \\ \star\ \mathit{codelist} \mapsto a\ \star\ R(a,\mathsf{null},\sigma) \end{array} \right\}$$

The master pattern: $\mu^k\ R_1(\boldsymbol{x}_1), \ldots, R_n(\boldsymbol{x}_n)\ .\ \mathscr{A}_1, \ldots, \mathscr{A}_n$

where the following grammar shows what form each $\mathscr{A}_i$ can take (here $i_1, \ldots, i_m$ and $i'_1, \ldots, i'_{m'}$ are in $\{1, \ldots, n\}$, and P, P' are formulae not containing any of $R_1, \ldots, R_n$):

$$S ::= \ \forall \boldsymbol{x}_1.\ \{\exists \boldsymbol{x}_2\ .\ R_{i_1}(\boldsymbol{e}_1) \star \cdots \star R_{i_m}(\boldsymbol{e}_m) \star P\} \cdot (\boldsymbol{x}_3)\ \{\exists \boldsymbol{x}_4\ .\ R_{i'_1}(\boldsymbol{e}'_1) \star \cdots \star R_{i'_{m'}}(\boldsymbol{e}'_{m'}) \star P'\}$$

$$T ::= \quad t \mapsto S(\cdot) \mid t \mapsto x \wedge S(x) \qquad \mathscr{A}_i ::= \quad T_1 \star \cdots \star T_k \mid \text{lseg}{<}S{>}(\boldsymbol{x})$$

Fig. 6. Specification patterns for recursion through the store

Specification $\mathit{\Phi}_1$ allows the code pointed to by g to update itself (like our OSCILLATE example). Similarly, specification $\mathit{\Phi}_2$ allows pieces of code in g_1 and g_2 to update themselves and additionally to update each other. Such updates are permitted as long as the update is with code that behaves in the same way[1,2].

Note that although in this paper we will focus on proving memory safety, our patterns encompass full functional correctness specifications too. For instance, a factorial function that calls itself recursively through the store can be functionally specified using the following instance of the $\mathit{\Phi}_1$ pattern:

$$\mu^1 R\ .\ g \mapsto \forall x, n.\{x \mapsto n \star r \mapsto \_ \star R\} \cdot (x)\{x \mapsto 0 \star r \mapsto n! \star R\}$$

[1] Unlike the types used, say, in [2], our nested triples can handle updates that do not preserve code behaviour; examples in this paper do not use such updates, however.

[2] There are occasions when it is necessary to explicitly disallow update. This happens when one has a public and a (stronger) private specification for some code; allowing "external" updates to the code might preserve only the public specification.

Usage with a dynamic loader. As we pointed out, the preceding specifications permit in place update of code. This treats behaviour like that of our OSCILLATE module, which explicitly writes code onto the heap; but it does not account for behaviour like that of LOAD_OVERWRITE, where a loader function of the main program is invoked to load other code that is required. The specification Φ_{loader} in Fig. 6 describes a situation where two pieces of code are on the heap, calling themselves and each other recursively; but each may also call a loader procedure provided by the main program. Note the asymmetry in the specification of the loader, which could not be expressed using $\otimes$: R appears in the postcondition but nowhere in the precondition. Note also that we have omitted the analogous definition of *LoaderCode* (using μ^2) here for brevity.

Recursion via a list of code. The next step up in complexity is where a linked list is used to hold an arbitrary number of pieces of code. We suppose that each list node has three fields: the code, an ID number identifying the code, and a next pointer. The ID numbers allow the pieces of code to locate each other by searching through the list. We suppose that the cell at *codelist* contains a pointer to the start of the list. To reason about this setup, we use lseg$<T>$ (Fig. 5) to define recursively a predicate *CLseg* (Fig. 6) for segments of code lists. Note the existential quantifiers over a and σ in the auxiliary T_0: these mean that the pieces of code are free to extend or update the code list in any way they like, e.g. by updating themselves or adding or updating other code, as long as the new code also behaves again in the same way. One can constrain this behaviour by varying the specification in several ways; for instance, we can allow the pieces of code in the list to call each other but prohibit them from updating each other.

We point out the similarity between our idealised code lists and for example the *net_device* list that the Linux kernel uses to manage dynamically loaded and unloaded network device drivers [6].

Recursion via a set of pointers stored in a data structure. Instead of finding the right code to call explicitly within a list of type *CLseg* (using the ID numbers) the program might set up code pointers referencing code in such a list so that the pieces of code can invoke each other directly. We suppose that these direct code pointers live in the data structure D, writing $D(\tau)$ for a data structure whose code pointers collectively point to the set of addresses τ. The recursive specification we need is CLseg' (in Fig. 6); the constraint $\tau \subseteq \sigma$ says that all code pointers in D must point into the code list. Our example program combines this kind of recursion through the store with use of a loader function; we will see the specifications needed in Section 5 where the $D(\cdot)$ will be tree$(a, \cdot)$.

The master pattern. The last part of Fig. 6 presents a master pattern, which encompasses all the specification patterns seen so far. We have shown that recursive definitions of this form admit unique solutions; due to space constraints the proof of this, which uses the model and techniques of [18], is omitted. The $\otimes$ operator does not appear in the master pattern, but its effect can be expressed by unfolding its definition using the distributions axioms for $\otimes$ as found in [18].

$$\begin{aligned}
&\mathit{CLseg} \quad := \quad \mu^1\ \mathit{CLseg}(x,y,\sigma), \mathit{EvalCode}, \mathit{LoaderCode}, \mathit{SearchModsCode}\ .\\
&\quad \mathrm{lseg}{<}\ \mathit{Mod}\ {>}(x,y,\sigma),\\
&\quad \mathit{evalTree} \mapsto\ \mathit{EvalTriple}(\cdot)\ ,\\
&\quad \mathit{loader} \mapsto \forall \mathit{opID}, \mathit{codePtraddr}, \sigma_1\ .\\
&\quad\left\{\begin{array}{l} \exists a\ .\\ \quad \mathit{modlist} \mapsto a\\ \star\ \mathit{CLseg}(a, \mathsf{null}, \sigma_1)\\ \star\ \mathit{codePtraddr} \mapsto \_ \end{array}\right\} \cdot (\mathit{opID}, \mathit{codePtraddr}) \left\{\begin{array}{l} \exists a, r, \sigma_2.\quad \sigma_1 \cup \{r\} \subseteq \sigma_2 \wedge\\ \quad \mathit{modlist} \mapsto a\\ \star\ \mathit{CLseg}(a, \mathsf{null}, \sigma_2)\\ \star\ \mathit{codePtraddr} \mapsto r \end{array}\right\},\\
&\quad \mathit{searchMods} \mapsto \forall \mathit{opID}, \mathit{codePtraddr}, a, \sigma\ .\\
&\quad\left\{\begin{array}{l} \quad \mathit{modlist} \mapsto a\\ \star\ \mathit{CLseg}(a, \mathsf{null}, \sigma)\\ \star\ \mathit{codePtraddr} \mapsto \_ \end{array}\right\} \cdot (\mathit{opID}, \mathit{codePtraddr}) \left\{\begin{array}{l} \exists r.\quad (r \in \sigma \vee r = \mathsf{null}) \wedge\\ \quad \mathit{modlist} \mapsto a\\ \star\ \mathit{CLseg}(a, \mathsf{null}, \sigma)\\ \star\ \mathit{codePtraddr} \mapsto r \end{array}\right\}
\end{aligned}$$

where $\mathit{Mod}(F)$ abbreviates

$$\begin{aligned}
&\forall t, r, \tau_1, \sigma_1\ .\\
&\quad\left\{\begin{array}{l} \exists a.\quad \tau_1 \subseteq \sigma_1 \wedge\\ \quad \mathit{modlist} \mapsto a\\ \star\ \mathit{CLseg}(a, \mathsf{null}, \sigma_1)\\ \star\ \mathrm{tree}_{\mathrm{fork}}(t, \tau_1)\\ \star\ \mathit{EvalCode}\\ \star\ \mathit{LoaderCode}\\ \star\ \mathit{SearchModsCode}\\ \star\ r \mapsto \_ \end{array}\right\} F(t,r) \left\{\begin{array}{l} \exists a, \sigma_2, \tau_2.\quad \tau_2 \subseteq \sigma_2 \wedge \sigma_1 \subseteq \sigma_2 \wedge\\ \quad \mathit{modlist} \mapsto a\\ \star\ \mathit{CLseg}(a, \mathsf{null}, \sigma_2)\\ \star\ \mathrm{tree}(t, \tau_2)\\ \star\ \mathit{EvalCode}\\ \star\ \mathit{LoaderCode}\\ \star\ \mathit{SearchModsCode}\\ \star\ r \mapsto \_ \end{array}\right\}
\end{aligned}$$

and $\mathit{EvalTriple}(F)$ is the same, but with tree in place of $\mathrm{tree}_{\mathrm{fork}}$ in the precondition.

Fig. 7. Definitions of predicates used in the specification of our example program

5 Specifying and Proving Our Example Program

In this section we show how to specify memory safety of our example program; due to space restrictions we can only hint briefly at the proof. Our program uses three heap data structures (Fig. 1) as well as various procedures stored on the heap; Fig. 7 defines the predicates we use to describe these. All uses of μ fit the master pattern. By convention predicates named *Proc*Code assert that the variable *proc* points to some appropriate code stored in the heap. The abbreviation $\mathit{Mod}(F)$ used here says that code F behaves appropriately to be used as a module in our system. $\mathit{Mod}(F)$ is almost the same as the specification for *evalTree*; the difference is that modules may assume the tree is a fork, because *evalTree* has already checked for and handled the leaf case.

In the proof obligation of the main program (Fig. 2) we *assume* specifications for the procedures *loader* and *searchMods* (for which the code is not given):

$$\left\{\begin{array}{l} \mathit{modlist} \mapsto a \ \star \ \mathit{CLseg}(a, \mathsf{null}, \sigma) \star \ \mathrm{tree}(\mathit{tree}, \tau) \wedge \tau \subseteq \sigma \\ \star \ \mathit{LoaderCode} \star \ \mathit{SearchModsCode} \star \ \mathit{evalTree} \mapsto \_ \end{array}\right\} \ \text{MainProg} \ \{\text{True}\}$$

A more specific post-condition could be used to prove the absence of memory leaks etc., but True is sufficient for memory safety. For the proof we use some obvious adaptations of rules as presented in [18].

We remark that because the main program doesn't manipulate the association list at all, we can omit *AssocList* from our proof of the main program. Once our proof is complete, we can use the so-called *deep frame rule* ($\otimes$–Frame in [18]) to add *AssocList* everywhere it is needed. To prove memory safety of the modules we must show for each module implementation M that $\mathit{Mod}(M) \otimes \mathit{AssocList}$. For modules that do not directly access the association list (e.g. PLUS), we first prove $\mathit{Mod}(M)$ and then use $\otimes$–Frame to add *AssocList*. For modules that do access the association list, e.g. VAR and ASSIGN, one must prove $\mathit{Mod}(M) \otimes \mathit{AssocList}$ directly.

The proofs for the modules are very similar to that for the main program. One difference is that one must apply the $\star$-frame axiom to nested triples as well: e.g. in the module PLUS, the first eval works only on the left subtree, so we use $\star$-frame (see [18]) to add the root node and the right subtree as invariants to the triple for *evalTree*. Here we see the purpose of the constraint $\sigma_1 \subseteq \sigma_2$ in the postconditions of *Mod* and *EvalTriple*, which says that modules can update code in place, and add new code, but may not *delete* code from the code list.

6 Conclusions and Future Work

We extended the separation logic of [18] for higher-order store, adding several features needed to reason about larger, more realistic programs, including parameter passing and more general recursive specification patterns. We classified and discussed several such specification patterns, corresponding to increasingly complex uses of recursion through the store.

The work most closely related to ours (other than [18] on which we build) is that by Honda *et al.* [10], which also provides a Hoare logic with nested triples. It discusses the proof of a factorial function which calls itself through the store, but does not consider more complex patterns of recursion through the store. In [10] *content quantification* is used instead of separation logic, consequently ignoring frame rules, and total rather than partial correctness is treated.

We plan to extend our language and logic with (extensional) operations for generating code at runtime. We will also study the relationship between nested triples and the specifications based on *abstract predicate families* used in [16].

Finally, the complexity of the involved specifications and proofs demands tool support. We have developed a verification tool *Crowfoot* [1] supporting a logic and a language very similar to that used in this paper. We have used *Crowfoot*, for example, to verify correctness of runtime module updates [5], and to verify a version of the example presented in this paper. Use of $\star$-frame by *Crowfoot* is automatic, whereas the use of $\otimes$-frame is presently guided by user annotations. Our tool has been inspired by tools for separation logic like Smallfoot [3], jStar [7] and VeriFast [11] (a small survey of related work can be found in [12]).

Acknowledgements. We would like to thank J. Schwinghammer, H. Yang, F. Pottier, and L. Birkedal and for discussions on the usage of recursive families of specifications. Our research has been supported by EPSRC grant EP/G003173/1.

References

1. The Crowfoot website (includes a version of the example in this paper), `http://www.informatics.sussex.ac.uk/research/projects/PL4HOStore/crowfoot/`
2. Abadi, M., Cardelli, L.: A Theory of Objects. Springer-Verlag New York, Inc, Secaucus (1996)
3. Berdine, J., Calcagno, C., O'Hearn, P.W.: Smallfoot: Modular automatic assertion checking with separation logic. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 115–137. Springer, Heidelberg (2006)
4. Birkedal, L., Torp-Smith, N., Yang, H.: Semantics of separation-logic typing and higher-order frame rules for Algol-like languages. LMCS, vol. 2(5) (2006)
5. Charlton, N., Horsfall, B., Reus, B.: Formal reasoning about runtime code update. In: Abiteboul, S., Böhm, K., Koch, C., Tan, K.-L. (eds.) ICDE Workshops, pp. 134–138. IEEE, Los Alamitos (2011)
6. Corbet, J., Rubini, A., Kroah-Hartman, G.: Linux device drivers, 3rd edn. O'Reilly Media, Sebastopol (2005)
7. Distefano, D., Parkinson, M.J.: jStar: towards practical verification for Java. In: OOPSLA, pp. 213–226 (2008)
8. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
9. Henderson, B.: Linux loadable kernel module HOWTO, v1.09 (2006), `http://tldp.org/HOWTO/Module-HOWTO/`
10. Honda, K., Yoshida, N., Berger, M.: An observationally complete program logic for imperative higher-order functions. In: LICS, pp. 270–279 (2005)
11. Jacobs, B., Smans, J., Piessens, F.: A quick tour of the veriFast program verifier. In: Ueda, K. (ed.) APLAS 2010. LNCS, vol. 6461, pp. 304–311. Springer, Heidelberg (2010)
12. Krishnaswami, N.R., Aldrich, J., Birkedal, L., Svendsen, K., Buisse, A.: Design patterns in separation logic. In: TLDI, pp. 105–116 (2009)
13. Landin, P.J.: The mechanical evaluation of expressions. Computer Journal 6(4), 308–320 (1964)
14. Neamtiu, I., Hicks, M.W., Stoyle, G., Oriol, M.: Practical dynamic software updating for C. In: PLDI, pp. 72–83 (2006)
15. Ni, Z., Shao, Z.: Certified assembly programming with embedded code pointers. In: POPL, pp. 320–333 (2006)
16. Parkinson, M.J., Bierman, G.M.: Separation logic, abstraction and inheritance. In: POPL, pp. 75–86 (2008)
17. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, pp. 55–74 (2002)
18. Schwinghammer, J., Birkedal, L., Reus, B., Yang, H.: Nested hoare triples and frame rules for higher-order store. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 440–454. Springer, Heidelberg (2009)
19. Stoyle, G., Hicks, M., Bierman, G., Sewell, P., Neamtiu, I.: Mutatis mutandis: Safe and predictable dynamic software updating. ACM Trans. Program. Lang. Syst. 29(4) (2007)

Sub-computabilities[⋆]

Fabien Givors and Gregory Lafitte

Laboratoire d'Informatique Fondamentale de Marseille (LIF),
CNRS – Aix-Marseille Université,
39, rue F. Joliot-Curie, 13453 Marseille Cedex 13, France
{Fabien.Givors,Gregory.Lafitte}@lif.univ-mrs.fr

Abstract. Every recursively enumerable set of integers (r.e. set) is enumerable by a primitive recursive function. But if the enumeration is required to be one-one, only a proper subset of all r.e. sets qualify. Starting from a collection of total recursive functions containing the primitive recursive functions, we thus define a *sub-computability* as an enumeration of the r.e. sets that are themselves one-one enumerable by total functions of the given collection. Notions similar to the classical computability ones are introduced and variants of the classical theorems are shown. We also introduce sub-reducibilities and study the related completeness notions. One of the striking results is the existence of natural (recursive) sets which play the role of low (non-recursive) solutions to Post's problem for these sub-reducibilities. The similarity between sub-computabilities and (complete) computability is surprising, since there are so many missing r.e. sets in sub-computabilities. They can be seen as toy models of computability.

Introduction

Many proofs in (basic and also more involved) computability rely on the algebraic structure of the enumeration of r.e. sets, partial functions, *etc.*, and not really *per se* on the notion of "being computable". The structure is provided by the properties of an acceptable enumeration, and consequences of being acceptable, *e.g.*, Kleene's second recursion theorem. One motivation behind the work of this article is to develop results similar to the classical computability ones (from basic results to Turing completeness issues) but in a setting verifying only a proper subset of the elementary properties of classical computability. In our case, this translates as the quest of developing computabilities with most of the nooks and crannies of classical computability without being all of computability. Here, a computability is meant to be collections of sets and functions which portray what we consider in this setting to be the r.e. sets and partial computable functions. Sub-computabilities are computabilities where these collections are a proper subset of the classical ones and is the focus of this article. Higher

[⋆] The research presented in this paper has been made possible by the support of the French ANR grants *NAFIT* (ANR-08-DEFIS-008-01) and *EMC* (ANR-09-BLAN-0164-01).

computability could also be cast in this setting and it will be the subject of a subsequent article. Our setting can be seen as a toy model for computability, not based on models of machines, but on the algebraic structure of computability.

A sub-computability c is a pair $(\boldsymbol{\Phi}^{\mathsf{c}}_{\cdot}, \mathbf{W}^{\mathsf{c}}_{\cdot})$ of enumerations of a sufficiently closed collection $\mathfrak{F}_{\mathsf{c}}$ (called the foundation of c) of total recursive functions (called c-*fundamental*), and of a collection $\mathfrak{E}_{\mathsf{c}}$ (called the support of c) of r.e. sets (called c-*enumerable*), which are one-one enumerated[1] by functions in $\mathfrak{F}_{\mathsf{c}}$. These fundamental functions somehow measure the effectiveness of the constructions used in computability. A partial recursive function is said to be *somewhat* c-*computable* if its graph is c-enumerable. Building on an enumeration $\boldsymbol{\Phi}^{\mathsf{c}}_{\cdot}$, we can respectively build canonical enumerations $\mathbf{W}^{\mathsf{c}}_{\cdot}$ (resp. $\varphi^{\mathsf{c}}_{\cdot}$) of c-enumerable sets (resp. somewhat c-computable functions). Since the collection of c-enumerable sets (and somewhat c-computable functions) is completely and uniquely determined by the collection $\mathfrak{F}_{\mathsf{c}}$, we will identify c with the foundation ($\mathsf{c} = \mathfrak{F}_{\mathsf{c}}$). What could vary is the enumerations of $\mathfrak{F}_{\mathsf{c}}$ and $\mathfrak{E}_{\mathsf{c}}$, but we will see with the isomorphism theorem *à la* Rogers (Theorem 5) that there is no confusion here.

An example of a sub-computability is p, the r.e. sets one-one enumerated by primitive recursive functions. Koz'minyh [2] in 1972 proved a key lemma on p. We generalize this lemma (Lemma 1) to any sub-computability. It gives insights into the behavior of c-enumerable sets and somewhat c-computable functions. Its corollaries make classical constructions possible, even if we do not have the general μ recursion operator. The (primitive recursive) effectivity of these corollaries (and of most of our theorems) is especially useful.

Other sub-computability foundations (and thus sub-computabilities) stem from the field of *subrecursion*. It provides a great number of examples of closed collections of ever more increasing total functions which do not contain all of the total recursive functions. This study is strongly entangled with proof theory and provides a clear picture of "what is provable and what is not" in a theory for which one is able to construct an *ordinal analysis*. In particular, proof theory has identified for many theories T the set of total recursive functions whose totality is provable in T. This connection gives another motivation to our work.

Studying sub-computabilities amounts to classifying r.e. sets by measuring the difficulty of enumerating them by way of subrecursion. On that quest, one stumbles with the well-known at first surprising result that all r.e. sets are enumerable by primitive recursive functions. But if the enumeration is required to be one-one, the triviality evaporates: some r.e. sets are not one-one enumerable by primitive recursive functions, *e.g.*, the Ackermann function's range, but still many r.e. sets are primitively one-one enumerable, *e.g.*, the graph of the characteristic function of the range of the Ackermann function or the graph of a universal function.

If a set of integers is enumerable in a sub-computability, then it is recursively enumerable. If it is not enumerable, then it is either not recursively enumerable, or not feasibly enumerable in this sub-computability and necessitates more *power* to be seen as effectively enumerable. Thus, a sub-computability is an

[1] The complete definition (Def. 2) also incorporates finite sets and $\varnothing$.

approximation of the classical computability: we have the same kind of results than in computability but with variations on the notions used; Classical (complete) computability is the *union* of all sub-computabilities.

