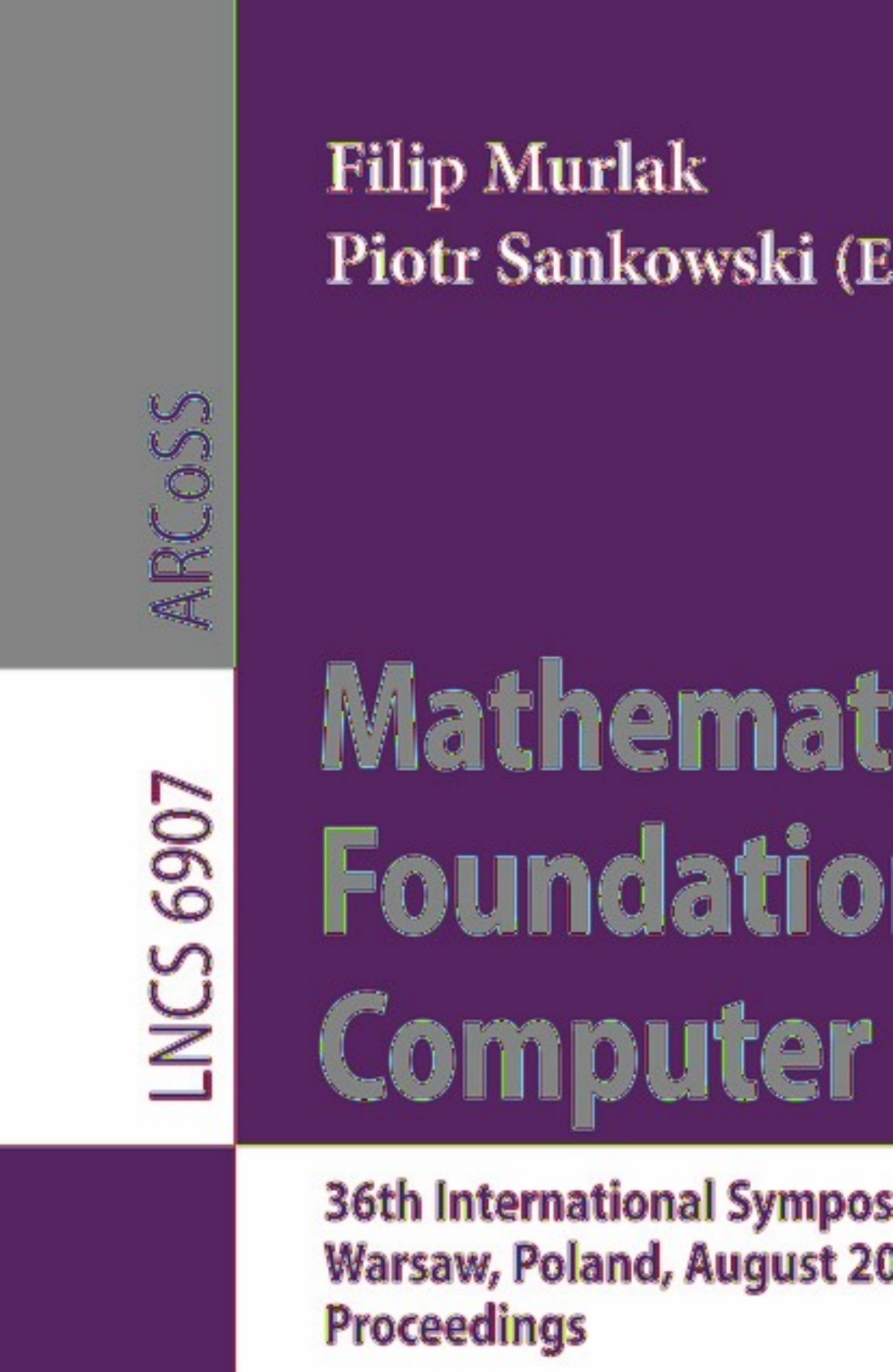

Filip Murlak
Piotr Sankowski (E
ARCoSS
LNCS 6907
Mathemat
Foundatio
Computer
36th International Sympos
Warsaw, Poland, August 20
Proceedings

Lecture Notes in Computer Science 6907

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

Advanced Research in Computing and Software Science
Subline of Lectures Notes in Computer Science

Filip Murlak Piotr Sankowski (Eds.)

Mathematical Foundations of Computer Science 2011

36th International Symposium, MFCS 2011
Warsaw, Poland, August 22-26, 2011
Proceedings

Springer

Volume Editors

Filip Murlak
Piotr Sankowski
University of Warsaw
Institute of Informatics
ul. Banacha 2, 02-097 Warsaw, Poland
E-mail: {fmurlak, sank}@mimuw.edu.pl

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-22992-3 e-ISBN 978-3-642-22993-0
DOI 10.1007/978-3-642-22993-0
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011933919

CR Subject Classification (1998): F.2, G.2, E.1, F.1, I.3.5, F.3

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 MFCS 2011: 36th International Symposium on Mathematical Foundations of Computer Science held during August 22–26, 2011 in Warsaw. It contains 48 contributed papers, selected by the Program Committee out of a total of 129 submissions. All submitted papers were peer reviewed and evaluated on the basis of originality, quality, soundness and significance. Each submission was reviewed by at least three Program Committee members with the help of external experts. The committee decided to give the Best Student Paper Award, sponsored by EATCS, to Remi Bonnet for the paper "The Reachability Problem for Vector Addition Systems with one Zero-Test."

The program also included six invited talks by:

- Alexandr Andoni, Microsoft Research Mountain View
- Jörg Flum, Albert-Ludwigs-Universität Freiburg
- Mai Gehrke, Radboud Universiteit Nijmegen
- Daniel Kirsten, Humboldt-Universität zu Berlin
- Prasad Raghavendra, Georgia Tech
- Paul Wollan, Sapienza University of Rome

As a special event, the Young Research Forum was organized in parallel with MFCS. The aim of the forum is to provide a platform for young researchers (excellent master students, PhD students, PostDocs) to present and discuss preliminary results or ongoing work in the field of theoretical computer science.

MFCS 2011 was organized by the Polish Mathematical Society and the Faculty of Mathematics, Informatics and Mechanics of the University of Warsaw in cooperation with the European Association for Theoretical Computer Science. We acknowledge with gratitude the support of all these institutions and thank the University of Warsaw for hosting the event. Special thanks are due to the Organizing Committee: Wojciech Czerwiński, Szczepan Hummel, Krystyna Jaworska, Agnieszka Kozubek, Marcin Pilipczuk, Jakub Radoszewski and Hanna Sokołowska. We also acknowledge EasyChair, an easy-to-use and freely available system for managing the work of the Program Committee and preparing conference proceedings.

June 2011

Filip Murlak
Piotr Sankowski

Organization

Program Committee

Andreas Abel	Ludwig Maximilians University Munich, Germany
Andris Ambainis	University of Latvia, Latvia
Aris Anagnostopoulos	Sapienza University of Rome, Italy
Henrik Björklund	Umeå University, Sweden
Tomáš Brázdil	Masaryk University, Czech Republic
Vince Bárány	University of Warsaw, Poland
Alessandra Carbone	Université Pierre et Marie Curie, France
Anuj Dawar	University of Cambridge, UK
Stephane Demri	CNRS, France
Xiaotie Deng	University of Liverpool, UK
Naveen Garg	IIT Delhi, India
Fabrizio Grandoni	University of Rome Tor Vergata, Italy
Bartek Klin	University of Cambridge, UK
Łukasz Kowalik	University of Warsaw, Poland
Leonid Libkin	University of Edinburgh, UK
Daniel Lokshtanov	UCSD, USA
Christof Löding	RWTH Aachen, Germany
Jerzy Marcinkowski	University of Wrocław, Poland
Jiří Matoušek	Charles University, Czech Republic
Vahab S. Mirrokni	Google Research
Filip Murlak	University of Warsaw, Poland
Paulo Oliva	Queen Mary University of London, UK
Krzysztof Onak	CMU, USA
Seth Pettie	University of Michigan, USA
Krzysztof Pietrzak	CWI, The Netherlands
Harald Raecke	University of Warwick, UK
Liam Roditty	Bar-Ilan University, Israel
Jacques Sakarovitch	CNRS / ENST, France
Piotr Sankowski	University of Warsaw, Poland
Rahul Savani	University of Liverpool, UK
Christian Scheideler	University of Paderborn, Germany
Mohit Singh	McGill University, Canada
Thomas Wilke	University of Kiel, Germany
Ryan Williams	IBM Almaden, USA
Ronald de Wolf	CWI Amsterdam, The Netherlands

Additional Reviewers

Anselmo, Marcella
Arrighi, Pablo
Arthan, Rob
Bala, Sebastian
Balbiani, Philippe
Beaudry, Martin
Berglund, Martin
Bernauer, Julie
Berwanger, Dietmar
Bienkowski, Marcin
Blocki, Jeremiah
Blumensath, Achim
Bollig, Benedikt
Bouyer, Patricia
Brain, Martin
Brandes, Ulrik
Braud, Laurent
Bro Miltersen, Peter
Brodal, Gerth Stølting
Brodsky, Alex
Brozek, Vaclav
Calude, Christian
Cao, Yixin
Carayol, Arnaud
Carbone, Marco
Carreiro, Facundo
Cerny, Pavol
Cesati, Marco
Chatterjee, Krishnendu
Chen, Ning
Choffrut, Christian
Codenotti, Paolo
Colcombet, Thomas
Courcelle, Bruno
Crouch, Michael
Crowston, Robert
Cygan, Marek
Dantchev, Stefan
de Mier, Anna
Dell, Holger
Diakonikolas, Ilias
Doyen, Laurent
Dziembowski, Stefan
Ehsani, Shayan
Escardo, Martin
Fearnley, John
Feret, Jerome
Feuillade, Guillaume
Fleischer, Rudolf
Forejt, Vojtech
Foschini, Luca
Friedmann, Oliver
Gaboardi, Marco
Gairing, Martin
Gaspers, Serge
Genest, Blaise
Gheerbrant, Amelie
Golas, Ulrike
Groth, Jens
Guillon, Pierre
Habermehl, Peter
Haeupler, Bernard
Heggernes, Pinar
Hertling, Peter
Hirvensalo, Mika
Hofman, Piotr
Holm, Bjarki
Horn, Florian
Hueffner, Falk
Högberg, Johanna
Hüllmann, Martina
Istrate, Gabriel
Jalaly, Pooya
Jancar, Petr
Jansen, David
Jeż, Artur
Jurdziński, Marcin
Jurdziński, Tomasz
Kaiser, Lukasz
Kasiviswanathan, Shiva
Kawamura, Akitoshi
Kieroński, Emanuel
Kitaev, Sergey
Kniesburges, Sebastian
Korula, Nitish
Koucky, Michal

Koutsopoulos, Andreas
Kral, Daniel
Kratsch, Dieter
Krcal, Marek
Kretinsky, Jan
Kufleitner, Manfred
Kun, Gabor
Kurucz, Agi
Lampis, Michael
Larchey-Wendling, Dominique
Larose, Benoit
Lattanzi, Silvio
Lin, Anthony Widjaja
Litak, Tadeusz
Loebl, Martin
Lohrey, Markus
Lombardy, Sylvain
Madhusudan, P.
Mahabadi, Sepideh
Maletti, Andreas
Mantaci, Roberto
Markey, Nicolas
Marx, Daniel
Matsliah, Arie
Matthews, William
Mayr, Richard
Mazza, Damiano
McGregor, Andrew
Mckenzie, Pierre
Meer, Klaus
Michaliszyn, Jakub
Mnich, Matthias
Møgelberg, Rasmus Ejlers
Nagano, Kiyohito
Niehren, Joachim
Novotny, Petr
Nussbaum, Yahav
Obdrzalek, Jan
Palano, Beatrice
Pilipczuk, Marcin
Pilipczuk, Michal
Plandowski, Wojciech
Pucci, Geppino
Qi, Qi
Radoszewski, Jakub
Rehak, Vojtech
Reinhardt, Klaus
Reutter, Juan
Rossmanith, Peter
Rothe, Jörg
Saia, Jared
Sangnier, Arnaud
Saurabh, Saket
Schewe, Sven
Schlotter, Ildikó
Schmitz, Sylvain
Schnoor, Henning
Segoufin, Luc
Seylan, Inanc
Shi, Xinwei
Skopalik, Alexander
Spirakis, Paul
Stachowiak, Grzegorz
Staton, Sam
Stephan, Frank
Sun, Xiaorui
Sénizergues, Géraud
Tan, Tony
Tang, Bo
Taslaman, Nina
Tendera, Lidia
Terzi, Evimaria
Teutsch, Jason
Thapen, Neil
Thilikos, Dimitrios
Tracol, Mathieu
Tranquilli, Paolo
Trivedi, Ashutosh
Turan, Gyorgy
Viola, Emanuele
Visconti, Ivan
Wahl, Thomas
Weimann, Oren
Wieczorek, Piotr
Wu, Bang Ye
Zhang, Jinshan
Zhang, Nan
Zhu, An
Zielonka, Wieslaw

Table of Contents

Invited Papers

Contributed Papers

Nearest Neighbor Search in High-Dimensional Spaces

Alexandr Andoni

Microsoft Research SVC

Nearest neighbor search in high-dimensional spaces is a ubiquitous problem in searching and analyzing massive data sets. In this problem, the goal is to preprocess a set of objects (such as images), so that later, given a new query object, one can efficiently return the object most similar to the query. This problem is of key importance in several areas, including machine learning, information retrieval, image/video/music clustering, and others. For instance, it forms the basis of a widely used classification method in machine learning: to label a new object, just find a similar but already-labeled object. Nearest neighbor search also serves as a primitive for other computational problems such as closest pair, minimum spanning tree, or variants of clustering.

In this talk, I will survey the state-of-the-art for the nearest neighbor search. I will give a flavor of the main algorithms and techniques involved, both some classical and some more recent ones. Along the way, I will highlight the current challenges in the area.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, p. 1, 2011.

Invariantization of Listings

Jörg Flum

Albert-Ludwigs-Universität Freiburg

We consider a halting problem for nondeterministic Turing machines and show via invariantization of listings the relationship of its complexity to

- the existence of almost optimal algorithms for the set of propositional tautologies;
- the existence of hard sequences for all algorithms deciding a given problem;
- the existence of logics capturing polynomial time.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, p. 2, 2011.

Duality and Recognition

Mai Gehrke

Radboud University Nijmegen, The Netherlands

Abstract. The fact that one can associate a finite monoid with universal properties to each language recognised by an automaton is central to the solution of many practical and theoretical problems in automata theory. It is particularly useful, via the advanced theory initiated by Eilenberg and Reiterman, in separating various complexity classes and, in some cases it leads to decidability of such classes. In joint work with Jean-Éric Pin and Serge Grigorieff we have shown that this theory may be seen as a special case of Stone duality for Boolean algebras extended to a duality between Boolean algebras with additional operations and Stone spaces equipped with Kripke style relations. This is a duality which also plays a fundamental role in other parts of the foundations of computer science, including in modal logic and in domain theory. In this talk I will give a general introduction to Stone duality and explain what this has to do with the connection between regular languages and monoids.

1 Stone Duality

Stone type dualities is *the* fundamental tool for moving between linguistic specification and spatial dynamics or transitional unfolding. As such, it should come as no surprise that it is a theory of central importance in the foundations of computer science where one necessarily is dealing with syntactic specifications and their effect on physical computing systems.

In 1936, M. H. Stone initiated duality theory by presenting what, in modern terms, is a dual equivalence between the category of Boolean algebras and the category of compact Hausdorff spaces having a basis of clopen sets, so-called Boolean spaces [13]. The points of the space corresponding to a given Boolean algebra are not in general elements of the algebra – just like states of a system are not in general available as entities in a specification language but are of an entirely different sort. In models of computation these two different sorts, specification expressions and states, are given a priori but in unrelated settings. Via Stone duality, the points of the space may be obtained from the algebra as homomorphisms into the two-element Boolean algebra or equivalently as ultrafilters of the algebra. In logical terms these are valuations or models of the Boolean algebra. In computational terms they are possible states of the system. Each element of the Boolean algebra corresponds to the set of all models in which it is true, or all states in which it holds, and the topology of the space is generated by these sets. A main insight of Stone is that one may recover the original algebra as the Boolean algebra of clopen subsets of the resulting space.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 3–18, 2011.

In Boole's original conception, Boolean algebras were meant to capture the arithmetic of propositions and he thought of propositions as 'classes' or sets of entities modelling them. In this sense Stone's theorem closes the circle by showing that every Boolean algebra is indeed isomorphic to a field of sets with the set theoretic operations of intersection, union, and complement as the Boolean operations. Stone duality is thus, in part, a representation theorem showing that the axioms of Boolean algebras exactly capture the fields of sets just like Cayley's theorem shows that the axioms of groups exactly capture the groups of permutations.

However, the fact that, with the topology in play, we obtain mathematical objects with their own and very separate theory and intuitions which fully capture the original Boolean algebras as well as their morphisms is the real power of Stone duality. The *duality* (as opposed to equivalence) aspect turns more complicated constructions such as quotients into simpler ones such as subobjects, it turns additional connectives on the algebras into transition structure on state spaces. This ability to translate faithfully between algebraic specification and spacial dynamics has often proved itself to be a powerful theoretical tool as well as a handle for making practical problems decidable. This principle was applied first by Stone himself in functional analysis, followed by Grothendieck in algebraic geometry who represented rings in terms of sheaves over the dual spaces of distributive lattices (i.e., 'positive' Boolean algebras) and has since, over and over again, proved itself central in logic and its applications in computer science. One may specifically mention Scott's model of the λ-calculus, which is a dual space, Esakia's duality [4] for Heyting algebras and the corresponding frame semantics for intuitionist logics, Goldblatt's paper [8] identifying extended Stone duality as the theory for completeness issues for Kripke semantics in modal logic, and Abramsky's path-breaking paper [1] linking program logic and domain theory. Our work with Grigorieff and Pin [7,9,6], with Pippenger [10] as a precursor, shows that the connection between regular languages and monoids also is a case of Stone duality.

1.1 Duality for Finite Distributive Lattices

Lattices are partial orders with infima (meets) and suprema (joins) of finite sets, but may also be seen as algebras $(L, \wedge, \vee, 0, 1)$ satisfying certain equations, see [2] for the basics of lattice theory. A lattice is distributive provided the binary meet ($\wedge$) and the binary join ($\vee$) distribute over each other. Distributive lattices corresponds to the negation-free reduct of classical propositional logic, and if in a distributive lattice every element a has a complement (that is, an element b so that $a \wedge b = 0$ and $a \vee b = 1$) then the lattice is a Boolean algebra.

The restriction of Stone's duality for distributive lattices to finite objects yields a duality between finite posets and finite distributive lattices which restricts further to a duality between finite sets and finite Boolean algebras. This duality was a precursor to Stone's duality and is due to Birkhoff. We begin with a description of Birkhoff duality as the essential features are easiest to understand in this setting. This duality is based on the fact that each element in a finite

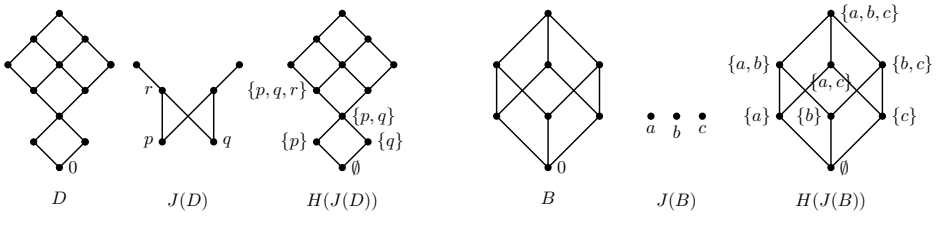
lattice is the join of all join irreducible elements below it and that all down-sets of join irreducible elements yield distinct elements if the lattice is distributive. The component facts are proved in [2, Lemma 5.11, page 117].

Definition 1. *An element p in a bounded lattice D is join irreducible provided $p \neq 0$ and $p = x \vee y$ in D implies $p = x$ or $p = y$. An element p in a bounded lattice D is join prime provided $p \neq 0$ and $p \leqslant x \vee y$ in D implies $p \leqslant x$ or $p \leqslant y$.*

We denote by $\mathcal{D}_{fin}$ the category of finite bounded distributive lattices with bounded lattice homomorphisms and by $\mathcal{P}_{fin}$ the category of finite posets with order preserving maps. Birkhoff duality is given by two functors

$$J : \mathcal{D}_{fin} \to \mathcal{P}_{fin} \qquad \text{and} \qquad H : \mathcal{P}_{fin} \to \mathcal{D}_{fin}$$

that establish the dual equivalence of the two categories. The functor J sends a finite bounded distributive lattice D, to the poset $J(D)$ of join irreducible elements of D with the order induced from D. For a finite poset P, the dual lattice $H(P)$ is the lattice of all down-sets of P with meet and join given by intersection and union. On the object level the dual equivalence of the categories $\mathcal{D}_{fin}$ and $\mathcal{P}_{fin}$ is given by the isomorphisms: $D \cong H(J(D)), a \mapsto \downarrow a \cap J(D)$ and $P \cong J(H(P)), p \mapsto \downarrow p$, see [2, Chapter 5]. The following figure provides two examples. Note that an element of a Boolean algebra is join irreducible if and only if it is an atom, i.e., an element right above 0, and thus the dual of a Boolean algebra is just a set.



The fact that the above correspondence extends to a categorical duality is what makes it so powerful. In order to specify the categorical duality we have also to give the correspondence between the morphisms in the two categories. This correspondence is essentially based on the notion of adjoint maps.

Definition 2. *Let D and E be finite lattices. Let $f : D \to E$ and $g : E \to D$ be functions satisfying for all $d \in D$ and for all $e \in E$:*

$$f(d) \leqslant e \qquad \Longleftrightarrow \qquad d \leqslant g(e).$$

Then g is called an upper adjoint of f and f is called a lower adjoint of g.

It is easy to see that adjoints are unique when they exist and that a map between complete lattices has an upper adjoint if and only if it preserves arbitrary joins and order dually for lower adjoints. If f has an upper adjoint, we will denote it by $f^\sharp$ and if g has a lower adjoint, we will denote it by $g^\flat$. Note that a bounded

lattice homomorphism between finite lattices $h : D \to E$ preserves arbitrary joins and meets. So it has both an upper adjoint and a lower adjoint. The duality for maps is based on the fact that a map $f : E \to D$ (such as $h^\flat$) has an upper adjoint which has an upper adjoint if and only if it sends join irreducible elements to join irreducible elements.

Definition 3. *Let D and E be finite distributive lattices, $h : D \to E$ a bounded lattice homomorphism. The dual of h is*

$$J(h) = h^\flat \restriction J(E),$$

that is, the restriction of the lower adjoint $h^\flat$ of h viewed as a map from $J(E)$ to $J(D)$. For finite posets P and Q and $f : P \to Q$ an order preserving map, we define $H(f) = (f^{\rightarrow})^\sharp$ where $f^{\rightarrow} : H(P) \to H(Q)$ is the forward image map, $S \mapsto f[S]$. Note that $H(f) = (f^{\rightarrow})^\sharp$ is then actually the inverse image map $T \mapsto f^{-1}(T)$ because the inverse image map is the upper adjoint of the forward image map.

Using the uniqueness of upper and lower adjoints, it is easy to show that J and H on morphisms in the two categories establish one-to-one correspondences as needed for the duality.

In closing, we note that the functors J and H can be extended to a duality between the category $\mathcal{DL}^+$ of down-set lattices with complete lattice homomorphisms and the category $\mathcal{P}$ of posets with order preserving maps by replacing binary meets and joins by arbitrary ones in the definitions above. However, this duality does not encompass distributive lattices in general (as can be seen, e.g. from the example at the end of the next subsection).

1.2 Duality for Bounded Distributive lattices

The basic idea of the dualities is to represent a distributive lattice by its set of join irreducible elements. However, for infinite lattices, there may not be enough of these, and idealised elements, in the form of ideals or filters, must be considered. Let D be a bounded distributive lattice. A subset I of D is an *ideal* provided it is a down-set closed under finite joins. We denote by $Idl(D)$ the set of all ideals of D partially ordered by inclusion. The embedding $D \to Idl(D), a \mapsto \downarrow a$ is the free $\bigvee$-completion of the lattice D. In this sense one should think of an ideal as standing for the element which would be the supremum of the ideal if the supremum existed. A subset F of D is a *filter* provided it is an up-set closed under finite meets. We denote by $Filt(D)$ the partially ordered set of all filters of D. Filters represent (possibly non-existing) infima and thus the order on filters is given by *reverse* inclusion. The embedding $D \to Filt(D), a \mapsto \uparrow a$ is the free $\bigwedge$-completion of the lattice D. An ideal or filter is *proper* provided it isn't the entire lattice. A proper ideal I is *prime* provided $a \wedge b \in I$ implies $a \in I$ or $b \in I$. A proper filter F is *prime* provided $a \vee b \in F$ implies $a \in F$ or $b \in F$.

Note that a filter is prime if and only if its complement is a (prime) ideal so that prime filters and prime ideals come in complementary pairs. In particular this means that the set of prime ideals with the inclusion order is isomorphic

to the set of prime filters with the reverse inclusion order. For a bounded distributive lattice D we will denote this partially ordered set by X_D or just X. Since there are so many set theoretic levels, we will revert to lower case letters $x, y, z \ldots$ for elements of X and to make clear when we talk about the corresponding prime filter or the complementary ideal we will denote these by F_x and I_x, respectively. In the case of a finite distributive lattice, filters and ideals are all principal generated by their meet and their join, respectively. In this case, the meets of prime filters are exactly the join prime elements of the lattice while the joins of the prime ideals are exactly the meet prime elements of the lattice. Thus these come in pairs $p, \kappa(p) = \bigwedge\{a \in D \mid p \not\leqslant a\}$ which split the lattice in two disjoint pieces, that is,

$$\forall a \in D \qquad (\ p \not\leqslant a \quad \Longleftrightarrow \quad a \leqslant \kappa(p)\ )$$

In a finite Boolean algebra, the meet of a prime filter is necessarily an atom while a meet irreducible is a co-atom and $\kappa(p) = \neg p$ in finite Boolean algebras.

In the infinite case prime filters play the role of the join irreducible elements, and it is not hard to verify that the following map is a bounded lattice homomorphism

$$\begin{aligned} \eta_D : D &\to \mathcal{P}(X_D) \\ a &\mapsto \eta_D(a) = \{x \in X_D \mid a \in F_x\} \end{aligned}$$

Using the Axiom of Choice one may in addition show that any distributive lattice has enough prime filters in the sense that this map also is injective.

One may also show that the sets in the image of η_D are down-sets in the reverse order of inclusion. However, for an infinite distributive lattice, it is never the set of all such down-sets. Stone's insight was to generate a topology with the sets in the image of η_D. This works but yields a non-Hausdorff space in the non-Boolean case. A slight variant of Stone duality was later developed by Priestley and this is what we will use here. The (Priestley) dual space of bounded distributive lattice D is the ordered topological space $(X_D, \leqslant, \pi)$ where X_D is the set of prime filters of D under reverse inclusion order and π is the topology on X_D generated by the subbasis

$$\{\eta_D(a), (\eta_D(a))^c \mid a \in D\}.$$

The space $(X_D, \leqslant, \pi)$ is then compact and *totally order disconnected*, that is, for $x, y \in X_D$ with $x \not\leqslant y$ there is a clopen down-set U with $y \in U$ and $x \notin U$. The dual of a homomorphism $h : D \to E$ is the restriction of the inverse image map to prime filters, $h^{-1} : X_E \to X_D$, and, for any homomorphism $h : D \to E$, the map $h^{-1} : X_E \to X_D$ is continuous and order preserving.

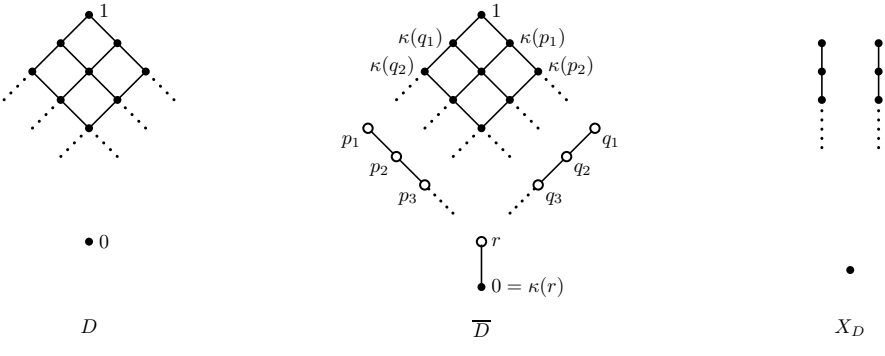
A *Priestley space* is an ordered topological space that is compact and totally order disconnected and the morphisms of Priestley spaces are the order preserving continuous maps. The dual of a Priestley space $(X, \leqslant, \pi)$ is the bounded distributive lattice $ClopD(X, \leqslant, \pi)$ of all subsets of X that are simultaneously clopen and are down-sets. For $f : X \to Y$ a morphism of

Priestley spaces, the restriction of the inverse image map to clopen down-sets, $f^{-1} : ClopD(Y) \to ClopD(X)$, is a bounded lattice homomorphism and is the dual of f under Priestley duality.

This accounts for Priestley duality. The point is that, for each distributive lattice D, the lattice $ClopD(X_D, \leqslant, \pi)$ is isomorphic to D via the map η_D as described above and these isomorphisms transform homomorphisms between lattices into their double dual homomorphisms. Similarly, any Priestley space is order isomorphic and homeomorphic to its double dual via the map which assigns to any point its neighbourhood filter and the double duals of order preserving continuous functions are naturally isomorphic to the original maps. This very tight relationship between the two categories allows one to translate essentially all structure, concepts, and problems back and forth between the two sides of the duality.