For example, Post's theorem —stating that a set is recursive if and only if it is both recursively enumerable and co-recursively enumerable— does not hold anymore, leading the way to more restrictive notions of "being recursive". Another example is Kleene's second recursion theorem, which only partially holds in various ways in our sub-computabilities (Theorem 3).

We also define reducibilities which are refinements of Turing (or stronger) reducibilities and yield a structure of the associated degrees that looks very similar to the classical structure of the Turing degrees. Two of the interesting aspects of this study are the simplicity of some of the proofs of the degree structure, and the existence of natural (recursive) sets which play the role of low (non-recursive) solutions to Post's problem for these sub-reducibilities. To have this setting as a *real* toy model for computability, it would be nice to be able to have ways of transferring those results back in classical computability.

Our study is different from other *subrecursive degree* theories, especially when considering the fact that our objects of study are not necessarily recursive. Nevertheless, one of our future goals is to build bridges with these theories, *e.g.*, *honest elementary degrees* (see [3,4,5]), to be able to use their techniques and have their applications, especially the minimal pairs result with a proof-theoretical twist in [6].

1 Sub-computability Basics

We consider a *sub-computability* $\mathbb{C}$ to be a pair $(\mathfrak{\Phi}^{\mathbb{C}}_{\cdot},\mathfrak{W}^{\mathbb{C}}_{\cdot})$ of enumerations of the *foundation* $\mathfrak{F}_{\mathbb{C}}$ and of the *support* $\mathfrak{E}_{\mathbb{C}}$ of the sub-computability. A support $\mathfrak{E}_{\mathbb{C}}$ is a collection of r.e. sets (the $\mathbb{C}$-*enumerable* sets). It is constituted by sets one-one enumerated by total recursive functions (the $\mathbb{C}$-*fundamental* functions) belonging to $\mathfrak{F}_{\mathbb{C}}$. Since the $\mathfrak{E}_{\mathbb{C}}$ is uniquely determined by this foundation, $\mathbb{C}$ will denote both the sub-computability and the foundation. It will always be easy to distinguish between those two uses of notation.

We start with the definition of a *sub-computability foundation.*

Definition 1. A *sub-computability foundation* $\mathbb{C}$ is a set of recursive total functions, called the *fundamental* functions, containing (at least) the primitive recursive initial functions, closed by composition and primitive recursion, along with a coding scheme providing fundamental-functions indices in $\mathbb{N}$ for the fundamental functions, such that we can primitively compute a recursive-functions index from any fundamental-functions index, and such that the characteristic function of the set $\{x : x \text{ is an index for fundamental } f\}$ is primitive recursive, with a primitive recursive padding function $\mathfrak{p}$ ($\mathfrak{p}^n(x)$ gives an index for the fundamental function for which x is an index, which is different from all n *previous* indices, x, $\mathfrak{p}(x)$, ..., $\mathfrak{p}^{n-1}(x)$), and with a primitive recursive composition function[2] $\mathfrak{c}$.

[2] $\mathfrak{c}(x,y)$ gives an index for the fundamental function which is the composition of the two fundamental functions for which x and y are respectively indices.

$\mathsf{c}[F]$ designates $\mathsf{c} \cup F$ (the foundation c to which the functions of F are added) closed under composition and primitive recursion. $\mathsf{c}[f]$, $\mathsf{c}[f, g]$, *etc.*, designate respectively $\mathsf{c}[\{f\}]$, $\mathsf{c}[\{f, g\}]$, *etc.*

$\boldsymbol{\Phi}^{\mathsf{c}}_e(x)$ designates the computation on x of the c-fundamental function whose index is e.

On top of foundations, we build *sub-computabilities.*

Definition 2. A set of integers X is a c-*enumerable* set if there is a function $f \in \mathsf{c}$ such that the maximal initial one-one part[3] of f enumerates X.

A *sub-computability* is a pair $(\boldsymbol{\Phi}^{\mathsf{c}}_{\cdot}, \mathbf{W}^{\mathsf{c}}_{\cdot})$ of enumerations of a foundation c and of a collection $\mathfrak{E}_{\mathsf{c}}$ of recursively enumerable sets, the c-enumerable sets. $\mathfrak{E}_{\mathsf{c}}$ is called the *support* of the sub-computability. c will also denote the sub-computability.

In the following, we will denote by p the collection of primitive recursive functions and the sub-computability based on this foundation.

A partial function is *somewhat* c-*computable* if its graph is c-enumerable.

A partial somewhat c-computable function f is said to converge on x if $f(x)$ is defined. It is denoted by $f(x) \downarrow$. Otherwise, it is said to diverge on x and is denoted by $f(x) \uparrow$.

e is a c-*index* for a c-enumerable set W if e is a fundamental index for a c-fundamental function that one-one maximally initially enumerates W. We denote W by $\mathbf{W}^{\mathsf{c}}_e$.

e is a c-*index* for a somewhat c-computable function f if e is a c-index for the graph of f. We denote $f(x)$ by $\varphi^{\mathsf{c}}_e(x)$.

Notice that the honest functions (see for example [3]) are somehow defined in a similar way, but with an elementary foundation (which does not make possible enumerations of non recursive graphs). Notice also that there are indices for the empty set (and thus the nowhere-defined function).

Without the c, $(W_e)_{e \in \mathbb{N}}$ denotes the classical enumeration of recursively enumerable sets, and $(\varphi)_{e \in \mathbb{N}}$ the classical enumeration of recursive functions.

Definition 3. A *universal* function U_{c} over a sub-computability support c is a partial function $(c, x) \mapsto \mathsf{U}_{\mathsf{c}}(c, x)$ such that for each somewhat c-computable function φ^{c}_e, for all x, if $\varphi^{\mathsf{c}}_e(x) \uparrow$, then $\mathsf{U}_{\mathsf{c}}(e, x) \uparrow$, else $\mathsf{U}_{\mathsf{c}}(e, x) \downarrow$ and $= \varphi^{\mathsf{c}}_e(x)$.

A c-*fundamentally universal* function $\mathsf{U}^{\boldsymbol{\Phi}}_{\mathsf{c}}$ is a function $(c, x) \mapsto \mathsf{U}^{\boldsymbol{\Phi}}_{\mathsf{c}}(c, x)$ such that for each c-fundamental function $\boldsymbol{\Phi}^{\mathsf{c}}_e$, for all x, $\boldsymbol{\Phi}^{\mathsf{c}}_e(x) = \mathsf{U}^{\boldsymbol{\Phi}}_{\mathsf{c}}(e, x)$.

As we will see later, a universal function over a sub-computability support c will *surprisingly* be somewhat c-computable. The notation U_{c} will at the same time designate the afore defined partial function U_{c} and its c-index.

Definition 2 involving discarding the last enumerated element of the finite set is a better choice than having the empty set represented artificially because we

[3] f is said to *one-one maximally initially enumerate* (in short, *one-one enumerate*) X if $X = \begin{cases} f``Y & \text{if } Y = \mathbb{N}, \\ f``Y \setminus \{f(\max(Y)\} & \text{otherwise, that is } Y \text{ is finite,} \end{cases}$ where Y is the greatest initial subset of $\mathbb{N}$ (Y is equal to $\mathbb{N}$ or to some $n \in \mathbb{N}$) such that $f \upharpoonright Y$ is one-one.

want the set of indices of nowhere-defined somewhat c-computable functions to be as less[4] c-*computable* as possible.

Our enumerability notion sticks well with the intuition adopted in classical computability: a set is c-enumerable if and only if it is the domain of a somewhat c-computable function. This property and Definition 2 call for the following result: a c-enumerable set W, for which x is a c-index, has infinitely many other indices computable from x in a p-fundamental computable way. To prove it, we naturally use the padding function $\mathfrak{p}$ for the indices of the c-fundamental functions. The s-m-n theorem also still holds for fundamental functions.

Universal functions are absent of the fundamental functions. Therefore, even if we have s-m-n, we cannot prove Kleene's second recursion theorem for fundamental functions. It will however partially hold for somewhat c-computable functions (Theorem 3), for which s-m-n does not hold anymore.

Koz'minyh [2] in 1972 proved the following lemma (for the p part), which constitutes a mile stone for understanding sub-computabilities. We call it the *heredity lemma.*

Lemma 1. *If W_e is an infinite c-enumerable set, and $W_i \supset W_e$ is recursively enumerable, then W_i is c-enumerable.*

The following useful corollaries come to mind when proving this lemma. First, there exists a primitive recursive function H_{c} such that $\mathbf{W}^{\mathsf{c}}_{\mathsf{H}_{\mathsf{c}}(\langle i,j\rangle)} = \mathbf{W}^{\mathsf{c}}_i \cup \mathbf{W}^{\mathsf{c}}_j$. Also, if X is a c-enumerable set and g a c-fundamental function, then the following function is somewhat c-computable: $f(\langle x,y\rangle) = g(\langle x,y\rangle)$ if $x \in X$, $\uparrow$ otherwise.

The first idea to build sub-computabilities is to define a set of total recursive functions, called *fundamental.* From these total functions, we can define a collection of sets, called *support*, that can each be enumerated in a non-repetitive way[5]. Finally, thinking of this support as a collection of graphs of partial functions gives rise to a notion of *somewhat* computable functions. We review these notions below.

c-*fundamental functions.* As they will be our raw material, it is important to see what are their capabilities, and limitations. In general, there will not be any universal function among the fundamental functions. However, thanks to the Normal Form Theorem, we know we can primitively simulate any finite number of step of any recursive function φ_e: there is a *primitive recursive*, function $\mathrm{sim}_{\mathsf{p}}$ such that $\mathrm{sim}_{\mathsf{p}}(e,i,s) = \langle e,i,\varphi_e(i)\rangle + 1$ for at most a unique s, and 0 otherwise.

c-*computable sets.* The recursiveness of a set in classical computability has many characterizations. These definitions are no longer equivalent in sub-computabilities.

[4] It is not χ-c-computable, but nonetheless c-enumerable and weakly-c-computable. (See Def. 4).

[5] As indicates the Normal Form theorem, every recursively enumerable set is enumerable by a primitive recursive function. For example, $f_e : \langle n,s\rangle \mapsto \varphi_e(n)$ if it converges in less than s steps, $\varphi_e(0)$ otherwise. However, such a function will not be one-one in general.

Definition 4. Let E be a set of integers, E is weakly c-computable if E and $\overline{E}$ are c-enumerable. E is somewhat c-computable if χ_E is somewhat c-computable. E is strongly c-enumerable if E is c-enumerable, enumerated by an increasing one-one c-fundamental function Φ_e^{c}. E is strongly c-computable if E and $\overline{E}$ are strongly c-enumerable and χ-c-computable if E has a c-fundamental characteristic function.

It is easy to see that each of these *sub-recursive* notions implies classical computability: for a set, being strongly c-enumerable, strongly c-computable, weakly c-computable, somewhat c-computable or χ-c-computable implies being recursive. There are also straightforward implications between these sub-recursive notions: if E is a strongly c-enumerable (resp. strongly c-computable) set, then E is c-enumerable (resp. weakly c-computable) and χ-c-computable. Then, if E is strongly c-computable then E is weakly c-computable and χ-c-computable. If E is a weakly c-computable set or a χ-c-computable set, then E is somewhat c-computable. Hence, all our sub-recursive notions for sets imply being somewhat c-computable.

Notice that, as in the classical setting, every infinite c-enumerable set A contains an infinite strongly c-enumerable set E. Moreover, E is χ-c-computable.

Somewhat c-*computable functions.* First, recall that a partial function is *somewhat* c-*computable* if its graph is c-enumerable. Using our *simulation* function $\mathrm{sim}_{\mathsf{p}}$, it is not too difficult to create a one-one enumeration of the graph of an universal function for all recursive functions. Hence, it is easy to prove, as a corollary of the heredity lemma (Lemma 1), that there is a somewhat p-computable function $\mathsf{U}_{T,\mathsf{p}}$ universal for all recursive functions. The following theorem legitimates the notion of an effective enumeration of somewhat c-computable functions:

Theorem 2. *There exists a somewhat* p*-computable universal function* U_{c} *over* c*.*

Remarkable functions. There is a universal function among the somewhat c-computable functions. However, not all recursive functions belong to that class. As expected, the Ackermann function is not part of the fundamental functions of our primitive computability. It is not even somewhat p-computable. This result may seem contradictory with Theorem 2. It just shows again that enumerating all functions in one is easier than enumerating some. It also shows that the somewhat computable functions are not closed by composition. Nevertheless, the Ackermann function range is χ-p-computable, because its inverse does belong to p. Thus, the Ackermann function is linked to somewhat p-computable functions: if a recursive function f is somewhat p-computable, then it is less than the Ackermann function on an infinite subset of the domain.

Halting problem for different classes of function of sub-computabilities. The halting problem is central in classical computability and is linked to the diagonal set $\mathsf{K} = \{e : \varphi_e(e)\downarrow\}$. In our sub-computabilities, we can define a couple of different K's depending on the set of functions we are interested in (all recursive functions: $\mathsf{K} = \{e : \varphi_e(e)\downarrow\}$, somewhat computable (partial) functions: $\mathsf{K}_{\mathsf{c}} = \{e : \mathsf{U}_{\mathsf{c}}(e,e)\downarrow\} = \{e : \varphi^{\mathsf{c}}_e(e)\downarrow\}$, fundamental functions: $\mathsf{K}^{\Phi}_{\mathsf{c}} = \{e : \mathsf{U}^{\Phi}_{\mathsf{c}}(e,e) > 0\} = \{e : \Phi^{\mathsf{c}}_e(e) > 0\}$). Notice that the *hardness* of the different halting problems gives more sense to the distinctions between our sub-computability notions.

K is p-enumerable but not strongly p-enumerable. For every sub-computability c, $\overline{\mathsf{K}}$ is not c-enumerable, the set $\mathsf{K}^{\Phi}_{\mathsf{c}}$ is weakly p-computable but not χ-c-computable and the set K_{c} is p-enumerable but not strongly p-enumerable, nor recursive and $\overline{\mathsf{K}_{\mathsf{c}}}$ is not c-enumerable. Finally, the diagonal set on somewhat functions is complete: $\mathsf{K}_{\mathsf{c}} \equiv_m \mathsf{K}$. The proof[6] of the Turing (many-one really) completeness uses the Half recursion theorem (Theorem 3).

Strong c-enumerability implies χ-c-computability, thus we have that there exists a weakly c-computable but not strongly c-enumerable set. We summarize the properties of these sets in table 1.

Table 1. Properties in c of particular sets

	r.e	co-r.e	p-r.e	s-c-r.e	co-c-r.e	co-s-c-r.e	w-c-comp	s-c-comp	χ-c-comp
K	✓	×	✓	×	×	×	×	×	×
$\mathsf{K}^{\Phi}_{\mathsf{c}}$	✓	✓	✓	×	✓	×	✓	×	×
K_{c}	✓	×	✓	×	×	×	×	×	×

c*-computabilities' specimens.* It is possible to exhibit an infinite family of different c-computabilities using a natural extension of the primitive recursive recursion: α-recursion[7]. The different growing hierarchies give us an easy way to exhibit functions that require a given α-recursion operator. For example, the Ackermann function does not belong to p, but it belongs to[8] $\mathsf{c}_{\omega^\omega}$. And Goodstein's sequences function will only appear in c_{ϵ_0}. Using fundamental sequences, we can easily design other such examples up to Γ_0 thanks to Veblen's hierarchy (See [12]) and even beyond.

[6] Let a be the c-fund. index of a nowhere null function. Show that there is $f_x \in \mathsf{c}$ such that $\varphi^{\mathsf{c}}_{f_x(e)} : y \mapsto \begin{cases} \varphi_x(x) & \text{if } y = a \text{ or } y = e \\ 0 & \text{otherwise.} \end{cases}$. Use the set $A = \left\{\mathfrak{p}^{(n)}(z) : z \geqslant 1\right\}$ to apply the Half recursion part of Theorem 3 and effectively obtain a fixed-point n of f_x. Notice that $\varphi^{\mathsf{c}}_{f_x(n)} \cong \varphi^{\mathsf{c}}_n$ since $a \notin A$. Then, $n \in \mathsf{K}_{\mathsf{c}}$ iff $x \in \mathsf{K}$, for n c-computable from x.

[7] Let $\langle A, \lhd\rangle$ an elementary ordinal representation system (EORS) and $\alpha \in A$ such that $\overline{0} \lhd \alpha$. A function f on natural numbers is called *α-recursive* if it can be generated like the primitive recursive functions albeit plus the following operation:

$$f(m, \boldsymbol{n}) = \begin{cases} h(m, \boldsymbol{n}, f(\theta(m, \boldsymbol{n}), \boldsymbol{n})) & \text{if } \overline{0} \lhd m \lhd \alpha \text{ and } \theta(m,n) \lhd m, \\ g(m, \boldsymbol{n}) & \text{otherwise,} \end{cases}$$

where g,h,θ are α-recursive. See [1,10] for more details.

[8] Denote by c_α the set of total α-recursive functions based on EORS' provided by proof theory.

2 Common Sub-computability

Apart from the particular functions (like the diagonal Ackermann function[9] Ack or the ones built in the previous section), that dominate every c-fundamental functions, which we have for certain sub-computabilities c, there is a simple way to build for every c a total recursive function which is not c-fundamental. The function $x \mapsto \mathsf{U}^{\mathbf{\Phi}}_{\mathsf{c}}(x, x)$ is a total recursive function that is not c-fundamental, but is somewhat p-computable. Also, apart from the diagonal set adapted to c-computability, K_{c}, we can define c-versions of the *busy beaver* functions of Tibor Radó: $\mathsf{BB}_{\mathsf{c}}(x) = \max\{\varphi^{\mathsf{c}}_i(0) : i \leqslant x\}$ and $\mathsf{BB}^{\mathbf{\Phi}}_{\mathsf{c}}(x) = \max\{\mathbf{\Phi}^{\mathsf{c}}_i(0) : i \leqslant x\} + x$. The classical results and proofs carry through: BB_{c} is not somewhat c-computable. Moreover, $\overline{\text{graph}(\mathsf{BB}_{\mathsf{c}})}$ is not c-enumerable. $\mathsf{BB}^{\mathbf{\Phi}}_{\mathsf{c}}$ is recursive, but not somewhat c-computable.

There is a natural superset of the foundation c of a sub-computability c: Denote by $\textcircled{\mathsf{c}}$ the sub-computability of foundation $\mathsf{c}[\mathsf{BB}^{\mathbf{\Phi}}_{\mathsf{c}}]$. Notice that for any sub-computability c, $f \in \mathsf{c}$ does not necessarily[10] imply $f^{-1} \in \mathsf{c}$. Notice also that, for any sub-computability c, all the functions in $\textcircled{\mathsf{c}}$ stay recursive. This operation can be seen as some kind of jump since it provides a new sub-computability $\textcircled{\mathsf{c}}$ in which there is a fundamental universal function for c-fundamental functions: There is a $\textcircled{\mathsf{c}}$-fundamental universal $\mathsf{U}^{\mathbf{\Phi}}_{\mathsf{c}}$ function for c-fundamental functions. As a corollary, we have that $\chi_{\mathsf{K}^{\mathbf{\Phi}}_{\mathsf{c}}} \in \mathsf{c}[\mathsf{BB}^{\mathbf{\Phi}}_{\mathsf{c}}]$ but $\mathsf{BB}^{\mathbf{\Phi}}_{\mathsf{c}} \notin \mathsf{c}[\chi_{\mathsf{K}^{\mathbf{\Phi}}_{\mathsf{c}}}]$.

It is easy to see that the fundamental functions of $\textcircled{\mathsf{p}}$ are exactly the functions of $\mathsf{p}[\text{Ack}]$. First, notice that Ack is in $\textcircled{\mathsf{p}}$ since the p-fundamental index of $m \mapsto \text{Ack}_2(n, m)$ is computable from n and can be simulated with the universal function for p-fundamental functions. The converse is true since for any *acceptable* enumeration $\mathbf{\Phi}^{\mathsf{c}}_{.}$ of p-fundamental functions, there exists a primitive recursive function f such that $\forall e, \forall n, \text{Ack}_2(f(e), n) \leqslant \mathbf{\Phi}^{\mathsf{c}}_e(n)$[11]. Hence, the Ackermann function permits us to compute the p-fundamental busy beaver.

Following our s-m-n theorem for c-fundamental functions, we obtain generalizations of Kleene's second recursion theorem to sub-computabilities.

Theorem 3 (Recursion theorem *à la* Kleene). *Let f be a c-fundamental function.*

1. *(*Unbalanced fundamental recursion*) There is an $n \in \mathbb{N}$ such that $\mathbf{\Phi}^{\textcircled{\mathsf{c}}}_n \cong \mathbf{\Phi}^{\mathsf{c}}_{f(n)}$.*
2. *(*Unbalanced recursion*) There is an $n \in \mathbb{N}$ such that $\varphi^{\textcircled{\mathsf{c}}}_n \cong \varphi^{\mathsf{c}}_{f(n)}$.*
3. *(*Half recursion*) Let h be a somewhat c-computable function defined on an infinite c-enumerable and co-r.e. set A. There is an $n \in \mathbb{N}$ such that $\varphi^{\mathsf{c}}_n|_{\overline{A}} \cong \varphi^{\mathsf{c}}_{f(n)}|_{\overline{A}}$ and $\varphi^{\mathsf{c}}_n|_A \cong h$.*

[9] Ack_2 denotes the classical binary function and Ack the diagonal unary version.