Note that in the case where the lattice D is a Boolean algebra, that is, each element a has a complement $\neg a$, then the order on prime filters (which are in this case the same as the ultrafilters or the maximal proper filters) is trivial, and since $(\eta_D(a))^c = \eta_D(\neg a)$, the image of η is already closed under complementation. In this case, the Priestley duality agrees with the original Stone duality and we may refer to it as Stone duality rather than as Priestley duality.

We close this subsection with an example. Let $D = \mathbf{0} \oplus (\mathbb{N}^{op} \times \mathbb{N}^{op})$ be the first lattice in the figure below. Note that D has *no* join irreducible elements whatsoever. The prime filters of D correspond to the hollow points in the lattice $\overline{D}$ (by taking the intersection with D of their individual up-sets) and the prime ideals of D are all principal down-sets given by the points as marked with κ's. The dual space X_D consists of two chains with a common lower bound, the image of η_D consists of the cofinite down-sets, and the topology is that of the one point compactification of a discrete space where the limit point is the least element. We recover D as the clopen down-sets.



1.3 Duality for Additional Operations

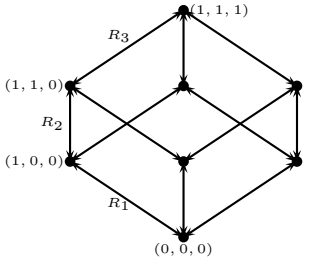
Any further structure on either side of the duality may be translated as corresponding structure on the other side. The translation of additional operations on lattices and Boolean algebras to their dual spaces is particularly important.

We give the simplest but prototypical example in the finite setting. Let B be a finite Boolean algebra and $f : B \to B$ a unary join and 0 preserving operation on B. This is usually called a normal modal (possibility) operator and it comes about in many applications. In order to dualise such an operation in case B is finite, we just need to know where the elements of $J(B)$ get sent. That is,

$$R_f = \{(x, y) \in X_B \times X_B \mid x \leqslant f(y)\}$$

encodes f as a binary relation on the dual of B. This relation is not a function unless f actually was a homomorphism on B. We illustrate this with an example.

Example 1. Consider the situation of the muddy children puzzle: there are n children each of whom may or may not have a muddy forehead thus giving rise to the Boolean algebra B whose atoms are the complete conjunctions over these n statements. Each child can see the foreheads of the others but not his own and we want to consider modal operators $<i>$, for each i from 1 to n, where $<i>\phi$ means ϕ is possible according to child i.



X_B

The dual space of B is the set of the 2^n atoms of B which may be thought of as n tuples each specifying which children have muddy foreheads and which don't. As explained above, the modal operators, $<i>$, are given dually by relations R_i where xR_iy for $x, y \in 2^n$ if and only if $x \leqslant <i>y$. Since the order in a Boolean algebra is the order of implication and x implies $<i>y$ precisely when x and y differ at most in the i^{th} coordinate, the relational image of each point in 2^n consists of precisely two points. In the case of three children for example, the dual space has 8 elements and the three relations each partition the points in two element sets along each of the three dimensions. Thus R_2 identifies points vertically above/below each other. This dual structure is quite simple and it is indeed also what one usually works with when analysing the associated dynamic epistemic puzzle [5].

For infinite lattices one first has to extend the operation to be dualised to the filters or the ideals (depending on whether it preserves join or meet) in order to have the operation defined on prime filters or ideals and thus on points of the dual space. Despite this slight complication, we get here too, for an n-ary operation, an $(n + 1)$-ary relation on the dual space. The dual relations will have appropriate topological properties. For a unary modality these amount to R being point-closed ($R[x] = \{y \mid xRy\}$ is closed for each x) and pre-images of clopens are clopen ($R^{-1}[U] = \{x \mid \exists y \in U \text{ with } xRy\}$ is clopen for each clopen U). One can also describe the duals of morphisms of lattices with additional operations. These are are often called bounded morphisms or p-morphisms. Altogether this yields what is known as *extended Stone or Priestley duality*. The details for a fairly large class of additional operations may be found in the first section of [8].

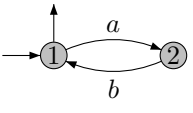
2 Monoids and Recognition by Automata

The starting point of the algebraic approach to automata theory is the classical result that one can effectively assign a finite monoid to each finite state automaton. We give a derivation of this basic result using extended duality.

An *automaton* is a structure $\mathcal{A} = (Q, A, \delta, I, F)$ where Q is a finite set whose elements are called *states*, A is a finite alphabet for the automaton, and $\delta \subseteq Q \times A \times Q$ is the *transition relation* which specifies the transition behaviour of the machine when it is fed a letter in a given state. The set I of *initial states* is a subset of Q as is the set F of *final states*.

We denote by A^* the free monoid over the alphabet A and any subset $L \subseteq A^*$ is called a *language* over A^*. The *language recognised by* $\mathcal{A}$, denoted $L(\mathcal{A})$, is the subset of A^* of all words $a_1 \dots a_n$ over the alphabet A such that there are states $q_0, \dots, q_n$ in Q with $(q_{i-1}, a_i, q_i) \in \delta$ for each i with $1 \leqslant i \leqslant n$ and $q_0 \in I$ and $q_n \in F$.

Example 2. Let $\mathcal{A} = (Q, A, \delta, I, F)$ where $Q = \{1, 2\}, A = \{a, b\}$, and δ is as specified in the picture. That is, $(q, x, q') \in \delta$ if and only if there is an arrow from q to q' labelled by x. The initial and final states are $I = \{1\} = F$.



The language recognised by $\mathcal{A}$ consists of all those words that may be read by starting in 1 and ending in state 1. That is, $L(\mathcal{A}) = (ab)^*$ where S^* denotes the submonoid generated by $S \subseteq A^*$ and $u^* = \{u\}^*$ for a word $u \in A^*$.

There may be many different automata that produce a given language but some languages recognised by automata require inherently more complex machines than others. A fundamental insight is that we can get at the essential features of the machines recognising a given language in a purely algebraic way from the language. As we shall see, we may think of the underlying transition system of an automaton as a kind of state space, and the languages recognised by it with various choices of initial and final states as a dual algebra of sets. Then, given what we know about duality, it should come as no surprise that the operations on languages dual to concatenation are given by adjunction.

Let A be a finite alphabet. The concatenation operation on A^* gives rise to a *residuated* or *adjoint family* of operations on the set of all languages over A^* as follows. *Complex* or *lifted concatenation* on $\mathcal{P}(A^*)$ is given by

$$KL = \{uv \mid u \in K \text{ and } v \in L\}. \tag{1}$$

The *residuals* of this operation are uniquely determined by the *residuation* or *adjunction laws*:

$$\begin{array}{lll} \forall K, L, M \in \mathcal{P}(A^*) & KM \subseteq L & \iff \quad M \subseteq K\backslash L \\ & & \iff \quad K \subseteq L/M. \end{array} \tag{2}$$

One easily sees from this that $K\backslash L = \{u \in A^* \mid \forall v \in K\ vu \in L\}$. In particular, for $K = \{x\}$ a singleton language $x\backslash L = \{u \in A^* \mid xu \in L\}$. The operations

$L \mapsto x\backslash L$ are widely used in language theory and usually $x\backslash L$ is denoted by $x^{-1}L$ and these operations are referred to as *quotients*.

One may now easily verify that the quotient operations on the left and the right correspond to moving the initial and final states along words respectively.

Proposition 1. *Let $L = L(\mathcal{A})$ be a language recognised by an automaton $\mathcal{A} = (Q, A, \delta, I, F)$. Then the languages $x^{-1}Ly^{-1}$ for $x, y \in A^*$ are recognised by automata $\mathcal{A}' = (Q, A, \delta, I', F')$ obtained from $\mathcal{A}$ by altering only the sets of initial and final states. Consequently, the set*

$$\{x^{-1}Ly^{-1} \mid x, y \in A^*\}$$

is finite.

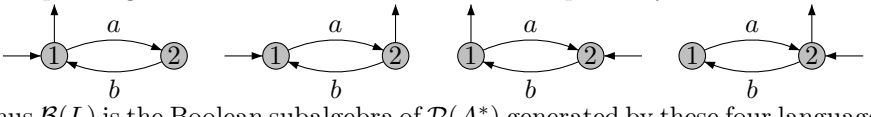
Definition 4. *Let A be a finite alphabet and $L \subseteq A^*$ a language over A. Let $\mathcal{B}(L)$ be the Boolean subalgebra of $\mathcal{P}(A^*)$ generated by the set $\{x^{-1}Ly^{-1} \mid x, y \in A^*\}$. We will call $\mathcal{B}(L)$ the* quotienting ideal *generated by L. More generally a quotienting ideal of $\mathcal{P}(A^*)$ is a Boolean subalgebra which is closed under the quotienting operations $x^{-1}(\ )$ and $(\ )y^{-1}$ for all $x, y \in A^*$.*

Using the fact that the quotienting operations $x^{-1}(\ )$ preserve all the Boolean operations and that $S\backslash(\ ) = \bigcap_{x \in S} x^{-1}(\ )$ (and the same on the right), we can then prove the following proposition.

Proposition 2. *Let L be a language recognised by some automaton. Then $\mathcal{B}(L)$ is closed under the operations $S\backslash(\ )$ and $(\ )/S$ for all $S \subseteq A^*$. In particular, $\mathcal{B}(L)$ is closed under the binary operations $\backslash$ and $/$.*

The stronger property that the Boolean subalgebra $\mathcal{B}(L)$ of $\mathcal{P}(A^*)$ has of being closed under residuation with arbitrary denominators, we call being a *residuation ideal*.

Example 3. For the language L of Example 2, it is clear that moving the final and initial states around along transitions yields the four automata given below corresponding to L, Lb^{-1}, $a^{-1}L$, and $a^{-1}Lb^{-1}$, respectively.



Thus $\mathcal{B}(L)$ is the Boolean subalgebra of $\mathcal{P}(A^*)$ generated by these four languages. It is not hard to see that this is the Boolean algebra generated by the atoms 1, $(ab)^+$, $a(ba)^*$, $b(ab)^*$, $(ba)^+$, and 0, where $1 = \{\varepsilon\} = L \cap a^{-1}Lb^{-1}$, and 0 is the complement of the union of the four generating languages. Note that $\mathcal{B}(L)$ is *not* closed under the lifted multiplication.

Theorem 1. *Let L be a language recognised by an automaton, then the extended dual of the Boolean algebra with additional operations $(\mathcal{B}(L), \backslash, /)$ is the syntactic monoid of L. In particular, it follows that the syntactic monoid of L is finite and is effectively computable.*

Proof. It is not hard to see that the atoms of the Boolean algebra generated by the finite collection $\mathcal{C} = \{x^{-1}Ly^{-1} \mid x, y \in A^*\}$ are the equivalence classes of the finite indexed equivalence relation

$$\begin{aligned} u \approx_L v \quad & \text{if and only if} \quad \forall x, y \in A^* \ (u \in x^{-1}Ly^{-1} \iff v \in x^{-1}Ly^{-1}) \\ & \text{if and only if} \quad \forall x, y \in A^* \ (xuy \in L \iff xvy \in L) \end{aligned}$$

and the set $A^*/\approx_L = S(L)$ is in fact the set underlying the syntactic monoid of L. It is a general fact that all the operations of a residuated family have the same dual relation up to the order of the coordinates. So we focus on the operation $\backslash$. It turns joins in the first coordinate into meets and meets in the second coordinate into meets. For this reason some swapping between join irreducible and meet irreducible elements using κ is needed. For $X, Y, Z \in A^*/\approx_L$ we have

$$\begin{aligned} R_\backslash(X, Y, Z) &\iff X\backslash\kappa(Y) \subseteq \kappa(Z) \\ &\iff X\backslash(Y^c) \subseteq Z^c \\ &\iff Z \not\subseteq X\backslash Y^c \\ &\iff XZ \not\subseteq Y^c \\ &\iff \exists x \in X, z \in Z \text{ with } xz \in Y \\ &\iff \exists x, z \text{ with } X = [x]_{\approx_L}, Z = [z]_{\approx_L}, Y = [xz]_{\approx_L} \end{aligned}$$

so that $R_\backslash$ is the graph of the operation on the quotient. Thus the dual space $(X_{\mathcal{B}(L)}, R_\backslash)$ is the quotient monoid $(A^*/\approx_L, \cdot/\approx_L)$. □

Example 4. Continuing with our running example $L = (ab)^*$, we have seen that the Boolean algebra $\mathcal{B}(L)$ has six atoms, namely $1 = \{\varepsilon\}$, $(ab)^+$, $a(ba)^*$, $b(ab)^*$, $(ba)^+$, and 0 (the latter consisting of all words in A^* containing two consecutive identical letters). Note that the product of two languages in $\mathcal{B}(L)$ may intersect several languages in $\mathcal{B}(L)$ (e.g., $(ab)^*(ba)^*$ intersects 1, $(ab)^+$, $(ba)^+$, and 0). However, the product of any two of the atoms is entirely contained in a unique other atom (while we didn't quite prove that above, it is a consequence of what we proved). It should be clear that in this example, the element 1 will be the neutral element, 0 will be absorbing, and the multiplication of the remaining four elements will be as given in the adjoining table.

	$(ab)^+$	$a(ba)^*$	$b(ab)^*$	$(ba)^+$
$(ab)^+$	$(ab)^+$	$a(ba)^*$	0	0
$a(ba)^*$	0	0	$(ab)^+$	$a(ba)^*$
$b(ab)^*$	$b(ab)^*$	$(ba)^+$	0	0
$(ba)^+$	0	0	$b(ab)^*$	$(ba)^+$

Duality theory is not just about objects but also about maps, and it is straight forward to check that the dual of the inclusion map $\iota : \mathcal{B}(L) \to \mathcal{P}(A^*)$ is the quotient map $\varphi_L : A^* \to A^*/\approx_L$, where $A^*/\approx_L = S(L)$ is the syntactic monoid of L. The content of this fact is that the dual of φ_L, which is $\varphi_L^{-1} : \mathcal{P}(S(L)) \to \mathcal{P}(A^*)$, is naturally isomorphic to $\iota : \mathcal{B}(L) \to \mathcal{P}(A^*)$:

$$\begin{array}{ccc} \mathcal{B}(L) & \xrightarrow{\iota} & \mathcal{P}(A^*) \\ \Big\updownarrow \cong & & \Big\updownarrow = \\ \mathcal{P}(S(L)) & \xrightarrow{\varphi_L^{-1}} & \mathcal{P}(A^*). \end{array}$$

This in turn precisely means that $\mathcal{B}(L) = \{\varphi_L^{-1}(P) \mid P \subseteq S(L)\}$.

3 Recognisable Subsets of an Algebra and Profinite Completions

Let A be any algebra, $\varphi : A \to B$ a homomorphism into a finite algebra. A subset $L \subseteq A$ is said to be *recognised by* φ provided there is a subset $P \subseteq B$ with $L = \varphi^{-1}(P)$. A subset $L \subseteq A$ is said to be *recognised by* B provided there is a homomorphism $\varphi : A \to B$ which recognises L. The last observation of the previous section can then be phrased as saying that the residuation ideal generated by a regular language L consists precisely of those languages that are recognised by the quotient map φ_L onto the syntactic monoid of L and, in particular, that every language recognised by an automaton also is recognised by a finite monoid. It is not hard to see that the converse of the latter statement also holds so that the languages recognised by finite automata precisely are the languages recognised by finite monoids. This is then the starting point of the algebraic theory of automata. The next, and most crucial step, is the link to profinite completions.

In this section we describe the duality theoretic link between recognition and profinite completion. The technical result is Theorem 2 at the end of the section. This result is crucial for the applications in Section 4. For the reader who finds the proof given here too abstract, a more pedestrian proof may be found in [6] where this result in the case of monoids occurs as Theorem 3 and a different proof is given.

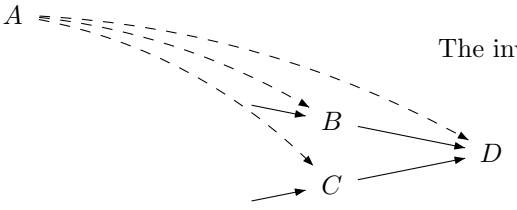
Let A be an algebra. A subset $L \subseteq A$ is said to be *recognisable* provided there is a finite algebra B such that L is recognised by B. We denote the Boolean algebra of all recognisable subsets of A by $\mathrm{Rec}(A)$. We have:

$$\begin{aligned} \mathrm{Rec}(A) &= \{\varphi^{-1}(P) \mid \varphi : A \to B \supseteq P \text{ with } \varphi \text{ an onto morphism and } B \text{ finite}\} \\ &= \bigcup \{\varphi^{-1}(\mathcal{P}(B)) \mid \varphi : A \to B \text{ is an onto morphism and } B \text{ is finite}\} \end{aligned}$$

By placing this definition in a more category-theoretic context we will be able to apply the dualities of Section 1. First note that the finite quotients of A are in one-to-one correspondence with the set $Con_\omega(A)$ of all congruences θ of A of finite index (i.e. for which the quotient algebra A/θ is finite). Also, $Con_\omega(A)$ is ordered by reverse set inclusion and is directed in this order since the intersection of two congruences of finite index is again a congruence of finite index. Thus we obtain an inverse limit system, $\mathcal{F}_A$, indexed by $Con_\omega(A)$ as follows:

1. For each $\theta \in Con_\omega(A)$ we have the finite algebra A/θ;

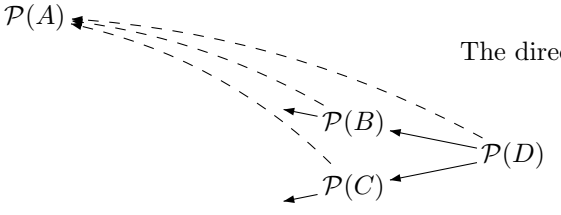
2. Whenever $\theta \subseteq \psi$ we have a (unique) homomorphism $A/\theta \to A/\psi$ which commutes with the quotient maps $q_\theta : A \to A/\theta$ and $q_\psi : A \to A/\psi$.



The inverse limit system $\mathcal{F}_A$

The *profinite completion* of an algebra A, denoted $\widehat{A}$, is by definition the inverse limit, $\varprojlim \mathcal{F}_A$, of the system $\mathcal{F}_A$ viewed *as a system of topological algebras*. Note that $\widehat{A}$ is not a topological algebra but the finite quotients are trivially so with the discrete topology.

Applying the discrete duality of Section 1.1, we get the direct limit system dual to $\mathcal{F}_A$:



The direct limit system $\mathcal{G}_A$

In Section 2, we saw that for a regular language L, the dual of the residuation algebra $(\mathcal{B}(L), \backslash, /)$ is the syntactic monoid of L. One can actually show that the extended Stone dual of any finite algebra is the Boolean algebra with residuation operations obtained by taking the powerset of the algebra with the residuals of the liftings of the operations of the algebra (as illustrated for a binary operation in (1) and (2) in Section 2). Further, the quotient maps in the system $\mathcal{F}_A$ are dual to the inverse image maps which provide embeddings between the finite powerset algebras, and one can show that the fact that the maps in $\mathcal{F}_A$ are algebra homomorphisms corresponds to the fact that the maps in $\mathcal{G}_A$ embed the residuation algebras as quotienting ideals (in the case of a single binary operation, see Definition 4). All in all, we obtain a direct limit system $\mathcal{G}_A$ of finite residuation ideals of $\mathcal{P}(A)$. It is well-known in algebra that a direct limit of subalgebras of an algebra simply is the union of these subalgebras. Thus we get

$$\begin{aligned}\varinjlim \mathcal{G}_A &= \bigcup \{\varphi^{-1}(\mathcal{P}(B)) \mid \varphi : A \to B \text{ is an onto morphism and } B \text{ is finite}\} \\ &= \mathrm{Rec}(A).\end{aligned}$$

We have outlined the proof of the following theorem.

Theorem 2. *Let A be an abstract algebra. Then* $\mathrm{Rec}(A)$ *is residuation ideal in* $\mathcal{P}(A)$ *and the profinite completion, $\widehat{A}$, of the algebra A is homeomorphic as a topological algebra to the extended Stone dual of* $\mathrm{Rec}(A)$ *viewed as a Boolean algebra with residuation operations.*

4 Eilenberg-Reiterman: Sub vs. Quotient Duality

In automata theory, deciding membership in a class of recognisable languages and separating two such classes are central problems. Classes of interest arise by restricting the class of automata allowed for recognition, or they arise via the description of languages by regular expressions by putting some restriction on the form of the expressions that are allowed. There is also, via Büchi's logic on words, a correspondence between recognisable languages and monadic second order sentences of this logic. Thus natural subclasses arise as the classes of languages corresponding to fragments of monadic second order logic.

The classical example is that of the star-free languages. These are the languages obtainable from the singleton languages by closure under the Boolean connectives and the lifted concatenation product (but *without* using the Kleene star). In terms of Büchi's logic, these are precisely the languages that are models of sentences of the first order fragment. While both of these descriptions of the star-free languages are nice, neither allows one readily to decide whether or not the language recognised by a given automaton is star-free or not. Schützenberger [12] made a breakthrough in the mid-sixties by using syntactic monoids to give an algebraic characterisation of the star-free languages which also provides a decision procedure for the class: a regular languages is star-free if and only if its syntactic monoid is aperiodic (for any element x in a finite monoid, there are m and n so that $x^{m+n} = x^m$; aperiodicity means the n uniformly can be taken to be equal to 1). This is clearly a decidable property of a finite monoid.

Eilenberg [3] greatly contributed to the success of the algebraic theory by isolating an essential feature of the above example: the finite aperiodic monoids form a *variety* of finite algebras. That is, a class of finite algebras closed under subalgebras, quotients, and finite Cartesian products. Further he showed that such varieties are in one-to-one correspondence with certain classes of regular languages, which he called *varieties of languages*. Later, Reiterman proved that profinite words (that is, elements of the profinite completion of A^* for A finite) can be viewed as $|A|$-ary term functions on any finite monoid, and that each variety of finite monoids is given by a set of identities in profinite words [11]. In conjunction we have: A class of regular languages is a variety of languages if and only if it can be defined by a set of profinite identities. If a variety of languages has a finite basis for its identities and the identities can be checked effectively, then it follows that the class is decidable. This has become a standard route to decidability, both for varieties of languages and for various generalisations for which Eilenberg-Reiterman theorems have subsequently been proved.

With Grigorieff and Pin, we gave a general and modular Eilenberg-Reiterman theorem based on duality theory [7]. The idea is the following. Let $\mathcal{C}$ be a sublattice of $\mathrm{Rec}(A^*)$, that is

$$\mathcal{C} \quad \hookrightarrow \quad \mathrm{Rec}(A^*).$$

Then, the Priestley dual $X_{\mathcal{C}}$ is a quotient space of the dual space of $\mathrm{Rec}(A^*)$, which we know to be $\widehat{A^*}$:

$$\widehat{A^*} \quad \twoheadrightarrow \quad X_{\mathcal{C}}.$$

That is, $\mathcal{C}$ is fully described by describing $X_{\mathcal{C}}$, and $X_{\mathcal{C}}$ is fully described by describing, in the case of a Boolean subalgebra of $\mathrm{Rec}(A^*)$, the equivalence relation on $\widehat{A^*}$ that yields the quotient $X_{\mathcal{C}}$. In the sublattice case, not only are some points identified going from $\widehat{A^*}$ to $X_{\mathcal{C}}$, but the order may also be strengthened. Thus sublattices correspond to certain quasiorders, called Priestley quasiorders, on $\widehat{A^*}$. This may be seen as the underlying source of profinite identities.

Fundamental to this relationship between sublattices and quotients is the following binary *satisfaction* relation between pairs $(u, v) \in \widehat{A^*} \times \widehat{A^*}$ and elements $L \in \mathrm{Rec}(A^*)$:

$$L \ \text{ satisfies } \ (u, v) \quad \Longleftrightarrow \quad \big(\eta_{\mathrm{Rec}(A^*)}(L) \in v \ \Rightarrow \ \eta_{\mathrm{Rec}(A^*)}(L) \in u\big).$$

A language L satisfies (u, v) provided L lies in a sublattice of $\mathrm{Rec}(A^*)$ corresponding to a Priestley quotient of $\widehat{A^*}$ in which u ends up being below v. For this reason we write $u \to v$ for these profinite inequations instead of just (u, v).

Theorem 3. *The assignments*

$$\Sigma \ \mapsto \ \mathcal{C}_\Sigma = \{L \in \mathrm{Rec}(A^*) \mid \forall\, u \to v \in \Sigma \ (L \ \textit{ satisfies } \ u \to v)\}$$

for $\Sigma \subseteq \widehat{A^*} \times \widehat{A^*}$ *and*

$$\mathcal{K} \ \mapsto \ \Sigma_{\mathcal{K}} = \{u \to v \in \widehat{A^*} \times \widehat{A^*} \mid \forall\, L \in \mathcal{K} \ (L \ \textit{ satisfies } \ u \to v)\}$$

for $\mathcal{K} \subseteq \mathrm{Rec}(A^*)$ *establish a Galois connection whose Galois closed sets are the Priestley quasiorders on* $\widehat{A^*}$ *and the bounded sublattices of* $\mathrm{Rec}(A^*)$*, respectively.*

Thus, for any sublattice $\mathcal{C}$ of $\mathrm{Rec}(A^*)$, we have $\mathcal{C}_{\Sigma_{\mathcal{C}}} = \mathcal{C}$ so that $\mathcal{C}$ is determined by a set of inequations. Also, we may look for bases $\Sigma \subseteq \Sigma_{\mathcal{C}}$ with $\mathcal{C}_\Sigma = \mathcal{C}$. In addition, if $\mathcal{C}$ is closed under the quotienting operations $a^{-1}(\ )$ and $(\ )a^{-1}$ on languages then we know that the corresponding Priestley quotient is also a monoid quotient, or equivalently, the corresponding Priestley quasiorder is a monoid congruence. In this case, we know that, with each $u \to v$ in $\Sigma_{\mathcal{C}}$, the inequation $xuy \to xvy$ is also in $\Sigma_{\mathcal{C}}$ and we can abbreviate this whole family of inequalities as $u \leqslant v$. Similarly, for Boolean sublattices $u \to v$ in $\Sigma_{\mathcal{C}}$ implies $v \to u$ in $\Sigma_{\mathcal{C}}$, and it can be shown that $\mathcal{C}$ being closed under inverse images for various kinds of homomorphisms of A^* corresponds to the set of inequations for $\mathcal{C}$ being closed with respect to various kinds of substitutions. The ensuing family of Eilenberg-Reiterman theorems thus obtained is summed up in the following table.

Closed under	**Equations**	**Definition**
$\cup, \cap$	$u \to v$	$\hat{\varphi}_L(v) \in P_L \Rightarrow \hat{\varphi}_L(u) \in P_L$
quotienting	$u \leqslant v$	for all $x, y,\ \ xuy \to xvy$
complement	$u \leftrightarrow v$	$u \to v$ and $v \to u$
quotienting and complement	$u = v$	for all $x, y,\ \ xuy \leftrightarrow xvy$
Closed under inverses of morphisms		**Interpretation of variables**
all morphisms		words
nonerasing morphisms		nonempty words
length multiplying morphisms		words of equal length
length preserving morphisms		letters

In order to understand the interpretation of profinite words in finite monoid quotients of A^*, it is important to realise that, by duality, any map $\varphi : A^* \to F$ has a unique extension $\widehat{\varphi} : \widehat{A^*} \to F$ obtained as the Stone dual of the Boolean algebra homomorphism $\varphi^{-1} : \mathcal{P}(F) \to \mathrm{Rec}(A^*)$.

Example 5. The class of star-free languages is axiomatised by $x^\omega = x^{\omega+1}$ with the interpretation of x ranging over all profinite words. The fact that the class is closed under the quotient operations and the Boolean operations means that L is star-free if and only if $\varphi_L^{-1}(P)$ is star-free for each $P \subseteq S(L)$, not just for P_L. Now, it can be shown that for $u \in \widehat{A^*}$ we have $\widehat{\varphi}_L(u^\omega) = e(\widehat{\varphi}_L(u))$ where $e : S(L) \to S(L)$ is the map that sends any element m of $S(L)$ to the unique idempotent in the cyclic monoid generated by m. Also, since each element of $S(L)$ is $\widehat{\varphi}_L(u)$ for some $u \in \widehat{A^*}$, we have L is star-free if and only if

$$\forall\ P \subseteq S(L)\ \forall m \in S(L) \qquad (\ e(m) \in P \iff e(m)\, m \in P\ ).$$

Since this has to hold in particular for singleton subsets P of $S(L)$, a language L is star-free if and only if $S(L)$ satisfies the identity $e(x) = e(x)x$. Here we get a genuine identity, that is, an equation scheme closed under substitution because the class of star-free languages is closed under inverse images of arbitrary morphisms between free finitely generated monoids. Finally we note that our language $L = (ab)^*$ is star-free since the elements 1, 0, $(ab)^+$, and $(ba)^+$ are idempotent and $e(a(ba)^*) = e(b(ab)^*) = 0$ is absorbent.

A *regular language with zero* is a regular language whose syntactic monoid has a 0. It is not hard to see that the class of regular languages with zero is closed under the quotient operations and the Boolean operations, but not under inverse images of arbitrary morphisms. Regular languages with 0 are given by the A-specific identities $x\rho_A = \rho_A = \rho_A x$ where ρ_A is an idempotent in the minimal (closed) ideal of $\widehat{A^*}$ and x can range over all elements of $\widehat{A^*}$. As in the case of star-freeness, the closure under the Boolean and the quotient operations allows us to quantify over all the subsets P of $S(L)$ and thus we must have

$$\forall m \in S(L) \qquad \big(\ m\,\widehat{\varphi}_L(\rho_A) = \widehat{\varphi}_L(\rho_A) = \widehat{\varphi}_L(\rho_A)\,m\ \big).$$

Since $\widehat{\varphi}_L(\rho_A)$ will necessarily belong to the minimum ideal of $S(L)$, it is easy to see that it will evaluate to 0 if and only if $S(L)$ has a zero so that these profinite equations precisely say that $S(L)$ has a zero.

For more details, see [9] where the various declensions of our theorem are illustrated with various examples.

References

1. Abramsky, S.: Domain Theory in Logical Form. Ann. Pure Appl. Logic 51, 1–77 (1991)
2. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (2002)
3. Eilenberg, S.: Automata, languages, and machines, vol. B. Academic Press (Harcourt Brace Jovanovich Publishers), New York (1976)
4. Esakia, L.L.: Topological Kripke models. Soviet Math. Dokl. 15, 147–151 (1974)
5. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
6. Gehrke, M.: Stone duality and the recognisable languages over an algebra. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 236–250. Springer, Heidelberg (2009)
7. Gehrke, M., Grigorieff, S., Pin, J.-É.: Duality and equational theory of regular languages. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 246–257. Springer, Heidelberg (2008)
8. Goldblatt, R.: Varieties of complex algebras. Ann. Pure App. Logic 44, 173–242 (1989)
9. Pin, J.-E.: Profinite methods in automata theory. In: Albers, S., Marion, J.-Y. (eds.) 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, pp. 31–50 (2009)
10. Pippenger, N.: Regular languages and Stone duality. Theory Comput. Syst. 30(2), 121–134 (1997)
11. Reiterman, J.: The Birkhoff theorem for finite algebras. Algebra Universalis 14(1), 1–10 (1982)
12. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Inform. Control 8, 190–194 (1965)
13. Stone, M.: The theory of representations for Boolean algebras. Trans. Amer. Math. Soc. 40, 37–111 (1936)

Some Variants of the Star Height Problem*

Daniel Kirsten**

Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, D-10099 Berlin, Germany

Abstract. Given a family of recognizable languages $L_1, \ldots, L_m$ and recognizable languages $K_1 \subseteq K_2$, the relative inclusion star height problem means to compute the minimal star height of some rational expression r over $L_1, \ldots, L_m$ satisfying $K_1 \subseteq L(r) \subseteq K_2$. We show that this problem is of elementary complexity and give an analysis of its complexity.

1 Introduction

The *star height problem* was raised by L.C. Eggan in 1963 [10]: Is there an algorithm which computes the star height of recognizable languages? Like L.C. Eggan, we consider star height concerning rational expressions with union, concatenation, and iteration in contrast to extended star height which also allows intersection and complement. For several years, the star height problem was considered as the most difficult problem in the theory of recognizable languages, and it took 25 years until K. Hashiguchi showed the existence of such an algorithm which is one of the most important results in the theory of recognizable languages [16]. His solution to the star height problem relies on distance automata and yields an algorithm of non-elementary complexity, and it remained open to deduce any upper complexity bound from K. Hashiguchi's approach (cf. [23, Annexe B]).

Recently, the author presented another approach to the star height problem which relies on a generalization of distance automata, the distance desert automata. He showed that the star height of the language of a non-deterministic automaton is computable in double exponential space which is the first upper complexity bound to the star height problem [19, 21].

K. Hashiguchi also considered the *relative star height problem*: Given a finite family of recognizable languages $L_1, \ldots, L_m$ and some recognizable language K, compute the minimal star height over all rational expressions r over $L_1, \ldots, L_m$ satisfying $L(r) = K$ [16]. In 1991, he considered inclusion variants of these problems, as the *inclusion star height problem*: Given two recognizable languages $K_1 \subseteq K_2$, compute the minimal star height over all rational expressions r satisfying $K_1 \subseteq L(r) \subseteq K_2$ [17]. Finally, K. Hashiguchi considered

* See [22] for a full version.

** The author was supported by the postgraduate program "Knowledge Representation" (GK 446) of the German Research Foundation (Deutsche Forschungsgemeinschaft) and by the Centre National de la Recherche Scientifique (CNRS).

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 19–33, 2011.

the *relative inclusion star height problem* which is a joint generalization of the relative and the inclusion star height problem. In [17], K. HASHIGUCHI showed the decidability of all these variants of the star height problem. The proofs in [17] are complicated. Moreover, [17] is a continuation of the difficult series of papers [14, 15, 16]. As for the star height problem, it remained open to deduce upper complexity bounds from [17].

In the present paper, we utilize distance desert automata and develop techniques from [19, 21] to give a concise decidability proof and an elementary upper complexity bound to the relative inclusion star height problem: it is decidable in triple exponential space.

2 Preliminaries

2.1 Notations, Rational Expressions, and Automata

We denote by $\mathcal{P}(M)$ the power set of some set M. We let $\mathbb{N} := \{0, 1, 2, \dots\}$.

Let Σ be some finite alphabet. We denote the empty word by ε. We denote by $|w|$ the length of some word $w \in \Sigma^*$.

We denote the set of rational expressions over Σ by $\mathsf{REX}(\Sigma)$ and define it as the least set of expressions which includes Σ, ε, $\emptyset$ and is closed such that for $r, s \in \mathsf{REX}(\Sigma)$, the expressions rs, $r \cup s$ and r^* belong to $\mathsf{REX}(\Sigma)$. We denote the language of some rational expression r by $L(r)$.

The *star height* of rational expressions is defined inductively: we set $\mathsf{sh}(\emptyset) := 0$, $\mathsf{sh}(\varepsilon) := 0$, and $\mathsf{sh}(a) := 0$ for every $a \in \Sigma$. For $r, s \in \mathsf{REX}(\Sigma)$, we set $\mathsf{sh}(rs) = \mathsf{sh}(r \cup s) := \max\{\mathsf{sh}(r), \mathsf{sh}(s)\}$, and $\mathsf{sh}(r^*) := \mathsf{sh}(r) + 1$.

For some language $L \subseteq \Sigma^*$, we define the *star height* of L by

$$\mathsf{sh}(L) \;:=\; \min\big\{\mathsf{sh}(r) \;\big|\; L = L(r)\big\}.$$

We recall some standard terminology in automata theory [5, 11, 25, 26, 28]. A (non-deterministic) *automaton* is a quadruple $\mathcal{A} = [Q, E, I, F]$ where

1. Q is a finite set of *states*,
2. $E \subseteq Q \times \Sigma \times Q$ is a set of *transitions*, and
3. $I \subseteq Q$, $F \subseteq Q$ are sets called *initial* resp. *accepting states*.

Let $k \geq 1$. A *path* π in $\mathcal{A}$ of length k is a sequence $(q_0, a_1, q_1)\ (q_1, a_2, q_2) \dots (q_{k-1}, a_k, q_k)$ of transitions in E. We say that π starts at q_0 and ends at q_k. We call the word $a_1 \dots a_k$ the *label of* π. We denote $|\pi| := k$. As usual, we assume for every $q \in Q$ a path which starts and ends at q and is labeled with ε.

We call π *successful* if $q_0 \in I$ and $q_k \in F$. For every $0 \leq i \leq j \leq k$, we denote $\pi(i, j) := (q_i, a_i, q_{i+1}) \dots (q_{j-1}, a_{j-1}, q_j)$ and call $\pi(i, j)$ a *factor of* π. For every $p, q \in Q$ and every $w \in \Sigma^*$, we denote by $p \overset{w}{\rightsquigarrow} q$ the set of all paths with the label w which start at p and end at q.

We denote the *language of* $\mathcal{A}$ by $L(\mathcal{A})$ and define it as the set of all words in Σ^* which are labels of successful paths. We call some $L \subseteq \Sigma^*$ *recognizable*, if L is the language of some automaton. We denote by $\mathsf{REC}(\Sigma^*)$ the class of all recognizable languages over Σ^*.

We call an automaton $\mathcal{A} = [Q, E, I, F]$ *normalized* if there are $q_I, q_F \in Q$ such that $I = \{q_I\}$, $\{q_F\} \subseteq F \subseteq \{q_I, q_F\}$, and $E \subseteq (Q \setminus \{q_F\}) \times \Sigma \times (Q \setminus \{q_I\})$. It is well known that each automaton can be transformed in an equivalent normalized automaton by adding at most two states.

2.2 Distance Desert Automata

Distance desert automata were introduced by the author in [19, 21]. They include K. Hashiguchi's distance automata [13] and S. Bala's and the author's desert automata [3, 4, 18, 20] as particular cases. In the recent years, several authors developed more general automata models, e.g., R-, S- and B-automata. See [1, 2, 9, 8, 7] for recent developments.

Let $h \geq 0$ and $V_h := \{\angle_0, \curlyvee_0, \angle_1, \curlyvee_1, \ldots, \curlyvee_{h-1}, \angle_h\}$. We define a mapping $\Delta : V_h^* \to \mathbb{N}$. Let us give an intuitive explanation. We imagine a machine with $h+1$ counters which can store non-negative integers. The machine processes words over V_h. Initially, the counters are set to zero. Whenever the machine processes a symbol $\curlyvee_i \in V_h$ for some i, it sets the counters $0, \ldots, i$ to zero and leaves the counters $i+1, \ldots, h$ unchanged. Whenever the machine processes a symbol $\angle_i \in V_h$ for some i, it sets the counters $0, \ldots, i-1$ to zero, increments counter i, and leaves the counters $i+1, \ldots, h$ unchanged. For every $\pi \in V_h^*$, we set $\Delta(\pi)$ as the highest value of some counter when the machine processes π.

The explanation using counters is due to [8, 9], a different explanation is given in [19, 21]. For a precise definition, let $\pi \in V_h^*$. For every $0 \leq g \leq h$, we consider every factor π' of π satisfying $\pi' \in \{\angle_0, \curlyvee_0, \ldots, \angle_g\}^* = V_g^*$, count the number of occurrences of $\angle_g$, and choose the maximum of these values. For $0 \leq g \leq h$ and $\pi' \in V_h^*$, let $|\pi'|_g$ be the number of occurrences of the letter $\angle_g$ in π'. Let

$$\Delta_g(\pi) := \mathsf{max}_{\substack{\pi' \in V_g^* \\ \pi' \text{ is a factor of } \pi}} |\pi'|_g \quad \text{and} \quad \Delta(\pi) := \mathsf{max}_{0 \leq g \leq h} \Delta_g(\pi).$$

An *h-nested distance desert automaton* (for short *distance desert automaton*) is a tuple $\mathcal{A} = [Q, E, I, F, \theta]$ where $[Q, E, I, F]$ is an automaton and $\theta : E \to V_h$.

Let $\mathcal{A} = [Q, E, I, F, \theta]$ be an h-nested distance desert automaton. For every transition $e \in E$, we say that e is *marked by* $\theta(e)$. We extend θ to a homomorphism $\theta : E^* \to V_h^*$. For $w \in \Sigma^*$, let

$$\Delta_{\mathcal{A}}(w) := \mathsf{min}_{p \in I,\, q \in F,\, \pi \in p \overset{w}{\rightsquigarrow} q} \Delta(\theta(\pi)).$$

We have $\Delta_{\mathcal{A}}(w) = \infty$ iff $w \notin L(\mathcal{A})$. Hence, $\Delta_{\mathcal{A}}$ is a mapping $\Delta_{\mathcal{A}} : \Sigma^* \to \mathbb{N} \cup \{\infty\}$.

If there is a bound $d \in \mathbb{N}$ such that $\Delta_{\mathcal{A}}(w) \leq d$ for every $w \in L(\mathcal{A})$, then we say that $\mathcal{A}$ is *limited by* d or for short that $\mathcal{A}$ is *limited*. Let $R \subseteq \Sigma^*$. If there is a bound $d \in \mathbb{N}$ such that $\Delta_{\mathcal{A}}(w) \leq d$ for every $w \in L(\mathcal{A}) \cap R$, then we say that $\mathcal{A}$ is *limited on* R. We need the following result.

Theorem 1 ([19, 21, 22]). *Given a distance desert automaton $\mathcal{A}$ and a nondeterministic automaton $\mathcal{A}'$, it is PSPACE-complete to decide whether $\mathcal{A}$ is limited on $L(\mathcal{A}')$.*

The particular case $L(\mathcal{A}') = \Sigma^*$ is shown in [19, 21], the general case is reduced to the case $L(\mathcal{A}') = \Sigma^*$ by a product construction [22].

3 Variants of the Star Height Problem

The *star height problem* was raised by L.C. Eggan in 1963 [10]: Given some recognizable language K, compute the star height of K. Or equivalently, given some recognizable language K and some integer h, decide whether $\mathsf{sh}(K) \leq h$. For several years, in particular after R. McNaughton refuted some promising ideas in 1967 [24], the star height problem was considered as the most difficult problem in the theory of recognizable languages, and it took 25 years until K. Hashiguchi showed its decidability [16]. The complexity of Hashiguchi's algorithm is extremely large, and it remained open to deduce an upper complexity bound (cf. [23, Annexe B]). However, the author showed the following result:

Theorem 2 ([19, 21]). *Let $h \in \mathbb{N}$ and K be the language accepted by an n-state non-deterministic automaton. It is decidable in $2^{2^{\mathcal{O}(n)}}$ space whether $\mathsf{sh}(K) \leq h$.*

An instance of the *inclusion star height problem* is a pair (K_1, K_2) of recognizable languages K_1 and K_2 satisfying $K_1 \subseteq K_2$. The *inclusion star height* of (K_1, K_2) is defined by

$$\mathsf{sh}(K_1, K_2) := \min\big\{\mathsf{sh}(r) \,\big|\, K_1 \subseteq L(r) \subseteq K_2\big\}.$$

The case $K_1 = K_2$ is Eggan's star height problem.

For the relative star height problem, one does not build up expressions from letters but from a given family of recognizable languages $L_1, \ldots, L_m$. An instance of the *relative star height problem* is a triple (K, m, σ) where

- K is a recognizable language,
- $m \geq 1$, and $\sigma : \Gamma \to \mathsf{REC}(\Sigma^*)$ where $\Gamma = \{b_1, \ldots, b_m\}$.

The languages $\sigma(b_1), \ldots, \sigma(b_m)$ play the role of the family $L_1, \ldots, L_m$, above. The mapping σ extends to a homomorphism $\sigma : \big(\mathcal{P}(\Gamma^*), \cup, \cdot\big) \to \big(\mathcal{P}(\Sigma^*), \cup, \cdot\big)$. For $r \in \mathsf{REX}(\Gamma)$, we denote $\sigma(L(r))$ by $\sigma(r)$.

The *relative star height* of (K, m, σ) is defined by

$$\mathsf{sh}(K, m, \sigma) := \min\big\{\mathsf{sh}(r) \,\big|\, r \in \mathsf{REX}(\Gamma),\, \sigma(r) = K\big\}$$

where the minimum of the empty set is defined as ∞. The reader should be aware that a rational expression r satisfying $\sigma(r) = K$ does not necessarily exist, e.g., it might be the case that K is finite but $\sigma(b_1), \ldots, \sigma(b_m)$ are infinite languages.

Assume $m = |\Sigma|$, $\Sigma = \{a_1, \ldots, a_m\}$, and $\sigma(b_i) = \{a_i\}$ for $i \in \{1, \ldots, m\}$. Clearly, we have $\mathsf{sh}(K) = \mathsf{sh}(K, m, \sigma)$ for every $K \in \mathsf{REC}(\Sigma^*)$. Hence, Eggan's star height problem is a particular case of the relative star height problem.

The finite power problem (FPP) means to decide whether some given recognizable language L has the finite power property, i.e., whether there exists some integer k such that $L^* = \cup_{i=0}^{k} L^i$. It was raised by J.A. Brzozowski in 1966, and it took more than 10 years until I. Simon and K. Hashiguchi independently showed its decidability [27, 12].

Let $L \subseteq \Sigma^*$ be a recognizable language and set $m := 1$ and $\sigma(b_1) := L$. We have $\mathsf{sh}(L^*, m, \sigma) \leq 1$ since $\sigma(b_1^*) = L^*$. Moreover, $\mathsf{sh}(L^*, m, \sigma) = 0$ iff L has the finite power property [22]. Consequently, the FPP is a particular case of the relative star height problem.

An instance of the *relative inclusion star height problem* is a quadruple (K_1, K_2, m, σ) where

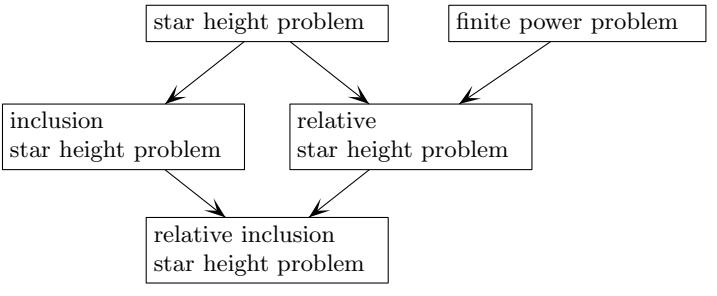
- K_1, K_2 are recognizable languages satisfying $K_1 \subseteq K_2$,
- m and σ are defined as for the relative star height problem.

The *relative inclusion star height* of (K_1, K_2, m, σ) is defined by

$$\mathsf{sh}(K_1, K_2, m, \sigma) := \min\left\{\mathsf{sh}(r) \,\middle|\, r \in \mathsf{REX}(\Gamma),\ K_1 \subseteq \sigma(r) \subseteq K_2\right\}.$$

Given some instance (K_1, K_2, m, σ), we call some $r \in \mathsf{REX}(\Gamma)$ a *solution of* (K_1, K_2, m, σ) if $\mathsf{sh}(r) = \mathsf{sh}(K_1, K_2, m, \sigma)$ and $K_1 \subseteq \sigma(r) \subseteq K_2$.

The following figure shows the relations between the five above problems. The arrows go from particular to more general problems.



In 1991, K. Hashiguchi showed that the relative inclusion star height problem is effectively computable [17]. Here, we show the first complexity bound:

Theorem 3 ([22]). *Let (K_1, K_2, m, σ) be an instance of the relative inclusion star height problem.*

For $i \in \{1, 2\}$, let n_i be the number of states of non-deterministic automata recognizing K_i. Denote by n_σ the total number of states of non-deterministic automata recognizing $\sigma(b_i)$ for $i \in \{1, \ldots, n\}$.

1. *If (K_1, K_2, m, σ) has a solution, then $\mathsf{sh}(K_1, K_2, m, \sigma) \leq 2^{2^{n_2}}$.*
2. *Given $h \in \mathbb{N}$, $\mathsf{sh}(K_1, K_2, m, \sigma) \leq h$ is decidable in*

$$n_1 \cdot n_\sigma \cdot 2^{h \cdot 2^{\mathcal{O}(n_2)}}$$

space.

3. *The relative inclusion star height $\mathsf{sh}(K_1, K_2, m, \sigma)$ is computable in*

$$n_1 \cdot n_\sigma \cdot 2^{2^{2^{\mathcal{O}(n_2)}}}$$

space.

The integer m does not explicitly occur in the complexity bounds in (2) and (3). However, m implicitly occurs in the factor n_σ since $n_\sigma \geq m$.

4 The Proof of Theorem 3

In this section, we prove Theorem 3. We assume an instance (K_1, K_2, m, σ) of the relative inclusion star height problem. To avoid some technical overhead, we assume $\varepsilon \notin \sigma(b_i)$ for $i \in \{1, \ldots, m\}$. Moreover, we assume $\varepsilon \notin K_1$, and hence, $\varepsilon \notin K_2$. The general case is shown in [22] by a reduction to this particular case.

We assume that every language $\sigma(b_i)$ is given by some non-deterministic automaton $\mathcal{B}_i$. We denote the number of states of all these $\mathcal{B}_i$ by n_σ.

For $i \in \{1, 2\}$, we assume non-deterministic automata $\mathcal{A}_i = [Q_i, E_i, I_i, F_i]$ satisfying $L(\mathcal{A}_i) = K_i$ and denote $n_i = |Q_i|$.

4.1 Limitedness and Substitutions

We show a certain closure property for mappings defined by nested distance desert automata. Let $h \geq 1$ and $\mathcal{A} = [Q, E, I, F, \theta]$ be an h-nested distance desert automaton over Γ.

We define a mapping $\Delta' : \Sigma^* \to \mathbb{N} \cup \{\infty\}$ by setting

$$\Delta'(w) := \min\left\{\Delta_{\mathcal{A}}(u) \,\middle|\, u \in \Gamma^*,\ w \in \tau(u)\right\}$$

for every $w \in \Sigma^*$.

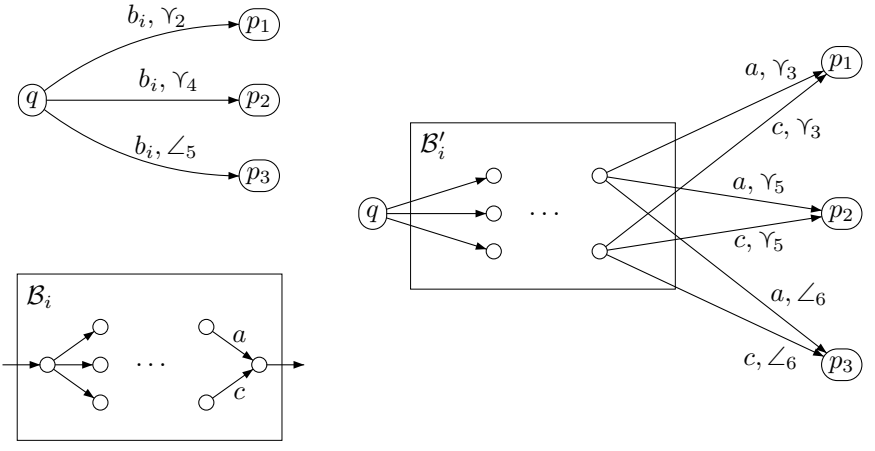
Proposition 1. *We can effectively construct an $(h+1)$-nested distance desert automaton $\mathcal{A}'$ over Σ with at most $|Q| \cdot (n_\sigma - 2m + 1)$ states which computes Δ'.*

The condition $\varepsilon \notin \tau(b_i)$ for $i \in \{1, \ldots, m\}$ is necessary for Proposition 1 [22].

Proof (Sketch). At first, we deal with some preliminaries. We define a homomorphism *lift* $\ell : V_h^* \to V_{h+1}^*$ by setting for every $i \in \{0, \ldots, h+1\}$, $\ell(\curlywedge_i) := \curlywedge_{i+1}$ and for every $i \in \{0, \ldots, h\}$, $\ell(\curlyvee_i) := \curlyvee_{i+1}$. It is easy to verify that for every $\pi \in V_h^*$, we have $\Delta(\pi) = \Delta(\ell(\pi))$.

To construct $\mathcal{A}'$, we replace the transitions in $\mathcal{A}$ by copies of $\mathcal{B}_i$. Let $q \in Q$ and $i \in \{1, \ldots, m\}$ such that there exists at least one transition of the form $\{q\} \times \{b_i\} \times Q$ in E. Let P be the states $p \in Q$ which admit a transition $(q, b_i, p) \in E$. We create $|P|$ copies of the accepting state of $\mathcal{B}_i$. We insert the new automaton $\mathcal{B}_i'$ into $\mathcal{A}$ and merge q and the initial state of $\mathcal{B}_i'$ and we merge each state in P and one accepting state of $\mathcal{B}_i'$.

The idea of the transition marks in $\mathcal{A}'$ is the following: For every $(p, b_i, q) \in E$ and every word $w \in \tau(b_i)$ there is some path $\pi \in p \overset{w}{\rightsquigarrow} q$ in $\mathcal{A}'$ such that $\theta(\pi) = \curlyvee_0^{|w|-1} \ell\big(\theta((p, b_i, q))\big)$.



We proceed this insertion for every $q \in Q$, $i \in \{1, \ldots, m\}$ provided that there exists at least one transition of the form $\{q\} \times \{b_i\} \times Q$ in E. One can easily verify that the constructed automaton computes Δ'. □

4.2 On the Existence of a Solution

As already mentioned, a solution of (K_1, K_2, m, σ) does not necessarily exist. In this section, we show that the existence of a solution is decidable. Moreover, we show that if there is some (not necessarily recognizable) language $R \subseteq \Gamma^*$ such that $K_1 \subseteq \sigma(R) \subseteq K_2$, then there exists a solution.

The language

$$L = \{w \in \Gamma^* \mid \sigma(w) \subseteq K_2\},$$

which is by its definition the largest subset $L \subseteq \Gamma^*$ satisfying $\sigma(L) \subseteq K$, plays a central role.

Proposition 2. *We can effectively construct a total, non-deterministic automaton $\mathcal{A}_{\overline{L}}$ which recognizes $\Gamma^* \setminus L$.*

Given an automaton $\mathcal{A}'_2$ which recognizes $\Sigma^* \setminus K_2$, the number of states of $\mathcal{A}_{\overline{L}}$ is at most the number of states of $\mathcal{A}'_2$.

Proof (Sketch). Let $\mathcal{A}'_2 = [Q'_2, E'_2, I'_2, F'_2]$ be an automaton which recognizes $\Sigma^* \setminus K_2$. The automaton $\mathcal{A}'_2$ might be constructed by a determinization and complementation of $\mathcal{A}_2$, but in general, we do not require $\mathcal{A}'_2$ to be deterministic.

We define a new set of transitions $E_{\overline{L}}$. For every $p, q \in Q'_2$, $b \in \Gamma$, the triple (p, b, q) belongs to $E_{\overline{L}}$ iff there exists some word $w \in \sigma(b)$ such that $\mathcal{A}'_2$ admits a path from p to q which is labeled with w. This condition is decidable in polynomial time since it means to decide whether the language of $[Q'_2, E'_2, p, q]$ and $\sigma(b)$ are disjoint.

Clearly, $\mathcal{A}_{\overline{L}} = [Q'_2, E_{\overline{L}}, I'_2, F'_2]$ recognizes $\Gamma^* \setminus L$. □

By Proposition 2, we have also shown that L is recognizable. Now, the following conditions are equivalent:

1. There is some $R \subseteq \Gamma^*$ satisfying $K_1 \subseteq \sigma(R) \subseteq K_2$.
2. The instance (K_1, K_2, m, σ) has a solution.
3. $K_1 \subseteq \sigma(L)$

For (3) $\Rightarrow$ (2), let $r \in \mathsf{REX}(\Gamma)$ such that $L(r) = L$. For (2) $\Rightarrow$ (1), set $R = \sigma(r)$ whereas r is a solution. For (1) $\Rightarrow$ (3), note that $\sigma(R) \subseteq K_2$ implies $R \subseteq L$ by the definition of L.

Since (3) is decidable, we can decide the existence of a solution.

Proof (Theorem 3(1)). As another consequence, we have $\mathsf{sh}(K_1, K_2, m, \sigma) \leq \mathsf{sh}(L)$ if (K_1, K_2, m, σ) has a solution. We can construct an automaton recognizing L by complementing the automaton from Proposition 2. Consequently, L is recognized by an automaton having at most $2^{2^{n_2}}$ states, i.e., we have

$$\mathsf{sh}(K_1, K_2, m, \sigma) \leq 2^{2^{n_2}}$$

if (K_1, K_2, m, σ) has a solution. □

4.3 String Expressions

We recall the notion of a string expression from R.S. Cohen [6].

Every word $w \in \Sigma^*$ is a *single string expression* of star height $\mathsf{sh}(w) = 0$ and *degree* $\mathsf{dg}(w) := |w|$. Let $n \geq 1$ and $r_1, \ldots, r_n$ be single string expressions. We call $r := r_1 \cup \cdots \cup r_n$ a *string expression* of star height $\mathsf{sh}(r) = \max\{\mathsf{sh}(r_i) \mid 1 \leq i \leq n\}$ and *degree* $\mathsf{dg}(r) := \max\{\mathsf{dg}(r_i) \mid 1 \leq i \leq n\}$. The empty set $\emptyset$ is a *string expression* of star height $\mathsf{sh}(\emptyset) = 0$ and *degree* $\mathsf{dg}(\emptyset) := 0$.

Let $n \geq 2$, $a_1, \ldots, a_n \in \Sigma$, and $s_1, \ldots, s_{n-1}$ be string expressions. We call the expression $s := a_1 s_1^* a_2 s_2^* \ldots s_{n-1}^* a_n$ a *single string expression* of star height $\mathsf{sh}(s) = 1 + \max\{\mathsf{sh}(s_i) \mid 1 \leq i < n\}$ and *degree* $\mathsf{dg}(s) := \max\big(\{n\} \cup \{\mathsf{dg}(s_i) \mid 1 \leq i < n\}\big)$.

The following lemma is due to R.S. Cohen [6].

Lemma 1 ([6, 19, 21]). *Let $L \subseteq \Sigma^*$ be a recognizable language. There is a string expression s such that we have $L = L(s)$ and $\mathsf{sh}(s) = \mathsf{sh}(L)$.*

Lemma 2 ([19, 21]). *Let $L \subseteq \Sigma^*$ be recognizable. We have $\mathsf{sh}(L) = \mathsf{sh}(L \setminus \{\varepsilon\})$.*

4.4 The $T_{d,h}(P, R)$-Hierarchy

Let $\mathcal{A}_{\overline{L}} = [Q_{\overline{L}}, E_{\overline{L}}, I_{\overline{L}}, F_{\overline{L}}]$ be the automaton recognizing $\Gamma^* \setminus L$ by Proposition 2.

Let $\delta_{\overline{L}} : \mathcal{P}(Q_{\overline{L}}) \times \Gamma^* \to \mathcal{P}(Q_{\overline{L}})$ be defined by

$$\delta_{\overline{L}}(P, w) := \{r \in Q_{\overline{L}} \mid P \overset{w}{\rightsquigarrow} r \neq \emptyset\}$$

for every $P \subseteq Q_{\overline{L}}$, $w \in \Gamma^*$. For every $P, R \subseteq Q_{\overline{L}}$ let

$$\mathcal{T}(P, R) := \{w \in \Gamma^+ \mid \delta_{\overline{L}}(P, w) \subseteq R\}.$$

Consequently, $\mathcal{T}\big(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}}\big) \,=\, \Gamma^+ \setminus L(\mathcal{A}_{\overline{L}}) \,=\, L \setminus \{\varepsilon\}$.