[10] For example, take f a total computable function not in c. Transform it into f' such that $\forall n, f'(n) = \langle f(n),\ \text{steps}(f, n)\rangle$ (where steps(f, n) is the number of steps needed by f on the input n to halt). f' still does not belong to c, but f'^{-1} does.

[11] This function works by recursion on the code of $\mathbf{\Phi}^{\mathsf{c}}_e$, see [9], thm VII.8.10 p299-300.

Proof. The function $u \mapsto g_x(u) = \begin{cases} \varphi_a^{\mathsf{C}}(u) & \text{if } u \in A, \\ \varphi_{\mathbf{\Phi}_x^{\mathsf{C}}(x)}^{\mathsf{C}}(u) & \text{otherwise,} \end{cases}$ (where a is a C-index for h) is somewhat C-computable and of C-index computable from x by $d_a \in \mathsf{C}$. Let e_a be a C-fundamental index for $f \circ d_a$. Choose $n = d_a(e_a)$, since $\varphi_{d_a(e_a)}^{\mathsf{C}}(u) \cong \varphi_a^{\mathsf{C}}(u)$ if $u \in A$, and $\varphi_{d_a(e_a)}^{\mathsf{C}}(u) \cong \varphi_{\mathbf{\Phi}_{e_a}^{\mathsf{C}}(e_a)}^{\mathsf{C}}(u) \cong \varphi_{f \circ d_a(e_a)}^{\mathsf{C}}(u)$ otherwise.

We also have the usual corollary of a C-recursion theorem *with parameters.* As it is visible in the various proofs of Theorem 3, the function providing the *fixed point* for each parameter is a C-fundamental function.

As we have already remarked, we do not have a general s-m-n theorem for somewhat computable functions, only one for fundamental functions[12]. The principal reason is the fact that somewhat computable functions are not closed by composition[13]. And at the same time, there is no[14] (balanced) full fixed-point theorem on fundamental or somewhat computable functions (staying in a particular sub-computability and not using higher sub-computabilities in an unbalanced way like in Theorem 3).

[12] There is a duality at play between the fundamental functions and the somewhat C-computable functions. Duality both in their use in theorems, usually generalizations (or really restrictions) of classical computability theorems, and in the theorems they verify. The former is used to construct the latter and verifies padding and s-m-n (or composition) but not fixed-point, nor universality, while the latter verifies padding, fixed-point and universality, but not s-m-n (nor composition). An example of the use in theorems is the fixed point theorem: the fixed point is of a fundamental function over the enumeration of the somewhat computable functions. These facts are to be put in perspective with the classical results that enumeration alone or even enumeration and fixed point are not enough to ensure having an acceptable computation system, *i.e.* a computation system isomorphic to the canonical enumeration of all r.e. sets, while enumeration and composition are sufficient: Enumeration is not enough since a Friedberg enumeration, *i.e.*, without repetition, of r.e. sets will obviously not verify the padding lemma. Enumeration and fixed point is not enough by the following counter-example: if $e \mapsto \varphi_{h(e)}$ is a Friedberg enumeration without repetition of the partial recursive functions, then consider the enumeration $e \mapsto \psi_e = \varphi_{h(\varphi_e(0))}$. It is not hard to show that $\psi_\cdot$ verifies the fixed-point theorem ($\exists$ recursive f, $\forall$ total ψ_i, $\psi_{\psi_i(f(i))} \cong \psi_{f(i)}$) but each partial recursive function is equal to a $\varphi_{h(i)}$ for a unique i and thus equal to a ψ_e only if $\varphi_e(0) = i$. Thus, the set of ψ-indices of each partial recursive function is r.e., which is in opposition to the fact that the nowhere-defined function $(x \mapsto \uparrow)$ cannot have an r.e. set of indices in an acceptable enumeration.

[13] For well-behaving functions, composition is still possible and in a primitively effective way. For example, composition $f \circ g$ is possible for somewhat C-computable functions f and g such that g^{-1} is C-fundamental and there is a C-enumerable subset of the intersection of the range of g and the graph of f. If you take the function $\mathbf{J}_{\mathsf{C}} : x \mapsto \varphi_x^{\mathsf{C}}(x)$ as f, then f being somewhat P-computable and having a graph containing many P-enumerable subsets, many functions g will make $f \circ g$ somewhat C-computable.

[14] Both fundamental and somewhat (balanced) full fixed-point theorems for P would provide fixed points which would be indices of the Ackermann function. The function $n \mapsto$ fundamental C-index of $y \mapsto \sum_{x \leqslant y} \mathbf{\Phi}_n^{\mathsf{C}}(x) + 1$ is also designed not to have any of those full fixed-points.

Even though, several corollaries of s-m-n still carry through: the effective union given by H_{c} and the following corollary of the existence of a c-universal function in $\textcircled{\mathsf{c}}$: if f is a c-fundamental function, then there exists a $\textcircled{\mathsf{c}}$-fundamental function g such that for all x, $\mathbf{W}^{\mathsf{c}}_{g(x)} = f``\mathbf{W}^{\mathsf{c}}_{x}$.

It is worthwhile to remark that a universal function (for fundamental functions) and $x \mapsto \mathbf{\Phi}^{\mathsf{c}}_{x}(x)$ provide examples of functions which are not c-fundamental but which are total and somewhat c-computable. Notice also that fundamentals go beyond total somewhat c-computable functions: $\mathsf{BB}^{\mathbf{\Phi}}_{\mathsf{c}}$ is not somewhat c-computable but is $\textcircled{\mathsf{c}}$-fundamental.

We should also notice that as in classical computability, not all partial somewhat c-computable functions are extendible to total somewhat c-computable functions. A counterexample is the function $x \mapsto \varphi^{\mathsf{c}}_{x}(x) + 1$, since it is somewhat c-computable and differs with all somewhat c-computable functions and thus also with all total ones.

We generalize Rice's theorem to sub-computabilities. To this end, we define the following reducibility and notion of set of indices: a set A is somewhat c-reducible to a set B (designated by $A \preccurlyeq^{\mathsf{c}}_{\sim} B$) if there is a somewhat c-computable function f such that $x \in A$ if and only if $f(x) \in B$. A set $A \subseteq \mathbb{N}$ is a *set of* c-*somewhat indices* (resp. *a set of* c-*fundamental indices*) if for all x, y, $(x \in A \text{ and } \varphi^{\mathsf{c}}_{x} \cong \varphi^{\mathsf{c}}_{y}) \implies y \in A$, (resp. $(x \in A \text{ and } \mathbf{\Phi}^{\mathsf{c}}_{x} \cong \mathbf{\Phi}^{\mathsf{c}}_{y}) \implies y \in A$).

Theorem 4 (Rice for sub-computabilities). *If A is a non-trivial ($A \neq \varnothing, \mathbb{N}$) set of c-indices (resp. c-fundamental indices), then we have either $\mathsf{K}_{\mathsf{c}} \preccurlyeq^{\mathsf{c}}_{\sim} A$ or $\mathsf{K}_{\mathsf{c}} \preccurlyeq^{\mathsf{c}}_{\sim} \overline{A}$ (resp. $\mathsf{K}^{\mathbf{\Phi}}_{\mathsf{c}} \preccurlyeq^{\textcircled{\mathsf{c}}}_{1} A$ or $\mathsf{K}^{\mathbf{\Phi}}_{\mathsf{c}} \preccurlyeq^{\textcircled{\mathsf{c}}}_{1} \overline{A}$).*

Productivity and creativity can also be non-trivially ported to our setting: A set X is c-*productive* if there exists a one-one c-fundamental function ψ, called the *productive function* for X, such that $\forall x$, $\mathbf{W}^{\mathsf{c}}_{x} \subseteq X \implies \psi(x) \in X \setminus \mathbf{W}^{\mathsf{c}}_{x}$. A c-enumerable set X is c-*creative* if $\overline{X}$ is c-productive.

It is a non-trivial generalization of the classical notion, since Ack $``\overline{\mathsf{K}}$ is productive, but not p-productive. As expected, K_{c} is still c-creative. And variants of the properties carry through: If X is c-productive, then X is neither c-enumerable nor weakly c-computable. (Hence $\mathsf{K}^{\mathbf{\Phi}}_{\mathsf{c}}$ is not c-creative.) If X is c-productive, then X contains a c-enumerable infinite subset. And, if X is c-productive and $X \preccurlyeq^{\mathsf{c}}_{1} A$ by a function f such that f^{-1} is also one-one c-fundamental, then A is $\textcircled{\mathsf{c}}$-productive[15].

We would like to carry through the characterization of creativity by K (creativity is equivalent[16] to m-completeness) to sub-computabilities but we need for that to have the right notion of m-reducibility. The third property above hints that it could be challenging.

[15] $\mathsf{K}^{\mathbf{\Phi}}_{\mathsf{c}}$ is not c-creative, but it is $\mathbf{\Phi}_{\mathsf{c}}$-creative where $\left(\mathbf{W}^{\mathsf{c}}_{n}\right)_{n \in \mathbb{N}}$ is replaced by $\left({}^{\mathbf{\Phi}}\mathbf{W}^{\mathsf{c}}_{n} = \{y : \mathbf{\Phi}^{\mathsf{c}}_{n}(y) > 0\}\right)_{n \in \mathbb{N}}$ in the definition of productivity.

[16] If X is productive and $X \preccurlyeq_{1} A$, then A is productive. And if X is productive then $\overline{\mathsf{K}} \preccurlyeq_{1} X$.

We now generalize Rogers' isomorphism theorem to sub-computabilities. To this end, we first show that the constructive version, due to Myhill, of the Cantor-Schröder-Bernstein Theorem can be adapted to $\mathbb{C}$-computabilities: (Myhill for sub-computabilities)[17] $A \equiv_1^{\mathbb{C}} B \iff A \equiv^{\mathbb{C}} B$.

We can generalize the notion of "acceptable system of indices" to sub-computabilities:

Definition 5. An *acceptable system of* $\mathbb{C}$*-computability* is a pair $(\boldsymbol{\Psi}^{\mathbb{C}}_{\cdot}, \psi^{\mathbb{C}}_{\cdot})$ providing maps from $\mathbb{N}$ onto the set of $\mathbb{C}$-fundamental functions and the set of somewhat $\mathbb{C}$-computable functions such that there are $\mathbb{C}$-fundamental functions f, g, $\mathbf{f}$ and $\mathbf{g}$ such that for all e $\psi^{\mathbb{C}}_e \cong \varphi^{\mathbb{C}}_{f(e)}$ and $\varphi^{\mathbb{C}}_e \cong \psi^{\mathbb{C}}_{g(e)}$, $\boldsymbol{\Psi}^{\mathbb{C}}_e \cong \boldsymbol{\Phi}^{\mathbb{C}}_{\mathbf{f}(e)}$ and $\boldsymbol{\Phi}^{\mathbb{C}}_e \cong \boldsymbol{\Psi}^{\mathbb{C}}_{\mathbf{g}(e)}$ and g is one-one on indices of finite domain functions.

An acceptable system of sub-computability inherits many properties of the respective canonical sub-computability. In particular, the Half recursion[18] for sub-computabilities (Theorem 3), parametrization[19] for fundamentals and the padding lemma[20] for somewhat $\mathbb{C}$-computable functions holds in any acceptable system of sub-computability.

Theorem 5 (Isomorphism theorem *à la* Rogers). *Let* $\mathbb{C}$ *be a sub-computability. For any acceptable system of* $\mathbb{C}$*-computability,* $(\boldsymbol{\Psi}^{\mathbb{C}}_{\cdot}, \psi^{\mathbb{C}}_{\cdot})$*, there is a* $\mathbb{C}$*-fundamental permutation* h *of* ω *such that for all* e*,* $\varphi^{\mathbb{C}}_e \cong \psi^{\mathbb{C}}_{h(e)}$.

We thus notice that acceptable systems of indices provide the same structure theory for a sub-computability as the *standard* one we have defined and been using. Things happen not because of the coding used: a foundation induces a unique sub-computability.

3 Sub-reducibilities

Our interest in sub-reducibilities in the context of sub-computabilities is twofold.

[17] A set A is $\mathbb{C}$-reducible to a set B (designated by $A \preccurlyeq_1^{\mathbb{C}} B$) if there is a one-one $\mathbb{C}$-fundamental function f such that $x \in A$ if and only if $f(x) \in B$. A set A is $\mathbb{C}$-equivalent to a set B (designated by $A \equiv_1^{\mathbb{C}} B$) if $A \preccurlyeq_1^{\mathbb{C}} B$ and $B \preccurlyeq_1^{\mathbb{C}} A$. A set A is $\mathbb{C}$-isomorphic to B (designated by $A \equiv^{\mathbb{C}}$) if there exists a $\mathbb{C}$-fundamental permutation p such that $p''(A) = B$.

[18] There is a $\boldsymbol{\Psi}^{\mathbb{C}}_{\cdot}$-fundamental index Kleene_ψ such that for all i, $\mathbf{j}$, if $\mathrm{domain}(\psi^{\mathbb{C}}_i)$ is infinite and co-r.e., then $\psi^{\mathbb{C}}_{\boldsymbol{\Psi}^{\mathbb{C}}_{\mathrm{Kleene}_\psi}(i,\mathbf{j})}|_{\overline{\mathrm{domain}(\psi^{\mathbb{C}}_i)}} \cong \psi^{\mathbb{C}}_{\boldsymbol{\Psi}^{\mathbb{C}}_{\mathbf{j}}(\boldsymbol{\Psi}^{\mathbb{C}}_{\mathrm{Kleene}_\psi}(i,\mathbf{j}))}|_{\overline{\mathrm{domain}(\psi^{\mathbb{C}}_i)}}$ and $\psi^{\mathbb{C}}_{\boldsymbol{\Psi}^{\mathbb{C}}_{\mathrm{Kleene}_\psi}(i,\mathbf{j})} \cong \psi^{\mathbb{C}}_i$.

[19] There is a $\boldsymbol{\Psi}^{\mathbb{C}}_{\cdot}$-fundamental index param_ψ such that for all i_0, i_1, $\mathbf{j}$, the function $\psi^{\mathbb{C}}_{\boldsymbol{\Psi}^{\mathbb{C}}_{\boldsymbol{\Psi}^{\mathbb{C}}_{\mathrm{param}_\psi}(\langle i_0, i_1, \mathbf{j}\rangle)}(e)}$ is equal (when defined) to $\begin{cases} \psi^{\mathbb{C}}_{i_0} \text{ if } \boldsymbol{\Psi}^{\mathbb{C}}_{\mathbf{j}}(e) = 0, \\ \psi^{\mathbb{C}}_{i_1} \text{ otherwise.} \end{cases}$

[20] In any acceptable system of $\mathbb{C}$-computability, given one index of a somewhat $\mathbb{C}$-computable function with infinite domain, we can $\mathbb{C}$-fundamentally generate infinitely many indices of the same function.

As we have already seen with the previous theorems, the same kind of results than the classical ones appear in most sub-computabilities. For example, we have recursion theorems *à la* Kleene and an isomorphism theorem *à la* Rogers. It is even more striking with the degree structure as will be outlined in this section. We will see in particular that each Turing degree is divided into infinitely many different degrees, our sub-reducibilities being a recursive refinement of the classical ones. Some computable sets are shown to play a role similar to the classical solutions of Post's problem, from our sub-reducibilities point of view. As these results are quite natural and easy to prove, it would be interesting, and it is one of our goal for the future, to prove some kind of degree-structure homogeneity theorem to be able to somehow carry the sub-computability proofs back to the classical computability setting.

Another goal in the study of sub-computabilities is to find real classical computability sets, *e.g.*, solutions to Post's problem, that are already in a sub-computability but not in *weaker* sub-computabilities, *i.e.*, a r.e. set $\varnothing <_T X <_T \varnothing'$ that first appears in the c-r.e. sets.

Since $A \preccurlyeq_T B$ means that A is computable in B (really, computable in χ_B), and in sub-computabilities, we have several notions of recursiveness (for a set), we now introduce several associated Turing (or stronger) reducibilities:

Definition 6 (Sub-reducibilities). A function f is χ-c-*Turing* reducible to B ($f \preccurlyeq^{\chi}_{\mathsf{c}\text{-}T} B$) if f is in $\mathsf{c}[\chi_B]$.[21] A set A is χ-c-*Turing* reducible to B if its characteristic function is.

A function f is somewhat c-*Turing* reducible to B ($f \preccurlyeq^{\sim}_{\mathsf{c}\text{-}T} B$) if f is somewhat computable over $\mathsf{c}[\chi_B]$. A set A is somewhat c-*Turing* reducible to B if its characteristic function is.

A set A is χ-c-r.e. in B if A is one-one enumerable by a function χ-c-*Turing* reducible to B.

A set A is weakly c-*Turing* reducible to B ($A \preccurlyeq^{\mathrm{w}}_{\mathsf{c}\text{-}T} B$) if A and $\overline{A}$ are one-one enumerable by functions in $\mathsf{c}[\mathfrak{e}_B, \mathfrak{e}_{\overline{B}}]$, for any function $\mathfrak{e}_X$ which one-one enumerates X.

A set A is strongly-c-*Turing* reducible to B ($A \preccurlyeq^{\mathrm{s}}_{\mathsf{c}\text{-}T} B$) if A is enumerable by an increasing function in $\mathsf{c}[p_B, p_{\overline{B}}]$, where p_X is the increasing enumeration of X.

A function is reducible to B if its graph is reducible to B.

When the type of c-reducibility is not specified, it usually means that it is true for each of these reducibilities.

K_{c} is (weak/strong/χ) c-Turing complete: $\mathsf{K} \equiv_{\mathsf{c}\text{-}T} \mathsf{K}_{\mathsf{c}} \equiv_{\mathsf{c}\text{-}T} \varnothing'$. There are also new incomparable sets: for some A and some c, $A \not\preccurlyeq^{\chi}_{\mathsf{c}\text{-}T} A'$, for example, for $A = \text{range(Ack)}$ and $\mathsf{c} = \mathsf{p}$.

$\preccurlyeq^{\chi}_{\mathsf{c}\text{-}T}$ is strictly stronger than the truth table reducibility $\preccurlyeq_{tt}$ (and thus also than the weak truth table reducibility $\preccurlyeq_{wtt}$ and the Turing reducibility $\preccurlyeq_T$).

[21] $\mathsf{c}[g]$ is the primitive closure of $\mathsf{c} \cup \{g\}$ (See Definition 1). We could take a stronger closure since c could be closed by more than composition and primitive recursion but for our purposes, taking the primitive closure seems to be enough.

$\preccurlyeq^{\mathrm{w}}_{\mathsf{C}\text{-}T}$ and $\preccurlyeq^{\mathrm{s}}_{\mathsf{C}\text{-}T}$ are strictly stronger than the Turing reducibility (the inclusion is trivial and range(Ack) $\preccurlyeq_T \varnothing$ but range(Ack) $\not\preccurlyeq^{\mathrm{w,\,s}}_{\mathsf{C}\text{-}T} \varnothing$). $\preccurlyeq^{\mathrm{w}}_{\mathsf{C}\text{-}T}$ is incomparable[22] with $\preccurlyeq_{tt}$ and $\preccurlyeq_{wtt}$. Ack $\preccurlyeq^{\chi}_{\mathsf{C}\text{-}T} \varnothing$ but Ack $\not\preccurlyeq^{\mathrm{s}}_{\mathsf{C}\text{-}T} \varnothing$, thus $\preccurlyeq^{\chi}_{\mathsf{C}\text{-}T}$ does not imply $\preccurlyeq^{\mathrm{s}}_{\mathsf{C}\text{-}T}$. $\preccurlyeq^{\chi}_{\mathsf{C}\text{-}T}$ implies $\preccurlyeq^{\sim}_{\mathsf{C}\text{-}T}$, which in turn implies $\preccurlyeq_{wtt}$.

A generalization of *Martin-Arslanov-Lachlan's completeness criterion* holds: Given a C-enumerable set A, $\varnothing' \preccurlyeq^{\sim}_{\mathsf{C}\text{-}T} A$ is equivalent to the existence of a weakly C-computable set B and function $f \preccurlyeq^{\sim}_{\mathsf{C}\text{-}T} A$, such that f is FPF$[B]$[23].

One of the interests of considering these reducibilities is to see what sub-computabilities have to say about the computability of sets of integers that are not necessarily recursively enumerable. A lot of results of classical computability, especially concerning Δ^0_2 sets, can be extended (or really restricted) to sub-computabilities but we will only consider here ways of providing solutions to Post's problem of finding an incomputable yet incomplete recursively enumerable set.

When considering Post's problem with regard to these strong reducibilities $\preccurlyeq_{\mathsf{C}\text{-}T}$, it becomes fast obvious that finding solutions is very simple.

Theorem 6 (Solutions to Post's sub-problem). *There exists an r.e. set X such that $\varnothing \prec_{\mathsf{C}\text{-}T} X \prec_{\mathsf{C}\text{-}T} \varnothing'$ for $\prec_{\mathsf{C}\text{-}T} = \prec^{\chi}_{\mathsf{C}\text{-}T}$ or $\prec^{s}_{\mathsf{C}\text{-}T}$.*

A proof of this theorem may use the fact that the χ-C-jump[24] of $\mathsf{K}^{\Phi}_{\mathsf{C}}$ is χ-C-reducible to $\mathsf{K} \equiv^{\chi}_{\mathsf{C}\text{-}T} \mathsf{K}_{\mathsf{C}} \equiv^{\chi}_{\mathsf{C}\text{-}T} \varnothing'$. $\mathsf{K}^{\Phi}_{\mathsf{C}}$ is thus a C-low C-r.e. set; it represents a low r.e. set in the C sub-computability. The set graph$(\mathsf{BB}^{\Phi}_{\mathsf{C}})$ is another example of a C-low set, albeit not C-r.e.