Let $d \geq 1$ and $P, R \subseteq Q_{\overline{L}}$. We define

$$T_{d,0}(P,R) := \big\{ w \in \Gamma^+ \,\big|\, \delta_{\overline{L}}(P,w) \subseteq R,\, |w| \leq d \big\}.$$

We have

$$T_{d,0}(P,R) = \bigcup_{\substack{1 \leq c \leq d,\\ P_0,\ldots,P_c \subseteq Q_{\overline{L}},\\ P = P_0,\, P_c \subseteq R}} T_{1,0}\big(P_0,P_1\big)\, T_{1,0}\big(P_1,P_2\big)\, \ldots\, T_{1,0}\big(P_{c-1},P_c\big).$$

It is easy to see that $\mathcal{T}(P,R) \,=\, \bigcup_{d\geq 1} T_{d,0}(P,R)$.

Now, let $h \in \mathbb{N}$, and assume by induction that for every $P, R \subseteq Q_{\overline{L}}$, the language $T_{d,h}(P,R)$ is already defined. We define

$$T_{d,\,h+1}(P,R) := \bigcup_{\substack{1 \leq c \leq d,\\ P_0,\ldots,P_c \subseteq Q_{\overline{L}},\\ P = P_0,\, P_c \subseteq R}} T_{1,0}(P_0,P_1)\Big(T_{d,h}(P_1,P_1)\Big)^* \ldots$$

$$\ldots T_{1,0}(P_1,P_2)\Big(T_{d,h}(P_2,P_2)\Big)^* \ldots T_{1,0}(P_{c-1},P_c).$$

Let $d \geq 1$, $h \in \mathbb{N}$, and $P, R \subseteq Q_{\overline{L}}$ be arbitrary. We have $\varepsilon \notin T_{d,h}(P,R)$.

Lemma 3. *Let $d \geq 1$, $h \in \mathbb{N}$, and $P, R \subseteq Q_{\overline{L}}$. We have*

$$\big(T_{d,h}(P,P)\big)^* T_{1,0}(P,P) \big(T_{d,h}(P,P)\big)^* \;\subseteq\; \big(T_{d,h}(P,P)\big)^*.$$

Proof. The assertion follows, because $T_{1,0}(P,P) \subseteq T_{d,h}(P,P)$ and $\big(T_{d,h}(P,P)\big)^*$ is closed under concatenation. □

For every $d' \geq d$, $h' \geq h$, $R' \supseteq R$, $T_{d,h}(P,R) \subseteq T_{d',h'}(P,R)$. For fixed $P, R \subseteq Q_{\overline{L}}$, the sets $T_{d,h}(P,R)$ form a two-dimensional hierarchy. Whenever we use the notion $T_{d,h}(P,R)$-*hierarchy*, we regard $P, R \subseteq Q_{\overline{L}}$ and $h \in \mathbb{N}$ as fixed, i.e., it is a one-dimensional hierarchy with respect to the parameter $d \geq 1$.

By induction, we can easily construct a string expression r with $L(r) = T_{d,h}(P,R)$ such that $\mathsf{sh}(r) \leq h$ and $\mathsf{dg}(r) \leq d$, and hence, $\mathsf{sh}\big(T_{d,h}(P,R)\big) \leq h$. However, we cannot assume that there is a string expression r with $L(r) = T_{d,h}(P,R)$ such that $\mathsf{sh}(r) = h$ and $\mathsf{dg}(r) = d$. In the inductive construction of r, several sets $T_{1,0}(P_{i-1},P_i)$ may be empty, and then, the star-height (resp. degree) of r is possibly smaller than h (resp. d). Just consider the case $T_{d,h}(P,R) = \{a\}$ but $h > 1$, $d > 1$.

Lemma 4. *Let $d \geq 1$, $h \in \mathbb{N}$, and $P, R \subseteq Q_{\overline{L}}$. We have $T_{d,h}(P,R) \subseteq \mathcal{T}(P,R)$.*

Proof (Sketch). The proof is done for some fixed d. The case $h = 0$ follows directly from the definition, the case $h > 0$ is shown by induction on h. □

We have for every $h \in \mathbb{N}$ and $P, R \subseteq Q_{\overline{L}}$:

$$\mathcal{T}(P,R) \;=\; \bigcup_{d\geq 1} T_{d,0}(P,R) \;\subseteq\; \bigcup_{d\geq 1} T_{d,h}(P,R) \;\subseteq\; \mathcal{T}(P,R).$$

4.5 The Collapse of the $T_{d,h}(P,R)$-Hierarchy

We say that the $T_{d,h}(P,R)$-hierarchy *collapses for some* $h \in \mathbb{N}$ if there is some $d \geq 1$ such that $T_{d,h}(P,R) = \mathcal{T}(P,R)$. Below, we will observe that the $T_{d,h}(P,R)$-hierarchy collapses for some h iff $h \geq \mathsf{sh}\big(\mathcal{T}(P,R)\big)$.

For the relative inclusion star height problem, we are rather interested in $\sigma\big(T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})\big)$ than in $T_{d,h}(P,R)$. In particular, it is interesting whether for some given $h \in \mathbb{N}$, there exists some d such that $K_1 \subseteq \sigma\big(T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})\big)$. For this, the following lemma will be very useful.

Lemma 5. *Let r be a string expression, $d \geq \mathsf{dg}(r)$, and $h \geq \mathsf{sh}(r)$. Let $P, R \subseteq Q_{\overline{L}}$ such that $L(r) \subseteq \mathcal{T}(P,R)$. We have $L(r) \subseteq T_{d,h}(P,R)$.*

Proof. We assume $L(r) \neq \emptyset$. By $L(r) \subseteq \mathcal{T}(P,R)$, we have $\varepsilon \notin L(r)$.

Assume $\mathsf{sh}(r) = 0$. There are some $k \geq 1$ and $w_1, \ldots, w_k \in \Gamma^+$ such that $r = w_1 \cup \cdots \cup w_k$ and for every $1 \leq i \leq k$, we have $|w_i| \leq d$, and moreover, $\delta(P, w_i) \subseteq R$. By the definition of $T_{d,0}(P,R)$, we have $w_i \in T_{d,0}(P,R)$, i.e., $L(r) \subseteq T_{d,0}(P,R) \subseteq T_{d,h}(P,R)$.

Now, let $\mathsf{sh}(r) \geq 1$, and assume that the claim is true for every string expression r' with $\mathsf{sh}(r') < \mathsf{sh}(r)$.

Clearly, it suffices to consider the case that r is a single string expression. Let $c \geq 2$ and $a_1, \ldots, a_c \in \Gamma$ and $r_1, \ldots, r_{c-1}$ be string expressions of a star height less that $\mathsf{sh}(r)$ such that $r = a_1 r_1^* a_2 r_2^* \ldots r_{c-1}^* a_c$. Let $d \geq \mathsf{dg}(r)$ and $h \geq \mathsf{sh}(r)$. Let $P, R \subseteq Q_{\overline{L}}$ such that $L(r) \subseteq \mathcal{T}(P,R)$.

Let $P_0 := P$, and for $1 \leq i < c$, let $P_i := \delta\big(P_{i-1}, a_i L(r_i^*)\big)$. Finally, let $P_c := \delta(P_{c-1}, a_c)$. To show $L(r) \subseteq T_{d,h}(P,R)$, we apply the definition of $T_{d,h}(P,R)$ with $P_0, \ldots, P_c$. We defined $P_0 = P$, and we can easily show $P_c = \delta\big(P_0, L(r)\big) \subseteq R$. Clearly, $c \leq d$. To complete the proof, we show the following two assertions:

1. for every $1 \leq i \leq d'$, we have $a_i \in T_{1,0}(P_{i-1}, P_i)$, and
2. for every $1 \leq i < d'$, we have $L(r_i) \subseteq T_{d,h-1}(P_i, P_i)$.

(1) Clearly, $\delta(P_{i-1}, a_i) \subseteq \delta\big(P_{i-1}, a_i L(r_i)^*\big) = P_i$. Hence, $a_i \in T_{1,0}(P_{i-1}, P_i)$ follows from the definition of $T_{1,0}(P_{i-1}, P_i)$.

(2) We have $\mathsf{sh}(r_i) < h$ and $\mathsf{dg}(r_i) \leq d$. In order to apply the inductive hypothesis, we still have to show $\delta(P_i, L(r_i)) \subseteq P_i$. We have $a_i L(r_i)^* L(r_i) \subseteq a_i L(r_i)^*$. Thus, we obtain

$$\delta(P_i, L(r_i)) \;=\; \delta\Big(\delta\big(P_{i-1},\, a_i L(r_i)^*\big),\, L(r_i)\Big) \;=$$

$$\delta\big(P_{i-1},\, a_i L(r_i)^* L(r_i)\big) \;\subseteq\; \delta\big(P_{i-1},\, a_i L(r_i)^*\big) \;=\; P_i. \qquad \square$$

Let $P, R \subseteq Q_{\overline{L}}$ and $h \geq \mathsf{sh}\big(\mathcal{T}(P,R)\big)$. By Lemma 1, there is a string expression r such that $L(r) = \mathcal{T}(P,R)$ and $h \geq \mathsf{sh}(r)$. Let $d := \mathsf{dg}(r)$. We have

$$\mathcal{T}(P,R) \;=\; L(r) \overset{\text{Lemma 5}}{\subseteq} T_{d,h}(P,R) \overset{\text{Lemma 4}}{\subseteq} \mathcal{T}(P,R),$$

i.e., the $T_{d,h}(P,R)$-hierarchy collapses for h.

Conversely, let $h \in \mathbb{N}$, $P, R \subseteq Q_{\overline{L}}$ and assume that the $T_{d,h}(P,R)$-hierarchy collapses for h. Let $d \geq 1$ such that $T_{d,h}(P,R) = \mathcal{T}(P,R)$. As already seen, we can construct a string expression r such that $L(r) = T_{d,h}(P,R)$, $\mathsf{sh}(r) \leq h$, and $\mathsf{dg}(r) \leq d$. Thus, $h \geq \mathsf{sh}\big(\mathcal{T}(P,R)\big)$.

To sum up, the $T_{d,h}(P,R)$-hierarchy collapses for h iff $h \geq \mathsf{sh}\big(\mathcal{T}(P,R)\big)$.

Proposition 3. *Let $h \in \mathbb{N}$. There exists some $d \geq 1$ such that $K_1 \subseteq \sigma\big(T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})\big)$ iff $\mathsf{sh}(K_1, K_2, m, \sigma) \leq h$.*

Proof. $\cdots \Rightarrow \cdots$ Let r be a string expression such that $L(r) = T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})$, $\mathsf{sh}(r) \leq h$, and $\mathsf{dg}(r) \leq d$. From $L(r) \subseteq \mathcal{T}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}}) = L \setminus \{\varepsilon\}$, it follows $\sigma(L(r)) \subseteq \sigma(L) \subseteq K_2$.

Moreover, $K_1 \subseteq \sigma(L(r))$. Consequently, $h \geq \mathsf{sh}(r) \geq \mathsf{sh}(K_1, K_2, m, \sigma)$.

$\cdots \Leftarrow \cdots$ Let s be a solution of (K_1, K_2, m, σ), i.e., $K_1 \subseteq \sigma(L(s)) \subseteq K_2$. Our aim is to apply Lemma 5 to show that $L(s)$ is subsumed by the set $T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})$ for some $d \geq 1$. However, the empty word causes some trouble. Since $\varepsilon \notin K_1$, we obtain

$$K_1 \subseteq \sigma\big(L(s) \setminus \{\varepsilon\}\big) \subseteq K_2.$$

By Lemmas 1 and 2, we can transform s into a string expression r by preserving the star height such that $L(r) = L(s) \setminus \{\varepsilon\}$. Thus,

$$K_1 \subseteq \sigma(L(r)) \subseteq K_2.$$

From $L(r) \subseteq L(s) \subseteq L$ and $\varepsilon \notin L(r)$, it follows $L(r) \subseteq L \setminus \{\varepsilon\} = \mathcal{T}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})$.

Let $d := \mathsf{dg}(r)$. Since $h \geq \mathsf{sh}(r)$, we can apply Lemma 5 and get $L(r) \subseteq T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})$, i.e.,

$$K_1 \subseteq \sigma(L(r)) \subseteq \sigma\big(T_{d,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})\big). \qquad \square$$

4.6 A Reduction to Limitedness

In this section, we construct for given $h \in \mathbb{N}$ and $P, R \subseteq Q_{\overline{L}}$ a $(h+1)$-nested distance desert automaton $\mathcal{A}_h(P,R)$ over the alphabet Γ. This automaton associates to each word $w \in \Gamma^+$ the least integer d such that $w \in T_{d+1,\,h}(P,R)$. It computes ∞ if such an integer d does not exist, i.e., if $w \notin \mathcal{T}(P,R)$.

The automaton $\mathcal{A}_h(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})$ will be of particular interest. By applying Proposition 1, we transform $\mathcal{A}_h(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})$ to a distance desert automaton which associates to each word $w \in \Sigma^*$ the least integer d such that $w \in \sigma\big(T_{d+1,h}(I_{\overline{L}}, Q_{\overline{L}} \setminus F_{\overline{L}})\big)$.

In combination with Proposition 3 and the decidability of limitedness (Theorem 1), this construction allows us to decide whether $\mathsf{sh}(K_1, K_2, m, \sigma) \leq h$.

Proposition 4. *Let $h \in \mathbb{N}$ and $P, R \subseteq Q_{\overline{L}}$. We can construct an $(h+1)$-nested distance desert automaton $\mathcal{A}_h(P,R) = [Q, E, q_I, q_F, \theta]$ with the following properties:*

1. $E \subseteq (Q \setminus \{q_F\}) \times \Gamma \times (Q \setminus \{q_I\})$,
2. $|Q| \leq k^{h+1} + \dfrac{k^h - 1}{k-1} + 1$ *where* $k = 2^{|Q_{\overline{L}}|}$,

3. *for every* $(p,a,q) \in E$, *we have* $\theta((p,a,q)) = \curlyvee_h$ *if* $p = q_I$, *and* $\theta((p,a,q)) \in \{\curlyvee_0, \ldots, \curlyvee_{h-1}, \curlyangle_0, \ldots, \curlyangle_h\}$ *if* $p \neq q_I$,
4. *for every* $w \in \Gamma^*$, $\Delta_{\mathcal{A}}(w) + 1 = \mathsf{min}\{d \geq 1 \mid w \in T_{d,h}(P,R)\}$.

Proof (Sketch). We employ the mapping $\delta_{\overline{L}}$ from the beginning of Section 4.4. We proceed by induction on h. Let $P, R \subseteq Q_{\overline{L}}$ be arbitrary.

Let $h = 0$. At first, we construct an automaton which accepts every word w with $\delta(P,w) \subseteq R$. We use $\mathcal{P}(Q_{\overline{L}})$ as set of states. For every $S, T \subseteq Q_{\overline{L}}$, $b \in \Gamma$, we set a transition (S,b,T) iff $\delta_{\overline{L}}(S,b) \subseteq T$. The initial state is P, every non-empty subset of R is an accepting state. We apply to this automaton a standard construction to get an automaton $[Q, E, q_I, q_F]$ which satisfies (1) where $Q = \mathcal{P}(Q_{\overline{L}}) \cup \{q_I', q_F'\}$. For every transition $(q_I, b, q) \in E$, we set $\theta((q_I,b,q)) = \curlyvee_0$. For every transition $(p,b,q) \in E$ with $p \neq q_I$, we set $\theta((p,b,q)) = \curlyangle_0$. This completes the construction of $\mathcal{A}_0(P,R) = [Q, E, q_I, q_F, \theta]$.

Now, let $h \in \mathbb{N}$. We assume that the claim is true for h and show the claim for $h+1$. At first, we construct an automaton $\mathcal{A}' := [Q', E', q_I, q_F]$. Let $Q' := \mathcal{P}(Q_{\overline{L}}) \cup \{q_I, q_F\}$.

Let $b \in \Gamma$ and $S, T \subseteq Q_{\overline{L}}$ be arbitrary. If $S \neq T$ and $\delta(S,b) \subseteq T$, then we put the transition (S,b,T) into E'. If $\delta(P,b) \subseteq T$, then we put the transition (q_I,b,T) into E'. If $\delta(S,b) \subseteq R$, then we put the transition (S,b,q_F) into E'. Finally, if $\delta_{\overline{L}}(P,b) \subseteq R$, then we put the transition (q_I,b,q_F) into E'. For every word w which $\mathcal{A}'$ accepts, we have $w \in \mathcal{T}(P,R)$.

We define $\theta' : E' \to \{\curlyvee_{h+1}, \curlyangle_{h+1}\}$. For every transition $(q_I,b,q) \in E'$, let $\theta'((q_I',b,q)) = \curlyvee_{h+1}$. For every transition $(p,b,q) \in E'$ with $p \neq q_I$, we set $\theta'((p,b,q)) = \curlyangle_{h+1}$.

We construct $\mathcal{A}_{h+1}(P,R)$. For every $S \subseteq Q_{\overline{L}}$, we assume by induction an automaton $\mathcal{A}_h(S,S)$ which satisfies $(1, \ldots, 4)$. We assume that the sets of states of the automata $\mathcal{A}_h(S,S)$ are mutually disjoint. We construct $\mathcal{A}_{h+1}(P,R) = [Q, E, q_I, q_F, \theta]$ as a disjoint union of $\mathcal{A}'$ and the automata $\mathcal{A}_h(S,S)$ for every $S \subseteq Q_{\overline{L}}$ and unifying both the initial and accepting state of $\mathcal{A}_h(S,S)$ with the state S in $\mathcal{A}'$. Because we did not allow self loops in $\mathcal{A}'$, the union of the transitions is disjoint, and hence, θ arises in a natural way as union of θ' and the corresponding mappings of the automata $\mathcal{A}_h(S,S)$. If $\theta(t) \in \{\curlyvee_{h+1}, \curlyangle_{h+1}\}$ for some $t \in E$, then t stems from $\mathcal{A}'$. Conversely, if $\theta(t) \in \{\curlyvee_0, \ldots, \curlyvee_h, \curlyangle_0, \ldots, \curlyangle_h\}$ for some $t \in E$, then t stems from some automaton $\mathcal{A}_h(S,S)$.

Let π be some path in $\mathcal{A}_{h+1}(P,R)$ and assume that for every transition t in π, we have $\theta(t) \in \{\curlyvee_0, \ldots, \curlyvee_h, \curlyangle_0, \ldots, \curlyangle_h\}$. Then, the entire path π stems from some automaton $\mathcal{A}_h(S,S)$, i.e., π cannot visit states in $\mathcal{P}(Q_{\overline{L}}) \setminus \{S\}$. Conversely, if π is a path in $\mathcal{A}_{h+1}(P,R)$, and two states $S, T \subseteq Q_{\overline{L}}$ with $S \neq T$ occur in π, then π contains some transition t with $\theta(t) = \curlyangle_{h+1}$.

Clearly, $\mathcal{A}_{h+1}(P,R)$ satisfies (1) and (3). It is just a technical calculation to verify (2). To prove (4), we show the following two claims:

4a. Let $d \geq 1$. For every $w \in T_{d,h+1}(P,R)$, there is a successful path π in $\mathcal{A}_{h+1}(P,R)$ with the label w and $\Delta(\theta(\pi)) + 1 \leq d$.
4b. Let π be a successful path in $\mathcal{A}_{h+1}(P,R)$ with the label w. We have $w \in T_{\Delta(\theta(\pi))+1,\, h+1}(P,R)$.

Claim (4a) (resp. 4b) proves "$\ldots \leq \ldots$" (resp. "$\ldots \geq \ldots$") in (4). Thus, (4) is a conclusion from (4a) and (4b). For a proof of (4a) and (4b), the reader is referred to [22]. □

Proposition 5. *Let $h \in \mathbb{N}$. We can construct an $(h+2)$-nested distance desert automaton $\mathcal{A}$ over Σ such that for every $w \in \Sigma^*$*

$$\Delta_{\mathcal{A}}(w) + 1 \;=\; \min\big\{d \geq 1 \,\big|\, w \in \sigma(T_{d,h}(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}}))\big\}.$$

In particular, $\mathcal{A}$ has at most

$$\Big(k^{h+1} + \frac{k^h - 1}{k-1} + 1\Big)(n_\sigma - 2m + 1)$$

states where $k = 2^{|Q_{\overline{L}}|}$.

Proof. The initial point of our construction is the automaton $\mathcal{A}_h(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}})$ from Proposition 4. We denote its mapping by $\Delta_{\mathcal{A}_h}$.

We consider the following mapping $\Delta' : \Sigma^* \to \mathbb{N} \cup \{\infty\}$

$$\Delta'(w) := \min\big\{\Delta_{\mathcal{A}_h}(u) \,\big|\, u \in \Gamma^*,\, w \in \sigma(u)\big\}.$$

If $\Delta'(w) \in \mathbb{N}$, then there exists some $u \in \Gamma^*$ such that $w \in \sigma(u)$ and $\Delta_{\mathcal{A}_h}(u) = \Delta'(w)$. By Proposition 4(4), we have $u \in T_{\Delta_{\mathcal{A}}(u)+1,\,h}(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}}) \subseteq T_{\Delta'(w)+1,h}(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}})$. Thus, $w \in \sigma\big(T_{\Delta'(w)+1,\,h}(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}})\big)$.

Conversely, let $d \geq 1$ and assume $w \in \sigma\big(T_{d,h}(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}})\big)$. There is some $u \in T_{d,h}(I_{\overline{L}},\, Q_{\overline{L}} \setminus F_{\overline{L}})$ such that $w \in \sigma(u)$. By Proposition 4(4), we have $\Delta_{\mathcal{A}_h}(u) + 1 \leq d$, and hence, $\Delta'(w) + 1 \leq d$.

To prove the proposition, we construct by Proposition 1 an $(h+2)$-nested distance desert automaton $\mathcal{A}$ which computes Δ'. The bound on the number of states follows from Propositions 1 and 4(2). □

Finally, we can prove Theorem 3:

Proof (Theorem 3(2)(3)). (2) Given $h \in \mathbb{N}$, we can construct $\mathcal{A}$ from Proposition 4. By Proposition 3, $\mathsf{sh}(K_1, K_2, m, \sigma) \leq h$ iff $K_1 \subseteq \sigma(L) = L(\mathcal{A})$ and $\mathcal{A}$ is limited on K_1. The latter condition is decidable by Theorem 1.

As seen in Proposition 2, we have $|Q_{\overline{L}}| \leq 2^{n_2}$. Thus, the number of states of $\mathcal{A}$ lies in

$$n_\sigma \cdot 2^{h \cdot 2^{\mathcal{O}(n_2)}}.$$

The factor n_1 arises in the decision of $K_1 \subseteq L(\mathcal{A})$ and limitedness of $\mathcal{A}$ on K_1.

(3) To compute $\mathsf{sh}(K_1, K_2, m, \sigma)$, we have do decide the existence of a solution and to decide $\mathsf{sh}(K_1, K_2, m, \sigma) \leq h$ for every $0 \leq h \leq 2^{2^{n_2}}$. □

Acknowledgements. The author thanks Filip Murlak and Piotr Sankowski for the invitation to MFCS 2011.

References

[1] Abdulla, P.A., Krcal, P., Yi, W.: R-automata. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 67–81. Springer, Heidelberg (2008)

[2] Abdulla, P.A., Krcal, P., Yi, W.: Universality of R-automata with value copying. In: INFINITY 2008 Proceedings (2008)

[3] Bala, S.: Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 596–607. Springer, Heidelberg (2004)

[4] Bala, S.: Complexity of regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. Theory Of Computing Systems, Special Issue of Selected Best Papers from STACS 2004 39(1), 137–163 (2006)

[5] Berstel, J.: Transductions and Context-Free Languages. B.G.Teubner, Stuttgart (1979)

[6] Cohen, R.S.: Star height of certain families of regular events. Journal of Computer and System Sciences 4, 281–297 (1970)

[7] Colcombet, T.: The theory of stabilisation monoids and regular cost functions. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 139–150. Springer, Heidelberg (2009)

[8] Colcombet, T., Löding, C.: The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 416–430. Springer, Heidelberg (2008)

[9] Colcombet, T., Löding, C.: The non-deterministic mostowski hierarchy and distance-parity automata. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 398–409. Springer, Heidelberg (2008)

[10] Eggan, L.C.: Transition graphs and the star height of regular events. Michigan Math. Journal 10, 385–397 (1963)

[11] Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press, New York (1974)

[12] Hashiguchi, K.: A decision procedure for the order of regular events. Theoretical Computer Science 8, 69–72 (1979)

[13] Hashiguchi, K.: Limitedness theorem on finite automata with distance functions. Journal of Computer and System Sciences 24, 233–244 (1982)

[14] Hashiguchi, K.: Regular languages of star height one. Information and Control 53, 199–210 (1982)

[15] Hashiguchi, K.: Representation theorems of regular languages. Journal of Computer and System Sciences 27(1), 101–115 (1983)

[16] Hashiguchi, K.: Algorithms for determining relative star height and star height. Information and Computation 78, 124–169 (1988)

[17] Hashiguchi, K.: Algorithms for determining relative inclusion star height and inclusion star height. Theoretical Computer Science 91, 85–100 (1991)

[18] Kirsten, D.: Desert automata and the finite substitution problem. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 305–316. Springer, Heidelberg (2004)

[19] Kirsten, D.: Distance desert automata and the star height one problem. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 257–272. Springer, Heidelberg (2004)

[20] Kirsten, D.: A Burnside approach to the finite substitution problem. Theory of Computing Systems, Special Issue of Selected Best Papers from STACS 2004 39(1), 15–50 (2006)
[21] Kirsten, D.: Distance desert automata and star height substitutions. In: Habilitationsschrift, Universität Leipzig, Fakultät für Mathematik und Informatik (2006)
[22] Kirsten, D.: On the complexity of the relative inclusion star height problem. Advances in Computer Science and Engineering 5(2), 173–211 (2010)
[23] Lombardy, S.: Approche structurelle de quelques problèmes de la théorie des automates. PhD thesis, École nationale supérieure des télécommunications, Paris (2001)
[24] McNaughton, R.: The loop complexity of pure-group events. Information and Control 11, 167–176 (1967)
[25] Sakarovitch, J.: Eléments de théorie des automates. Vuibert (2003)
[26] Sakarovitch, J.: Elements of Automata Theory. Encyclopedia of Mathematics and Applications. Cambridge University Press, Cambridge (2009)
[27] Simon, I.: Limited subsets of a free monoid. In: Proceedings of the 19th IEEE Annual Symposium on Foundations of Computer Science, pp. 143–150. IEEE Computer Society Press, Long Beach (1978)
[28] Yu, S.: Regular Languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, Vol. 1, Word, Language, Grammar, pp. 41–110. Springer, Berlin (1997)

Generic Techniques to Round SDP Relaxations

Prasad Raghavendra

Georgia Tech

This talk will survey two general approaches to rounding solutions to SDP relaxations.

'Squish and Solve' Rounding:
This technique of rounding SDPs for constraint satisfaction problems generalizes and unifies a large body of SDP-based algorithms for CSPs. More specifically, it yields a a generic algorithm that for every CSP, achieves an approximation at least as good as the best known algorithm in literature. The generic algorithm is guaranteed to achieve an approximation ratio equal to the integrality gap of an SDP relaxation known to be optimal under Unique Games Conjecture. This is based on joint work with David Steurer.

Rounding Using Correlations:
Despite the numerous applications of semidefinite programming, in all but very few cases, the algorithms rely on arguably the simplest SDP relaxation. Hence the power of stronger semidefinite programming relaxations is not yet harnessed, leaving open the possibility that fundamental optimization problems like Max-Cut and Vertex Cover could be approximated better using SDPs. The dearth of algorithms based on stronger SDP relaxations stems from the lack of general techniques to round these relaxations.

In this work, we present a technique to round SDP hierarchies using the underlying correlations between variables. To demonstrate the technique, we present two applications of the technique, one to the problem of MaxBisection and other to general 2-CSPs on random graphs. This is based on joint works with David Steurer, Ning Tan and Boaz Barak.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, p. 34, 2011.

New Proofs in Graph Minors

Paul Wollan

Sapienza University of Rome

The theory of Graph Minors developed by Robertson and Seymour has had wide reaching consequences in graph theory and theoretical computer science. The main result of the theory, known as the graph minor structure theorem, approximately describes the structure of graphs which do not contain a fixed graph as a minor. The structure theorem is the central piece of many of the algorithmic applications of graph minor theory. Unfortunately, the statement of this structure theorem is quite complex and the only proof until now requires a series of 16 technical papers.

In joint work with Ken-ichi Kawarabayashi and Robin Thomas, we have worked to make these techniques more accessible to the wider community. In this talk, I will discuss a new constructive proof which is much simpler than the original proof. Additionally, the new proof also immediately gives a polynomial time algorithm to find the decomposition and dramatically improves the constants obtained in the theorem.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, p. 35, 2011.

The Least-Core of Threshold Network Flow Games

Yoram Bachrach

Abstract. Network flow games model domains where a commodity can flow through a network controlled by selfish agents. Threshold Network Flow Games (TNFGs) are a form of such games where an agent coalition wins if it manages to send a flow exceeding a certain threshold between a source and a target vertex. Cooperative game theory predicts the agents' actions in such settings with solutions such as the core, the set of stable distributions of a coalition's gains among its members. However, some games have empty cores, so every distribution is inherently unstable. When the core is empty, one must use a more relaxed notion of stability, such as the least-core. We examine several problems regarding the least-core in general and restricted TNFGs.

1 Introduction

Game theory analyzes interactions between selfish agents, with implications ranging from auctions to electronic commerce. A key aspect of agent interaction is *cooperation*. Cooperation between selfish agents requires careful planning as a stable coalition can only be formed if the resulting gains are distributed in appropriately. Game theory suggests possible distributions of the gains, formalized in *solution concepts* such as the core [11], and least-core [14].

We consider Threshold Network Flow Games (TNFGs), which model situations where a commodity (goods, information or traffic) can flow through a network where selfish agents control parts of the network. In TNFGs, agents form coalitions to guarantee a certain bandwidth (or a required flow) from a source to a target node. In our model, a principal offers a single unit of reward if a coalition can send a certain minimal flow between the source and the target, but is not willing to offer a higher reward for achieving a higher flow.