There are obvious solutions to Post's (real) problem in our sub-computabilities, *i.e.*, a C-r.e. set X such that $\varnothing <_T X <_T \varnothing'$. But the nice thing is that we can manage to make them appear only starting from a given C-computability. And the same goes for C-r.e. low degrees.

These results put together hint to the following lemma: in each 1-degree, one can find a set as high in the sub-computability hierarchy as desired. More generally, for any sub-computability C, each r.e. 1-degree has a non-C-enumerable r.e. member and a C-enumerable member.

A corollary that completes the previous result is that, there exists a non-C-enumerable r.e. set X, such that $\varnothing <_T X <_T \varnothing'$. A consequence is that there exists a low r.e. set W which is not C-enumerable.

Using a priority argument, we can even ensure the low promptly simple set X to be C-low ($X'_{\mathsf{C}} \preccurlyeq_{\mathsf{C}\text{-}T} \varnothing'$) but not C^--low for sub-computabilities C^- weaker[25]

[22] Let A be a set having the property of being Turing complete but not truth table complete, *e.g.*, an effectively hypersimple set. Then, $B = \{3a : a \in A\} \cup \{3n+1 : n \in \mathbb{N}\}$ is weakly C-Turing complete and not truth table complete.

[23] f is fixed point free relatively to B (FPF$[B]$) if $\forall x, \exists n \in B, \varphi^{\mathsf{C}}_x(n) \neq \varphi^{\mathsf{C}}_{f(x)}(n)$.

[24] Since this section is, for now, only an introduction to sub-reducibilities in sub-computabilities, we will only introduce the χ-C-jump in this footnote. The (degree of the) χ-C-jump of A, denoted by A'_{C}, is the $\preccurlyeq^{\chi}_{\mathsf{C}\text{-}T}$ maximal equivalence class containing a set which is χ-C-r.e. in A.

[25] In the sense that C^- is strictly included in C.

than $\mathbb{C}$, in the same way we can build a low simple set that is not superlow (see [7, Ex. 1.6.7, p. 386]).

As we saw multiple times in this paper, sub-computabilities are strongly linked with fast growing functions, which gives evidence of their *recursive power.* As this kind of reasoning also appears in honest elementary degrees theory we hope to soon be able to develop the links between these degree theories.

Other immediate goals are to port some essential computability results to sub-computabilities, especially various characterization of r.e. sets and reducibilities.

Acknowledgements. We would like to thank Bruno Durand for many challenging discussions on the ideas of this paper. Many thanks also to the anonymous referee whose comments and remarks have helped us to clarify its presentation.

References

1. Friedman, H., Sheard, M.: Elementary descent recursion and proof theory. Annals of Pure and Applied Logic 71(1), 1–45 (1995)
2. Koz'minyh, V.V.: On a presentation of partial recursive functions by compositions. Algebra i Logika 11(3), 270–294 (1972) (in Russian)
3. Kristiansen, L.: Papers on subrecursion theory. Ph.D. thesis, Department of Informatics, University of Oslo (1996)
4. Kristiansen, L.: A jump operator on honest subrecursive degrees. Archive for Mathematical Logic 37(2), 105–125 (1998)
5. Kristiansen, L.: Low$_n$, high$_n$, and intermediate subrecursive degrees. In: Calude, D. (ed.) Combinatorics, Computation and Logic, pp. 286–300. Springer, Singapore (1999)
6. Kristiansen, L., Schlage-Puchta, J.C., Weiermann, A.: Streamlined subrecursive degree theory. Annals of Pure and Applied Logic (to appear, 2011)
7. Nies, A.: Computability and Randomness. Oxford University Press, Oxford (2009)
8. Odifreddi, P.: Classical Recursion Theory. Elsevier, North-Holland, Amsterdam (1988)
9. Odifreddi, P.: Classical Recursion Theory, vol. II. North Holland - Elsevier, Amsterdam (1999)
10. Rathjen, M.: The realm of ordinal analysis. In: Cooper, S.B., Truss, J. (eds.) Sets and Proofs, pp. 219–279. Cambridge University Press, Cambridge (1999)
11. Rose, H.E.: Subrecursion: Functions and Hierarchies, Oxford Logic Guides, vol. 9. Oxford University Press, USA (1984)
12. Veblen, O.: Continuous increasing functions of finite and transfinite ordinals. Trans. Amer. Math. Soc. 9, 280–292 (1908)

Functions That Preserve p-Randomness[⋆]

Stephen A. Fenner

Computer Science and Engineering Department,
University of South Carolina,
Columbia, SC 29208 USA
fenner.sa@gmail.com

Abstract. We show that polynomial-time randomness (p-randomness) is preserved under a variety of familiar operations, including addition and multiplication by a nonzero polynomial-time computable real number. These results follow from a general theorem: If $I \subseteq \mathbb{R}$ is an open interval, $f : I \to \mathbb{R}$ is a function, and $r \in I$ is p-random, then $f(r)$ is p-random provided

1. f is p-computable on the dyadic rational points in I, and
2. f varies sufficiently at r, i.e., there exists a real constant $C > 0$ such that either
$$(\forall x \in I - \{r\}) \left[\frac{f(x) - f(r)}{x - r} \geq C \right]$$
or
$$(\forall x \in I - \{r\}) \left[\frac{f(x) - f(r)}{x - r} \leq -C \right] .$$

Our theorem implies in particular that any analytic function about a p-computable point whose power series has uniformly p-computable coefficients preserves p-randomness in its open interval of absolute convergence. Such functions include all the familiar functions from first-year calculus.

Keywords: Randomness, p-randomness, complexity, polynomial-time, measure, martingale, real analysis.

Subject Classification: Computational complexity.

1 Introduction

Informally, we might call an infinite binary sequence "random" if we see no predictable patterns in the sequence. Put another way, a sequence is random if it looks "typical," that is, it enjoys no easily identifiable properties not shared by almost all other sequences. Here, the notion of "almost all" comes from Lebesgue measure on the unit interval $[0, 1]$. What we mean by "easily identifiable," on the other hand, can vary greatly with the situation. In statistics, random sequences are useful to avoid bias in sampling or in simulating processes (e.g., queueing

[⋆] Partially supported by NSF grants CCF-0515269 and CCF-0915948.

systems) that are too complex for us to determine exactly. In statistics, desirable properties for random sequences include instances of the law of large numbers: a fixed sequence of length n should occur in the sequence asymptotically a 2^{-n} fraction of the time, for example. Other examples include the law of the iterated logarithm. In cryptography and network security, "easily identifiable" must be strengthened to "unpredictable by an adversary." In computer science generally, random sequences should produce successful results most of the time when used in various randomized algorithms.

There is always a trade-off between the amount of randomness possessed by a sequence and the ease with which it can be produced. Random sequences that can be produced algorithmically (i.e., pseudorandom sequences) are of course desirable, provided they have enough randomness for the task at hand. The study of algorithmic randomness has a long and rich history (see, for example, [4,3] for references to the literature). Complexity theoretic notions of randomness were first suggested by Schnorr, and resource-bounded measure and randomness were developed more fully by Lutz (see [8]). For a survey on the subject, see [1].

A natural trade-off in the context of polynomial-time computation is the notion of polynomial-time randomness, or p-randomness for short (see Definition 1, below), which is closely tied with the notion of p-measure introduced by Lutz [6,7]. There are p-random sequences that can be computed in exponential time; in fact, almost all sequences in **EXP** (in a resource-bounded measure theoretic sense) are p-random. Yet p-random sequences are still strong enough for many common tasks, both statistical and computational. For example, p-random sequences satisfy the laws of large numbers and the iterated logarithm (see [13]), and they provide adequate sources for **BPP** computations and have many other desirable computational properties (see [8]).

The current work addresses some geometric aspects of p-random sequences. Recently, connections between the geometry of Euclidean space and effective and resource-bounded measure and dimension have been found [9,10]. The question of how the complexity or measure theoretic properties of a real number are altered when it is transformed via a real-valued function goes back at least to Wall [12], who showed that adding or multiplying a nonzero rational number to a real number whose base-k expansion is normal[1] yields another real with a normal base-k expansion. Doty, Lutz, & Nandakumar recently extended Wall's result, showing that the finite-state dimension of the base-k expansion of a real number is preserved under addition or multiplication by a nonzero rational number [2]. At the other extreme of the complexity spectrum, it is not hard to show that algorithmic randomness (Martin-Löf randomness [11]) is preserved under addition or multiplication by a nonzero computable real, regardless of the base of the expansion.

In this paper we take a middle ground, considering how polynomial-time computable functions mapping reals to reals preserve p-randomness. We show

[1] An infinite sequence s over a k-letter alphabet Σ is *normal* iff for any finite string $w \in \Sigma^*$, there are $nk^{-|w|}(1 + o(1))$ occurrences of w as a substring among the first n letters of s, as n tends to infinity.

(Theorem 1, below) that such a function f maps a p-random real r to a p-random real $f(r)$ provided f satisfies a kind of anti-Lipschitz condition in some neighborhood of r: $f(x)$ varies from $f(r)$ at least linearly in $x - r$. (This result still holds even if f is not monotone in any neighborhood of r, or if f is only polynomial-time computable on dyadic rational inputs, or if f enjoys no particular continuity properties.)

Our result has a number of corollaries: p-randomness is preserved under addition and multiplication by nonzero p-computable reals (complementing the results in [12,2] and the folklore result about algorithmically random reals); it is also preserved by polynomial and rational functions (with p-computable coefficients) and all the familiar transcendental functions on the reals, e.g., exponential, logarithmic, and trigonometric functions.

The polynomial-time case presents some technical challenges not present in the resource-unbounded case. Roughly speaking, given a polynomial-time approximable function $f : \mathbb{R} \to \mathbb{R}$, our goal is to define a betting strategy (i.e., a martingale; see Section 2) that bets on the next bit of the binary expansion of a real number r, given previous bits. The strategy is based on the behavior of an assumed strategy d that successfully bets on $f(r)$. If we had no resource bounds, then we could approximate f at various points as closely as needed to obtain a good sample of d's behavior on f applied to those points, allowing us to mimic d and thus succeed on r. Since we are polynomial-time-bounded, however, we have no such luxury, and we have to settle for rougher approximations of f. For example, d could succeed on $f(0.0111111111\cdots)$ (where there is a long string of 1's before the next 0 in the argument to f) but lose everything on $f(0.10000\cdots)$, which is close by. If we only have a poor approximation to f, then we cannot distinguish the two cases above, and so d is no good at telling us how to bet on the first digit after the decimal point. Fortunately, we may assume that d is conservative—in the sense that it does not bet drastically—so that d's assets are relatively insensitive to slight variations in the real numbers corresponding to the sequences it bets on.

Section 2 has basic definitions, including martingales and p-randomness. Section 3 describes the conditions on real-valued functions sufficient to preserve p-randomness. Our main results are in Section 4, where we prove that these conditions indeed suffice; Theorem 1 is the main result of that section. In Section 5, we show that these conditions hold for a variety of familiar functions. We suggest further research in Section 6.

This paper is an extended abstract, with most of the proofs omitted. A complete draft with all proofs can be found in [5].

2 Notation and Basic Definitions

We let $\mathbb{N} = \{0, 1, 2, \ldots\}$. We let $\mathbb{Q}$ be the set of rational numbers. A *dyadic rational* is some $q \in \mathbb{Q}$ expressible as $\pm a/2^b$ for some $a, b \in \mathbb{N}$. We let $\mathbb{Q}_2$ denote the set of dyadic rational numbers.

For real $x > 0$, we let $\lg x$ denote $\log_2 x$.

In this paper, we only consider the binary expansions of real numbers. If need be, all our results can easily be modified to other bases.

Our basic notions and results about p-computability, martingales, and randomness in complexity theory are standard. See, for example, [7,8,1].

Let $w \in \{0,1\}^*$ and $s \in \{0,1\}^\infty$. We let $|w|$ denote the length of w, and for any $0 \le i < |w|$ we let $w[i]$ be the $(i+1)$st bit of w. Similarly, for any $i \in \mathbb{N}$ we let $s[i]$ denote the $(i+1)$st bit of s. For any $n \in \mathbb{N}$ we let $s[0\ldots(n-1)] = s[0]s[1]\cdots s[n-1] \in \{0,1\}^*$ denote the first n bits of s. We let $\{0,1\}^n$ denote the set of strings in $\{0,1\}^*$ of length n. If $v \in \{0,1\}^* \cup \{0,1\}^\infty$, we let $w \sqsubseteq v$ mean that w is a prefix of v, and we let $w \sqsubset v$ mean that w is a proper prefix of v. We denote the empty string by λ.

Recall that a *martingale* is a function $d : \{0,1\}^* \to \mathbb{R}$ such that for every $w \in \{0,1\}^*$,

$$0 \le d(w) = \frac{d(w0) + d(w1)}{2} .$$

We will also assume without loss of generality that $d(\lambda) \le 1$. We say that d *succeeds* on a sequence $s \in \{0,1\}^\infty$ iff

$$\limsup_{n\to\infty} d(s[0\ldots(n-1)]) = \infty .$$

We say that d *strongly succeeds* on s iff

$$\liminf_{n\to\infty} d(s[0\ldots(n-1)]) = \infty .$$

A function $d : \{0,1\}^* \to \mathbb{R}$ is *p-computable* if there is a polynomial time computable function $\hat{d} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Q}$ such that

$$\left| d(w) - \hat{d}(w, 0^r) \right| \le 2^{-r}$$

for every $w \in \{0,1\}^*$ and $r \in \mathbb{N}$. We say that $\hat{d}$ is a *p-approximator* for d. A real number c is *p-computable* if the constant function $\{0,1\}^* \to \{c\}$ is p-computable, and we may suppress the first argument in a p-approximator for c.

Definition 1. *A sequence $s \in \{0,1\}^\infty$ is* p-random *if no p-computable martingale succeeds on s.*

Definition 2. *We will say that a martingale d is* conservative *iff*

1. *for any $w \in \{0,1\}^*$ and $b \in \{0,1\}$,*

$$\frac{d(w)}{2} \le d(wb) \le \frac{3d(w)}{2} ,$$

and

2. *for any $s \in \{0,1\}^\infty$, if d succeeds on s, then d strongly succeeds on s.*

Note that if d is conservative, then $d(w) \leq (3/2)^{|w|}$ for all w. It is well-known (and easy to show) that if there is a p-computable martingale that succeeds on s, then there is a conservative p-computable martingale that succeeds on s.

We identify a sequence $s \in \{0,1\}^\infty$ with a real number $0.s \in [0,1]$ via the usual binary expansion: $0.s := \sum_{i=0}^{\infty} s[i]2^{-(i+1)}$. This correspondence is one-to-one except on $\mathbb{Q}_2$, where it is two-to-one. For every $x \in \{0,1\}^*$, we define $0.x := 0.x000\cdots$, and we define the *dyadic interval*

$$\Gamma_x := [0.x, 0.x + 2^{-|x|}] = \{0.s : s \in \{0,1\}^\infty \wedge x \sqsubset s\} .$$

For $s \in \{0,1\}^\infty$, we define $0.s$ to be p-random iff s is p-random. If $x \in \mathbb{R}$, then we define x to be p-computable (p-random) just as we do for $x - \lfloor x \rfloor$. It is well-known that no p-computable real number is p-random.

3 Functions of Interest

We will restrict our attention to certain types of real-valued functions of a real variable. We are only interested in the behavior of these functions on p-random inputs. For simplicity, we will only consider functions with domain $[0,1]$, but this is in no way an essential restriction. Our functions will possess a certain p-computability property and a certain strong variation property. Both these properties are *local* in the sense that we only care about them in the vicinity of a p-random number.

Definition 3. *A function $f : [0,1] \to \mathbb{R}$ is* weakly p-computable *if there exists a polynomial-time computable function $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Q}$ such that for any $w \in \{0,1\}^*$ and $r \in \mathbb{N}$,*

$$\left|\hat{f}(w, 0^r) - f(0.w)\right| \leq 2^{-r} .$$

Note that a weakly p-computable function can behave arbitrarily on $[0,1] - \mathbb{Q}_2$.

Definition 4. *Let $f : [0,1] \to \mathbb{R}$ be a function and let $\Gamma_y \subseteq [0,1]$ be some dyadic interval with $y \in \{0,1\}^*$. We say that f is* weakly p-computable on Γ_y *iff there exists a ptime computable function $\hat{f} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Q}$ such that for any $w \in \{0,1\}^*$ and $r \in \mathbb{N}$,*

$$\left|\hat{f}(w, 0^r) - f(0.(yw))\right| \leq 2^{-r} .$$

If $x \in [0,1]$, then we say that f is weakly p-computable at x *iff f is weakly p-computable on some dyadic interval containing x.*

We say that f is locally weakly p-computable *if f is weakly p-computable at all p-random points in $[0,1]$.*

[Note that $0.(yw) \in \mathbb{Q}_2$ is the dyadic rational number corresponding to the string yw (the concatenation of y and w).]

In other words, f is weakly p-computable at x iff we can approximate f on the dyadic rationals in some dyadic interval containing x in polynomial time. Notice that we are *not* insisting that f have any continuity properties. This means in particular that $\hat{f}$ may not uniquely determine f on Γ_x. Notice also that a function may be locally weakly p-computable but not "globally" p-computable, being patched together nonuniformly with various p-computable functions on different dyadic intervals.

We can extend Definition 4 to weak p-computability at an arbitrary point $x \in \mathbb{R}$ in the natural way.

Definition 5. *Let $I \subseteq \mathbb{R}$ be an interval, let $f : I \to \mathbb{R}$ be a function, and let $x \in I$ be some point. We say that f* strongly varies at x on I *iff there is some real constant $C > 0$ such that either*

1. *for all $z \in I - \{x\}$,*
$$\frac{f(z) - f(x)}{z - x} \geq C ,$$
or
2. *for all $z \in I$,*
$$\frac{f(z) - f(x)}{z - x} \leq -C .$$

In case (1) we say that f strongly increases at x on I*, and in case (2) f* strongly decreases at x on I*.*

We say that f strongly varies at x *if f strongly varies at x on N for some open interval N containing x. We define f strongly increasing/decreasing at x analogously.*

The notion of strong variation is illustrated in Figure 1.

Example 1. If f is C^1 in a neighborhood of x and $f'(x) \neq 0$, then f strongly varies at x.

4 Main Result

Here is our main technical theorem, from which the other results in the paper follow easily.

Theorem 1. *Let $I \subseteq \mathbb{R}$ be some interval and $f : I \to \mathbb{R}$ some function. Suppose r is a p-random point in the interior of I. If f is weakly p-computable at r and strongly varies at r, then $f(r)$ is p-random.*

4.1 Establishing Theorem 1

We start this section with two easy observations which we give without proof.

Observation 1. *Let n be any integer, and let $a \in \mathbb{Q}_2$. A number $r \in \mathbb{R}$ is p-random if and only if $2^n r$ is p-random, if and only if $r + a$ is p-random, if and only if $-r$ is p-random.*

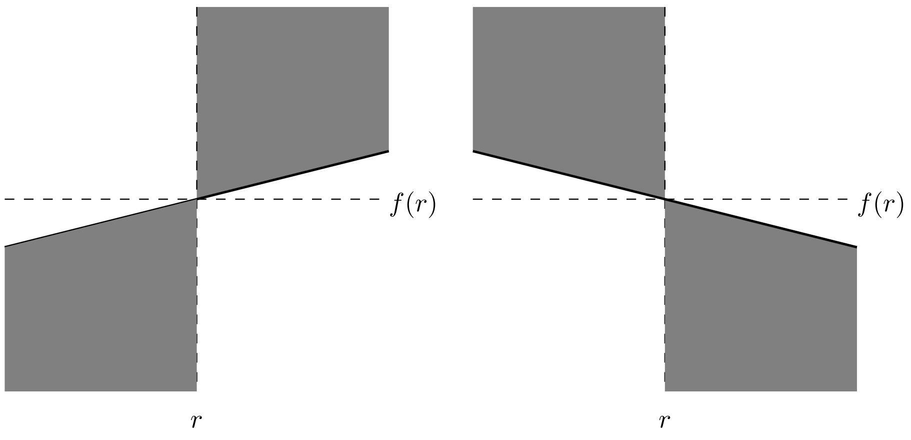


Fig. 1. For f to strongly vary at r, its graph must confine itself to the shaded region on the left (if strongly increasing) or the right (if strongly decreasing) in some neighborhood of r. The thick line on the left has slope C (satisfying $y - f(r) = C(x - r)$), and the line on the right has slope $-C$ (satisfying $y - f(r) = -C(x - r)$), for some constant $C > 0$.