Our Contribution: We examine stable gain distributions in TNFGs. When the TNFG's core is empty , a less demanding solution, the ϵ-core, must be applied to find a "less unstable" distribution. We show that computing the ϵ-core of TNFGs is a hard, but provide polynomial algorithms for unit capacity networks and bounded layer-graphs with bounded integer capacities.

1.1 Preliminaries

A transferable utility coalitional game is composed of a set of n agents, $I = \{1, 2, \ldots, n\}$, and a characteristic function mapping any subset (coalition) of the agents to a rational value $v : 2^I \rightarrow \mathbb{Q}$, indicating the total utility these agents

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 36–47, 2011.

achieve together. We denote the set of all the agents except agent i as $I_{-i} = I \setminus \{i\}$. A game is *monotone* if for all coalitions $C' \subset C$ we have $v(C') \leq v(C)$. In a *simple* game, v only gets values of 0 or 1 ($v : 2^I \rightarrow \{0, 1\}$). We say a coalition $C \subset I$ *wins* if $v(C) = 1$, and say it *loses* if $v(C) = 0$. In simple monotone games we say i is a veto agent if she is present in all winning coalitions, so when $v(C) = 1$ we have $i \in C$. It is easy to see that in such games, i is a veto agent iff $v(I) = 1$ but $v(I_{-i}) = 0$. The characteristic function only defines the gains a *coalition* achieves, but does not say how they are to be distributed among the coalition's agents. An *imputation* $(p_1, \ldots, p_n)$ is a division of the gains of the grand coalition I among the agents, where $p_i \in \mathbb{Q}$, such that $\sum_{i=1}^{n} p_i = v(I)$. We call p_i the payoff of agent i, and denote the payoff of a coalition C as $p(C) = \sum_{i \in C} p_i$. Cooperative Game theory allows choosing the appropriate imputation for the game.

A basic requirement for a good imputation is *individual rationality*, stating that for all agents $i \in C$, we have $p_i \geq v(\{i\})$—otherwise, some agent is incentivized to work alone. Similarly, we say a coalition B *blocks* the payoff vector $(p_1, \ldots, p_n)$ if $p(B) < v(B)$, since B's members can split from the original coalition, derive the gains of $v(B)$ in the game, give each member $i \in B$ its previous gains p_i—and still some utility remains, so each member can get more utility. If a blocked payoff vector is chosen, the coalition is unstable. A prominent solution focusing on stability is the core [11]. The core of a game is the set of all imputations $(p_1, \ldots, p_n)$ that are not blocked by any coalition, so that for any coalition C, we have: $p(C) \geq v(C)$. The core can be empty, so every imputation is blocked by some coalition. In this case we must relax the core's requirements to get a solution. One model for this assumes that coalitions that only have a small incentive to drop-off from the grand coalition would not do so. This solution, which slightly core's inequalities is the ϵ-core [14]. The ϵ-core is the set of all imputations $(p_1, \ldots, p_n)$ such that for any coalition $C \subseteq I$, $p(C) \geq v(C) - \epsilon$.

Given an imputation p, the *excess* of C is the difference between C's value and payoff: $e(C) = v(C) - p(C)$. Under an ϵ-core imputation, the excess of *any* coalition is at most ϵ. If ϵ is large enough, the ϵ-core is non-empty. A natural question is finding the smallest ϵ such that the ϵ-core is non-empty, known as the *least-core*. Given a game G we consider $\{\epsilon | \text{the } \epsilon\text{-core of G is not empty}\}$. It is easy to see that this set is compact, and has a minimal element ϵ_{min}. The least-core of the game G is the ϵ_{min}-core of G.

2 The Least-Core in TNFGs

Network domains give rise to natural game theoretic problems. A flow network consists of a directed graph $G = \langle V, E \rangle$ with capacities on the edges $c : E \rightarrow \mathbb{Q}+$, a distinguished source vertex $s \in V$ and a sink vertex $t \in V$. A flow through the network is a function $f : E \rightarrow \mathbb{Q}+$ which obeys the capacity constraints and conserves the flow at each vertex (except for the source and sink), meaning that the total flow entering a vertex must equal the total flow leaving that vertex. The value of a flow f (denoted $|f|$) is the net amount flowing out of the source

(and into the sink). A maximum flow is a valid flow f^* whose magnitude $|f^*|$ is maximal of all possible flows, so for any valid flow f' we have $|f^*| \geq |f'|$.

A *threshold network flow domain* consists of a network flow graph $G = \langle V, E \rangle$, with capacities on the edges $c : E \to \mathbb{Q}$, a source vertex s, a target vertex t, and a set I of agents, where agent i controls the edge $e_i \in E$, and flow threshold k. A coalition $C \subseteq I$, controls the edges $E_C = \{e_i | i \in C\}$. Given a coalition C we denote by $G_C = \langle V, E_C \rangle$ the induced graph where the only edges are the edges that belong to the agents in C. We denote by f_C the maximum flow between s and t in G_C. In a *threshold network flow game (TNFG)*, a coalition C wins if it can achieve a k-flow between s and t and loses otherwise. The characteristic function of the game, v, is:

$$v(C) = \begin{cases} 1 & \text{if } f_C \geq k, \text{ so } E_C \text{ allows a flow of } k \text{ from } s \text{ to } t; \\ 0 & \text{otherwise;} \end{cases}$$

TNFGs are simple games, since v can only get a value of 0 or 1. A *Cardinal* Network Flow Game (CNFG) is defined in a similar manner, except the value of a coalition is the maximum flow it can send from s to t. In CNFGs the characteristic function is $v_{CNFG}(C) = f_C$.

We consider stable payoff distributions in TNFGs. Given a reward for allowing a certain flow between the source and the target, the coalition must decide how to distribute this reward to the agents who own the links. We wish to distribute the gains in a way that minimizes the incentives of any subset of agents to split off from the main coalition and form their own network. The appropriate game theoretic solution for this is the core. TNFGs are *simple* and *monotone* games, where the core is closely related to veto players. A folklore theorem (see [15]) states that in simple monotone games, if there are no veto agents then the core is empty, and if there are veto agents then the core is the set of imputations that distribute all the gains solely to the veto agents. Thus, computing the core in such games simply requires returning a list of veto players in that game, and checking if the core is non-empty simply requires testing if the game has any veto players. Unfortunately, TNFGs rarely have veto agents. If there are two disjoint edge subsets that allow a k-flow, the core is empty [1].

Computational problems in TNFGS: we now define several important problems regarding TNFGs: IMP-C, IMP-EC, IMP-LC, CNE, ECNE, LCV, ME. The core, ϵ-core and the least-core are all solution concepts that may contain more than one possible imputation (or even, an infinite number of imputations). One key question given a TNFG is testing whether a certain imputation $p = (p_1, \ldots, p_n)$ is stable (i.e. whether it is in the core, ϵ-core or the least-core). We denote these problems for the core, $\epsilon - core$ and least-core as IMP-C, IMP-EC and IMP-LC. The core and ϵ-core may be empty, so another problem is testing for their emptiness, which we denote as CNE (for core non-emptiness)

[1] Consider a unit capacity graph, and a target flow of one unit. In this case, if there are two or more edge disjoint paths from the source to the target, then the core is empty.

and ECNE (ϵ-core non-emptiness). Another problem is LCV (least-core value), computing the ϵ_{min} value of the least-core (i.e the minimal ϵ such that the ϵ-core is non-empty). A solution for IMP-C and CNE in TNFGs was given in [5], using a simple polynomial method for finding veto agents. Since the core in TNFGs may be empty, we propose using the relaxed stability notions of the ϵ-core and the least-core. We show the above problems are closely related to the problem ME (maximal excees), where we are given a TNFG with characteristic function v and an imputation $p = (p_1, \ldots, p_n)$ and are asked to compute the maximal excess of any coalition: $\max_{C \subseteq I} v(C) - p(C)$. Both LCV and ME require outputting a rational value, but we consider the decision version of these problems, where in the input we are given a rational number q and are asked whether the solution to the LCV or ME problem is greater than the specified value.

We consider the relation between TNFGs and another class of games, Weighted Voting Games (WVGs). We show that the problem of LCV is NP-hard in TNFGs, using a similar hardness result [8] for WVGs. WVGs are simple games with n agents $I = \{1, 2, \ldots, n\}$, where each agent i has a weight w_i. Given a coalition $C \subseteq I$, we denote $w(C) = \sum_{i \in C} w_i$. The game has a threshold k, and a coalition C wins if $w(C) \geq k$, and loses otherwise. We use the fact that TNFGs can easily express any WVG to obtain several negative results.

Theorem 1. *LCV in TNFGs is NP-hard*

Proof. Elkind et al. show [8] that LCV is NP-hard in WVGs. We reduce an instance of LCV in a WVG to an instance of LCV in a TNFG. Given the WVG instance with weights $w_1, \ldots, w_n$, we construct a TNFG. The source s is connected to the target t by n edges, so that edge $e_i = (s, t)$ has capacity c_i equal to the weight w_i in the WVG. Any coalition C achieves a flow of $\sum_{\{i|e_i \in E\}} c_i = \sum_{i \in C} w_i = w(C)$. The TNFG has the same threshold as the WVG threshold. Under these thresholds, coalition C wins in the TNFG iff it wins in the original WVG. The original and reduced instances define the same game (have an identical characteristic function) and have the same least-core value, so LCV in a TNFG is NP-hard as well. □

We now provide a negative result for the ϵ-core, and then provide positive results for restricted TNFGs. We begin with the IMP-EC problem where we are given a TNFG, an imputation $p = (p_1, \ldots, p_n)$ and a value ϵ, and are asked whether p is in the $\epsilon - core$ of the game.

Theorem 2. *IMP-EC and ME are coNP-complete*

Proof. IMP-EC is *in* coNP, as one can compute the excess of a given coalition using any polynomial max-flow algorithm and test if it is higher than ϵ or not. The results in [8] show that testing if an imputation is in the ϵ-core in WVGs is NP-Hard. Using construction of Theorem 1, we express any WVG as a TNFG. An imputation in the constructed TNFG is in the ϵ-core iff it is in the ϵ-core of the original WVG. Finally, reducing IMP-EC to ME is trivial: by definition an imputation is in the ϵ-core if the maximal excess is at most ϵ. □

ME is coNP-complete in general graphs, so we examine restricted cases. One important restricted case is *unit-capacity* graphs, where all the edges have a capacity of 1. Several results for *cardinal* network flow games (CNFGs) in unit-capacity graphs are given in [13]. We provide positive results for *threshold* network flow games (TNFGs). One result regarding CNFGs on unit-capacity graphs is that they always have non-empty cores. However, is not the case in TNFGs, as shown in the following example.

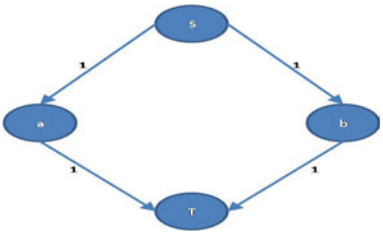


Fig. 1. Example of a unit capacity TNFG with empty core

Consider distributing a total reward of 1 among the edges in Figure 1. Under any such imputation, either the path $A = (s, a), (a, t)$ or the path $B = (s, b), (b, t)$ has a payoff of at most 0.5. Thus, the excess of at least one coalition C_A or C_B must be at least 0.5, and for any $\epsilon > 0.5$, the ϵ-core is empty, so the core is also empty. Note that the 0.5-core is not empty, as the imputation giving each edge 0.25 has a excess of exactly 0.5 for the coalitions C_A and C_B, and any other winning coalition has a lower excess.

We show that ME *is solvable in polynomial time* for *unit-capacity* graphs, and use this to compute the least-core for unit-capacity TNFGs. We first examine the relation between ME and the min-cost-flow problem [17], defined as follows. We are given a network flow graph, a target flow value d and a cost function $a : E \rightarrow \mathbb{Q}$, mapping each edge to the cost of sending a *unit* of flow through that edge. Given a flow $f(e)$ through edge e, the cost of the flow through the edge is $f(e) \cdot a(e)$. We define the cost of a flow f as $a(f) = \sum_{e \in E} f(e) \cdot a(e)$. We are asked to find a d-flow through the network, of minimal cost.

The Min-Cost Flow uses a pricing scheme where the cost of an edge is proportional to the flow sent through the edge. We call this pricing scheme *linear* pricing, as the cost of an edge increases linearly in the flow. A different pricing scheme is requiring a payment for *using* an edge, so the cost of an edge is $a(e)$ if $f(e) > 0$, and the cost is 0 if $f(e) = 0$. We call this pricing scheme the *All-Or-Nothing scheme* (AON pricing), as it either takes the full cost of an edge (for edges e with flow $f(e) > 0$), or takes nothing at all (for edges e for which the flow is $f(e) = 0$). The problem of Min-Cost Flow under AON-pricing is defined as follows. We are given a network flow graph, a target flow value d and a cost function $a : E \rightarrow \mathbb{Q}$, mapping each edge to the cost of sending *any non-zero* flow through that edge. Given a flow $f(e)$ through edge e, the AON-cost of the flow through e is $n(e) = a(e)$ if $f(e) > 0$ and $n(e) = 0$ if $f(e) = 0$. We define the

AON-cost of a flow f as $a^{AON}(f) = \sum_{e \in E} n(e)$. We are asked to find a d-flow through the network, f, of minimal AON-cost (i.e. to minimize $a^{AON}(f)$).

Lemma 1. *ME and AON-Min-Cost-Flow are equivalent problems when we set the AON-Min-Cost-Flow edge costs to be the edge payoff in the ME imputation: $a(e) = p_e$ for any edge e.*

Proof. Consider a k-flow f sending flow only through edges C_f. The payoff $p(C_f)$ under the imputation is exactly the AON-cost of f, $a^{AON}(f)$. ME finds the minimal $p(C_f)$ where f is a k-flow, and AON-Min-Cost finds the minimal $a^{AON}(f)$ where f is a k-flow, so their solution is the same. □

Theorem 2 shows that ME is coNP-complete, so due to Lemma 1, AON-Min-Cost-Flow is also coNP-complete. However, the similar Min-Cost Flow problem (under *linear* pricing) is solvable in polynomial time [17]. We show that for *unit-capacity* graphs, AON-Min-Cost is solvable in polynomial time. We use a lemma from [17] used in the analysis of the Min-Cost Flow problem, regarding augmenting min-cost flows [2]. A min-cost flow is a flow f, such that for any other flow f' such that $|f| = |f'|$ we have $a(f') \geq a(f)$, i.e. it is impossible to achieve a flow of equal magnitude with a lower cost (under linear pricing).

Lemma 2. *If f is a min-cost flow, then any flow obtained from f by augmenting along a minimum-cost path is also a min-cost flow.*

We show that in unit-capacity graphs, one of the optimal k-flows is an all-or-nothing flow. An *All-Or-Nothing Flow (AON Flow)* is a flow f such that any edge is used up to its maximal capacity, or completely unused, so for any edge e we either have $f(e) = c(e)$ or we have $f(e) = 0$.

Theorem 3. *For any unit-capacity graph G and an achievable integer flow threshold k, there is a min-cost k-flow f (under* linear *pricing) that is an AON-flow, i.e. either fully uses an edge, sending the maximal possible flow of 1 unit, or does not use the edge at all, sending a flow of 0 through it.*

Proof. One possible way of finding a min-cost flow is starting with an empty flow, and repeating the process of augmenting the flow through a min-cost augmenting path until the target flow threshold of k is reached [17]. Due to Lemma 2, after each augmentation we end up with a min-cost flow. The initial residual graph has capacities of 1 on all the edges. Thus, the first augmenting path has a capacity of one unit on all the edges along it. The first augmentation fully uses the edges along a certain path. Thus, in the residual graph after the augmentation, all the edges have either a capacity of 1, or -1 for the reverse direction. This, in turn, means that the next augmenting path would also have a capacity of 1 all along it. Thus the next augmentation either fully uses some edges, or completely stops

[2] For a complete introduction to the min-cost network flow problem we refer the reader to [17]. A min-cost augmenting path is simply a minimal cost s-t path in the residual graph. We use the lemma to prove certain properties of min-cost flow solutions.

using some previously unused edges. Thus, each augmentation is done through a path of capacity one, and after each augmentation some edges are changed from being unused to completely used, or from being fully used to completely unused. Since the flow's magnitude increases by one unit after each step, the process terminates after k such steps, with a flow such that each edge is either fully used or completely unused. □

Lemma 3. *In unit-capacity graphs, under the same cost function $a(e)$, the AON-cost of a flow f, $a^{AON}(F)$, is bounded from below by the linear cost of a flow, $a(f)$, i.e $a(f) \leq a^{AON}(f)$*

Proof. The graph is a unit-capacity graph, so for any edge $f(e) \leq 1$. We note that the cost of each edge only increases when we use AON-pricing. For used edges the full cost is incurred even if the edge is only partially used, and for unused edges both pricing schemes charge nothing: $a(f) = \sum_{e \in E} a(e) \cdot f(e) = \sum_{e \in E | f(e) > 0} a(e) \cdot f(e) + \sum_{e \in E | f(e) = 0} a(e) \cdot f(e) \leq \sum_{e \in E | f(e) > 0} a(e) \cdot 1 + \sum_{e \in E | f(e) = 0} a(e) \cdot f(e) = a^{AON}(f)$ □

Lemma 4. *In unit-capacity graphs, under the same cost function $a(e)$, the cost of an AON-flow is the same under linear-pricing and AON-pricing. In other words, if f is a AON-flow, then $a(f) = a^{AON}(f)$.*

Proof. Similarly to the analysis in Lemma 3 we decompose the flow to used and unused edges, and note that the cost for each edge is the same under the two pricing models. Under both models, an unused edge incurs no cost, and a fully used edge incurs a cost of $a(e)$, since for a AON-flow if $f(e) > 0$ then $f(e) = c(e) = 1$. Thus the following holds: $a(f) = \sum_{e \in E} a(e) \cdot f(e)$ $= \sum_{e \in E | f(e) > 0} a(e) \cdot f(e) + \sum_{e \in E | f(e) = 0} a(e) \cdot f(e) = \sum_{e \in E | f(e) > 0} a(e) \cdot 1 + \sum_{e \in E | f(e) = 0} a(e) \cdot 0 = a^{AON}(f)$ □

Theorem 4. *In unit-capacity graphs, under a certain cost function $a(e)$, a min-cost k-flow (under linear pricing) is also a AON-min-cost k-flow.*

Proof. Due to Theorem 3, a certain AON-flow f^* is a min-cost k-flow, under linear pricing. Thus, for any other k-flow f' we have $a(f') \geq a(f^*)$. Due to Lemma 4, since f^* is a AON-flow, then $a(f^*) = a^{AON}(f^*)$. Due to Lemma 3, for any k-flow f we have $a^{AON}(f) \geq a(f)$. Thus for any k-flow f, $a^{AON}(f) \geq a(f) \geq a(f^*) = a^{AON}(f^*)$, so f^* is a AON-min-cost k-flow. □

Corollary 1. *ME is in P for unit-capacity TNFGs.*

Proof. Lemma 1 shows that ME and AON-Min-Cost-Flow are equivalent. Due to Theorem 4 solving AON-Min-Cost simply requires solving Min-Cost Flow (under the linear pricing model). Polynomial algorithms for solving Min-Cost-Flow that rely on the all-or-nothing augmenting paths used in Theorem 3 are given in [17]. Note that the proof in Theorem 3 is constructive, and allows finding such a Min-Cost Flow which is also an AON-flow for unit capacity graphs, thus solving the ME problem for such graphs. □

Theorem 5. *ECNE is in P for unit-capacity TNFGs.*

Proof. We consider an exponential-size program linear program (LP) for computing an imputation in the ϵ-core. Consider all 2^n possible coalitions over n players, $C_1, \ldots, C_{2^n}$. The ϵ-core's definition is a LP over the variables $p_1, \ldots, p_n$, with a constraint for each coalition:

Table 1. Exponential LP for the ϵ-core

Feasible $(p_1, \ldots, p_n)$ **s.t.:**	
$\forall C : v(C_1) - \sum_{i \in C} p_i \leq \epsilon$	(Coalition constraints)
$\sum_{i=1}^{n} p_i = v(I)$	(Imputation constraint)

A violated constraint is a coalition C_j, such that $v(C_j) - \sum_{i \in C_j} p_i > \epsilon$. TNFGs are simple games, so any losing coalition C_j cannot yield a violated constraint, as $v(C_j) = 0$. Thus the violated constrain is due to a winning coalition such that $p(C_j) < v(C_j) - \epsilon = 1 - \epsilon$. The algorithm of Corollary 1 returns a maximal excess coalition C, where $e(C) = v(C) - \sum_{i \in C} p_i$ is maximal. If $e(C) > \epsilon$, we have a violating constraint: $v(C) - \sum_{i \in C_2} p_i > \epsilon$. If $e(C) \leq \epsilon$, then all the constrains hold: $e(C)$ is the maximal excess, so for any $C_j \neq C$ we have $d(C_j) = v(C_j) - \sum_{i \in C_j} p_i \leq e(C) \leq \epsilon$. Thus this algorithm can serve as a separation oracle for the above LP, and we can solve it in polynomial time. Thus, given ϵ we can, in polynomial time, compute an imputation in the ϵ-core if one exists, or reply that no such imputation exists. □

Corollary 2. *LCV is in P for unit-capacity TNFGs.*

Proof. Due to Theorem 5, for a given ϵ we can test if the ϵ-core is non-empty. We can perform a binary search on the minimal value of ϵ such that the ϵ-core is not empty. Since the maximal value of a coalition in TNFGs is 1, we search for ϵ between 0 and 1. This finds the minimal ϵ such that the ϵ-core is not empty, up to any given degree of accuracy. We can then use the algorithm from Theorem 5 to return a least-core imputation. Alternatively, we can consider program in Theorem 5's proof as a program over the variables $p_1, \ldots, p_n, \epsilon$ and set the target function to be $\min \epsilon$ (rather than being a feasibility program). □

We propose a polynomial algorithm for computing the least-core of a TNFG with *non-unit capacities* under restrictions on the graph's structure. A graph $G = \langle V, E \rangle$ is a *layer graph* with w layers if we can partition the vertices into layers $L_0, L_1, L_2, \ldots, L_w$ (where each L_i is a subset of vertices $L_i \subseteq V$ and the L_i's are disjoint so if $i \neq j$ then $L_i \cap L_j = \emptyset$) such that edges only occur between vertices of consecutive layers. In other words, we require that if $(u, v) \in E$ then we have $u \in L_i$ and $v \in L_{i+1}$ for some $0 \leq i < w$. A layer graph is *h-bounded* if each layer L_i has at most h vertices (so $|L_i| \leq h$ for any $0 \leq i \leq w$). For flow networks we denote the source s as the first layer so $L_0 = \{s\}$ and the target t as the last layer $L_w = \{t\}$ (in layer graphs vertices other than s or t

in the source/target layers do not influence the maximum flow). The bound h only applies to the number of vertices in each layer so the number of layers is unbounded. A flow network has *c-bounded integer capacities* if all edge capacities are integers bounded from above by c, so for any $e \in E$, $c(e) \in \{0, 1, 2, \ldots, c\}$.

In Lemma 1 we noted that solving ME is equivalent to AON-Min-Cost-Flow, where for any edge e we have $a(e) = p_e$ (edge costs are the payoffs in the ME imputation). We show that AON-Min-Cost-Flow can be solved in polynomial time in bounded layer-graphs with bounded integer capacities.

Theorem 6. *AON-Min-Cost-Flow is in P for bounded layer-graphs with bounded integer capacities.*

Proof. Consider a layer-graph with $w+1$ layers $L_0, L_1, \ldots, L_w$. Denote the source s as $L_0 = \{s\}$, the target as $L_w = \{t\}$, the bound on the number of vertices in each layer as h and the maximal capacity as c. Denote by $p(e)$ the AON-price of an edge e in the AON-Min-Cost-Flow input. The maximal achievable flow is at most $b = h \cdot c$ as there are at most h edges going out of s and each has a capacity at most c. The border between any two layers forms an $s-t$-cut in the graph, so the incoming flow into any vertex is at most $b = h \cdot c$, the bound on the flow leaving s. Given a layer $L_i \subset V$ of l_i vertices $v_1^i, v_2^i, \ldots, v_{l_i}^i$ we define an *incoming flow configuration* for that layer, representing the amount of flow entering each vertex in that layer. An *incoming flow configuration* for layer L_i is a sequence $(f_1, f_2, \ldots, f_{l_i})$ of integers where $0 \leq l_j \leq b$. A layer has at most h vertices and each has a possible incoming integer flow of $0 \leq f_j \leq b$, so there are at most $(b+1)^h = (h \cdot c + 1)^h$ possible flow configurations for any layer L_i. Denote the bound on the possible flow configurations for any layer as $q = (b+1)^h = (h \cdot c + 1)^h$. Since h and c are constants, q is constant.

We provide an algorithm to compute the minimal cost to achieve a d-flow to the target. It operates by iterating through the layers and computing the minimal cost to achieve a specific flow configuration. We say an edge $e = (u, v)$ occurs before layer L_i if $u \in L_a$ and $v \in L_b$ where $a \leq i$ and $b \leq i$ (since no edges are allowed *within* a layer, this also means that $a \leq i - 1$). We say a flow configuration $f = (f_1, f_2, \ldots, f_{l_i})$ for L_i is achievable with price p if there is a subset of edges E' that all occur before layer L_i with total cost $\sum_{e \in E'} p(e) \leq p$ which allows *simultaneously* sending a flow of f_1 to v_1^i, f_2 to v_2^i and so on through a flow of f_{l_i} to $v_{l_i}^i$ using only the edges of E'.

Consider a flow configuration $f^i = (f_1^i, f_2^i, \ldots, f_{l_i}^i)$ for L_i, a flow configuration $f^{i+1} = (f_1^{i+1}, f_2^{i+1}, \ldots, f_{l_{i+1}}^{i+1})$ for layer L_{i+1} and an edge subset E' that occur between L_i and L_{i+1}. Suppose that p is the minimal price such that f_i is achievable for layer L_i with price p. Given f_i and E' we may achieve various flow configurations in L_{i+1}: if each vertex in L_i has an incoming flow as specified in f_i, we may route this flow through E' to the vertices in L_{i+1} in several ways. It is possible to check if f^{i+1} is achievable from f^i through the edges E' in polynomial time. To do this, we create a small flow network with a source s connected to each of the vertices in L_1 through edge with capacities as specified

in f_i (i.e. s is connected to $v_j \in L_i$ through an edge of capacity f_j^i), maintain the edges E' connecting L_i and L_{i+1} with their current capacities and connect each vertex in L_{i+1} to t with capacities as specified in f^{i+1} (i.e. $v_j \in L_{i+1}$ is connected to t through an edge of capacity f_j^{i+1}). If this created network allows a flow of $\sum_{j=1}^{l_{i+1}} f_j^{i+1}$ then given flow configuration f^i for L_i we can achieve the flow configuration f^{i+1} for layer L_{i+1} by using only the edges E' between L_i and L_{i+1}. This check can be done in polynomial time using a max-flow algorithm. The total cost of the edge subset E' is $p(E') = \sum_{e \in E'} p_e$, so if f^i is achievable with cost p then f^{i+1} is achievable with cost of at most $p+p(E')$ by simply adding the edges E' and routing the flow through them appropriately. We denote by $achievable - flow(f_j^i, E', f_k^{i+1})$ the algorithm that checks whether given a flow configuration f_j^i for layer i and an edge subset E' (occurring between L_i and L_{i+1}) we can achieve a flow configuration f_k^{i+1} for layer L_{i+1}.

For each layer L_i, our algorithm maintains a bound p_j^i for the minimal cost to achieve any of the possible q flow configurations (for $j \in \{1, \ldots, q\}$). We denote the set of all flow configurations for layer i as Q^i. Layer L_w contains the single target vertex so $L_w = \{t\}$ and a valid flow configurations for L_w is simply the possible flow values to the target vertex. We denote the flow configuration where a flow of d enters the target vertex as f_d so we denote the cost of achieving a flow of k is $p_{f_d}^w$. Layer L_0 contains the single source vertex so $L_0 = \{s\}$ so similarly a valid flow configuration for L_0 is simply the possible flow values to the target vertex. We denote the cost of a flow configuration where s has an "incoming" flow of x as $p_{f_x}^0$. Since s is the source it may achieve an any "incoming" flow at a zero cost. When iterating through layer L_i our algorithm relaxes the bounds by examining any possible edge subset E' occurring between L_i and L_{i+1}. We denote by q_j^i the j'th flow configuration for the $i'th$ layer in the graph. There are at most q flow configurations for each layer, so $1 \leq j \leq q$.

1. For $i = 1$ to w:
   (a) For each $q_j^i \in Q^i$: $p_j^i = \infty$
   (b) For $x = 0$ to $b = h \cdot c$: $p_{f_x}^0 = 0$
2. For $i = 1$ to w:
   (a) For each $q_j^i \in Q^i$, each $q_k^{i+1} \in Q^{i+1}$ and each $E' \subseteq E_{L_i}$:
      i. If $achievable - flow(q_j^i, E', q_k^{i+1})$: $p_k^{i+1} = \min(p_k^{i+1}, p_j^i + p(E'))$
3. return $p_{f_d}^w$

An induction through the layers shows that the algorithm correctly computes p_j^i for any layer i and configuration j for layer i. It returns $p_{f_d}^w$, the minimal cost for a d-flow to the t, i.e. the AON-Min-Cost-Flow solution. The update loop's runtime is $w \cdot |Q^i| \cdot |Q^{i+1}| \cdot 2^{|E_{L_i}|}$. For any i the configuration number $|Q^i|$ is bounded by the constant q. Also, $|E_{L_i}|$ is the number of edges between L_i and L_{i+1}, bounded by the constant h^2. Thus the runtime is bounded by $w \cdot q^2 \cdot 2^{h^2}$ or by $g \cdot w$ where g is a constant, so this is a polynomial algorithm. The runtime

is bounded by $w \cdot q^2 \cdot 2^{h^2}$ where $q = (b+1)^h = (h \cdot c+1)^h$, so it can be expressed as $O(f(h,c) \cdot w)$ so this is a *fixed parameter tractable* (FPT) algorithm[3]. □

The equivalence of ME and AON-Min-Cost-Flow obtains the following.