Observation 2. *Let $I \subseteq \mathbb{R}$ be an interval, let $f : I \to \mathbb{R}$ be a function, let n be any integer, and let $a \in \mathbb{Q}_2$. Define*

$$\begin{aligned} g(x) &= 2^n f(x) \ , \\ h(x) &= f(x) + a \ , \\ j(x) &= f(2^n x) \ , \\ k(x) &= f(x + a) \ . \end{aligned}$$

Then f strongly varies at some $x \in I$ on I (respectively, is p-computable at x) if and only if all of $(-f), g, h$ strongly vary (respectively, are p-computable) at x on I, if and only if j strongly varies (respectively, is p-computable) at $2^{-n}x$ on $2^{-n}I$, if and only if k strongly varies (respectively, is p-computable) at $x - a$ on $I - a$. The sense of variation (strongly increasing or strongly decreasing) of f is the same as that of g, h, j, k and opposite that of $(-f)$.

Theorem 1 is a corollary of the next lemma, which gives the theorem its essential technical content. We prove this lemma later in this section. For convenience, we will assume that our function f is monotone ascending. We will show later that this is not an essential restriction.

Lemma 1. *Let $f : [0, 1] \to \mathbb{R}$ be weakly p-computable and monotone ascending on $[0, 1]$. Suppose that $x_0 \in [0, 1]$ and that f strongly increases at x_0 on $[0, 1]$. Then if $f(x_0)$ is not p-random, then x_0 is not p-random.*

To prove Lemma 1, we need to construct a martingale d_f that succeeds on x_0, given one that succeeds on $f(x_0)$. If martingale d succeeds on $f(x_0)$ then we can define $d_f(w)$ (for a given string w) by sandwiching it between a lower bound $d^-(w;n)$ and an upper bound $d^+(w;n)$. We get $d^+(w;n)$ by overestimating d's total contribution in an interval around $f(0.w)$ (Equation (1), below), and we get $d^-(w;n)$ by underestimating it (Equation (2)). These estimates become more refined as n increases, and, provided d is conservative, they reach a common limit as n goes to infinity, yielding a well-defined martingale d_f.

Definition 6. *Let $f : [0,1] \to [0,1]$ be monotone ascending on $[0,1]$ and let d be a martingale. For every $x \in \{0,1\}^*$, let Δ_x denote the interval $f(\Gamma_x) = [f(0.x), f(0.x + 2^{-|x|})]$, and for every $n \in \mathbb{N}$, define*

$$d^+(x;n) = 2^{-n} \sum_{y \in \{0,1\}^{|x|+n} \,:\, \Gamma_y \cap \Delta_x \neq \emptyset} d(y) \ , \tag{1}$$

and define

$$d^-(x;n) := 2^{-n} \sum_{y \in \{0,1\}^{|x|+n} \,:\, \Gamma_y \subseteq \Delta_x} d(y) \ . \tag{2}$$

The following lemma is routine and easy to check.

Lemma 2. *Let f and d be as in Definition 6. For any $x \in \{0,1\}^*$, if Γ_y is any dyadic interval contained in Δ_x (that is, $\Gamma_y \subseteq \Delta_x$), then letting $n = |y| - |x|$,*

$$\begin{aligned} 2^{-n} d(y) \le d^-(x;n) \le d^-(x;n+1) \le d^-(x;n+2) \le \\ \cdots \le d^+(x;2) \le d^+(x;1) \le d^+(x;0) \ . \end{aligned}$$

Proof. See [5].

Definition 7. *Let f and d be as in Definition 6. We define the* upper f-shift *of d to be the function defined for all $x \in \{0,1\}^*$ as*

$$d^+(x) := \lim_{n\to\infty} d^+(x;n) \ .$$

Similarly, we define the lower f-shift *of d to be*

$$d^-(x) := \lim_{n\to\infty} d^-(x;n) \ .$$

Since for any fixed $x \in \{0,1\}^*$, $d^+(x;n)$ and $d^-(x;n)$ are both monotone functions of n (decreasing and increasing, respectively) by Lemma 2, the limits in the definition above clearly exist, and

$$d^-(x;n) \le d^-(x) \le d^+(x) \le d^+(x;n)$$

for all n.

For some martingales, the upper and lower f-shifts may differ, but they coincide for conservative martingales.

Lemma 3. *Fix f and d as in Definition 6. Suppose further that d is conservative. For any $x \in \{0,1\}^*$ and $n \in \mathbb{N}$,*

$$d^+(x;n) - d^-(x;n) \le 2\left(\frac{3}{2}\right)^{|x|}\left(\frac{3}{4}\right)^n .$$

Proof. See [5].

Corollary 1. *Let f and d be as in Definition 6. If d is conservative, then $d^+(x) = d^-(x)$ for all $x \in \{0,1\}^*$.*

Proof. Immediate from Lemma 3.

Definition 8. *If f and d are as in Definition 6 and d is conservative, then we let $d_f(x)$ denote the common value $d^+(x) = d^-(x)$, and we call d_f the f-*pullback *of d.*

On input string x, $d_f(x)$ merely samples d over the the interval $\Delta_x = f(\Gamma_x)$.

Lemma 4. *If f and d are as in Definition 6 and d is conservative, then its f-pullback d_f is a martingale.*

Proof. See [5].

The next lemma is key. Here is where we make essential use of the strongly increasing property of f. (The hypothesis here is slightly weaker, though).

Lemma 5. *Let f and d be as in Definition 6 with d being conservative. Suppose that there exist $r, s \in \{0,1\}^\infty$ and a real $C > 0$ such that*

$$\frac{f(x) - 0.r}{x - 0.s} \ge C \tag{3}$$

for all $x \in [0,1] - \{0.s\}$. If d succeeds on r and $0.s \notin \mathbb{Q}_2$, then d_f succeeds on s.

Proof. See [5].

Finally, we need a lemma regarding p-computability. The challenge in the proof is in finding an easy (i.e., polynomial-time) way to approximate the $d^-(x;n)$ and $d^+(x;n)$.

Lemma 6. *Let d be a conservative martingale, let $f : [0,1] \to [0,1]$ be monotone ascending on $[0,1]$, and let d_f be the corresponding f-pullback of d. Also assume $f(1) = 1$. If d is p-computable and f is weakly p-computable on $[0,1]$, then d_f is p-computable.*

Proof. See [5].

Proof (of Lemma 1). Let f and x_0 be as in Lemma 1. If $x_0 \in \mathbb{Q}_2$, then it is clearly not p-random, and we are done. Otherwise, let $\ell = \lfloor f(0) \rfloor$, let $h = \lceil f(1) \rceil$, and let $m \geq 0$ be the least natural number such that $2^m \geq h - \ell$. For all $x \in [0, 1)$, define

$$g(x) := 2^{-m}(f(x) - \ell) ,$$

and define $g(1) := 1$. Then $g : [0, 1] \to [0, 1]$ is monotone ascending, weakly p-computable on $[0, 1]$, and strongly increasing at x_0 on $[0, 1]$ by Observation 2. Further, since $f(x_0)$ is not p-random, it follows from Observation 1 that $g(x_0) = 2^{-m}(f(x_0) - \ell)$ is not p-random, either. Thus there is a conservative, p-computable martingale d that succeeds on $g(x_0)$. By Lemmata 5 and 6 (letting $0.s$ be x_0), the g-pullback d_g of d succeeds on x_0 and is p-computable. Thus x_0 is not p-random.

To prove Theorem 1, we first show that the monotonicity assumption in Lemma 1 is dispensible. We do this by tweaking a nonmonotone function into a monotone one with the same desirable properties.

Lemma 7. *Let $f : [0, 1] \to \mathbb{R}$ be weakly p-computable on $[0, 1]$. Suppose that there exists $x_0 \in [0, 1]$ such that f strongly increases at x_0 on $[0, 1]$. Then there exists a monotone ascending function $g : [0, 1] \to \mathbb{R}$ that is weakly p-computable on $[0, 1]$, strongly increases at x_0 on $[0, 1]$, and satisfies $g(x_0) = f(x_0)$.*

Proof. See [5].

Proof (of Theorem 1). Let I, f, and r be as in the statement of the theorem. We can assume that f strongly increases at x, for otherwise we apply the foregoing argument to $-f$, using Observations 1 and 2 to get that $f(r)$ is p-random. We can choose some dyadic interval $\Gamma_w = [0.w, 0.w + 2^{-|w|}] \subseteq I$ containing r on which f is weakly p-computable and strongly increases at x. For all $x \in [0, 1]$, define

$$g(x) := f(0.w + 2^{-|w|}x) .$$

By Observation 2, g is weakly p-computable on $[0, 1]$ and strongly increases at the point $s := 2^{|w|}(r - 0.w)$ on $[0, 1]$. By Lemma 7, there is a monotone ascending function h that is weakly p-computable on $[0, 1]$, is strongly increasing at s on $[0, 1]$, and satisfies $h(s) = g(s)$. By Observation 1, s is p-random. By Lemma 1, $h(s)$ is p-random, and clearly, $h(s) = g(s) = f(r)$, which proves the theorem.

5 Some p-Randomness-Preserving Functions

Here is the class of functions we will consider:

Definition 9. *Let $I \subseteq \mathbb{R}$ be an open interval. A function $f : I \to \mathbb{R}$ is* well-behaved *on I if f is locally weakly p-computable and strongly varying at each of the p-random points in I.*

Theorem 1 gives us the following corollary:

Corollary 2. *If a function f is well-behaved on an interval I, then f preserves p-randomness, i.e., f maps p-random points in I to p-random points.*

A wide variety of functions are well-behaved and hence preserve p-randomness, including addition and multiplication by nonzero p-computable numbers, nonconstant polynomial and rational functions with p-computable coefficients, and all the familiar transcendental functions—exponential, logarithmic, trigonometric, etc. (Define a function to be 0 where it would otherwise be undefined.) Although these functions may not be strongly varying at all points, they are strongly varying at all p-random points.

Definition 10. *A sequence $c_0, c_1, c_2, \ldots \in \mathbb{R}$ is* uniformly p-computable *if there exists a polynomial-time function $\hat{c} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Q}$ such that for all $n, r \in \mathbb{N}$,*

$$|\hat{c}(0^n, 0^r) - c_n| \leq 2^{-r} .$$

Definition 11. *Let $I \subseteq \mathbb{R}$ be an open interval. We say that a function $f : I \to \mathbb{R}$ is* p-analytic on I *if there exists a p-computable point $x_0 \in I$ and a uniformly p-computable sequence $c_0, c_1, c_2, \ldots$ such that for all $x \in I$,*

$$f(x) = \sum_{n=0}^{\infty} c_n (x - x_0)^n ,$$

and the power series on the right converges absolutely for all $x \in I$.

Note that if f is p-analytic on I, then f is C^1 on I. In this section we prove the following theorem:

Theorem 2. *Let $I \subseteq \mathbb{R}$ be an open interval. If $f : I \to \mathbb{R}$ is nonconstant and p-analytic on I, then f is well-behaved on I.*

Theorem 2 follows from the two lemmas below:

Lemma 8. *Let $J \subseteq \mathbb{R}$ be an open interval and let I be a dyadic interval such that $I \subseteq J$. If f is p-analytic on J, then f is weakly p-computable on I.*

Proof. See [5].

Lemma 9. *Suppose f is p-analytic and nonconstant in some open interval I. If $r \in I$ satisfies $f(r) = 0$, then r is p-computable.*

Proof. See [5].

Proof (of Theorem 2). We know already that, since f has a continuous derivative, it strongly varies at any point r such that $f'(r) \neq 0$ (hence if r is p-random then so is $f(r)$). If $f'(r) = 0$, then r is p-computable by Lemma 9, and thus not p-random.

Corollary 3. *Let r be p-random. Then so are e^r, $\sin r$, and $\cos r$. If $r > 0$, then $\ln r$ is p-random. If f is any fixed rational function whose numerator and denominator have p-computable coefficients, and f is defined at r, then $f(r)$ is p-random.*

Proof. All these functions are p-analytic in some neighborhood of any point in their domains.

6 Further Research

P-randomness-preserving functions are clearly closed under composition. Are well-behaved functions closed this way also? Is there a converse to Theorem 1? Even a partial converse? For example, consider the following:

Conjecture 1. If f is weakly p-computable and monotone in a neighborhood of $r \in \mathbb{R}$ but is not strongly varying at r, then $f(r)$ is not p-random.

As evidence for this conjecture, one can concoct monotone functions that deviate only slightly from strongly varying, but none of whose outputs are p-random. For example, one could have $f(0.\sigma) = 0.\tau$, where the sequence τ results from the sequence σ by inserting zeros very sparsely but infinitely often, in places that are easy for a martingale to find and bet on.

Acknowledgments. I would like to thank Jack Lutz for suggesting this problem and for many interesting and valuable discussions. The presentation was also helped significantly from an earlier draft by comments from anonymous referees. I would also like to thank Lance Fortnow for his guidance at a crucial point.

References

1. Ambos-Spies, K., Mayordomo, E.: Resource-bounded measure and randomness. In: Complexity, Logic and Recursion Theory. Lecture Notes in Pure and Applied Mathematics, pp. 1–47 (1997)
2. Doty, D., Lutz, J.H., Nandakumar, S.: Finite-state dimension and real arithmetic. Information and Computation 205, 1640–1651 (2007)
3. Downey, R., Hirschfeldt, D.R., Nies, A., Terwijn, S.A.: Calibrating randomness. Bulletin of Symbolic Logic 12(3), 411–491 (2006)
4. Downey, R.G., Hirschfeldt, D.: Algorithmic Randomness and Complexity. Springer, Heidelberg (2010)
5. Fenner, S.A.: Functions that preserve p-randomness. Full draft (2011), `http://www.cse.sc.edu/~fenner/papers/p-rand-func.pdf`
6. Lutz, J.H.: Category and measure in complexity classes. SIAM Journal on Computing 19(6), 1100–1131 (1990)
7. Lutz, J.H.: Almost everywhere high nonuniform complexity. Journal of Computer and System Sciences 44, 220–258 (1992)
8. Lutz, J.H.: The quantitative structure of exponential time. In: Hemaspaandra, L.A., Selman, A.L. (eds.) Complexity Theory Retrospective II, pp. 225–260. Springer, Heidelberg (1997)
9. Lutz, J.H., Mayordomo, E.: Dimensions of points in self-similar fractals. SIAM Journal on Computing 38, 1080–1112 (2008)
10. Lutz, J.H., Weihrauch, K.: Connectivity properties of dimension level sets. Mathematical Logic Quarterly 54, 483–491 (2008)
11. Martin-Löf, P.: On the definition of infinite random sequences. Information and Control 9, 602–619 (1966)
12. Wall, D.D.: Normal Numbers. Ph.D. thesis, University of California, Berkeley, California, USA (1949)
13. Wang, Y.: Resource bounded randomness and computational complexity. Theoretical Computer Science 237, 33–55 (2000)

Reactive Turing Machines

Jos C.M. Baeten, Bas Luttik, and Paul van Tilburg

Eindhoven University of Technology
{j.c.m.baeten,s.p.luttik,p.j.a.v.tilburg}@tue.nl

Abstract. We propose reactive Turing machines (RTMs), extending classical Turing machines with a process-theoretical notion of interaction. We show that every effective transition system is simulated modulo branching bisimilarity by an RTM, and that every computable transition system with a bounded branching degree is simulated modulo divergence-preserving branching bisimilarity. We conclude from these results that the parallel composition of (communicating) RTMs can be simulated by a single RTM. We prove that there exist universal RTMs modulo branching bisimilarity, but these essentially employ divergence to be able to simulate an RTM of arbitrary branching degree. We also prove that modulo divergence-preserving branching bisimilarity there are RTMs that are universal up to their own branching degree. Finally, we establish a correspondence between RTMs and the process theory TCP_τ.

1 Introduction

The Turing machine [19] is widely accepted as a computational model suitable for exploring the theoretical boundaries of computing. Motivated by the existence of universal Turing machines, many textbooks on the theory of computation present the Turing machine not just as a theoretical model to explain which functions are computable, but as an accurate conceptual model of the computer. There is, however, a well-known limitation to this view. A Turing machine operates from the assumptions that: (1) all input it needs for the computation is available on the tape from the very beginning; (2) it performs a terminating computation; and (3) it leaves the output on the tape at the very end. Thus, the notion of Turing machine abstracts from two key ingredients of computing: *interaction* and *non-termination*. Nowadays, most computing systems are so-called *reactive systems* [13], systems that are generally not meant to terminate and consist of computing devices that interact with each other and with their environment.

Concurrency theory emerged from the early work of Petri [16] and has now developed into a mature theory of reactive systems. We mention three of its contributions particularly relevant for our work. Firstly, it installed the notion of transition system as the prime mathematical model to represent discrete behaviour. Secondly, it offered the insight that language equivalence is too coarse in a setting with interacting automata; instead one should consider automata up to some form of bisimilarity. Thirdly, it yielded many algebraic process calculi facilitating the formal specification and verification of reactive systems.

In this paper we propose a notion of *reactive* Turing machine (RTM), extending the classical notion of Turing machine with interaction in the style of concurrency theory. The extension consists of a facility to declare every transition to be either *observable*, by labelling it with an action symbol, or unobservable, by labelling it with τ. Typically, a transition labelled with an action symbol models an interaction of the RTM with its environment (or some other RTM), while a transition labelled with τ refers to an internal computation step. Thus, a conventional Turing machine can be regarded as a special kind of RTM in which all transitions are declared unobservable by labelling them with τ.

The semantic object associated with a conventional Turing machine is either the function that it computes, or the formal language that it accepts. The semantic object associated with an RTM is a behaviour, formally represented by a transition system. A function is said to be effectively computable if it can be computed by a Turing machine. By analogy, we say that a behaviour is effectively executable if it can be exhibited by an RTM. In concurrency theory, behaviours are usually considered modulo a suitable behavioural equivalence. In this paper we shall mainly use *(divergence-preserving) branching bisimilarity* [11], which is the finest behavioural equivalence in Van Glabbeek's spectrum (see [9]).

In Sect. 3 we set out to investigate the expressiveness of RTMs up to divergence-preserving branching bisimilarity. We establish that every computable transition system with a bounded branching degree can be simulated, up to divergence-preserving branching bisimilarity, by an RTM. If the divergence-preservation requirement is dropped, even every effective transition system can be simulated. These results will then allow us to conclude that the behaviour of a parallel composition of RTMs can be simulated on a single RTM.

In Sect. 4 we define a suitable notion of universality for RTMs and investigate the existence of universal RTMs. We shall find that there are some subtleties pertaining to the branching degree bound associated with each RTM. Up to divergence-preserving branching bisimilarity, an RTM can at best simulate other RTMs with the same or a lower bound on their branching degree. If divergence-preservation is not required, however, then universal RTMs do exist.

In Sect. 5, we consider the correspondence between RTMs and the process theory TCP_τ. We establish that every executable behaviour is, again up to divergence-preserving branching bisimilarity, definable by a finite recursive TCP_τ-specification [1]. Recursive specifications are the concurrency-theoretic counterparts of grammars in the theory of formal languages. Thus, the result in Sect. 5 may be considered as the process-theoretic version of the correspondence between Turing machines and unrestricted grammars.

Several extensions of Turing machines with some form of interaction have been proposed in the literature, already by Turing in [20], and more recently in [6,12,21]. The goal in these works is mainly to investigate to what extent interaction may have a beneficial effect on the power of sequential computation. The focus remains on the computational aspect, and interaction is not treated as a first-class citizen. Our goal, instead, is to achieve integration of automata and concurrency theory that treats computation and interactivity on equal footing.

2 Reactive Turing Machines

We fix a finite set $\mathcal{A}$ of *action symbols* that we shall use to denote the observable events of a system. An unobservable event will be denoted with τ, assuming that $\tau \notin \mathcal{A}$; we shall henceforth denote the set $\mathcal{A} \cup \{\tau\}$ by $\mathcal{A}_\tau$. We also fix a finite set $\mathcal{D}$ of *data symbols*. We add to $\mathcal{D}$ a special symbol $\square$ to denote a blank tape cell, assuming that $\square \notin \mathcal{D}$; we denote the set $\mathcal{D} \cup \{\square\}$ of *tape symbols* by $\mathcal{D}_\square$.

Definition 1. *A* reactive Turing machine *(RTM) $\mathcal{M}$ is a quadruple $(\mathcal{S}, \rightarrow, \uparrow, \downarrow)$ consisting of a finite set of* states $\mathcal{S}$*, a distinguished* initial state $\uparrow \in \mathcal{S}$*, a subset of* final states $\downarrow \subseteq \mathcal{S}$*, and a $(\mathcal{D}_\square \times \mathcal{A}_\tau \times \mathcal{D}_\square \times \{L, R\})$-labelled transition relation*

$$\rightarrow \subseteq \mathcal{S} \times \mathcal{D}_\square \times \mathcal{A}_\tau \times \mathcal{D}_\square \times \{L, R\} \times \mathcal{S} \ .$$

An RTM is deterministic *if $(s, d, a, e_1, M_1, t_1) \in \rightarrow$ and $(s, d, a, e_2, M_2, t_2) \in \rightarrow$ implies that $e_1 = e_2$, $t_1 = t_2$ and $M_1 = M_2$ for all $s, t_1, t_2 \in \mathcal{S}$, $d, e_1, e_2 \in \mathcal{D}_\square$, $a \in \mathcal{A}_\tau$, and $M_1, M_2 \in \{L, R\}$, and, moreover, $(s, d, \tau, e_1, M_1, t_1) \in \rightarrow$ implies that there do not exist $a \neq \tau$, e_2, M_2, t_2 such that $(s, d, a, e_2, M_2, t_2) \in \rightarrow$*

If $(s, d, a, e, M, t) \in \rightarrow$, we write $s \xrightarrow{a[d/e]M} t$. The intuitive meaning of such a transition is that whenever $\mathcal{M}$ is in state s and d is the symbol currently read by the tape head, then it may execute the action a, write symbol e on the tape (replacing d), move the read/write head one position to the left or one position to the right on the tape (depending on whether $M = L$ or $M = R$), and then end up in state t. RTMs extend conventional Turing machines by associating with every transition an element $a \in \mathcal{A}_\tau$. The symbols in $\mathcal{A}$ are thought of as denoting observable activities; a transition labelled with an action symbol in $\mathcal{A}$ will semantically be treated as observable. Observable transitions are used to model interactions of an RTM with its environment or some other RTM, as will be explained more in detail below when we introduce a notion of parallel composition for RTMs. The symbol τ is used to declare that a transition is unobservable. A classical Turing machine is an RTM in which all transitions are declared unobservable.