Corollary 3. *ME is in P for TNFGs over bounded layer-graphs with bounded integer capacities.*

Theorem 7. *ECNE and LCV are in P for TNFGs over bounded layer-graphs with bounded integer capacities.*

Proof. Similarly to Theorem 5, we use the ME algorithm of Corollary 3 as a separation oracle for the ϵ-core LP, so ECNE can be solved in polynomial time. Similarly to Theorem 2, we perform a binary search on the minimal value of ϵ that makes the ϵ-core non-empty, so LCV is solvable in polynomial time. □

3 Related Work and Conclusions

We examined computing the core,ϵ-core and the least-core in TNFGs. The core was introduced in [11], and the ϵ-core and least-core relaxations for the case where it is empty were presented in [14]. An early treatment of computational aspects of such solutions was given in [7]. One restricted domain that has received much attention is the weighted voting games (WVG) model of decision making. Computing the least-core of WVGs was analyzed in [9].

We focused on solving TNFGs. The similar CNFG model was introduced in [13], showing that unit capacity CNFGs have non-empty cores. Further core related questions in CNFGs were studied in [10]. Other network task games, such as network formation, connectivity and security were studied in [18,12,6,3]. Despite the superficial similarity between CNFGs and TNFGs, they have very different computational and game theoretic properties. TNFGs were introduced in [4], which considered the Banzhaf index. An extended version of that paper [5] also considered computing the core of TNFGs. Further work [16] focused on solving TNFGs using the core-related Cost of Stability introduced in [2,1] for weighted voting games and general cooperative games.

We showed that, as opposed to CNFGs, in TNFGs the core can be non-empty even if the graph has unit-capacities, and that for general graphs, even finding the maximal excess of a coalition under an imputation (the ME problem) is coNP-complete. We provided polynomial-time algorithms for computing the ϵ-core and least-core for unit-capacity graphs and bounded layer-graphs with bounded integer capacities. Some questions remain open. First, one might examine other game theoretic solutions in TNFGs. Also, it would be interesting to examine other restrictions that allow tractably computing the least-core.

[3] Fixed parameter tractability with parameter α (α-FPT) means the running time is $f(\alpha) \cdot p(n)$ where $f(\alpha)$ may be any function (e.g. an exponential or even superexponential function), and $p(n)$ is a polynomial function in the input length n.

References

1. Bachrach, Y., Elkind, E., Meir, R., Pasechnik, D., Zuckerman, M., Rothe, J., Rosenschein, J.S.: The cost of stability in coalitional games. In: Mavronicolas, M., Papadopoulou, V.G. (eds.) SAGT 2009. LNCS, vol. 5814, pp. 122–134. Springer, Heidelberg (2009)
2. Bachrach, Y., Meir, R., Zuckerman, M., Rothe, J., Rosenschein, J.S.: The cost of stability in weighted voting games. In: AAMAS 2009 (2009)
3. Bachrach, Y., Porat, E.: Path disruption games. In: AAMAS 2010 (2010)
4. Bachrach, Y., Rosenschein, J.S.: Computing the Banzhaf power index in network flow games. In: AAMAS 2007 (2007)
5. Bachrach, Y., Rosenschein, J.S.: Power in threshold network flow games. Autonomous Agents and Multi-Agent Systems (2009)
6. Bachrach, Y., Rosenschein, J.S., Porat, E.: Power and stability in connectivity games. In: AAMAS 2008 (2008)
7. Deng, X., Papadimitriou, C.H.: On the complexity of cooperative solution concepts. Math. Oper. Res. 19(2), 257–266 (1994)
8. Elkind, E., Goldberg, L.A., Goldberg, P.W., Wooldridge, M.: Computational complexity of weighted threshold games. In: AAAI 2007 (2007)
9. Elkind, E., Pasechnik, D.: Computing the nucleolus of weighted voting games. In: SODA 2009 (2009)
10. Fang, Q., Fleischer, R., Li, J., Sun, X.: Algorithms for core stability, core largeness, exactness, and extendability of flow games. Computing and Combinatorics, 439–447 (2007)
11. Gillies, D.B.: Some theorems on n-person games. PhD thesis (1953)
12. Goyal, S., Vega-Redondo, F.: Network formation and social coordination. Games and Economic Behavior 50(2), 178–207 (2005)
13. Kalai, E., Zemel, E.: Generalized network problems yielding totally balanced games. Operations Research 30, 998–1008 (1982)
14. Maschler, M., Peleg, B., Shapley, L.S.: Geometric properties of the kernel, nucleolus, and related solution concepts. Math. Oper. Res. (1979)
15. Osborne, M.J., Rubenstein, A.: A Course in Game Theory. The MIT Press, Cambridge (1994)
16. Resnick, E., Bachrach, Y., Meir, R., Rosenschein, J.S.: The cost of stability in network flow games. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 636–650. Springer, Heidelberg (2009)
17. Tarjan, R.E.: Data Structures and Network Algorithms. Society for Industrial and Applied Mathematics (1983)
18. van den Nouweland, A.: Models of network formation in cooperative games. Cambridge University Press, Cambridge (2005)

Adhesivity Is Not Enough: Local Church-Rosser Revisited*

Paolo Baldan[1], Fabio Gadducci[2], and Pawel Sobociński[3]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova
[2] Dipartimento di Informatica, Università di Pisa
[3] School of Electronics and Computer Science, University of Southampton

Abstract. Adhesive categories provide an abstract setting for the double-pushout approach to rewriting, generalising classical approaches to graph transformation. Fundamental results about parallelism and confluence, including the local Church-Rosser theorem, can be proven in adhesive categories, provided that one restricts to linear rules. We identify a class of categories, including most adhesive categories used in rewriting, where those same results can be proven in the presence of rules that are merely left-linear, i.e., rules which can merge different parts of a rewritten object. Such rules naturally emerge, e.g., when using graphical encodings for modelling the operational semantics of process calculi.

Keywords: Adhesive and extensive categories, double-pushout rewriting, local Church-Rosser property, parallel and sequential independence.

Introduction

The strength of graph transformation formalisms in the specification of distributed and concurrent systems lies on the relative ease of use, due to their visual nature. However, equally relevant is the fact that these formalisms often come equipped with a rich theory of concurrency, including confluence properties that can be pivotal in developing suitable verification techniques.

Focusing on the double pushout (DPO) approach to graph transformation, *parallel* and *sequential independence* are central properties in the corresponding concurrency theory. While the former is essentially a local confluence property, the latter identifies suitable conditions for establishing whenever two rewrite steps, even though performed in sequence, do not interfere with each other and thus can be potentially applied in any order (as well as concurrently).

The local Church-Rosser theorem tightly connects parallel and sequential independence: two sequentially independent steps can be applied to the same start graph, resulting in a pair of parallel independent steps; analogously, two parallel independent steps can be sequentialised in any order. This allows for defining concurrent rewrites as equivalence classes up to *shift equivalence* [2], identifying (as for the better-known permutation equivalence of λ-calculus) those rewrite sequences that differ for the scheduling of independent steps.

* Supported by the MIUR project `SisteR` and the University of Padua project `AVIAMO`.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 48–59, 2011.

Rewriting over *adhesive categories* (ACs) was proposed as an abstraction of the DPO approach to graph transformation. Many well-known categories with graph-like objects are adhesive. Moreover, since ACs subsume many properties of e.g. HLR categories [5], several results about parallelism and confluence, including the local Church-Rosser theorem, can be proven with no additional assumptions if one restricts to linear rules, i.e., to spans of monos [16].

The restriction to linear rules is common in the DPO literature. It has been folklore, though, that local Church-Rosser should "usually" hold for left-linear rules, i.e. where only the left leg of the span is required to be injective. This has been established within the concrete setting of a category of labelled directed graphs [9,13] and, in the context of HLR rewriting, sufficient axioms were exhibited in [7]. Those axioms have not been shown to be preserved by common operations such as functor category and slice and thus, to the best of our knowledge, this result has not been proved for a class of categories that would include, say, the category of hyper-graphs **HGraph** or of graphs with second order edges (both presheaf categories, thus adhesive) where linear rules may not suffice.

Whenever distinct parts of the state are to be fused as a result of a transformation, it becomes necessary to use a non-injective morphism as the right-hand side of the rule. A notable example is given by the encodings of process calculi as graph transformation systems, where the exchange of channel names and the creation of connections are modeled as node fusions [10]. In order to extend the results about independence and parallelism to left-linear rules, adhesivity does not appear to be enough. Roughly, while the distinguishing feature of ACs is that pushouts along monos are well behaved, for non-linear rules some of the pushouts involved in the technical lemmas are not necessarily of this kind.

Instead of looking for an *axiomatic* characterisation of the class of categories in which the result could be proven, this paper takes a different approach. First we show that the local Church-Rosser theorem for left-linear rules holds in any AC **C** with a strict initial object (hence extensive [16, Lemma 4.1]), where monos are coproduct injections and all pushouts exist and are stable under pullback, an example being **Set**, but not, for example, **Graph**. Then, we note that the technical results used in the proof of the theorem mention only pushouts and pullbacks; hence whenever they hold in **C**, they hold in any functor category over **C** as well as in any slice or coslice category over **C**. Since these operations can be iterated, the result holds in a family of categories that contains most known examples of ACs, such as, in particular, **Graph** and other graph-like categories.

An analogous result can be proven for quasi-adhesivity: a local Church-Rosser theorem for rules where the left-leg is a regular mono holds in any quasi-adhesive category **C** with a strict initial object (hence extensive [16, Lemma 6.3]), where *regular* monos are coproduct injections and pushouts are stable under pullback. Again, this extends to any category obtained from this class by iterating functor, slice and coslice category constructions (although here the presence of equalisers in the base category is needed [16, Lemma 6.6]). A notable example in the base class is **Inj**, the category of injective functions in **Set**. By the closure properties,

the category **IGraph** of *graphs with interface* (expressible as $\mathbf{Inj}^{\bullet \rightrightarrows \bullet}$) can be recast, thus subsuming the proposal for their rewriting [11].

The paper has the following structure. We first recall the basics of DPO rewriting in adhesive (and quasi-adhesive) categories and the local Church-Rosser theorem for linear rules. Then, we cut directly to our main contribution, providing a class of categories where the local Church-Rosser theorem holds for left-linear rules. Afterwards we provide an application of our results for the rewriting of graphs with interface, used in graphical encodings of nominal calculi. We conclude summing up our contribution and laying the plans for future work.

1 Background

In this section we introduce the basics of the double-pushout (DPO) approach to rewriting [2,4], including the notion of sequential and parallel independence. We also introduce adhesive categories [16] as an abstract setting for DPO rewriting.

1.1 DPO Rewriting

Hereafter $\mathbf{C}$ denotes a fixed category.

Definition 1 (rule and direct derivation). *A* (DPO) rule *on* $\mathbf{C}$ *is a span* $p\colon L \xleftarrow{l} K \xrightarrow{r} R$ *in* $\mathbf{C}$. *The objects* L, K, *and* R *are called left-hand side, context and right-hand side of the rule, respectively.*

A match *of* p *in an object* G *of* $\mathbf{C}$ *is an arrow* $m_L\colon L \to G$. *A* direct derivation $p/m\colon G \Longrightarrow H$ *from* G *to* H *via rule* p *at the match* m_L *is a diagram as depicted in Fig. 1, where the two squares are pushouts in* $\mathbf{C}$ *and* $m = \langle m_L, m_K, m_R \rangle$.

The notion of sequential independence aims at characterising direct derivations which, even if performed in sequence, do not interfere with each other and thus could be potentially applied in any order (and concurrently).

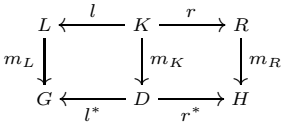


Fig. 1. A direct derivation

Definition 2 (sequential independence). *Let* p_1, p_2 *be rules and let* $p_1/m_1\colon G \Longrightarrow H_1$, $p_2/m_2\colon H_1 \Longrightarrow H$ *be direct derivations as in Fig. 2. They are called* sequential independent *if there exist arrows* $i_1\colon R_1 \to D_2$ *and* $i_2\colon L_2 \to D_1$ *such that* $l_2^* \circ i_1 = m_{R_1}$ *and* $r_1^* \circ i_2 = m_{L_2}$.

A strictly related notion is parallel independence which is aimed at characterising independent direct derivations starting from the same object.

Definition 3 (parallel independence). *Let* p_1, p_2 *be rules and* $p_1/m_1\colon G \Longrightarrow H_1$, $p_2/m_2\colon G \Longrightarrow H_2$ *direct derivations as in Fig. 3. They are called* parallel independent *if there exist arrows* $i_1\colon L_1 \to D_2$ *and* $i_2\colon L_2 \to D_1$ *such that* $l_2^* \circ i_1 = m_{L_1}$ *and* $l_1^* \circ i_2 = m_{L_2}$.

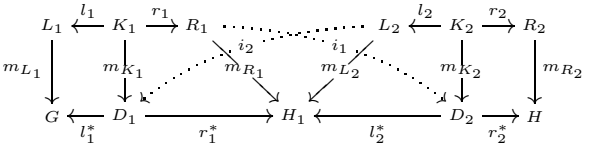


Fig. 2. Sequential independence for $p_1/m_1 \colon G \Longrightarrow H_1$ and $p_2/m_2 \colon H_1 \Longrightarrow H$

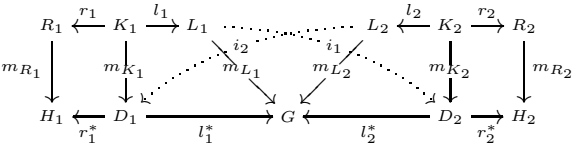


Fig. 3. Parallel independence for $p_1/m_1 \colon G \Longrightarrow H_1$ and $p_2/m_2 \colon G \Longrightarrow H_2$

The two notions are often connected by two properties, which are denoted under the collective name of the local Church-Rosser theorem. The first one is clearly the counterpart of standard local confluence.

Definition 4 (parallel confluence). *Let p_1, p_2 be rules and $p_1/m_1 \colon G \Longrightarrow H_1$, $p_2/m_2 \colon G \Longrightarrow H_2$ parallel independent derivations as in Fig. 3. Then, the derivations satisfy the* parallel local confluence property *if there exist a graph H and derivations $p_2/m_2^* \colon H_1 \Longrightarrow H$, with match $r_1^* \circ i_2$, and $p_1/m_1^* \colon H_2 \Longrightarrow H$, with match $r_2^* \circ i_1$, such that p_1/m_1 and p_2/m_2^* as well as p_2/m_2 and p_1/m_1^* are sequential independent.*

The second part moves instead from sequential independent derivations.

Definition 5 (sequential confluence). *Let p_1, p_2 be rules and $p_1/m_1 \colon G \Longrightarrow H_1$, $p_2/m_2 \colon H_1 \Longrightarrow H$ sequential independent derivations as in Fig. 2. Then, the derivations satisfy the* sequential local confluence property *if there exist a graph H_2 and a derivation $p_2/m_2^* \colon G \Longrightarrow H_2$, with match $l_1^* \circ i_2$, such that p_1/m_1 and p_2/m_2^* are parallel independent.*

Sequential and parallel local confluence are the basis of the concurrency theory of DPO rewriting. When they hold, concurrent derivations can be seen as equivalence classes of concrete derivations up to *shift equivalence* [2], identifying (as for the better-known *permutation equivalence* of λ-calculus) those derivations that differ only for the scheduling of independent steps.

1.2 Rewriting in Adhesive Categories and Local Confluence

Most categories of graph-based objects satisfy both local confluence properties when restricted to *linear* rules. A general setting where the theorem can be proven is that of *adhesive categories* (ACs) [16].

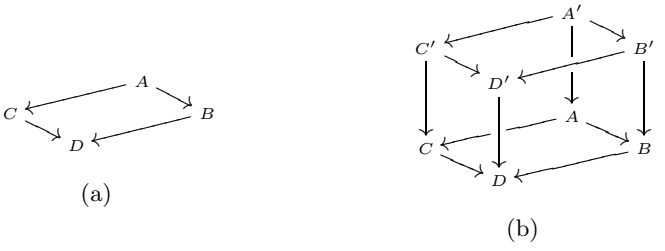


Fig. 4. A pushout square (a) and a commutative cube (b)

Definition 6 (adhesive categories). *A category* **C** *is called* adhesive *if*

- *it has pushouts along monos;*
- *it has pullbacks;*
- *pushouts along monos are* Van Kampen *(*VK*) squares.*

Referring to Fig. 4, a VK *square is a pushout like (a), such that for each commuting cube as in (b) having (a) as bottom face and the back faces of which are pullbacks, the front faces are pullbacks if and only if the top face is a pushout.*

The prototypical AC is **Set**, the category of sets and total functions. Exploiting the closure properties of ACs, it is immediate to deduce that also $\mathbf{Graph} = \mathbf{Set}^{\bullet \rightrightarrows \bullet}$, mentioned before, is adhesive. Likewise, the category of directed hyper-graphs **HGraph** is a category of presheaves and thus adhesive. In fact, $\mathbf{HGraph} = \mathbf{Set}^{\mathbf{M}}$ where **M** is a category with objects $\mathbb{N} \times \mathbb{N} \cup \{\bullet\}$, and where, from any (m, n) there are $m + n$ arrows into $\bullet$.

Example 1. Figure 7 depicts a rule in the AC **HGraph** of hyper-graphs. The left-hand side, context and right-hand side are separated by vertical lines. The arrows from the context to the left- and right-hand side are represented by the positions of the items and by the labels (ignoring for now $\{p\}$ and the dotted arrow). Graphically, nodes are circles and edges are boxes enclosing their label, with either incoming or outgoing tentacles, connecting edges to their source and target nodes, respectively. For the sake of readability, these tentacles are either ordered clock-wise, or when necessary, they are labelled by numbers 0, 1 and 2.

Intuitively, in this graphical interpretation, the application of a rule first removes all the items of G matched by $L - l(K)$, leading to object D. Then the items of $R - r(K)$ are added to D, thus obtaining H. When the rule is not linear, taking the second pushout can also cause some merging.

ACs subsume many properties of HLR categories [6], and this fact ensures the validity of several results about parallelism and confluence. In particular, a local Church-Rosser theorem for *linear* rules holds with no additional assumptions.

Definition 7 ((left-)linear rules in ACs). *A rule* $p\colon L \xleftarrow{l} K \xrightarrow{r} R$ *in an AC is called* left-linear *if* l *is mono, and* linear *if both* l *and* r *are so.*

Actually, the first half of local confluence holds for left-linear rules [16, Theorem 7.7], provided that the AC in question has enough pushouts.

Proposition 1 (parallel confluence in ACs). *Let p_1, p_2 be left-linear rules in an AC with all pushouts and $p_1/m_1 \colon G \Longrightarrow H_1$, $p_2/m_2 \colon G \Longrightarrow H_2$ parallel independent derivations, as in Fig. 3. Then, they satisfy the parallel local confluence property.*

Instead, the restriction to linear rules is needed for the second half of the local Church-Rosser theorem.

Proposition 2 (sequential confluence in ACs). *Let p_1, p_2 be linear rules in an AC and $p_1/m_1 \colon G \Longrightarrow H_1$, $p_2/m_2 \colon H_1 \Longrightarrow H$ sequential independent derivations as in Fig. 2. Then, they satisfy the sequential local confluence property.*

1.3 Quasi-adhesivity

A theory for rewriting can be developed also in the wider class of quasi-adhesive categories (QACs). Recall that a mono is *regular* if it is obtained as an equalizer.

Definition 8 (quasi-adhesive categories). *A category* **C** *is called* quasi-adhesive *if*

- *it has pushouts along regular monos;*
- *it has pullbacks;*
- *pushouts along regular monos are* VK *squares.*

Then, the two confluence properties can be established for derivations in QACs, exactly as for those in ACs, by adapting the notion of (left-)linear rule.

Definition 9 ((left-)linear rules in QACs). *A rule $p \colon L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ in a QAC is called* left-linear *if l is regular mono, and* linear *if both l and r are so.*

Even though every AC is a QAC, no confusion may arise: for ACs Definitions 7 and 9 denote the same class of rules since in ACs all monos are regular [15].

Example 2. A prototypical example of quasi-adhesive category is **Inj**: objects are injective functions in **Set** and arrows are pairs of functions between the corresponding sources and targets, making the diagram commute. An arrow is mono when both components are mono; it is a regular mono when it is mono and the resulting diagram is a pullback [15].

A relevant, graph-based example is the category of *graphs with interface* **IGraph**, whose objects are injective graph morphisms. Such category can be defined as $\mathbf{Inj}^{\bullet \rightrightarrows \bullet}$, hence it is quasi-adhesive by the closure properties in [16], since **Inj** is quasi-adhesive and has equalisers. Similarly, the category **IHGraph** of hyper-graphs with interface can be defined as $\mathbf{Inj}^{\mathbf{M}}$, for **M** the category used in Section 1.2 to present the category of hyper-graphs as a presheaf category.

Graphs with interface have been used in the modelling of process calculi, the basic idea being that nodes in the interface represent the free names of the

process itself [10]. They are also at the basis of the borrowed context approach to the labelled semantics of graph transformation [8].

Consider again rule p_π in Fig. 7. Its three components can be seen as graphs with interface. When the interface is discrete (i.e., it contains no edge, as it happens for the examples in this paper), it is simply represented as a set. For instance, the interface of the left-hand side is just $\{p\}$ and the dotted arrow indicates the image of p into the left-hand side graph. Regular monos in **IHGraph** are easily proven to be the pairs of injective graph morphisms such that the interface is reflected as well as preserved, hence rule p_π is left-linear.

2 Church-Rosser for Left-Linear Rules

The results about local confluence in (quasi-)adhesive categories cannot be proven when we consider rules which are only left-linear. Still, the result has been shown to hold in several concrete categories, most notably the category of directed graphs **Graph** [13]. The aim here is to prove the result in an abstract general setting, so that we can conclude that it holds in most of the categories used in DPO rewriting. We will first consider a setting intended to treat the adhesive case, and then generalise it to deal with QACs.

2.1 Adhesive Case

We first identify a class of ACs where Proposition 2 can be extended and shown to hold for rules that are merely left-linear.

Definition 10 (class $\mathbb{B}$). *We denote by $\mathbb{B}$ the class of adhesive categories* **C** *which satisfy the following properties*

i) **C** *has all pushouts;*
ii) **C** *has a* strict *initial object* 0 *(any arrow $f\colon a \to 0$ is an isomorphism);*
iii) its monos are coproduct injections *(for any mono $f\colon a \rightarrowtail b$, b is a coproduct of a and some c, and f is the corresponding injection);*
iv) all pushouts are stable under pullback *(for any cube like the one in Fig. 4, if the bottom face is a pushout and the lateral faces are pullbacks then the top face is a pushout).*

The category **Set** of sets and functions is clearly in $\mathbb{B}$. Moreover, recall that any AC satisfying strict initiality is also extensive [16, Lemma 4.1], thus all categories in $\mathbb{B}$ are so. Notice that membership in $\mathbb{B}$ is actually very restrictive; for example $\mathbb{B}$ does not include **Graph**, which has monos that are not coproduct injections, nor the category of sets and partial functions which does not have a strict initial object; both of which *are* examples of ACs. Indeed $\mathbb{B}$ is not in general closed under coslice nor functor category constructions.

In order to prove the local Church-Rosser theorem for left-linear rules we need just two technical lemmas. The first result concerns the validity of a decomposition property which generalises the one holding in ACs [16, Lemma 4.6].

Fig. 5. Diagrams for (a) mixed decomposition and (b) pushout decomposition

Definition 11 (mixed decomposition). *We say that a category* $\boldsymbol{C}$ *satisfies the* mixed decomposition *property if for any commuting diagram like the one depicted in Fig. 5(a) (where* w *is mono), whenever (1)+(2) is a pushout and (2) is a pullback, then (1) is a pushout.*

Lemma 1 (mixed decomposition in $\mathbb{B}$). *Let* $\boldsymbol{C}$ *be a category in the class* $\mathbb{B}$ *of Definition 10. Then, it satisfies the mixed decomposition property.*

With respect to [16, Lemma 4.6], we dropped the requirement enforcing arrows l, s, v and r to be monos. A similar result is proven in [14] for partial ACs, but with the additional requirement that the outer pushout is hereditary. As for the stricter case, the lemma above implies that all the squares of the diagram in Fig. 5(a) are both pushouts and pullbacks.

Definition 12 (pushout decomposition in $\mathbb{B}$). *We say that a category* $\boldsymbol{C}$ *satisfies the* pushout decomposition *property if for any commuting diagram like the one depicted in Fig. 5(b) (where* l*,* s *and* v *are mono), whenever the regions (1)+(2) and (2) are pushouts then (1) is a pushout.*

Lemma 2 (pushout decomposition in $\mathbb{B}$). *Let* $\mathbf{C}$ *be a category in class* $\mathbb{B}$*. Then, it satisfies the pushout decomposition property.*

The crucial observation is that Lemmas 1 and 2 mention only monos, pushouts and pullbacks, and since all these are built "pointwise" for product, functor category, slice and coslice, the lemmas hold in any category obtained from a category in class $\mathbb{B}$ by iterating these operations.

Proposition 3. *If mixed and pushout decompositions hold in a category* $\mathbf{C}$*, then they hold in* $\mathbf{C}/C$*,* $C/\mathbf{C}$ *and* $\mathbf{C}^{\mathbf{X}}$ *for any* $C \in \mathbf{C}$ *and any small category* $\mathbf{X}$*.*

Finally, since most known ACs are constructed from **Set** by iterating the above operations, the local Church-Rosser theorem for left-linear rules holds in all these categories. Examples include the aforementioned **Graph**, **HGraph** and their typed versions (slices) as well as the category of sets and partial functions.

Proposition 4 (sequential confluence for left-linear rules). *Let* $\mathbf{C}$ *be any category obtained from a category in class* $\mathbb{B}$ *by iterated application of the functor category, slice and coslice constructions; let* p_1*,* p_2 *be left-linear rules over* $\mathbf{C}$*, and let* $p_1/m_1 \colon G \Longrightarrow H_1$*,* $p_2/m_2 \colon H_1 \Longrightarrow H$ *be sequential independent derivations as in Fig. 2. Then they satisfy the sequential local confluence property.*

2.2 Quasi-adhesive Case

The theory above can be easily generalised to quasi-adhesivity, roughly by replacing ACs with QACs and monos with regular monos.

Definition 13. *Let $\mathbb{QB}$ be the class of quasi-adhesive categories* **C** *such that*

i) **C** *has all pushouts;*
ii) **C** *has a strict initial object;*
iii) regular monos are coproduct injections;
iv) all pushouts are stable under pullback.

Then Proposition 4 holds replacing class $\mathbb{B}$ with $\mathbb{QB}$, because Lemmas 1 and 2 can be reproved in a straightforward fashion.

It is not difficult to show that **Inj** is in $\mathbb{QB}$ using the facts that it is a quasi-topos and quasi-adhesive [15]; the fact that regular monos are coproduct injections is an easy exercise. We thus obtain the local Church-Rosser theorem for it and any functor category over it, hence in particular **IHGraph**, the category of hyper-graphs with interface in which we work in the following section.

3 Graph Rewriting for the Concurrent Semantics of π

A recently developed area of application for graphs with interface is the visual modelling of nominal calculi. Here we focus on the deterministic fragment of the π-calculus and on its graphical semantics, along the lines of [11] (see Section 2 there for the syntax and operational semantics of the calculus). The results in this paper are needed for formally describing the concurrency in the graphical semantics, thus obtaining one of the few such semantics available for π-calculus.

The idea is quite simple: we work in the (quasi-adhesive) category of graphs with interface **IHGraph** and each process is associated with a graph having a discrete interface. The topology of the graph represents a (simplified) syntactic tree of the process. The interface contains a node p, denoting the root of the graph, and a set of nodes denoting free names in the process (this can be larger than the actual set of free names of the process). As an example, the encoding of the process $(\nu a)(\overline{b}a.\overline{a}a \mid b(d).\overline{d}c)$ can be found in Fig. 6. The interface contains the root p and two other nodes representing the free names b and c (the different sorts for processes and names are visually represented by $\bullet$ and $\circ$ nodes, respectively). Each input or output prefix operator, like $\overline{b}a$, corresponds to an edge labelled by *in* or *out*, with one incoming tentacle and three (ordered) outgoing ones, denoting the continuation (labelled by 0) and the channel names, respectively. For the sake of readability, these outgoing tentacles are either ordered clock-wise, or whenever necessary, they are labelled by numbers 0, 1 and 2.

The restriction operator (νa) is modelled by an edge ν connecting the root to the restricted name a and by dropping the node a from the interface. Note the lack of an edge for the parallel operator: parallelism is reduced to being linked to the same node, as with the components $\overline{b}a.\overline{a}a$ and $b(d).\overline{d}c$ of our process.