Example 1. Assume that $\mathcal{A} = \{c!d, c?d \mid c \in \{i, o\} \ \&\ d \in \mathcal{D}_\square\}$. Intuitively, i and o are the input/output communication channels by which the RTM can interact with its environment. The action symbol $c!d$ ($c \in \{i, o\}$) then denotes the event that a datum d is sent by the RTM along channel c, and the action symbol $c?d$ ($c \in \{i, o\}$) denotes the event that a datum d is received by the RTM along channel c.

The left state-transition diagram in Fig. 1 specifies an RTM that first inputs a string, consisting of an arbitrary number of 1s followed by the symbol #, stores the string on the tape, and returns to the beginning of the string. Then, it performs a computation to determine if the number of 1s is odd or even. In the first case, it simply removes the string from the tape and returns to the initial state. In the second case, it outputs the entire string, removes it from the tape, and returns to the initial state. The right state-transition diagram in Fig. 1 outputs on channel i the infinite sequence $1\#11\#111\#\ldots\#1^n\#\ldots$ $(n \geq 1)$.

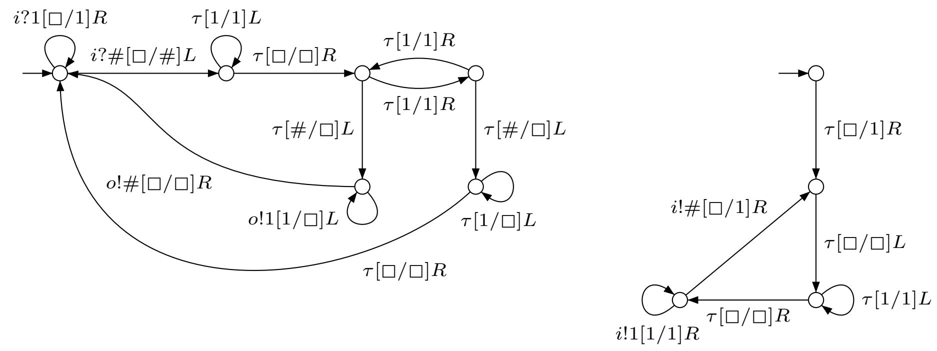


Fig. 1. Examples of reactive Turing machines

To formalise our intuitive understanding of the operational behaviour of RTMs we shall below associate with every RTM a transition system. An $\mathcal{A}_\tau$*-labelled transition system* T is a quadruple $(\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$ consisting of a set of *states* $\mathcal{S}$, an *initial* state $\uparrow \in \mathcal{S}$, a subset $\downarrow \subseteq \mathcal{S}$ of *final* states, and an $\mathcal{A}_\tau$-labelled *transition relation* $\longrightarrow \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathcal{S}$. If $(s, a, t) \in \longrightarrow$, we write $s \xrightarrow{a} t$. If s is a final state, i.e., $s \in \downarrow$, we write $s\downarrow$. The transition system T is *deterministic* if, for every state $s \in \mathcal{S}$ and for every $a \in \mathcal{A}_\tau$, $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ implies $s_1 = s_2$, and, moreover, $s \xrightarrow{\tau} s_1$ implies that there do not exist an action $a \neq \tau$ and a state s_2 such that $s \xrightarrow{a} s_2$.

With every RTM $\mathcal{M}$ we are going to associate a transition system $\mathcal{T}(\mathcal{M})$. The states of $\mathcal{T}(\mathcal{M})$ are the configurations of the RTM, consisting of a state of the RTM, its tape contents, and the position of the read/write head on the tape. We represent the tape contents by an element of $(\mathcal{D}_\Box)^*$, replacing precisely one occurrence of a tape symbol d by a *marked* symbol $\check{d}$, indicating that the read/write head is on this symbol. We denote by $\check{\mathcal{D}}_\Box = \{\check{d} \mid d \in \mathcal{D}_\Box\}$ the set of *marked* tape symbols; a *tape instance* is a sequence $\delta \in (\mathcal{D}_\Box \cup \check{\mathcal{D}}_\Box)^*$ such that δ contains exactly one element of $\check{\mathcal{D}}_\Box$. Formally, a *configuration* is now a pair (s, δ) consisting of a state $s \in \mathcal{S}$, and a tape instance δ.

Our transition system semantics defines an $\mathcal{A}_\tau$-labelled transition relation on configurations such that an RTM-transition $s \xrightarrow{a[d/e]M} t$ corresponds with a-labelled transitions from configurations consisting of the RTM-state s and a tape instance in which some occurrence of d is marked. The transitions lead to configurations consisting of t and a tape instance in which the marked symbol d is replaced by e, and either the symbol to the left or to right of this occurrence of e is replaced by its marked version, according to whether $M = L$ or $M = R$. If e happens to be the first symbol and $M = L$, or the last symbol and $M = R$, then an additional blank symbol is appended at the left or right end of the tape instance, respectively, to model the movement of the head.

We introduce some notation to concisely denote the new placement of the tape head marker. Let δ be an element of $\mathcal{D}_\Box^*$. Then by $\delta^<$ we denote the element of $(\mathcal{D}_\Box \cup \check{\mathcal{D}}_\Box)^*$ obtained by placing the tape head marker on the right-most symbol of δ if it exists, and $\check{\Box}$ otherwise. Similarly, by ${}^>\delta$ we denote the element of $(\mathcal{D}_\Box \cup \check{\mathcal{D}_\Box})^*$ obtained by placing the tape head marker on the left-most symbol of δ if it exists, and $\check{\Box}$ otherwise.

Definition 2. *Let $\mathcal{M} = (\mathcal{S}, \rightarrow, \uparrow, \downarrow)$ be an RTM. The* transition system $\mathcal{T}(\mathcal{M})$ associated with $\mathcal{M}$ *is defined as follows:*

1. *its set of states is the set of all configurations of $\mathcal{M}$;*
2. *its transition relation $\rightarrow$ is the least relation satisfying, for all $a \in \mathcal{A}_\tau$, $d, e \in \mathcal{D}_\Box$ and $\delta_L, \delta_R \in \mathcal{D}_\Box^*$:*

$$(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta_L{}^{<} e \delta_R) \text{ iff } s \xrightarrow{a[d/e]L} t \text{ , and}$$
$$(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta_L e\, {}^{>}\delta_R) \text{ iff } s \xrightarrow{a[d/e]R} t \text{ ;}$$

3. *its initial state is the configuration $(\uparrow, \check{\Box})$; and*
4. *its set of final states is the set of terminating configurations $\{(s, \delta) \mid s\downarrow\}$.*

Turing introduced his machines to define the notion of *effectively computable function.* By analogy, our notion of RTM can be used to define a notion of *effectively executable behaviour.*

Definition 3. *A transition system is* executable *if it is associated with an RTM.*

Parallel composition. To illustrate how RTMs are suitable to model a form of interaction, we shall now define on RTMs a notion of parallel composition, equipped with a simple form of communication. Let $\mathcal{C}$ be a finite set of *channels* for the communication of data symbols between one RTM and another. Intuitively, $c!d$ stands for the action of sending datum d along channel c, while $c?d$ stands for the action of receiving datum d along channel c.

First, we define a notion of parallel composition on transition systems. Let $T_1 = (\mathcal{S}_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ and $T_2 = (\mathcal{S}_2, \rightarrow_2, \uparrow_2, \downarrow_2)$ be transition systems, and let $\mathcal{C}' \subseteq \mathcal{C}$. The *parallel composition* of T_1 and T_2 is the transition system $[T_1 \parallel T_2]_{\mathcal{C}'} = (\mathcal{S}, \rightarrow, \uparrow, \downarrow)$, with $\mathcal{S}$, $\rightarrow$, $\uparrow$ and $\downarrow$ defined by

1. $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$;
2. $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ iff $a \in \mathcal{A}_\tau - \{c!d, c?d \mid c \in \mathcal{C}', d \in \mathcal{D}_\Box\}$ and either
   (a) $s_1 \xrightarrow{a} s_1'$ and $s_2 = s_2'$, or $s_2 \xrightarrow{a} s_2'$ and $s_1 = s_1'$, or
   (b) $a = \tau$ and either $s_1 \xrightarrow{c!d} s_1'$ and $s_2 \xrightarrow{c?d} s_2'$, or $s_1 \xrightarrow{c?d} s_1'$ and $s_2 \xrightarrow{c!d} s_2'$ for some $c \in \mathcal{C}'$ and $d \in \mathcal{D}_\Box$;
3. $\uparrow = (\uparrow_1, \uparrow_2)$; and
4. $\downarrow = \{(s_1, s_2) \mid s_1 \in \downarrow_1 \;\&\; s_2 \in \downarrow_2\}$.

Definition 4. *Let $\mathcal{M}_1 = (\mathcal{S}_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ and $\mathcal{M}_2 = (\mathcal{S}_2, \rightarrow_2, \uparrow_2, \downarrow_2)$ be RTMs, and let $\mathcal{C}' \subseteq \mathcal{C}$; by $[\mathcal{M}_1 \parallel \mathcal{M}_2]_{\mathcal{C}'}$ we denote the* parallel composition *of $\mathcal{M}_1$ and $\mathcal{M}_2$. The transition system $\mathcal{T}([\mathcal{M}_1 \parallel \mathcal{M}_2]_{\mathcal{C}'})$ associated with the parallel composition $[\mathcal{M}_1 \parallel_{\mathcal{C}} \mathcal{M}_2]_{\mathcal{C}'}$ of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the parallel composition of the transition systems associated with $\mathcal{M}_1$ and $\mathcal{M}_2$, i.e., $\mathcal{T}([\mathcal{M}_1 \parallel \mathcal{M}_2]_{\mathcal{C}'}) = [\mathcal{T}(\mathcal{M}_1) \parallel \mathcal{T}(\mathcal{M}_2)]_{\mathcal{C}'}$.*

Example 2. Let $\mathcal{A}$ be as in Example 1, let $\mathcal{M}$ denote the left-hand side RTM in Fig. 1, and let $\mathcal{E}$ denote the right-hand side RTM in Fig. 1. Then the parallel composition $[\mathcal{M} \parallel \mathcal{E}]_i$ exhibits the behaviour of outputting, along channel o, the string $11\#1111\#\cdots\#1^n\#$ ($n \geq 2$, n even).

Equivalences. In automata theory, Turing machines that compute the same function or accept the same language are generally considered equivalent. In fact, functional or language equivalence is underlying many of the standard notions and results in automata theory. Perhaps most notably, a *universal* Turing machine is a Turing machine that, when started with the code of some Turing machine on its tape, simulates this machine up to functional or language equivalence. A result from concurrency theory is that functional and language equivalence are arguably too coarse for reactive systems, because they abstract from all moments of choice (see, e.g., [1]). In concurrency theory many alternative behavioural equivalences have been proposed; we refer to [9] for a classification.

The results about RTMs that are obtained in the remainder of this paper are modulo *branching bisimilarity* [11]. We shall consider both the divergence-insensitive and the divergence-preserving variant. Let T_1 and T_2 be transition systems. If T_1 and T_2 are *branching bisimilar*, then we write $T_1 \leftrightarroweq_{\mathrm{b}} T_2$. If T_1 and T_2 are *divergence-preserving branching bisimilar*, then we write $T_1 \leftrightarroweq_{\mathrm{b}}^{\Delta} T_2$. (Due to space limitations, the formal definitions had to be omitted; the reader is referred to the full version [4], or to [10], where the divergence-preserving variant is called *branching bisimilarity with explicit divergence*.)

3 Expressiveness of RTMs

We shall establish in this section that every effective transition system can be simulated by an RTM up to branching bisimilarity, and that every boundedly branching computable transition system can be simulated up to divergence-preserving branching bisimilarity. We use this as an auxiliary result to establish that a parallel composition of RTMs can be simulated by a single RTM.

Let $T = (\mathcal{S}, \rightarrow, \uparrow, \downarrow)$ be a transition system; the mapping $out : \mathcal{S} \rightarrow 2^{\mathcal{A}_\tau \times \mathcal{S}}$ associates with every state its set of outgoing transitions, i.e., for all $s \in \mathcal{S}$, $out(s) = \{(a, t) \mid s \xrightarrow{a} t\}$, and $fin(\_)$ denotes the characteristic function of $\downarrow$.

Definition 5. *Let $T = (\mathcal{S}, \rightarrow, \uparrow, \downarrow)$ be an $\mathcal{A}_\tau$-labelled transition system. We say that T is* effective *if there exist suitable codings of $\mathcal{A}_\tau$ and $\mathcal{S}$ (into the natural numbers) such that $\rightarrow$ and $\downarrow$ are recursively enumerable. We say that T is* computable *if there exist suitable codings of $\mathcal{A}_\tau$ and $\mathcal{S}$ such that the functions $out(\_)$ and $fin(\_)$ are recursive.*

The notion of effective transition system originates with Boudol [7]. For the notion of computable transition system we adopt the definition from [2]. If $\rightarrow$ and $\downarrow$ are recursively enumerable, then there exist algorithms that enumerate the transitions in $\rightarrow$ and the states in $\downarrow$. If the functions $out(\_)$ and $fin(\_)$ are recursive, then there exists an algorithm that, given a state s, yields the list of outgoing transitions of s and determines if $s \in \downarrow$.

Proposition 1. *The transition system associated with an RTM is computable.*

Hence, unsurpisingly, if a transition system is not computable, then it is not executable either. It is easy to define transition systems that are not computable, so there exist behaviours that are not executable. The full version of this paper [4] contains an example that illustrates that there exist behaviours that are not even executable up to branching bisimilarity.

Phillips associates, in [17], with every effective transition system a *branching bisimilar* computable transition system of which, moreover, every state has a branching degree of at most 2. (Phillips actually establishes weak bisimilarity, but it is easy to see that branching bisimilarity holds.)

Definition 6. *Let $T = (\mathcal{S}, \rightarrow, \uparrow, \downarrow)$ be a transition system, and let B be a natural number. We say that T has a branching degree* bounded by B *if, for every state $s \in \mathcal{S}$, $|out(s)| \leq B$. We say that T is* boundedly branching *if there exists $B \in \mathbb{N}$ such that the branching degree of T is bounded by B.*

Proposition 2 (Phillips). *For every effective transition system T there exists a boundedly branching computable transition system T' such that $T \leftrightarroweq_b T'$.*

A crucial insight in Phillips' proof is that a divergence (i.e., an infinite sequence of τ-transitions) can be exploited to simulate a state of which the set of outgoing transitions is recursively enumerable, but not recursive. The following example, inspired by [8], shows that introducing divergence is unavoidable.

Example 3. (In this and later examples, we denote by φ_x the partial recursive function with index $x \in \mathbb{N}$ in some exhaustive enumeration of partial recursive functions, see, e.g., [18].) Assume that $\mathcal{A} = \{a, b\}$, and consider the transition system $T_1 = (\mathcal{S}_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ with $\mathcal{S}_1$, $\rightarrow_1$, $\uparrow_1$ and $\downarrow_1$ defined by

$$\begin{aligned}
&\mathcal{S}_1 = \{s_{1,x},\ t_{1,x} \mid x \in \mathbb{N}\}\ , \quad \uparrow_1 = s_{1,0}\ ,\\
&\rightarrow_1 = \{(s_{1,x}, a, s_{1,x+1}) \mid x \in \mathbb{N}\} \cup \{(s_{1,x}, b, t_{1,x}) \mid x \in \mathbb{N}\}\ , \text{ and}\\
&\downarrow_1 = \{t_{1,x} \mid \varphi_x(x) \text{ converges}\}\ .
\end{aligned}$$

If T_2 is a transition system such that $T_1 \leftrightarroweq_{\mathrm{b}}^{\Delta} T_2$, as witnessed by some divergence-preserving branching bisimulation relation $\mathcal{R}$, then it can be argued that T_2 is not computable. A detailed argument can be found in the full version [4].

By Proposition 2, in order to prove that every effective transition system can be simulated up to branching bisimilarity by an RTM, it suffices to prove that every boundedly branching computable transition system can be simulated by an RTM. Let $T = (\mathcal{S}_T, \rightarrow_T, \uparrow_T, \downarrow_T)$ be a boundedly branching computable transition system, say with branching degree bounded by B. It is reasonably straightforward to construct an RTM $\mathcal{M} = (\mathcal{S}_\mathcal{M}, \rightarrow_\mathcal{M}, \uparrow_\mathcal{M}, \downarrow_\mathcal{M})$, which we call the *simulator* for T, such that $\mathcal{T}(\mathcal{M}) \leftrightarroweq_{\mathrm{b}}^{\Delta} T$. The construction is detailed in [4].

Theorem 1. *For every boundedly branching computable transition system T there exists an RTM $\mathcal{M}$ such that $\mathcal{T}(\mathcal{M}) \leftrightarroweq_b^{\Delta} T$.*

Combining Theorem 1 with Proposition 2 we can conclude that RTMs can simulate effective transition systems up to branching bisimilarity, but, in view of Example 3, not in a divergence-preserving manner.

Corollary 1. *For every effective transition system T there exists a reactive Turing machine $\mathcal{M}$ such that $\mathcal{T}(\mathcal{M}) \leftrightarrows_b T$.*

All computations involved in the simulation of T are deterministic; if $\mathcal{M}$ is non-deterministic, then this is due to a state of which the menu includes some action a more than once. Clearly, if T is deterministic, then, for every state s in T, $|out(s)| \leq |\mathcal{A}_\tau|$. So a deterministic transition system is boundedly branching, and therefore we get the following corollary to Theorem 1.

Corollary 2. *For every deterministic computable transition system T there exists a deterministic RTM $\mathcal{M}$ such that $\mathcal{T}(\mathcal{M}) \leftrightarrows_b^\Delta T$.*

Using Theorem 1 we can now also establish that a parallel composition of RTMs can be simulated, up to divergence-preserving branching bisimilarity, by a single RTM. To this end, note that the transition systems associated with RTMs are boundedly branching and computable. Further note that the parallel composition of boundedly branching computable transition systems is again computable. It follows that the transition system associated with a parallel composition of RTMs is boundedly branching and computable, and hence, by Theorem 1, there exists an RTM that simulates this transition system up to divergence-preserving branching bisimilarity. Thus we get the following corollary.

Corollary 3. *For every pair of RTMs $\mathcal{M}_1$ and $\mathcal{M}_2$ and for every set of communication channels $\mathcal{C}$ there is an RTM $\mathcal{M}$ such that $\mathcal{T}(\mathcal{M}) \leftrightarrows_b^\Delta \mathcal{T}([\mathcal{M}_1 \parallel \mathcal{M}_2]_\mathcal{C})$.*

4 Universality

Recall that a *universal Turing machine* is some Turing machine that can simulate an arbitrary Turing machine on arbitrary input. The assumptions are that both the finite description of the to be simulated Turing machine and its input are available on the tape of the universal Turing machine, and the simulation is up to functional or language equivalence. We adapt this scheme in two ways. Firstly, we let the simulation start by inputting the description of an arbitrary RTM $\mathcal{M}$ along some dedicated channel u, rather than assuming its presence on the tape right from the start. This is both conceptually desirable —for our aim is to give interaction a formal status—, and technically necessary —for in the semantics of RTMs we have assumed that the tape is initially empty. Secondly, we require the behaviour of $\mathcal{M}$ to be simulated up to divergence-preserving branching bisimilarity.

Thus, we arrive at the following tentative definitions. For an arbitrary RTM $\mathcal{M}$, denote by $\overline{\mathcal{M}}$ an RTM that outputs a description of $\mathcal{M}$ along channel u and then terminates. A *universal* RTM is then an RTM $\mathcal{U}$ such that, for every RTM $\mathcal{M}$, the parallel composition $\left[\mathcal{U} \parallel \overline{\mathcal{M}}\right]_{\{u\}}$ simulates $\mathcal{T}(\mathcal{M})$.

Although such a universal RTM $\mathcal{U}$ exists up to branching bisimilarity as we shall see below, it does not exist up to divergence-preserving branching bisimilarity. To see this, note that the transition system associated with any particular RTM $\mathcal{U}$ has a branching degree that is bounded by some natural number B. It can then be established that, up to divergence-preserving branching bisimilarity, $\mathcal{U}$ can only simulate RTMs with a branching degree bounded by B. The argument is formalised in the following proposition; see [4] for a proof.

Proposition 3. *There does not exist an RTM $\mathcal{U}$ such that for all RTM $\mathcal{M}$ it holds that $\left[\mathcal{U} \parallel \overline{\mathcal{M}}\right]_{\{u\}} \mathrel{\underline{\leftrightarrow}}_b^{\Delta} \mathcal{T}(\mathcal{M})$.*

If we insist on simulation up to divergence-preserving branching bisimilarity, then we need to relax the notion of universality. Let B be a natural number. An RTM $\mathcal{U}_B$ is *universal up to B* if for every RTM $\mathcal{M}$ with $\mathcal{T}(\mathcal{M})$ bounded by branching degree B it holds that $\mathcal{T}(\mathcal{M}) \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}}^{\Delta} \left[\overline{\mathcal{M}} \parallel \mathcal{U}_B\right]_{\{u\}}$.

The construction of the simulator for a transition system of which the branching degree is bounded by B in the proof of Theorem 1 can be adapted to get the definition of an RTM $\mathcal{U}_B$ that is universal up to B. It suffices to slightly modify the initialisation fragment. Instead of writing the codes of the functions $out(\_)$ and $fin(\_)$ and the initial state directly on the tape, the *initialisation fragment* receives the code $\ulcorner\mathcal{M}\urcorner$ of an arbitrary $\mathcal{M}$ along some dedicated channel u. Then, it recursively computes the codes of the functions $out(\_)$ and $fin(\_)$, and the initial state of $\mathcal{T}(\mathcal{M})$ and stores these on the tape.

Theorem 2. *For every B there exists an RTM $\mathcal{U}_B$ such that, for all RTMs $\mathcal{M}$ with a branching degree bounded by B, it holds that $\mathcal{T}(\mathcal{M}) \mathrel{\underline{\leftrightarrow}}_b^{\Delta} \left[\overline{\mathcal{M}} \parallel \mathcal{U}_B\right]_{\{u\}}$.*

If we drop divergence-preservation as a requirement for the simulation, then a universal RTM does exist. At the heart of the argument is a trick, first described in [2] and adapted by Phillips in [17], to use a divergence with (infinitely many) states of at most a branching degree of 2 to simulate, up to branching bisimilarity, a single state of some arbitrary (even countably infinite) branching degree.

Theorem 3. *There exists an RTM $\mathcal{U}$ such that, for all RTMs $\mathcal{M}$, it holds that $\mathcal{T}(\mathcal{M}) \mathrel{\underline{\leftrightarrow}}_b \left[\mathcal{U} \parallel \overline{\mathcal{M}}\right]_{\{u\}}$.*

5 Recursive Specifications

A well-known result from the theory of automata and formal languages is that the formal languages accepted by Turing machines correspond with the languages generated by an unrestricted grammar. A *grammar* is a formal system for describing a formal language. The corresponding notion in concurrency theory is the notion of *recursive specification*, which is a formal system for describing behaviour. In this section, we show that the behaviours of RTMs correspond with the behaviours described by so-called TCP$_\tau$ recursive specifications. The process theory TCP$_\tau$ is a general theory for describing behaviour, encompassing the key features of the well-known process theories ACP$_\tau$ [5], CCS [15] and CSP [14].

We shall briefly introduce the syntax of TCP$_\tau$ and informally describe its operational semantics. We refer to the textbook [1] for an elaborate treatment. We reuse the finite set $\mathcal{C}$ of channels and set of data $\mathcal{D}_\Box$ introduced in Sect. 2; we introduce the set of special actions $\mathcal{I} = \{c?d, c!d \mid d \in \mathcal{D}_\Box, c \in \mathcal{C}\}$. The actions $c?d$ and $c!d$ denote the events that a datum d is received or sent along *channel* c. Let $\mathcal{N}$ be a countably infinite set of names. The set of *process expressions* $\mathcal{P}$ is generated by the following grammar ($a \in \mathcal{A}_\tau \cup \mathcal{I}, N \in \mathcal{N}, \mathcal{C}' \subseteq \mathcal{C}$):

$$p \ ::= \ \mathbf{0} \mid \mathbf{1} \mid a.p \mid p \cdot p \mid p + p \mid [p \parallel p]_{\mathcal{C}'} \mid N \ .$$

The constant $\mathbf{0}$ denotes *deadlock*, the unsuccessfully terminated process. The constant $\mathbf{1}$ denotes *skip*, the successfully terminated process. For each action $a \in \mathcal{A} \cup \mathcal{I}$ there is a unary operator $a.$ denoting action prefix; the process denoted by $a.p$ can do an a-transition to the process denoted by p. The binary operator $\cdot$ denotes *sequential composition*. The binary operator $+$ denotes *alternative composition* or *choice*. The binary operator $[\_ \parallel \_]_{\mathcal{C}'}$ denotes the special kind of *parallel composition* that we have also defined on RTMs. It enforces communication along the channels in $\mathcal{C}'$, and communication results in τ. (By including the restricted kind of parallel composition, we deviate from the definition of TCP$_\tau$ discussed in [1], but we note that our notion of parallel composition is definable with the operations $\parallel$, $\partial_\_(\_)$ and $\tau_\_(\_)$ of TCP$_\tau$ in [1].)

A *recursive specification* E is a set of equations of the form: $N \stackrel{\text{def}}{=} p$, with as left-hand side a name N and as right-hand side a TCP$_\tau$ process expression p. It is required that a recursive specification E contains, for every $N \in \mathcal{N}$, at most one equation with N as left-hand side; this equation will be referred to as the *defining equation* for N in $\mathcal{N}$. Furthermore, if some name occurs in the right-hand side of some defining equation, then the recursive specification must include a defining equation for it. Let E be a recursive specification, and let p be a process expression. There is a standard method to associate with p a transition system $\mathcal{T}_E(p)$. The details can be found, e.g., in [1].

In [4] we present the details of a construction that associates with an arbitrary RTM a TCP$_\tau$ recursive specification that defines its behaviour up to divergence-preserving branching bisimilarity. Thus, we get the following correspondence.

Theorem 4. *For every RTM $\mathcal{M}$ there exists a finite recursive specification E and a process expression p such that $\mathcal{T}(\mathcal{M}) \leftrightarrows_b^\Delta \mathcal{T}_E(p)$,*

As a corollary we find that every executable transition system is definable, up to divergence-preserving branching bisimilarity, by a TCP$_\tau$ recursive specification. Since there exist recursive specifications with an unboundedly branching associated transition system (see, e.g., [3], for the converse of the abovementioned theorem), we have to give up divergence-preservation. Since the transition system associated with a finite recursive specification is clearly effective, we do get, by Corollary 1, the following result.

Corollary 4. *For every finite recursive specification E and process expression p, there exists an RTM $\mathcal{M}$ such that $\mathcal{T}_E(p) \leftrightarrows_b \mathcal{T}(\mathcal{M})$.*

6 Concluding Remarks and Future Work

We have proposed a notion of reactive Turing machine and discussed its expressiveness in bisimulation semantics. Although it is not the aim of this work to contribute to the debate as to whether interactive computation is more powerful than traditional computation, our notion of RTM may nevertheless turn out to be a useful tool in the discussion. For instance, our result that the parallel composition of RTMs can be simulated by an RTM seems to contradict the implied conjecture in [12, Sect. 11] that interactive computation performed by multiple machines working together is more expressive than interactive computation performed by a single machine.

To be sure, however, we would need to firmly establish the robustness of our notion by showing that variations on its definition (e.g., multiple tracks or multiple tapes), and by showing that it can simulate the other proposals (persistent Turing machines [12], interactive Turing machines [21]). We also intend to consider interactive versions of other computational models. The λ-calculus would be an interesting candidate to consider, because of the well-known result that it is inherently sequential; this suggests that an interactive version of λ-calculus will be less expressive than RTMs. In particular, we conjecture that the evaluation of *parallel-or* or McCarthy's *amb* can be simulated with RTMs.

RTMs may also prove to be a useful tool in establishing the expressiveness of process theories. For instance, the transition system associated with a π-calculus expression is effective, so it can be simulated by an RTM, at least up to branching bisimilarity. The π-calculus can to some extent be seen as the interactive version of the λ-calculus. We conjecture that the converse —every executable transition system can be specified by a π-calculus expression— is also true.

Petri showed already in his thesis [16] that concurrency and interaction may serve to bridge the gap between the theoretically convenient Turing machine model of a sequential machine with unbounded memory, and the practically more realistic notion of extendable architecture of components with bounded memory. The specification we present in the proof of Theorem 4 (see [4]) is another illustration of this idea: the unbounded tape is modelled as an unbounded parallel composition. It would be interesting to further study the inherent tradeoff between unbounded parallel composition and unbounded memory in the context of RTMs, considering unbounded parallel compositions of RTMs with bounded memory.

Acknowledgement. We thank Herman Geuvers for discussions, and Clemens Grabmayer for suggesting to us the term *reactive Turing machine*, and the anonymous referees for their useful comments.

References

1. Baeten, J.C.M., Basten, T., Reniers, M.A.: Process Algebra (Equational Theories of Communicating Processes). Cambridge University Press, Cambridge (2009)
2. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.: On the consistency of Koomen's fair abstraction rule. Theoretical Computer Science 51, 129–176 (1987)

3. Baeten, J.C.M., Cuijpers, P.J.L., Luttik, B., van Tilburg, P.J.A.: A Process-Theoretic Look at Automata. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 1–33. Springer, Heidelberg (2010)
4. Baeten, J.C.M., Luttik, B., van Tilburg, P.J.A.: Reactive Turing machines. CoRR, abs/1104.1738 (2011)
5. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theoretical Computer Science 37, 77–121 (1985)
6. Blass, A., Gurevich, Y., Rosenzweig, D., Rossman, B.: Interactive small-step algorithms I: Axiomatization. Logical Methods in Computer Science 3(4) (2007)
7. Boudol, G.: Notes on algebraic calculi of processes. In: Apt, K.R. (ed.) Logics and Models of Concurrent Systems. NATO-ASI Series, vol. F13, pp. 261–303. Springer, Berlin (1985)
8. Darondeau, P.: Bisimulation and effectiveness. Inf. Process. Lett. 30(1), 19–20 (1989)
9. van Glabbeek, R.J.: The Linear Time – Branching Time Spectrum II. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993)
10. van Glabbeek, R.J., Luttik, B., Trcka, N.: Branching bisimilarity with explicit divergence. Fundam. Inform. 93(4), 371–392 (2009)
11. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. Journal of the ACM 43(3), 555–600 (1996)
12. Goldin, D.Q., Smolka, S.A., Attie, P.C., Sonderegger, E.L.: Turing machines, transition systems, and interaction. Inf. Comput. 194(2), 101–128 (2004)
13. Harel, D., Pnueli, A.: On the development of reactive systems. In: Apt, K.R. (ed.) Logics and Models of Concurrent Systems. NATO ASI Series, vol. F-13, pp. 477–498. Springer, New York (1985)
14. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
15. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
16. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2 (1962)
17. Phillips, I.C.C.: A note on expressiveness of process algebra. In: Burn, G.L., Gay, S., Ryan, M.D. (eds.) Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods, Workshops in Computing, pp. 260–264. Springer, Heidelberg (1993)
18. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Company, New York (1967); Reprinted by MIT Press (1987)
19. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 2(42), 230–265 (1936)
20. Turing, A.M.: Systems of logic based on ordinals. Proceedings of the London Mathematical Society 45(2), 161–228 (1939)
21. van Leeuwen, J., Wiedermann, J.: On algorithms and interaction. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 99–113. Springer, Heidelberg (2000)

Virtual Substitution for SMT-Solving

Florian Corzilius and Erika Ábrahám

RWTH Aachen University, Germany

Abstract. SMT-solving aims at deciding satisfiability for the existential fragment of a first-order theory. A SAT-solver handles the logical part of a given problem and invokes an embedded theory solver to check consistency of theory constraints. For efficiency, the theory solver should be able to work *incrementally* and generate *infeasible subsets*. Currently available decision procedures for *real algebra* – the first-order theory of the reals with addition and multiplication – do not exhibit these features. In this paper we present an adaptation of the virtual substitution method, providing these abilities.

1 Introduction

The *satisfiability problem* poses the question of whether a given logical formula is satisfiable, i.e., whether we can assign values to the variables contained in the formula such that the formula becomes true. The development of efficient algorithms and tools (*solvers*) for satisfiability checking form an active research area in computer science. A lot of effort has been put into the development of fast solvers for the propositional satisfiability problem, called SAT. To increase expressiveness, extensions of the propositional logic with respect to first-order *theories* can be considered. The corresponding satisfiability problems are called *SAT-modulo-theories* problems, short *SMT*. SMT-solvers exist, e.g., for equality logic, uninterpreted functions, predicate logic, and linear real arithmetic.

In contrast to the above-mentioned theories, less activity can be observed for SMT-solvers supporting the first-order theory of the real ordered field, what we call *real algebra*. Our research goal is to develop an SMT-solver for real algebra, being capable of solving Boolean combinations of polynomial constraints over the reals efficiently.

Even though decidability of real algebra has been known for a long time [Tar48], the first decision procedures were not yet practicable. Since 1974 it is known that the time complexity of deciding formulas of real algebra is in worst case doubly exponential in the number of variables (dimension) contained in the formula [DH88, Wei88].

Today, several methods are available which satisfy these complexity bounds, for example the cylindrical algebraic decomposition (CAD) [CJ98] , the Gröbner basis, and the virtual substitution method [Wei98]. An overview of these methods is given in [DSW97]. There are also tools available which implement these methods. The stand-alone application `QEPCAD` is a `C++` implementation of the CAD method [Bro03]. Another example is the REDLOG package [DS97] of the computer algebra system REDUCE based on Lisp, which offers an optimized combination of the virtual substitution, the CAD method, and real root counting.

Currently existing solvers are not suited to solve large formulas containing arbitrary combinations of real constraints. We want to combine the advantages of highly tuned SAT-solvers and the most efficient techniques currently available for solving conjunctions of real constraints, by implementing an SMT-solver for real algebra capable of efficiently solving arbitrary Boolean combinations of real constraints.

Theory solvers should satisfy specific requirements in order to embed them into an SMT-solver efficiently:

– *Incrementality:* The theory solver should be able to accept theory constraints one after the other. After it receives a new theory constraint it should check the conjunction of the new constraint with the earlier constraints for satisfiability. For efficiency it is important that the solver makes use of the result of earlier checks.
– *(Minimal) infeasible subsets:* If the theory solver detects a conflict, it should give a reason for the unsatisfiability. The usual way is to determine an unsatisfiable subset of the constraints which is ideally minimal in the sense that if we remove a constraint the remaining ones become satisfiable.
– *Backjumping:* If a conflict occurs, either in the Boolean or in the theory domain, the solver should be able to backtrack and continue the search for a solution at an earlier state, thereby reducing the search space as far as possible.

To our knowledge, these functionalities are currently not supported by the available solvers for real algebra. In this paper we extend the virtual substitution method to support incrementality, backjumping, and the generation of infeasible subsets.

We have chosen the virtual substitution method because it is a restricted but very efficient decision procedure for a subset of real algebra. The restriction concerns the degree of polynomials. The method uses solution equations to determine the zeros of (multivariate) polynomials in a given variable. As such solution equations exist for polynomials of degree at most 4, the method is a priori restricted in the degree of polynomials. In this paper we restrict ourselves to polynomials of degree 2, however it is possible to extend our approach to polynomials up to degree 4. Furthermore, we want to develop an incremental adaptation of the CAD method and to call this complete[1] but less efficient decision procedure in order to complete the incremental implementation of the virtual substitution method. This will be part of our future work.

Related work. The SMT-solvers Z3 [dMB08], HySAT [FHT+07] and ABsolver [BPT07] are able to handle nonlinear real-algebraic constraints. The algorithm implemented in HySAT and in its successor tool iSAT uses interval constraint

[1] The CAD method can handle full real algebra.

propagation to check real constraints. Though this technique is not complete, in practice it is more efficient than those based on exact methods [FHT+07]. The structures of `ABsolver` and `Z3` are more similar to our approach. However, `Z3` does not support full real-algebra. Papers on `ABsolver` do not address the issue of incrementality. Though `ABsolver` computes minimal infeasible subsets, we did not find any information on how they are generated. Other approaches as, e.g., implemented in the RAHD tool [PJ09], embed real-algebraic decision procedures in a theorem proving context. The virtual substitution was also used in its original form for temporal verification within the Stanford Temporal Solver [Bjo99]. The virtual substitution is used for linear arithmetic in [Bjo10].

The remaining part of the paper is structured as follows: We give an introduction to DPLL-based SMT-solving and to virtual substitution in Section 2. We introduce our incremental virtual substitution algorithm providing infeasible subsets and give some first experimental results in Section 3. We conclude the paper in Section 4. A more detailed description of our work can be found in [Cor11].

2 Preliminaries

In this paper we focus on satisfiability checking for a subset of the existential fragment of real algebra (quadratic and beyond). *Terms* or *polynomials* p, *constraints* c, and *formulas* φ can be built upon constants $0, 1$ and real-valued variables x according to the following abstract grammar:

$$\begin{array}{rcl} p & ::= & 0 \mid 1 \mid x \mid (p+p) \mid (p \cdot p) \\ c & ::= & p = p \mid p < p \\ \varphi & ::= & c \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\exists x \varphi) \end{array}$$

Syntactic sugar like *True* , *False* , $-$, $/$, $\vee$, $\rightarrow$, $\forall$, ... is defined as usual; the equality is added for convenience but could also be defined as syntactic sugar. We define that $\vee$ binds stronger than $\wedge$, and $\wedge$ binds stronger than $\exists$, and sometimes skip the parentheses. The semantics of real algebra is as expected. We call a variable x occurring in a formula $\exists x \varphi$ *bound*; not bound variables are called *free*. Formulas with no free variables are called *sentences*. With $\mathbb{R}[x_1, \ldots, x_n]$ we denote the set of all polynomials containing variables $x_1, \ldots, x_n$.

With the real numbers $\mathbb{R}$ as the domain, the set of all true real-algebraic sentences is the first-order theory of $(\mathbb{R}, +, \cdot, 0, 1, <)$, called *real algebra*. In this paper we restrict to the existential fragment, i.e., to formulas which can be transformed into the form $\exists x_1 \ldots \exists x_n \varphi$ with φ being quantifier-free.

The satisfiability problem for real-algebraic formulas is decidable as proved around the 1930s [Tar48]. We use DPLL-based SMT-solving, introduced in Section 2.1, for the satisfiability check. An SMT-solver combines a SAT-solver, handling the Boolean structure, and a theory solver to check the theory constraints. We apply the *virtual substitution method*, introduced in Section 3, as an algorithm for the theory solver, which is very efficient but restricted in the degree of polynomials that can be handled.

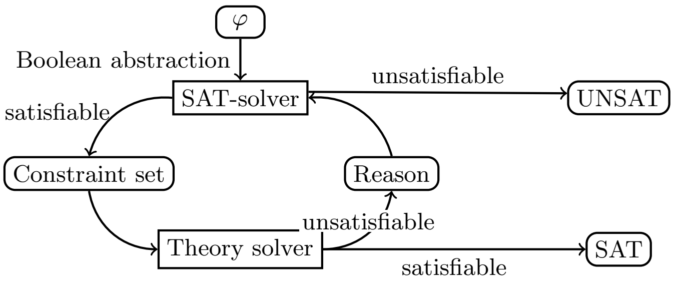


Fig. 1. The basic scheme of DPLL-based full lazy SMT-solving

2.1 Lazy SMT-Solving

The propositional satisfiability problem (SAT) where the variables range over the values 1 (*True*) and 0 (*False*) is NP-complete, but SAT-solvers are quite efficient in practice due to a vast progress in SAT-solving during recent years. One of the main achievements in the field of SAT-solving is the DPLL-algorithm, which is capable of solving existential Boolean formulas.

The DPLL-algorithm can be extended to handle more expressive logics. For this approach, which is called *SAT-modulo-theories* (*SMT*) solving, a SAT-solver gets combined with a decision procedure for the satisfiability check of constraints from the underlying theory (see, e.g., [KS08]).

The basic scheme of *full lazy* DPLL-based SMT-solving is roughly as follows (see Fig. 1). The SMT-solver first creates a Boolean skeleton of the input formula, replacing all theory constraints contained in the input formula by fresh Boolean variables. The resulting Boolean formula is passed to the SAT-solver, which searches for a satisfying assignment. If it does not succeed, the formula is unsatisfiable. Otherwise, the assignment found corresponds to certain truth values for the theory constraints and has to be verified by the theory solver. If the constraints are satisfiable, then the original formula is satisfiable. Otherwise, if the theory solver detects that the conjunction of the corresponding theory constraints is unsatisfiable, it then hands over a reason for the unsatisfiability, an *infeasible subset* of the theory constraints, to the SAT-solver. The SAT-solver uses this piece of information to exclude the detected conflict from further search. Afterwards, the SAT-solver computes an assignment for the refined Boolean problem again, which in turn has to be verified by the theory solver. Continuing this iteration decides the satisfiability of the input formula in the end.

The full lazy procedure is often disadvantageous in practice, because the SAT-solver may do a lot of needless work by extending an already (in the theory domain) contradictory partial assignment. *Less lazy* variants of the procedure call the theory solver more often, already handing over constraints corresponding to partial assignments. To do so efficiently, the theory solver should accept constraints in an *incremental* fashion, where computation results of previous steps can be reused.

Note that we strictly separate the satisfiability checks in the Boolean and in the theory domains, that means, we do not consider theory propagation embedded in the DPLL search like, e.g., Yices does.

2.2 Virtual Substitution

The virtual substitution method is a restricted but very efficient decision procedure for a subset of real algebra. In this paper we adapt it to support incrementality and infeasible subset generation (see Section 3). In this section we introduce virtual substitution in its original form.