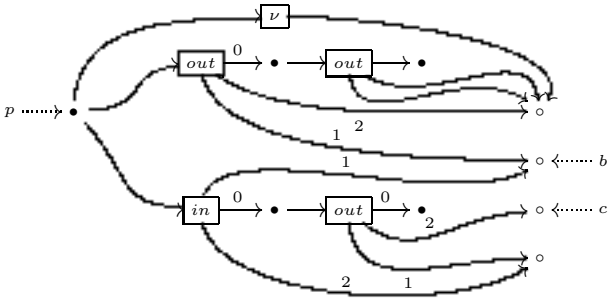


Fig. 6. The graphical encoding of process $(\nu a)(\overline{b}a.\overline{a}a \mid b(d).\overline{d}c)$

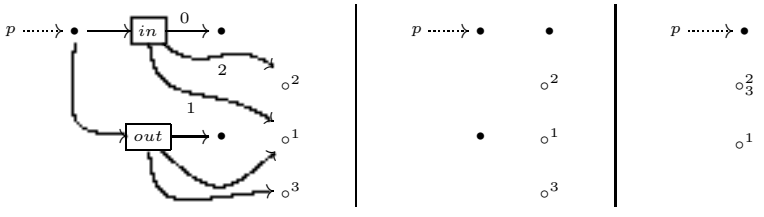


Fig. 7. The rule p_π for synchronization

A single rule p_π (depicted in Fig. 7) suffices for simulating process reduction. As explained before, a node may be the target of a dotted arrow, meaning that the node is in the image of a node of the interface (the source of the arrow). The nodes may be labeled by natural numbers, which are used for describing the (interface preserving) span of arrows constituting the rule. E.g. the nodes identified by 2 and 3 are merged by the rule.

The structural rules are taken care of by the matching mechanism: the embedding of a graph into a larger one models the closure of reduction with respect to contexts. The presence of the interface $\{p\}$ guarantees that a reduction may occur only on the top operators, i.e., never inside a prefix such as $\overline{b}a$ or $b(d)$. Graph isomorphism takes care of the closure with respect to structural congruence. For example, the graph on the left of Fig. 8 is the encoding of the target process of the reduction $(\nu a)(\overline{b}a.\overline{a}a \mid b(d).\overline{d}c) \rightarrow (\nu a)(\overline{a}a \mid \overline{a}c)$.

The presence of the interface node p in the context graph of p_π implies that p can be shared by concurrent reductions. This permits the simultaneous execution of reductions involving top operators.

However, rule p_π is left-linear only: the left leg of the rule is a regular mono (since it reflects the interface), while the right leg is not even mono. Hence, even though category **IHGraph** is quasi-adhesive, the results previously available for rewriting in QACs do not apply, and we need to resort to the theory in Section 2 to formally analyse the concurrency in the system.

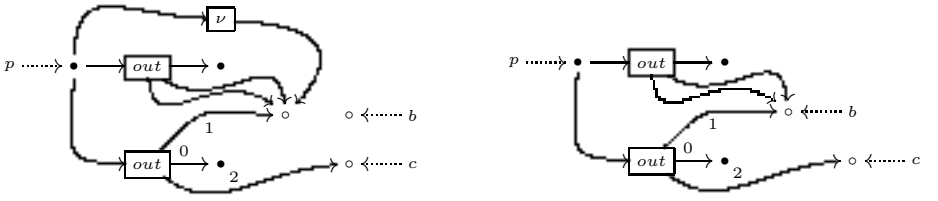


Fig. 8. The graphical encoding of process $(\nu a)(\overline{a}a \mid \overline{a}c)$ (left) and $\overline{b}b \mid \overline{b}c$ (right)

In order to make the example more illustrative, we add rule p_ν in Fig. 9. It models the revelation of a restricted name $(\nu a)P \to P\{^b/_a\}$, which is associated to the free name b occurring in the process, while the corresponding restriction operator is removed. As before, the rule is only left-linear.

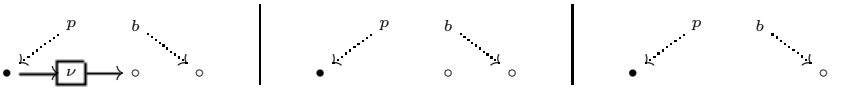


Fig. 9. The rule p_ν for revealing a restricted name

Rule p_ν can be applied to $(\nu a)(\overline{a}a \mid \overline{a}c)$, resulting in the process $\overline{b}b \mid \overline{b}c$: its graphical encoding is depicted on the right of Fig. 8. It is intuitively clear that the two direct derivations represented by first applying the synchronization, and then the revelation, are sequential independent, so they should be executable simultaneously. In order to prove this formally, since the involved rules are left-linear only, we need to resort to the quasi-adhesive variant of Proposition 4.

4 Conclusions

We have identified a class of categories where the local Church-Rosser theorem, a fundamental result in the DPO approach to rewriting, holds also for left-linear rules and arbitrary matches. This class includes most of the adhesive and quasi-adhesive categories actually used as domain categories for rewriting.

There are many examples where left-linear rules arise naturally. One that we consider relevant is related to the graphical encodings of nominal calculi: changes to the physical or logical topology of a system determined by phenomena like name passing or fusion or code mobility are naturally modelled by rules whose right-hand sides are not monomorphisms. By our results these calculi can be equipped with a concurrent semantics, as obtained by exploiting the local Church-Rosser theorem (see e.g. [10,12]).

A further advancement in the theory would be to consider situations where the left-linearity of the rules does not guarantee the existence of the pushout complement, as in the case of the (quasi-adhesive) category of term graphs [3] and of the category of graphs with equivalences [1], possibly constraining the match without necessarily requiring it to be a monomorphism.

References

1. Baldan, P., Gadducci, F., Montanari, U.: Concurrent rewriting for graphs with equivalences. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 279–294. Springer, Heidelberg (2006)
2. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, pp. 163–245. World Scientific, Singapore (1997)
3. Corradini, A., Gadducci, F.: On term graphs as an adhesive category. In: TERMGRAPH 2004. ENTCS, vol. 127(5), pp. 43–56. Elsevier, Amsterdam (2005)
4. Drewes, F., Habel, A., Kreowski, H.-J.: Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, pp. 95–162. World Scientific, Singapore (1997)
5. Ehrig, H., Ehrig, K., Prange, U., Täntzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. Springer, Heidelberg (2006)
6. Ehrig, H., Habel, A., Padberg, J., Prange, U.: Adhesive high-level replacement categories and systems. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 144–160. Springer, Heidelberg (2004)
7. Ehrig, H., Habel, A., Parisi-Presicce, F.: Basic results for two types of high-level replacement systems. In: GETGRATS Closing Workshop. ENTCS, vol. 51, pp. 127–138. Elsevier, Amsterdam (2002)
8. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. Mathematical Structures in Computer Science 16(6), 1133–1163 (2006)
9. Ehrig, H., Kreowski, H.-J.: Parallelism of manipulations in multidimensional information structures. In: Mazurkiewicz, A. (ed.) MFCS 1976. LNCS, vol. 45, pp. 284–293. Springer, Heidelberg (1976)
10. Gadducci, F.: Graph rewriting for the π-calculus. Mathematical Structures in Computer Science 17(3), 407–437 (2007)
11. Gadducci, F., Lluch Lafuente, A.: Graphical encoding of a spatial logic for the π-calculus. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 209–225. Springer, Heidelberg (2007)
12. Gadducci, F., Monreale, G.V.: A decentralized implementation of mobile ambients. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 115–130. Springer, Heidelberg (2008)
13. Habel, A., Müller, J., Plump, D.: Double-pushout graph transformation revisited. Mathematical Structures in Computer Science 11(5), 637–688 (2001)
14. Heindel, T.: Hereditary pushouts reconsidered. In: Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A. (eds.) ICGT 2010. LNCS, vol. 6372, pp. 250–265. Springer, Heidelberg (2010)
15. Johnstone, P.T., Lack, S., Sobociński, P.: Quasitoposes, quasiadhesive categories and artin glueing. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 312–326. Springer, Heidelberg (2007)
16. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. Theoretical Informatics and Applications 39(3), 511–545 (2005)

Quantitative Refinement for Weighted Modal Transition Systems

Sebastian S. Bauer[1], Uli Fahrenberg[2], Line Juhl[3], Kim G. Larsen[3], Axel Legay[2], and Claus Thrane[3]

[1] Ludwig-Maximilians-Universität München, Germany
[2] Irisa/INRIA Rennes, France
[3] Aalborg University, Denmark

Abstract. Specification theories as a tool in the development process of component-based software systems have recently attracted a considerable attention. Current specification theories are however qualitative in nature and hence fragile and unsuited for modern software systems. We propose the first specification theory which allows to capture quantitative aspects during the refinement and implementation process.

Keywords: reducing complexity of design, modal specifications, quantitative reasoning.

1 Introduction

Rigorous design of modern computer systems faces the major challenge that the systems are too complex to reason about [16]. Hence it is necessary to reason at the level of specification rather than at the one of implementations. Such specifications, which act as finite and concise abstractions for possibly infinite sets of implementations, allow not only to decrease the complexity of the design, but also permit to reason on subsystems independently.

Any reasonable specification theory is equipped with a *satisfaction relation* to decide whether an implementation matches the requirements of a specification, and a *refinement relation* that allows to compare specifications (hence sets of implementations). Moreover, the theory needs a notion of *logical composition* which allows to infer larger specifications as logical combinations of smaller ones. Another important ingredient is a notion of *structural composition* that allows to build overall specifications from subspecifications, mimicking at the implementation level *e.g.* the interaction of components in a distributed system. A partial inverse of this operation is given by the notion of *quotient* which allows to synthesize a subspecification from an overall specification and an implementation which realizes a part of the overall specification.

Over the years, there have been a series of advances on specification theories [2,14,5,13]. The predominant approaches are based on modal logics and process algebras but have the drawback that they cannot naturally embed both logical and structural composition within the same formalism. Moreover, such

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 60–71, 2011.

formalisms do not permit to reason from specification to implementation through stepwise refinement.

In order to leverage those problems, the concept of modal transition systems was introduced [12]. In short, modal transition systems are labeled transition systems equipped with two types of transitions: *must* transitions which are mandatory for any implementation, and *may* transitions which are optional for implementations. It is well admitted that modal transition systems match all the requirements of a reasonable specification theory (see *e.g.* [15] for motivations). Also, practical experience shows that the formalism is expressive enough to handle complex industrial problems [6,17].

In a series of recent work [3,10], the modal transition system framework has been extended in order to reason on *quantitative* aspects, hence providing a new specification theory for more elaborated structures, with the objective to better meet practical needs. In this quantitative setting however, the standard Boolean satisfaction and refinement relations are too fragile. Indeed, either an implementation satisfies a specification or it does not. This means that minor and major modifications in the implementation cannot be distinguished, as both of them may reverse the Boolean answer. As observed by de Alfaro *et al.* for the logical framework of CTL [1], this view is obsolete; engineers need quantitative notions on how modified implementations differ.

The main contribution of this paper is to mitigate the above problem by lifting the satisfaction and refinement relations into the quantitative framework, hence completing the quantitative approach to reason on modal transition systems. More precisely, and similarly to what has been proposed in the logical framework, we introduce a notion of *distance* between both specifications and implementations, which permits quantitative comparison. Given two implementations that do not necessarily satisfy a specification, we can decide through quantitative reasoning which one is the better match for the specification's requirements.

To facilitate this reasoning, we develop a notion of *modal* distance between specifications, which approximates the distances between their implementations. This preserves the relation between modal refinement and satisfaction checking in the Boolean setting. We show that computing distances between implementation sets is EXPTIME-hard, whereas modal distances are computable in NP $\cap$ CO-NP (which is higher than for Boolean modal refinement). Akin to *discounted games* [19] we can reason on behaviors in a discounted manner, giving more importance to differences that happen in the near future, while accumulating the amount by which the specifications fail to be compatible at each step. As for the games, the semantics of the outcome is considered application specific.

Modifying the semantic outcome of satisfaction has strong impact on operations between specifications. As a second contribution of this paper, we propose quantitative versions of structural composition and quotient which inherit the good properties from the Boolean setting. We also propose a new notion of *relaxation*, which is inherent to the quantitative framework and allows *e.g.* to calibrate the quotient operator.

However, there is no free lunch, and working with distances has a price: some of the properties of logical conjunction and determinization are not preserved in the quantitative setting. More precisely, conjunction is not the greatest lower bound with respect to refinement distance as it is in the Boolean setting, and deterministic overapproximation is too coarse. In fact we show that this is a fundamental limitation of *any* reasonable quantitative specification formalism.

Structure of the paper. We start out by introducing our quantitative formalism which has weighted transition systems as implementations and weighted modal transition systems as specifications. In Section 3 we introduce the distances we use for quantitative comparison of both implementations and specifications. Section 4 is devoted to a formalization of the notion of relaxation which is of great use in quantitative design. In the next section we see some inherent limitations of the quantitative approach, and Section 6 finishes the paper by showing that structural composition works as expected in the quantitative framework and links relaxation to quotients.

2 Weighted Modal Transition Systems

In this section we present the formalism we use for implementations and specifications. As implementations we choose the model of *weighted transition systems*, *i.e.* labeled transition systems with integer weights at transitions. Specifications both have a *modal* dimension, specifying discrete behavior which *must* be implemented and behavior which *may* be present in implementations, and a *quantitative* dimension, specifying intervals of weights on each transition which an implementation must choose from.

Let $\mathbb{I} = \left\{[x,y] \;\middle|\; x \in \mathbb{Z} \cup \{-\infty\}, y \in \mathbb{Z} \cup \{\infty\}, x \leq y\right\}$ be the set of closed extended-integer intervals and let Σ be a finite set of actions. Our set of *specification labels* is $\mathsf{Spec} = (\Sigma \times \mathbb{I}) \cup \{\bot\}$, where the special symbol $\bot$ models *inconsistency*. The set of *implementation labels* is defined as $\mathsf{Imp} = \Sigma \times \left\{[x,x] \;\middle|\; x \in \mathbb{Z}\right\} \approx \Sigma \times \mathbb{Z}$. Hence a specification imposes labels and integer *intervals* which constrain the possible weights of an implementation.

We define a partial order on $\mathbb{I}$ (representing inclusion of intervals) by $[x,y] \sqsubseteq [x',y']$ if $x' \leq x$ and $y \leq y'$, and we extend this order to specification labels by $(a,I) \sqsubseteq (a',I')$ if $a = a'$ and $I \sqsubseteq I'$, and $\bot \sqsubseteq (a,I)$ for all $(a,I) \in \mathsf{Spec}$. The partial order on Spec is hence a *refinement* order; if $k_1 \sqsubseteq k_2$, then no more implementation labels are contained in k_1 than in k_2.

Specifications and implementations are defined as follows:

Definition 1. A *weighted modal transition system* (WMTS) is a four-tuple $(S, s^0, \dashrightarrow, \longrightarrow)$ consisting of a set of states S with an initial state $s^0 \in S$ and *must* and *may* transition relations $\longrightarrow \subseteq \dashrightarrow \subseteq S \times \mathsf{Spec} \times S$. A WMTS is an *implementation* if $\longrightarrow = \dashrightarrow \subseteq S \times \mathsf{Imp} \times S$.

A WMTS is *finite* if S and $\dashrightarrow$ (and hence also $\longrightarrow$) are finite sets, and it is *deterministic* if it holds that for any $s \in S$ and $a \in \Sigma$, $\big(s,(a,I_1),t_1\big), \big(s,(a,I_2),t_2\big) \in \dashrightarrow$ imply $I_1 = I_2$ and $t_1 = t_2$. Hence a deterministic specification allows at

most one transition under each discrete action from every state. In the rest of the paper we will write $s \dashrightarrow^{k} s'$ for $(s,k,s') \in \dashrightarrow$ and similarly for $\longrightarrow$, and we will always write $S = (S, s^0, \dashrightarrow, \longrightarrow)$ or $S_i = (S_i, s_i^0, \dashrightarrow_i, \longrightarrow_i)$ for WMTS and $I = (I, i^0, \longrightarrow)$ for implementations. Note that an implementation is just a usual integer-weighted transition system.

The implementation semantics of a specification is given through modal refinement, as follows: A *modal refinement* of WMTS S_1, S_2 is a relation $R \subseteq S_1 \times S_2$ such that for any $(s_1, s_2) \in R$ and any *may* transition $s_1 \dashrightarrow^{k_1}_1 t_1$ in S_1, there exists $s_2 \dashrightarrow^{k_2}_2 t_2$ in S_2 for which $k_1 \sqsubseteq k_2$ and $(t_1, t_2) \in R$, and for any *must* transition $s_2 \xrightarrow{k_2}_2 t_2$ in S_2, there exists $s_1 \xrightarrow{k_1}_1 t_1$ in S_1 for which $k_1 \sqsubseteq k_2$ and $(t_1, t_2) \in R$. Hence in such a modal refinement, behavior which is required in S_2 is also required in S_1, no more behavior is allowed in S_1 than in S_2, and the quantitative requirements in S_1 are refinements of the ones in S_2. We write $S_1 \leq_m S_2$ if there is a modal refinement relation R for which $(s_1^0, s_2^0) \in R$. The implementation semantics of a specification can then be defined as the set of all implementations which are also refinements:

Definition 2. The *implementation semantics* of a WMTS S is the set $[\![S]\!] = \{I \mid I \leq_m S, I \text{ implementation}\}$.

We say that a WMTS S is *consistent* if it has an implementation, *i.e.* if $[\![S]\!] \neq \emptyset$. A useful over-approximation of consistency is *local consistency*: a WMTS S is said to be locally consistent if $s \xrightarrow{k} t$ implies $k \neq \bot$, *i.e.* if no $\bot$-labeled *must* transitions appear in S. Local consistency implies consistency, but the inverse is not true; *e.g.* the WMTS $s_0 \xrightarrow{a,2} s_1 \dashrightarrow^{a,9} s_2 \xrightarrow{\bot} s_3$ has an implementation $i_0 \xrightarrow{a,2} i_1$. Local inconsistencies may be removed recursively as follows:

Definition 3. For a WMTS S, let $\mathrm{pre} : 2^S \to 2^S$ be given by $\mathrm{pre}(B) = \{s \in S \mid s \xrightarrow{k} t \in B \text{ for some } k\}$, and let $S^{\bot} = \{s \in S \mid s \xrightarrow{\bot} t \text{ for some } t \in S\}$. If $s^0 \notin \mathrm{pre}^*(S^{\bot})$, then the *pruning* $\rho(S) = (S_\rho, s^0, \dashrightarrow_\rho, \longrightarrow_\rho)$ is defined by $S_\rho = S \setminus \mathrm{pre}^*(S^{\bot})$, $\dashrightarrow_\rho = \dashrightarrow \cap \big(S_\rho \times (\mathsf{Spec} \setminus \{\bot\}) \times S_\rho\big)$ and $\longrightarrow_\rho = \longrightarrow \cap \big(S_\rho \times (\mathsf{Spec} \setminus \{\bot\}) \times S_\rho\big)$.

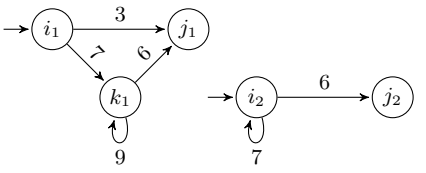
Note that if $\rho(S)$ exists, then it is locally consistent, and if $\rho(S)$ does not exist ($s^0 \in \mathrm{pre}^*(S^{\bot})$), then S is inconsistent. Also, $\rho(S) \leq_m S$ and $[\![\rho(S)]\!] = [\![S]\!]$.

3 Thorough and Modal Refinement Distances

For the quantitative specification formalism we have introduced in the last section, the standard Boolean notions of satisfaction and refinement are too fragile. To be able to reason not only whether a given quantitative implementation satisfies a given quantitative specification, but also *to what extent*, we introduce a notion of *distance* between both implementations and specifications.

We first define the distance between *implementations*; for this we introduce a distance on implementation labels by

$$d_{\mathsf{Imp}}\big((a_1, x_1), (a_2, x_2)\big) = \begin{cases} \infty & \text{if } a_1 \neq a_2, \\ |x_1 - x_2| & \text{if } a_1 = a_2. \end{cases} \tag{1}$$



$$
\begin{aligned}
&d(j_1, j_2) = 0\\
&d(i_1, j_2) = \infty\\
&d(j_1, i_2) = \infty\\
&d(k_1, j_2) = \infty\\
&d(k_1, i_2) = \max\{2 + .9\, d(k_1, i_2),\ .9\, \overbrace{d(j_1, j_2)}^{0}\}\\
&d(i_1, i_2) = \max\{3 + .9\, \underbrace{d(j_1, j_2)}_{0},\ .9\, d(k_1, i_2)\}
\end{aligned}
$$

Fig. 1. Two weighted transition systems with branching distance $d(I_1, I_2) = 18$

In the rest of the paper, let $\lambda \in \mathbb{R}$ with $0 < \lambda < 1$ be a *discounting factor*.

Definition 4. Let I_1, I_2 be implementations (weighted transition systems). The *implementation distance* $d : I_1 \times I_2 \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ between the states of I_1 and I_2 is the least fixed point of the equations

$$
d(i_1, i_2) = \max \begin{cases} \sup\limits_{i_1 \xrightarrow{k_1}_1 j_1} \inf\limits_{i_2 \xrightarrow{k_2}_2 j_2} d_{\mathsf{Imp}}(k_1, k_2) + \lambda d(j_1, j_2), \\ \sup\limits_{i_2 \xrightarrow{k_2}_2 j_2} \inf\limits_{i_1 \xrightarrow{k_1}_1 j_1} d_{\mathsf{Imp}}(k_1, k_2) + \lambda d(j_1, j_2). \end{cases}
$$

We define $d(I_1, I_2) = d(i_1^0, i_2^0)$.

Except for the symmetrizing max operation, this is precisely the *accumulating branching distance* which is introduced in [18]; see also [8,9] for a thorough introduction to linear and branching distances as we use them here. As the equations in the definition define a *contraction*, they have indeed a unique least fixed point; note that $d(i_1, i_2) = \infty$ is also a fixed point, *cf.* [11].

We remark that besides this accumulating distance, other interesting system distances may be defined depending on the application at hand, but we concentrate here on this distance and leave a generalization to other distances for future work.

Example 1. Consider the two implementations I_1 and I_2 in Figure 1 with a single action (elided for simplicity) and with discounting factor $\lambda = .9$. The equations in the illustration have already been simplified by removing all expressions that evaluate to ∞. What remains to be done is to compute the least fixed point of the equation $d(k_1, i_2) = \max\{2 + .9\, d(k_1, i_2), 0\}$ which is $d(k_1, i_2) = 20$. Hence $d(i_1, i_2) = \max\{3, .9 \cdot 20\} = 18$.

To lift implementation distance to specifications, we need first to consider the distance between *sets* of implementations. Given implementation sets $\mathcal{I}_1, \mathcal{I}_2$, we define

$$
d(\mathcal{I}_1, \mathcal{I}_2) = \sup_{I_1 \in \mathcal{I}_1} \inf_{I_2 \in \mathcal{I}_2} d(I_1, I_2)
$$

Note that in case $\mathcal{I}_2$ is finite, we have that for all $\varepsilon \geq 0$, $d(\mathcal{I}_1, \mathcal{I}_2) \leq \varepsilon$ if and only if for each implementation $I_1 \in \mathcal{I}_1$ there exists $I_2 \in \mathcal{I}_2$ for which $d(I_1, I_2) \leq \varepsilon$, hence this is a natural notion of distance. Especially, $d(\mathcal{I}_1, \mathcal{I}_2) = 0$ if and only

if $\mathcal{I}_1$ is a subset of $\mathcal{I}_2$ up to bisimilarity. For infinite $\mathcal{I}_2$, we have the slightly more complicated property that $d(\mathcal{I}_1, \mathcal{I}_2) \leq \varepsilon$ if and only if for all $\delta > 0$ and any $I_1 \in \mathcal{I}_1$, there is $I_2 \in \mathcal{I}_2$ for which $d(I_1, I_2) \leq \varepsilon + \delta$.

Note that in general, our distance on sets of implementations is *asymmetric*; we may well have $d(\mathcal{I}_1, \mathcal{I}_2) \neq d(\mathcal{I}_2, \mathcal{I}_1)$. We lift this distance to specifications as follows:

Definition 5. The *thorough refinement distance* between WMTS S_1 and S_2 is defined as $d_t(S_1, S_2) = d\big([\![S_1]\!], [\![S_2]\!]\big)$. We write $S_1 \leq_t^\varepsilon S_2$ if $d_t(S_1, S_2) \leq \varepsilon$.

Indeed this permits us to measure incompatibility of specifications; intuitively, if two specifications have thorough distance ε, then any implementation of the first specification can be matched by an implementation of the second up to ε. Also observe the special case where $S_1 = I_1$ is an implementation: then $d_t(I_1, S_2) = \inf_{I_2 \in [\![S_2]\!]} d(I_1, I_2)$, which measures how close I_1 is to satisfy the specification S_2.

To facilitate computation and comparison of refinement distance, we introduce modal refinement distance as an overapproximation. We will show in Theorem 2 below that similarly to the Boolean setting [4], computation of thorough refinement distance is EXPTIME-hard, whereas modal refinement distance is computable in NP ∩ CO-NP. First we generalize the distance on implementation labels from Equation (1) to specification labels so that for $k, \ell \in \mathsf{Spec}$ we define

$$d_{\mathsf{Spec}}(k, \ell) = \sup_{k' \sqsubseteq k, k' \in \mathsf{Imp}} \inf_{\ell' \sqsubseteq \ell, \ell' \in \mathsf{Imp}} d_{\mathsf{Imp}}(k', \ell').$$

Note that d_{Spec} is asymmetric, and that $d_{\mathsf{Spec}}(k, \ell) = 0$ if and only if $k \sqsubseteq \ell$. Also, $d_{\mathsf{Spec}}(k, \ell) = d_{\mathsf{Imp}}(k, \ell)$ for all $k, \ell \in \mathsf{Imp}$. Using the $\dot{-}$ operation defined on integers by $x_1 \dot{-} x_2 = \max(x_1 - x_2, 0)$, we can express d_{Spec} as follows:

$$\begin{aligned} d_{\mathsf{Spec}}\big((a_1, I_1), (a_2, I_2)\big) &= \infty \quad \text{if } a_1 \neq a_2 \\ d_{\mathsf{Spec}}\big((a, [x_1, y_1]), (a, [x_2, y_2])\big) &= \max(x_2 \dot{-} x_1, y_1 \dot{-} y_2) \\ d_{\mathsf{Spec}}\big(\bot, (a, I_2)\big) &= 0 \qquad\qquad d_{\mathsf{Spec}}\big((a, I_1), \bot\big) = \infty \end{aligned}$$

Definition 6. Let S_1, S_2 be WMTS. The *modal refinement distance* $d_m : S_1 \times S_2 \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ from states of S_1 to states of S_2 is the least fixed point of the equations

$$d_m(s_1, s_2) = \max \begin{cases} \sup\limits_{s_1 \xrightarrow{k_1}_1 t_1} \inf\limits_{s_2 \xrightarrow{k_2}_2 t_2} d_{\mathsf{Spec}}(k_1, k_2) + \lambda d_m(t_1, t_2), \\ \sup\limits_{s_2 \xrightarrow{k_2}_2 t_2} \inf\limits_{s_1 \xrightarrow{k_1}_1 t_1} d_{\mathsf{Spec}}(k_1, k_2) + \lambda d_m(t_1, t_2). \end{cases}$$

We define $d_m(S_1, S_2) = d_m(s_1^0, s_2^0)$, and we write $S_1 \leq_m^\varepsilon S_2$ if $d_m(S_1, S_2) \leq \varepsilon$.

The argument for existence and uniqueness of the least fixed point is exactly the same as for implementation distance in Definition 4. Like thorough refinement distance, modal refinement distance may be asymmetric.

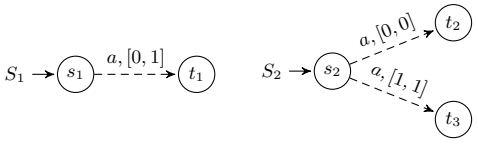


Fig. 2. Incompleteness of modal refinement distance

The next theorem shows that modal refinement distance indeed overapproximates thorough refinement distance, and that it is exact for deterministic WMTS. Note that nothing general can be said about the precision of the overapproximation in the nondeterministic case; as an example observe the two specifications in Figure 2 for which $d_t(S_1, S_2) = 0$ but $d_m(S_1, S_2) = \infty$.

The fact that modal refinement only equals thorough refinement for deterministic specifications is well-known from the theory of modal transition systems [12], and the special case of S_1 locally consistent and S_2 deterministic is important, as it can be argued [12] that indeed, deterministic specifications are sufficient for applications.

Theorem 1. *For WMTS S_1, S_2 we have $d_t(S_1, S_2) \leq d_m(S_1, S_2)$. If S_1 is locally consistent and S_2 is deterministic, then $d_t(S_1, S_2) = d_m(S_1, S_2)$.*

The complexity results in the next theorem show that modal refinement distance can serve as a useful approximation of thorough refinement distance.

Theorem 2. *For finite WMTS S_1, S_2 and $\varepsilon \geq 0$, it is* EXPTIME*-hard to decide whether $S_1 \leq_t^\varepsilon S_2$. The problem whether $S_1 \leq_m^\varepsilon S_2$ is decidable in* NP $\cap$ CO-NP.

4 Relaxation

We introduce here a notion of *relaxation* which is specific to the quantitative setting. Intuitively, relaxing a specification means to weaken the quantitative constraints, while the discrete demands on which transitions may or must be present in implementations are kept. A similar notion of strengthening may be defined, but we do not use this here.

Definition 7. For WMTS S, S' and $\varepsilon \geq 0$, S' is an *ε-relaxation* of S if $S \leq_m S'$ and $S' \leq_m^\varepsilon S$.

Hence the quantitative constraints in S' may be more permissive than the ones in S, but no new discrete behavior may be introduced. Also note that any implementation of S is also an implementation of S', and no implementation of S' is further than ε away from an implementation of S. The following proposition relates specifications to relaxed specifications:

Proposition 1. *If S_1' and S_2' are ε-relaxations of S_1 and S_2, respectively, then $d_m(S_1, S_2) - \varepsilon \leq d_m(S_1, S_2') \leq d_m(S_1, S_2)$ and $d_m(S_1, S_2) \leq d_m(S_1', S_2) \leq d_m(S_1, S_2) + \varepsilon$.*

On the syntactic level, we can introduce the following *widening* operator which relaxes all quantitative constraints in a systematic manner. We write $I \pm \delta = [x - \delta, y + \delta]$ for an interval $I = [x, y]$ and $\delta \in \mathbb{N}$.