We are interested in checking satisfiability of existentially quantified formulas in prenex normal form (PNF) $\exists x_1 \ldots \exists x_n \varphi$ with φ quantifier-free. The decision procedure based on virtual substitution produces a quantifier-free equivalent of a given input formula, by successively eliminating all bound variables starting with the innermost one. Below we explain how the innermost quantified variable is eliminated using virtual substitution.

Let $\exists y_1 \ldots \exists y_n \exists x \varphi$ be the input formula, where φ is a quantifier-free Boolean combination of polynomial constraints. In this paper we handle constraints, whose degree in x is at most two ($y_1, \ldots, y_n$ may occur with higher degree). Thus we assume that all polynomial constraints in φ are of the form $p \sim 0$, $\sim \in \{=, <, >, \leq, \geq, \neq\}$, where $p = ax^2 + bx + c$ is at most quadratic[2] in x.

Considering the problems domain, each constraint containing x splits it into values which satisfy the constraint and values which do not. More precisely, the satisfying values can be merged to a finite number of intervals whose endpoints are elements of $\{\infty, -\infty\} \cup \mathbb{L}_x$, where $\mathbb{L}_x$ are the zeros of p in x:

Linear in x :	$x_0 = -\frac{c}{b}$	, if $a = 0 \ \wedge \ b \neq 0$
Quadratic in x, first solution:	$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$	, if $a \neq 0 \ \wedge \ b^2 - 4ac \geq 0$
Quadratic in x, second solution:	$x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$	, if $a \neq 0 \ \wedge \ b^2 - 4ac \geq 0$

The conditions on the right are called *side conditions.* Note that in the case $a = b = c = 0$ (constant in x) the solution interval for x is $(-\infty, \infty)$, which does not have finite endpoints.

Assume that $\{c_1, \ldots, c_n\}$ is the set of constraints in φ that contain x. Each constraint c_i, $1 \leq i \leq n$, has a set of solution intervals $\{I_{i,1}, \ldots, I_{i,k_i}\}$ for x. If the constraints have a common solution for x, then for all $i \in \{1, \ldots, n\}$ there exists a $j_i \in \{1, \ldots, k_i\}$ with

$$I = \Big(\bigcap_{i \in \{1, \ldots, n\}} I_{i,j_i} \Big) \neq \emptyset.$$

The intersection I is an interval, whose endpoints are both endpoints of some of the intervals we intersect. If I is left-closed, its left endpoint is in I; otherwise

[2] The coefficients of x are again polynomials, but do not contain x.

there exists an infinitesimal value we can add to the left endpoint, such that the result is an element of I. In both cases we found an element of I being a solution for x. Candidates for the left endpoint of I are the left endpoints of the possible solution intervals, i.e. $-\infty$ and all finite endpoints $x_0,\ x_1,\ x_2$ for all constraints. When searching for a satisfying solution for x, it is sufficient to consider those candidates if they belong to a left-closed interval, or those candidates plus an infinitesimal if the corresponding interval is left-open.

In other words, we check if (1) one of the left endpoints of the left-closed solution intervals, or (2) one of the left endpoints plus an infinitesimal ϵ of the left-opened intervals or (3) a very small value, which we denote by $-\infty$, fulfills all constraints. We call those points belonging to (1), (2), or (3) *test candidates.* For the proof of soundness and completeness[3] of the method see [Wei88].

Basically, the virtual substitution recursively eliminates all bound variables x in φ by (i) determining all test candidates for x in all constraints in φ containing x, and (ii) checking if one of these test candidates satisfies φ.

To check whether a test candidate t for x satisfies a constraint $\overline{p} \sim 0$ in φ, we substitute all occurrences of x by t in $\overline{p}$, yielding $\overline{p}[t/x] \sim 0$, and check the resulting constraint under the test candidate's side conditions for consistency. Note that neither $\overline{p}[t/x] \sim 0$ nor the solution conditions refer to x, but they may contain other bound variables. Thus the consistency check may involve further quantifier eliminations.

Standard substitution could lead to terms not contained in real algebra, since the test candidates include $-\infty$, square roots, and infinitesimals ϵ. Virtual substitution however, avoids these expressions in the resulting terms: it defines substitution rules yielding formulas of real algebra that are equivalent to the result of the standard substitution. However, these substitution rules may increase the degree of the polynomials.

Assume T is the set of all possible test candidates for x. Given a test candidate $t \in T$ with side conditions C_t, the virtual substitution method applies the substitution rules to all constraints in the input formula φ and conjugates the result with C_t. Considering all possible test candidates results in the formula

$$\exists y_1 \ldots \exists y_n \bigvee_{t \in T} (\varphi[t/x] \wedge C_t).$$

Note that test candidates of type (3) do not have side conditions. The virtual substitution method continues with the elimination of the next variable.

3 The Adapted Virtual Substitution Method

As discussed in Section 1, a theory solver should support incrementality, infeasible subset generation, and backjumping in order to be suited for an efficient embedding into a less lazy SMT-solver.

The original virtual substitution method does not provide these functionalities yet. Nevertheless, it can be embedded into an SMT-solver. Full lazy SMT-solving

[3] For the restricted logic with quadratic constraints.

does not require incrementality, but is not very profitable compared to a less lazy approach with an incremental theory solver. We could also embed a non-incremental theory solver into a less lazy SMT-solver. However, in this case the theory solver has to re-do a lot of work.

In this section we propose an incremental version of the virtual substitution method, which is also able to generate an infeasible subset of the checked constraints in case they are inconsistent. We show, by way of an example, how the consistency check works, and give first experimental results using our prototype implementation.

3.1 Data Structure of the Theory Solver

Assume that the theory solver receives some constraints $\varphi = c_1, \dots,\ c_n$. The algorithm chooses a variable x to eliminate first and computes the set T of all test candidates that the constraints provide for x (see Section 2). The formula φ is satisfied if there is a test candidate $t \in T$ such that $\varphi_t = c_1[t/x] \wedge \dots \wedge c_n[t/x] \wedge C_t$ is satisfiable, where C_t are the side conditions of t. In contrast to the original algorithm, we proceed using a *depth-first search*, first checking consistency for a single test candidate t as long as φ_t does not turn out to be inconsistent, and switch to another test candidate otherwise. Applying the substitution $c_i[t/x]$, $1 \leq i \leq n$, may lead to further case distinctions from which we can again choose a first branch until it turns out to be unsatisfiable. If we succeed to eliminate all variables without getting any conflict in a certain branch, then the problem is satisfiable and we stop the search. Otherwise, if all branches lead to conflicts, the problem is unsatisfiable.

This search structure can be represented by a tree as shown in the examples of Figure 2. The framed nodes of this tree contain conjunctions of real constraints and are called *conjunction nodes*. The other nodes are either indexed substitutions, called *substitution nodes*, or *conflict nodes*. The indices of the substitutions are the side conditions of the test candidate it considers and the labels on the edges to a substitution are the constraints, which provided the substitutions test candidate. The successors of a node are formed as follows: Let us consider a conjunction node N containing the constraints $c_1, \dots,\ c_n$. Let x and T be as defined before. For each $t \in T$ there exists at most one successor of N being the substitution $[t/x]$ indexed by the side conditions of t. The successors of a substitution node $[t/x]_{C_t}$ are the substitution cases as explained above (*). If all of these cases contain a variable-free inconsistent constraint, than the substitution node has a conflict node as successor (**). We remove variable-free consistent constraints from the constraint sets. The index of a constraint in a conjunction node refers to the constraint in the root of the tree, from which it stems. For a constraint c indexed by c_i it means, that we got c from c_i by applying substitutions. We call c_i the *original constraint* of c. The index of a conflict node is formed by a disjunction of the original constraints of the constraints, which caused this conflict. It means, that applying the substitution of the antecessor node to such a constraint just results (according to the substitution rules) in cases containing variable-free inconsistent constraints.

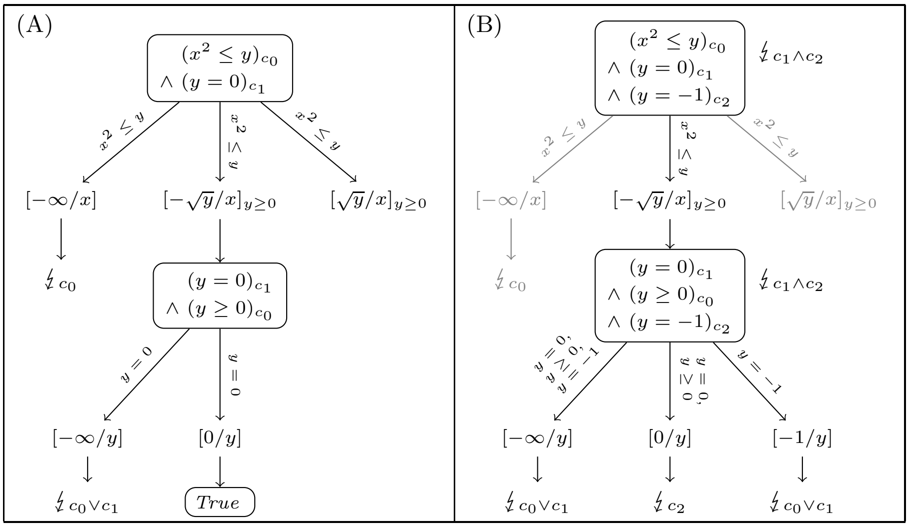

(A)
$(x^2 \le y)_{c_0} \wedge (y = 0)_{c_1}$
$x^2 \le y$
$x^2 \ge y$
$x^2 \le y$
$[-\infty/x]$
$[-\sqrt{y}/x]_{y\ge 0}$
$[\sqrt{y}/x]_{y\ge 0}$
$\lightning\, c_0$
$(y = 0)_{c_1} \wedge (y \ge 0)_{c_0}$
$y = 0$
$y = 0$
$[-\infty/y]$
$[0/y]$
$\lightning\, c_0 \vee c_1$
True
(B)
$(x^2 \le y)_{c_0} \wedge (y = 0)_{c_1} \wedge (y = -1)_{c_2}$
$\lightning\, c_1 \wedge c_2$
$[-\infty/x]$
$[-\sqrt{y}/x]_{y\ge 0}$
$[\sqrt{y}/x]_{y\ge 0}$
$\lightning\, c_0$
$(y = 0)_{c_1} \wedge (y \ge 0)_{c_0} \wedge (y = -1)_{c_2}$
$\lightning\, c_1 \wedge c_2$
$y = 0, y \ge 0, y = -1$
$y = 0, y \ge 0$
$y = -1$
$[-\infty/y]$
$[0/y]$
$[-1/y]$
$\lightning\, c_0 \vee c_1$
$\lightning\, c_2$
$\lightning\, c_0 \vee c_1$

Fig. 2. Solver state after (A) adding the constraints $c_0 : (x^2 \le y)$ and $c_1 : (y = 0)$ followed by a consistency check, and (B) adding another constraint $c_2 : (y \le -1)$ and performing another consistency check

The just described tree represents the data structure the theory solver maintains. As you can see in the next subsection, it provides all the information we need for an incremental implementation of the virtual substitution method and for the generation of infeasible subsets. Moreover, the tree structure permits an informed depth-first solution search instead of the breadth-first search given by the original virtual substitution.

3.2 Functionality of the Theory Solver

Using the above data structure, this subsection shows, by means of an example, how to initialize the solver, how to add constraints incrementally, how to perform a consistency check, and how to generate an infeasible subset of the added constraints, if the consistency check fails.

We initialize the solver's data structure with a single conjunction node containing the empty conjunction (representing *True*). First we add two constraints $c_0 : (x^2 \le y)$ and $c_1 : (y = 0)$ to the solver, whose conjunction is inserted into the root.

Now we provoke the consistency check of $c_0 \wedge c_1$ (Figure 2 part (A) shows the data structure after the consistency check). The solver repeats choosing a node in the data structure and creating successors for it until it either finds an empty conjunction node or no more nodes, for which we can create successors, exist. The former implies consistency, the latter inconsistency. Currently, there is only one node to choose, namely the root. As it is a conjunction node, its successors are substitution nodes. Hence, we fix a variable to eliminate next and form the test candidates for this variable, which the constraints in the root provide. Instead of creating all successors by determining all test candidates the constraints provide, the data structure allows us to create only successors corresponding to the test candidates of a single constraint. We choose the variable x for elimination and the constraint $x^2 \leq y$ for test candidate generation, as it is the only constraint containing x. The created successors are formed by the substitution $[-\infty/x]$ without a side condition and the substitutions $[-\sqrt{y}/x]$ and $[\sqrt{y}/x]$ both with the side condition $y \geq 0$.

The data structure now contains four nodes. However, we can only create successors for the three recently created substitution nodes, since the only constraint in the root containing x has already been used to create successors. We choose the left-most substitution node. Its successors are created by applying $[-\infty/x]$ to the conjunction $x^2 \leq y \wedge y = 0$ of its antecessor. It results in a conflict node with index c_0 (statement (**) in Section 3.1), denoting that the test candidate $-\infty$ for x is not a solution for $x^2 \leq y$. We call c_0 the *conflict reason* for the already fixed assignments of variables to test candidates (here just x to $-\infty$).

The next node we choose is $[-\sqrt{y}/x]_{y\geq 0}$. Applying the substitution yields a single conjunction node as successor (statement (*) in Section 3.1). We continue with the newly created node, in which only y can be eliminated. We create successors for the test candidates provided by $y = 0$, resulting in the substitution nodes $[-\infty/y]$ and $[0/y]$. We choose the former one to continue and create its successor being a conflict node with the index $c_0 \vee c_1$. The test candidate $-\infty$ for y is neither a solution $y = 0$ nor for $y \geq 0$. Therefore, the conflict reason is either c_0 or c_1.

We continue with the substitution node $[0/y]$ and create its only successor being an empty conjunction node. Hence, the consistency check terminates and reports consistency of $c_0 \wedge c_1$.

Next we add the constraint $c_2 : (y = -1)$ belatedly to the solver in order to demonstrate incrementality. The solver uses the resulting data structure of the previous consistency check, adds $y = -1$ to its root, and marks all nodes not being the root or one of its direct successors in order to avoid choosing them during a subsequent consistency check.

The solver checks the consistency of $c_0 \wedge c_1 \wedge c_2$ as follows (Figure 2 part (B) shows the solvers data structure after the consistency check). The only unmarked nodes for which we can create new successors are the root, $[-\sqrt{y}/x]_{y\geq 0}$, and $[\sqrt{y}/x]_{y\geq 0}$. We choose $[-\sqrt{y}/x]_{y\geq 0}$ and apply virtual substitution to the belatedly added constraint $y = -1$. The virtual substitution gives a real algebraic formula $C_1 \vee \ldots \vee C_n$ that is equivalent to the result of the standard substitution but does not contain square roots or fractions. To handle the case splitting, we copy each successor of the substitution node (including its subtree) n times and for each i, $1 \leq i \leq n$, we add the conjunction C_i, and to the ith copy. All nodes in these copied subtrees except the roots and their direct successors get marked in order to avoid choosing them in the next step. In the example there is a single successor and the resulting formula consists of one constraint, hence we just add this constraint to the successor and unmark it and its direct successors.

Now, we can choose each node in the data structure, except the empty conjunction node. We choose $[0/y]$ and create its successor, which is a conflict node with conflict reason c_2.

The next node we choose is the conjunction node $y = 0 \wedge y \geq 0 \wedge y = -1$. Until now we created only the successors considering the test candidates provided by $y = 0$. The constraint $y \geq 0$ provides the same test candidates. The constraint $y = -1$ provides another test candidate -1 for y. We create the according successor and choose it to continue the consistency check. Its resulting successor is a conflict node with conflict reason $c_0 \vee c_1$.

Now, we have created all possible successors of the considered conjunction node $y = 0 \wedge y \geq 0 \wedge y = -1$ and all lead to conflicts. It implies that its conjunction is not consistent. A conflict reason of this node must imply the conflict reason of each of its successors. We can choose for example the conflict reason $c_1 \wedge c_2$ and, indeed, $y = 0 \wedge y = -1$, which is the a result of applying $(c_1 \wedge c_2)[-\sqrt{y}/x]$, is inconsistent. Moreover, $y = 0 \wedge y = -1$ also appears in the preceding conjunction node, the root. Therefore it is also a conflict reason of the root and an infeasible subset of the constraints added to the solver.

We can use this information to jump back to the last node that implies the generated conflict reason. The constraints of that node are already conflicting, i.e., we do not need to continue the search in that subtree. We call this approach *backjumping*. In our example, we jump back to the root, which thus contains a conflict, and we report inconsistency.

3.3 Experimental Results

We are currently building a prototype implementation of the introduced procedures. We have already completed an *incremental* theory solver and a *non-incremental* one, both supporting the computation of *infeasible subsets*.

To get first results for the performance of an *SMT-solver* involving our theory solver, we embedded our implementation into an existing SMT-solver. However, it does *not* yet make use of the computed infeasible subsets. The SMT-embedding allows *lazy* or *less lazy* theory solver invocation. A lazy invocation calls the theory solver only when a complete solutions for the Boolean skeleton is found. The less lazy variant invokes the theory solver for consistency check after the completion of each decision level of the SAT-solver.

Since we did not yet embed an underlying complete theory solving procedure, to test our approach we need examples that during solving do not lead to polynomials of degree higher than two. Only after embedding, e.g., the CAD method, we will be able to handle more relevant case studies.

As benchmarks we have created a random set of test formulas of the form

$$(\ x_0 x_1 = c\ ) \wedge \bigwedge_{i=1}^{10} \bigvee_{j=1}^{5} (\ x_{i,j,1} + x_{i,j,2} + x_{i,j,3} + x_{i,j,4} + x_{i,j,5} = c_{i,j}\ )$$

Table 1. Running times (in seconds) for 4 benchmarks

Solver	Example 1	Example 2	Example 3	Example 4
Less lazy, non-incremental, with backjumping	18.879	15.805	22.769	13.981
Less lazy, incremental	23.879	43.200	20.103	47.124
Less lazy, non-incremental	49.217	99.714	109.582	68.955
Full lazy, non-incremental	550.988	144.826	162.805	102.458
`REDLOG`	1147.889	2104.883	1850.559	251.474

where x_0, x_1, and all $x_{i,j,k}$ are real-valued variables from a set *Var* with $|Var| = 20$ such that x_0 and x_1 are syntactically different, and c and all $c_{i,j}$ are constants from the set $\{1, \dots, 50\}$. The position of the single-literal clause, which is listed first above, is determined randomly, i.e., it is not always the first clause. Table 1 shows results, which are characteristic for this kind of input formula. All listed example formulas are satisfiable.

The running times show the expected result. The benefit of involving an SMT-solver can be observed by comparing the third and fourth row with the last one. The second row shows the running times of the SMT-solver using an incremental theory solver, whereas the Solver in the first row makes use of backjumping but without incrementality. Combining backjumping with incrementality has not yet been implemented but we expect it to lead to even shorter running times. The most promising improvement will be the inclusion of the infeasible subsets in the SMT environment.

4 Conclusion

In this paper we proposed an incremental adaptation of the virtual substitution method, and introduced methods for backjumping and for the generation of infeasible subsets. Our next step will be to make use of the infeasible subsets in our SMT-solver.

Our data model provides possibilities for different decision heuristics, which we also want to explore. Additionally, we will extend the degree of the constraints we can handle to its theoretical maximum of 4, and invoke a complete fall-back procedure.

References

[Bjo99] Bjorner, N.S.: Integrating Decision Procedures for Temporal Verification. PhD thesis, Stanford University (1999)

[Bjo10] Bjørner, N.: Linear quantifier elimination as an abstract decision procedure. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 316–330. Springer, Heidelberg (2010)

[BPT07] Bauer, A., Pister, M., Tautschnig, M.: Tool-support for the analysis of hybrid systems and models. In: DATE 2007, pp. 924–929. European Design and Automation Association (2007)

[Bro03] Brown, C.W.: QEPCAD B: A program for computing with semi-algebraic sets using CADs. SIGSAM Bull. 37, 97–108 (2003)

[CJ98] Caviness, B.F., Johnson, J.R. (eds.): Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation. Springer, Heidelberg (1998)

[Cor11] Corzilius, F.: Virtual substitution in SMT solving. Master's thesis, RWTH Aachen University (2011), http://www-i2.informatik.rwth-aachen.de/i2/publications/

[DH88] Davenport, J.H., Heinz, J.: Real quantifier elimination is doubly exponential. Journal of Symbolic Computation 5, 29–35 (1988)

[dMB08] de Moura, L., Bjørner, N.S.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)

[DS97] Dolzmann, A., Sturm, T.: REDLOG: Computer algebra meets computer logic. In: SIGSAM 1997, vol. 31, pp. 2–9 (1997)

[DSW97] Dolzmann, A., Sturm, T., Weispfenning, V.: Real quantifier elimination in practice (1997)

[FHT$^+$07] Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling, and Computation 1, 209–236 (2007)

[KS08] Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View. Springer, Heidelberg (2008)

[PJ09] Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) MKM 2009, Held as Part of CICM 2009. LNCS, vol. 5625, pp. 122–137. Springer, Heidelberg (2009)

[Tar48] Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley (1948)

[Wei88] Weispfenning, V.: The complexity of linear problems in fields. Journal of Symbolic Computation 5, 3–27 (1988)

[Wei98] Weispfenning, V.: A new approach to quantifier elimination for real algebra. In: Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation, pp. 376–392. Springer, Heidelberg (1998)

Author Index