Definition 8. Given $\delta \in \mathbb{N}$, the *δ-widening* of a WMTS S is the WMTS $S^{+\delta}$ with transitions $s \overset{a,I\pm\delta}{\dashrightarrow} t$ in $S^{+\delta}$ for all $s \overset{a,I}{\dashrightarrow} t$ in S, and $s \overset{a,I\pm\delta}{\longrightarrow} t$ in $S^{+\delta}$ for all $s \overset{a,I}{\longrightarrow} t$ in S.

Widening and relaxation are related as follows; note also that as widening is a global operation whereas relaxation may be achieved entirely locally, not all relaxations may be obtained as widenings.

Proposition 2. *The δ-widening of any WMTS S is a $(1-\lambda)^{-1}\delta$-relaxation.*

There is also an implementation-level notion which corresponds to relaxation:

Definition 9. The *ε-extended implementation semantics*, for $\varepsilon \geq 0$, of a WMTS S is $[\![S]\!]^{+\varepsilon} = \{I \mid I \leq_m^{\varepsilon} S, I \text{ implementation}\}$.

Proposition 3. *If S' is an ε-relaxation of S, then $[\![S']\!] \subseteq [\![S]\!]^{+\varepsilon}$.*

It can be shown that there are WMTS S, S' such that S' is an ε-relaxation of S but the inclusion $[\![S']\!] \subseteq [\![S]\!]^{+\varepsilon}$ is strict.

5 Limitations of the Quantitative Approach

In this section we turn our attention towards some of the standard operators for specification theories; determinization and logical conjunction. Quite surprisingly, we show that in the quantitative setting, there are problems with these notions which do not appear in the Boolean theory. More specifically, we show that there is no determinization operator which always yields a *smallest* deterministic overapproximation, and there is no conjunction operator which acts as a greatest lower bound.

Theorem 3. *There is no unary operator $\mathcal{D}$ on WMTS for which it holds that*

(3.1) *$\mathcal{D}(S)$ is deterministic for any WMTS S,*
(3.2) *$S \leq_m \mathcal{D}(S)$ for any WMTS S,*
(3.3) *$S \leq_m^{\varepsilon} D$ implies $\mathcal{D}(S) \leq_m^{\varepsilon} D$ for any WMTS S, any deterministic WMTS D, and any $\varepsilon \geq 0$.*

In the standard Boolean setting, there is indeed a determinization operator which satisfies properties similar to the above, and which is useful because it enables checking thorough refinement, *cf.* Theorem 1. Likewise, the greatest-lower-bound property of logical conjunction in the Boolean setting ensures that the set of implementations of a conjunction of specifications is precisely the intersection of the implementation sets of the two specifications.

Theorem 4. *There is no partial binary operator $\wedge$ on WMTS for which it holds that*

(4.1) *$S_1 \wedge S_2 \leq_m S_1$ and $S_1 \wedge S_2 \leq_m S_2$ for all locally consistent WMTS S_1, S_2 for which $S_1 \wedge S_2$ is defined,*

(4.2) *for any locally consistent WMTS S and all deterministic and locally consistent WMTS S_1, S_2 such that $S \leq_m S_1$ and $S \leq_m S_2$, $S_1 \wedge S_2$ is defined and $S \leq_m S_1 \wedge S_2$,*

(4.3) *for any $\varepsilon \geq 0$, there exist $\varepsilon_1 \geq 0$ and $\varepsilon_2 \geq 0$ such that for any locally consistent WMTS S and all deterministic and locally consistent WMTS S_1, S_2 for which $S_1 \wedge S_2$ is defined, $S \leq_m^{\varepsilon_1} S_1$ and $S \leq_m^{\varepsilon_2} S_2$ imply $S \leq_m^{\varepsilon} S_1 \wedge S_2$.*

The counterexamples used in the proofs of Theorems 3 and 4 are quite general and apply to a large class of distances, rather than only to the accumulating distance discussed in this paper. Hence it can be argued that what we have exposed here is a fundamental limitation of any quantitative approach to modal specifications.

6 Structural Composition and Quotient

In this section we show that in our quantitative setting, notions of structural composition and quotient can be defined which obey the properties expected of such operations. In particular, structural composition satisfies independent implementability [2], hence the refinement distance between structural composites can be bounded by the distances between their respective components.

First we define partial synchronization operators $\oplus$ and $\ominus$ on specification labels which will be used for synchronizing transitions. We let $(a_1, I_1) \oplus (a_2, I_2)$ and $(a_1, I_1) \ominus (a_2, I_2)$ be undefined if $a_1 \neq a_2$, and otherwise

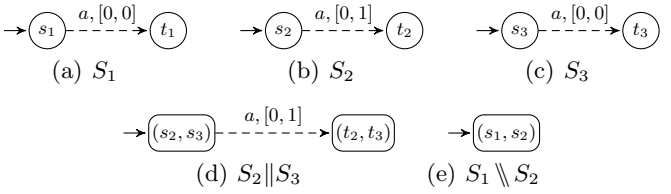
$$\begin{aligned} \big(a, [x_1, y_1]\big) \oplus \big(a, [x_2, y_2]\big) &= \big(a, [x_1 + x_2, y_1 + y_2]\big), \\ (a, I_1) \oplus \bot = \bot \oplus (a, I_2) &= \bot; \end{aligned}$$

$$\begin{aligned} \big(a, [x_1, y_1]\big) \ominus \big(a, [x_2, y_2]\big) &= \begin{cases} \bot & \text{if } x_1 - x_2 > y_1 - y_2, \\ \big(a, [x_1 - x_2, y_1 - y_2]\big) & \text{if } x_1 - x_2 \leq y_1 - y_2, \end{cases} \\ (a, I_1) \ominus \bot = \bot \ominus (a, I_2) &= \bot. \end{aligned}$$

Note that we use CSP-style synchronization, but other types of synchronization can easily be defined. Also, defining $\oplus$ to add intervals (and $\ominus$ to subtract them) is only one particular choice; depending on the application, one can also *e.g.* let $\oplus$ be intersection of intervals or some other operation. It is not difficult to see that these alternative synchronization operators would lead to properties similar to those we show here.

Definition 10. Let S_1 and S_2 be WMTS. The *structural composition* of S_1 and S_2 is $S_1 \| S_2 = \big(S_1 \times S_2, (s_1^0, s_2^0), \mathsf{Spec}, \dashrightarrow, \longrightarrow\big)$ with transitions given as follows:

$$\frac{s_1 \overset{k_1}{\dashrightarrow}_1 t_1 \quad s_2 \overset{k_2}{\dashrightarrow}_2 t_2, \quad k_1 \oplus k_2 \text{ defined}}{(s_1, s_2) \overset{k_1 \oplus k_2}{\dashrightarrow} (t_1, t_2)} \qquad \frac{s_1 \overset{k_1}{\longrightarrow}_1 t_1 \quad s_2 \overset{k_2}{\longrightarrow}_2 t_2 \quad k_1 \oplus k_2 \text{ defined}}{(s_1, s_2) \overset{k_1 \oplus k_2}{\longrightarrow} (t_1, t_2)}$$



(a) S_1 (b) S_2 (c) S_3 (d) $S_2 \| S_3$ (e) $S_1 \setminus\setminus S_2$

Fig. 3. WMTS for which $d_m(S_3, S_1 \setminus\setminus S_2) \neq d_m(S_2 \| S_3, S_1)$

The *quotient* of S_1 by S_2 is $S_1 \setminus\setminus S_2 = \rho\big(S_1 \times S_2 \cup \{u\}, (s_1^0, s_2^0), \mathsf{Spec}, \dashrightarrow, \longrightarrow\big)$ with transitions given as follows:

$$\frac{s_1 \overset{k_1}{\dashrightarrow}_1 t_1 \quad s_2 \overset{k_2}{\dashrightarrow}_2 t_2 \quad k_1 \ominus k_2 \text{ defined}}{(s_1, s_2) \overset{k_1 \ominus k_2}{\dashrightarrow} (t_1, t_2)} \qquad \frac{s_1 \overset{k_1}{\longrightarrow}_1 t_1 \quad s_2 \overset{k_2}{\longrightarrow}_2 t_2 \quad k_1 \ominus k_2 \text{ defined}}{(s_1, s_2) \overset{k_1 \ominus k_2}{\longrightarrow} (t_1, t_2)}$$

$$\frac{s_1 \overset{k_1}{\longrightarrow}_1 t_1 \quad \forall s_2 \overset{k_2}{\longrightarrow}_2 t_2 : k_1 \ominus k_2 \text{ undefined}}{(s_1, s_2) \overset{\perp}{\longrightarrow} (s_1, s_2)}$$

$$\frac{k \in \mathsf{Spec} \quad \forall s_2 \overset{k_2}{\dashrightarrow}_2 t_2 : k \oplus k_2 \text{ undefined}}{(s_1, s_2) \overset{k}{\dashrightarrow} u} \qquad \frac{k \in \mathsf{Spec}}{u \overset{k}{\dashrightarrow} u}$$

Note that we ensure that the quotient $S_1 \setminus\setminus S_2$ is locally consistent by recursively removing $\perp$-labeled *must* transitions using pruning, see Definition 3. The following theorem shows that structural composition is well-behaved with respect to modal refinement distance in the sense that the distance between the composed systems is bounded by the distances of the individual systems. Note also the special case in the theorem of $S_1 \leq_m S_2$ and $S_3 \leq_m S_4$ implying $S_1 \| S_3 \leq_m S_2 \| S_4$.

Theorem 5 (Independent implementability). *For WMTS S_1, S_2, S_3, S_4 we have $d_m\big(S_1 \| S_3, S_2 \| S_4\big) \leq d_m(S_1, S_2) + d_m(S_3, S_4)$.*

The following theorem expresses the fact that quotient is a partial inverse to structural composition. Intuitively, the theorem shows that the quotient $S_1 \setminus\setminus S_2$ is maximal among all WMTS S_3 with respect to any distance $S_2 \| S_3 \leq_m^{\varepsilon} S_1$; note the special case of $S_3 \leq_m S_1 \setminus\setminus S_2$ if and only if $S_2 \| S_3 \leq_m S_1$.

Theorem 6 (Soundness and maximality of quotient). *Let S_1, S_2 and S_3 be locally consistent WMTS such that S_2 is deterministic and $S_1 \setminus\setminus S_2$ is defined. If $d_m(S_3, S_1 \setminus\setminus S_2) < \infty$, then $d_m(S_3, S_1 \setminus\setminus S_2) = d_m(S_2 \| S_3, S_1)$.*

The example depicted in Figure 3 shows that the condition $d_m(S_3, S_1 \setminus\setminus S_2) < \infty$ in Theorem 6 is necessary. Here $d_m(S_2 \| S_3, S_1) = 1$, but $d_m(S_3, S_1 \setminus\setminus S_2) = \infty$ because of inconsistency between the transitions $s_1 \overset{a,[0,0]}{\dashrightarrow}_1 t_1$ and $s_2 \overset{a,[0,1]}{\dashrightarrow}_2 t_2$ for which $k_1 \ominus k_2$ is defined.

As a practical application, we notice that *relaxation* as defined in Section 4 can be useful when computing quotients. The quotient construction in Definition 10

introduces local inconsistencies (which afterwards are pruned) whenever there is a pair of transitions $s_1 \overset{k_1}{\dashrightarrow}_1 t_1$, $s_2 \overset{k_2}{\dashrightarrow}_2 t_2$ (or $s_1 \overset{k_1}{\longrightarrow}_1 t_1$, $s_2 \overset{k_2}{\longrightarrow}_2 t_2$) for which $k_1 \ominus k_2 = \bot$. Looking at the definition of $\ominus$, we see that this is the case if $k_1 = (a, [x_1, y_1])$ and $k_2 = (a, [x_2, y_2])$ are such that $x_1 - x_2 > y_1 - y_2$; hence these local inconsistencies can be avoided by *enlarging* k_1.

Enlarging quantitative constraints is exactly the intuition of relaxation, thus in practical cases where we get a quotient $S_1 \parallel S_2$ which is "too inconsistent", we may be able to solve this problem by constructing a suitable ε-relaxation S_1' of S_1. Theorems 5 and 6 can then be used to ensure that also $S_1' \parallel S_2$ is a relaxation of $S_1 \parallel S_2$.

7 Conclusion and Further Work

We have shown in this paper that within the quantitative specification framework of weighted modal transition systems, refinement and implementation distances provide a useful tool for robust compositional reasoning. Note that these distances permit us not only to reason about differences between implementations and from implementations to specifications, but they also provide a means by which we can compare specifications directly at the abstract level.

We have shown that for some of the ingredients of our specification theory, namely structural composition and quotient, our formalism is a conservative extension of the standard Boolean notions. We have also noted however, that for determinization and logical conjunction, the properties of the Boolean notions are not preserved, and that this is a fundamental limitation of any reasonable quantitative specification theory. The precise practical implications of this for the applicability of our quantitative specification framework are subject to future work.

Acknowledgment. The authors wish to thank Jiří Srba for fruitful discussions during the preparation of this work.

References

1. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and branching system metrics. IEEE Trans. Software Eng. 35(2), 258–273 (2009)
2. de Alfaro, L., Henzinger, T.: Interface-based design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, T. (eds.) Engineering Theories of Software Intensive Systems. NATO Science Series II: Mathematics, Physics and Chemistry, vol. 195, pp. 83–104. Springer, Heidelberg (2005)
3. Bauer, S.S., Juhl, L., Larsen, K.G., Legay, A., Srba, J.: Extending modal transition systems with structured labels (submitted, 2011)
4. Beneš, N., Křetínský, J., Larsen, K.G., Srba, J.: Checking Thorough Refinement on Modal Transition Systems Is EXPTIME-Complete. In: Leucker, M., Morgan, C. (eds.) ICTAC 2009. LNCS, vol. 5684, pp. 112–126. Springer, Heidelberg (2009)
5. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Mang, F.Y.C.: Synchronous and Bidirectional Component Interfaces. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 414–427. Springer, Heidelberg (2002)

6. STREP COMBEST (COMponent-Based Embedded Systems design Techniques), http://www.combest.eu/home/
7. Condon, A.: The complexity of stochastic games. Information and Computation 96(2), 203–224 (1992)
8. Fahrenberg, U., Larsen, K.G., Thrane, C.: A quantitative characterization of weighted Kripke structures in temporal logic. Computing and Informatics 29(6+), 1311–1324 (2010)
9. Fahrenberg, U., Thrane, C., Larsen, K.G.: Distances for weighted transition systems: Games and properties. In: Proc. QAPL 2011, Electronic Proceedings in Theoretical Computer Science (to be published, 2011)
10. Juhl, L., Larsen, K.G., Srba, J.: Modal transition systems with weight intervals. Journal of Logic and Algebraic Programming (to be published, 2011)
11. Larsen, K.G., Fahrenberg, U., Thrane, C.: Metrics for weighted transition systems: Axiomatization and complexity. Theoretical Computer Science (2011), doi:10.1016/j.tcs.2011.04.003
12. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
13. Lynch, N., Tuttle, M.R.: An introduction to input/output automata. CWI-Quarterly 2(3) (1989)
14. Nyman, U.: Modal Transition Systems as the Basis for Interface Theories and Product Lines. PhD thesis, Aalborg University, Department of Computer Science (September 2008)
15. Raclet, J.-B.: Residual for component specifications. Electr. Notes in Theor. Comput. Sci. 215, 93–110 (2008)
16. Sifakis, J.: A vision for computer science – the system perspective. Central European Journal of Computer Science 1(1), 108–116 (2011)
17. SPEEDS (SPEculative and Exploratory Design in Systems Engineering), http://www.speeds.eu.com/
18. Thrane, C., Fahrenberg, U., Larsen, K.G.: Quantitative simulations of weighted transition systems. Journal of Logic and Algebraic Programming 79(7), 689–703 (2010)
19. Zwick, U., Paterson, M.: The complexity of mean payoff games. In: Li, M., Du, D.-Z. (eds.) COCOON 1995. LNCS, vol. 959, pp. 1–10. Springer, Heidelberg (1995)

Faster Coupon Collecting via Replication with Applications in Gossiping

Petra Berenbrink[1], Robert Elsässer[2], Tom Friedetzky[3], Lars Nagel[3], and Thomas Sauerwald[4]

[1] School of Computing Science, Simon Fraser University, Canada
[2] Institute for Computer Science, University of Paderborn, Germany
[3] School of Engineering and Computing Sciences, Durham University, U.K.
[4] Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. We consider an extension of the well-known *coupon collecting* (CC) problem. In our model we have a player who is allowed to deterministically select one box per time step. The player plays against a random sequence of box choices $r_1, r_2, \ldots$ In each step, the contents of both boxes are merged.

The goal of the player is to collect all coupons in one box (the standard model), or to have a copy of each coupon in *all* boxes. We consider three information models, depending on the knowledge of the random choices that the player has before he has to fix his deterministic choices: (i) full prior knowledge of the whole random sequence; (ii) knowledge of the random sequence up to the previous step (but not the current or any subsequent step); (iii) all decisions must be made in advance without any knowledge of the random sequence.

Our main results are lower and asymptotically matching constructive upper bounds for all three models. We also show that network gossiping (similar in spirit to all-in-all CC) is asymptotically no harder than collecting coupons.

1 Introduction

The *coupon collecting* (henceforth referred to as *CC*) problem is one of the most well-studied mathematical models. It is defined as follows. We are given n boxes and n distinguishable coupons, one coupon per box. The goal of the player is to collect one of each of the coupons. In every step the player randomly picks one box, with replacement, and obtains a copy of the corresponding coupon. It is well known, and can be proven with elementary methods, that it takes $\Theta(n \log n)$ many steps in expectation for the player to obtain copies of all n coupons, see e.g. [13]. The coupon collecting problem and variations thereof are used as basic building block in the analysis of many algorithms. Much like balls into bins games, it is an extremely useful and versatile process. In this paper we consider a variant of the standard CC problem. In our model we have a player who is allowed to deterministically select one box per time step. The player plays against a random sequence of box choices $r_1, r_2, \ldots$ Throughout this paper, d_t

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 72–83, 2011.

denotes the player's box choice in step t. If C_t^i denotes the contents of box i after step t then $C_t^{r_t} = C_t^{d_t} := C_{t-1}^{r_t} \cup C_{t-1}^{d_t}$. That is, the contents of both boxes will be merged (under union). The goal of the player is to collect all coupons in one box (we refer to this as CC[1]), or to have a copy of each coupon in *every* box, which we will denote as CC[$*$]. We consider three information models, depending on the knowledge of the random choices that the player has before he has to fix his deterministic choices.

In our main model the player must make all decisions in advance without any knowledge of the random sequence. In the second model we allow knowledge of the random sequence up to the previous step (but not the current or any subsequent step). In the last model we allow the player full prior knowledge of the whole random sequence. To give an example, to "simulate" the standard CC problem the player would choose $d_t = 1$ in every step. Then Box 1 would collect all coupons (this is of course not what our protocol does).

Our main results are lower bounds and asymptotically matching constructive upper bounds for all three models. The results for our main model can be translated into upper and lower bounds for gossiping in the well-studied oblivious phone call model.

1.1 Previous Results

Results for the standard CC problem can be found in many textbooks (see [13] as an example).

The random phone-call model was introduced by Demers et al. [6]. Karp et al. [11] proved that it is possible to design a randomized procedure performing $O(n \log \log n)$ transmissions that accomplishes broadcasting in time $O(\log n)$, with probability $1 - n^{-1}$. For more results about broadcasting in the oblivious phone call model see [8,3,7,10].

In [12] the authors analyse a simple gossip-based protocols for the computations of sums, averages, random samples, quantiles, and other aggregate functions. They show that their protocols converge exponentially fast to the true answer when using uniform gossip. In [5] Chen and Pandurangan consider gossip-based algorithms to aggregate information. For their lower bound in the random phone-call model, they assume a scenario in which all nodes have to follow exactly the same rules in each step. The decision if a node sends a message to its communication partner does not depend on the partner's address nor on the knowledge acquired in the previous rounds. The authors show a lower bound of $\Omega(n \log n)$ on the message complexity of any gossiping algorithm, and this bound holds regardless of the running time of the algorithm. Another recent study of randomized gossiping can be found in the context of resilient information exchange in [2]. The authors propose an algorithm with the optimal $O(n)$ message complexity that tolerates oblivious faults. Note, however, that the model of communication adopted in [2] assumes that every process involved in exchanging information is allowed to contact multiple processes at any round, as well as to maintain open connections over multiple rounds.

In [4] the authors show that $O(\log n)$-time gossiping based on exchange of $O(n \log\log n)$ messages cannot be obtained in the random phone-call model. For more details on the relationship to gossiping (and [4]) see Section 1.3.

1.2 Model and Our Results

Let $\mathcal{B} = \{1, \ldots, n\}$ denote the set of n boxes (types of coupons). Let T be an arbitrary integer, let $r_1, \ldots, r_T$ denote the sequence of *random choices* of boxes, and let $d_1, \ldots, d_T$ denote the sequence of *deterministic choices* of boxes. Step i may be expressed as $s_i = (r_i, d_i)$. We assume that in any step i the contents of boxes r_i and d_i will be merged; if C_i^t denotes the contents of box i prior to step t then $C_{r_t}^t = C_{d_t}^t := C_{r_t}^{t-1} \cup C_{d_t}^{t-1}$. We call $s_1, s_2, \ldots, s_T$ the *step sequence*. We assume that initially, each box i contains one coupon denoted by c_i, that is, $C_i^0 = \{i\}$. For any coupon c, b_c is the unique box which has a copy of the coupon in the very beginning. The main model we consider is the following.

Oblivious: The player has to make all choices $d_i, \ldots, d_T$ in advance, without seeing any of the random choices $r_1, \ldots, r_T$.

For reasons of completeness we also consider two more models that differ in the knowledge of the random choices that the player is allowed to take into account before the deterministic decisions are made.

Clairvoyant: The player knows all the random choices in advance and may adjust $d_1, \ldots, d_T$ accordingly.

Adaptive: The player has to choose d_i *before* r_i is revealed, but it may take $r_1, \ldots, r_{i-1}$ into account. This means that the player knows which coupons have a copy in which box.

There are two natural questions to consider. How long does it take until at least *one* box contains all coupons (CC[1]), and how long does it take until *every* box contains all coupons (CC[$*$])? Somewhat surprisingly, it turns out that, up to constant factors, in *all three* models, CC[$*$] is as easy CC[1]. More formally, we show the following results. We remark that all our upper bounds are not only *existential* but also *constructive*, in the sense that we also provide easily constructed sequences of deterministic choices that achieve the upper bound.

For the oblivious model the player is not allowed to use any knowledge about the random choices for constructing its deterministic sequence. One method for the player might be to collect all coupons in one box (deterministic choice is always the same, resulting in standard Coupon Collecting), or to initialize the deterministic choices randomly. Both approaches will take $\Theta(n \log n)$ steps. We show the following result.

Theorem 1. *In the oblivious model,* CC[1] *and* CC[$*$] *can be solved in* $\mathcal{O}(n \log\log n)$ *steps, with high probability.*

Theorem 2. *Consider the oblivious model. Let* $\epsilon > 0$ *be a sufficiently small constant and fix an arbitrary sequence of deterministic choices. With high probability,* CC[1] *and* CC[$*$] *are not finished after* $\epsilon n \log\log n$ *steps.*

The authors found themselves somewhat surprised by the results. The standard CC process can be accelerated – that is, the runtime can be reduced from $\Theta(n \log n)$ to $\Theta(n \log \log n)$ – by essentially having the player first build a proper basis from which to reach the eventual goal; the original CC process is, of course, utterly incapable of this strategy. This observation is strengthened in Lemma 7, see Section 5 The lemma is essentially stating that whatever one's strategy is, there *must* exist many coupons with many copies each, supports this intuition.

Note that there is a trivial lower bound of $n-1$ for CC[1] (and CC[$*$]) which proves the tightness of our results for the clairvoyant and adaptive models up to constant factors.

Theorem 3. *In the clairvoyant model,* CC[1] *can be solved in* $n-1$ *steps and* CC[$*$] *can be solved in* $2n-2$ *steps with probability one.*

Theorem 4. *In the adaptive model, both* CC[1] *and* CC[$*$] *can be solved in* $\mathcal{O}(n)$ *steps with high probability*[1].

1.3 Relationship to Gossiping

The work in this paper was motivated by our research for lower bound for broadcasting and gossiping on complete graphs in the *random phone call model* The random phone call model is one of the simplest communication models for broadcasting and gossiping. In this model, every node v calls a randomly chosen neighbour w in every step to establish a communication channel with w. The channel can be used for bi-directional communication, but only during that step. From the point of view of v, the channel is an outgoing channel. For w it is an incoming channel. Note that every node has only one outgoing channel per step but it might have several incoming ones. It is assumed that the nodes can transmit over several open channels in one step. In the case of gossiping it is also assumed that the nodes can combine messages. The nodes can now chose which of the channels are used for a transmission. At the end of the step all open channels are closed. One of the classical papers in this area is [11] by Karp, Schindelhauer, Shenker, and Vöcking, where the authors introduce the notion of *address oblivious* algorithms. In this model the decision if a processor sends a message over an open channel must not depend on the identity of that neighbour. In particular, the authors prove a lower bound on the message transmissions of $\Omega(n \log \log n)$ regardless of the number of rounds taken by address-oblivious algorithm. Note that our model is not address oblivious in the sense of [11], and that their lower bound on broadcasting does not apply to our model. Moreover, the analysis of [11] can not be used for our model. For a nice treatment on gossiping we refer the reader to e.g. [5] and references therein.

In [4] the authors provide a lower bound for gossiping the phone call model. The bound shows a $\Omega(n \log n)$ message complexity for any $O(\log n)$-time randomized gossiping algorithm in the oblivious phone call model. For the sake of

[1] By with high probability, we refer to an event that holds with probability at least $1 - n^{-c}$ for some constant $c > 0$.

the proof the authors assume that in every time step an arbitrary subset of the nodes uses the opened channels. If a channel between v and w is used then v will send all its coupons over to w and vice versa. Hence, v and w exchange all their tokens. For this proof it is crucial that the runtime of the algorithm is $O(\log n)$, meaning no node can participate in more than $O(\log n)$ exchange operations.

It is easy to see that Theorem 2 also proves a lower bound of $cn \log\log n$ messages for gossiping in the phone call model for an algorithm with arbitrary runtime in the (our) oblivious model. The relationship between CC[$*$] in the oblivious model and gossiping can be described as follows. Every step of gossiping algorithm A is modelled by several steps (r_i, d_i) of our model. Let us consider the first step of A and let us assume that A uses the outgoing channels opened by the nodes $v_1^1, \ldots v_{\ell_1}^1$ (meaning $v_1, \ldots v_{\ell_1}$ exchange their messages with randomly chosen nodes). Then the player will have the steps $(r_1^1, v_1^1), \ldots (r_{\ell_1}^1, v_{\ell_1}^1)$ in the beginning of its step sequence. If A uses the outgoing channels opened by the nodes $v_1^2, \ldots v_{\ell_2}^2$ in Step 2, the sequence $(r_1^2, v_1^2), \ldots (r_{\ell_2}^2, v_{\ell_1}^2,)$ is appended to the step sequence of the player, and so on. Hence, the deterministic choices of the player model the processors which send messages, and the random choices are used to model the random choices of these nodes. This means that a lower bound on the number of steps it takes to collect all coupons translates directly into a lower bound on the number of messages in the phone call model.

Note that the algorithm of our upper bound in Theorem 1 can also be translated into a gossiping algorithm with runtime $O(n \log\log n)$ sending $O(n \log\log n)$ messages. In fact, Phase 1 of Algorithm I could also be realised in $k \log\log n$ steps of the phone call model (every node appears only once in the deterministic choices of n succeeding steps). But Phase 2 of the Algorithm still needs $kn \log\log n$ steps since all communications use Box 1 as deterministic choice.

The CC[$*$] algorithm for the clairvoyant model and the adaptive model can also be translated into a gossiping algorithm with runtime $O(n)$, but the nodes have to use global information (random choices of other nodes). The CC[$*$] algorithm for the clairvoyant model can also not be realized in the oblivious model.

2 The Oblivious Model

In this section we consider the model where the player does not know any of the random choices before he has to specify the deterministic choices $d_1, \ldots, d_t$. Hence, the player does not know the coupon distribution in the boxes.

2.1 Upper Bound

In this section we will prove Theorem 1. We first consider CC[1]. Algorithm I (Fig. 1) works in two phases. After the first phase every coupon will have $\mathcal{O}(\log n)$ copies in random boxes. The second phase is responsible for moving a copy of each coupon into box 1.

To analyze the algorithm we first show the following result; the proof is omitted due to space limitations.

Oblivious Algorithm I

Phase 1
  For $i \in \{1, \ldots, kn \cdot \log\log(n)\}$ **do**
    play $s_i = (r_i, (i \mod n) + 1)$

Phase 2
  For $i \in \{kn \cdot \log\log(n) + 1, \ldots, 2kn \cdot \log\log(n)\}$ **do**
    play $s_i = (r_i, 1)$

Oblivious Algorithm II

Phase 1
  For $i = 1$ to $2n$ **do**
    play $s_i = (r_i, 1)$

Phase 2
  $\ell = 2n + 1$
  For $i = 1$ to $\mathcal{O}(\log\log n)$ **do**
    For $j = 1$ to n **do**
      play $s_\ell = (r_\ell, j)$
      $\ell = \ell + 1$

Clairvoyant Algorithm III

Choose box $b \notin R$
$\ell = 1$
For $i = 1$ to $n - 1$ **do**
  If $(r_i, i) \in R'$
    play $s_i = (r_i, b)$
  else
    play $s_i = (r_i, u_\ell)$
    $\ell = \ell + 1$

Clairvoyant Algorithm IV

For $i = n$ to $2n - 2$ **do**
  If r_i is empty
    play $s_i = (r_i, b)$
  else
    pick arbitrary empty box u
    play $s_i = (r_i, u)$

Adaptive Algorithm V

Phase $i \in \{0, \ldots, 3\log\log(n) - 1\}$
  For $j \in \{0, \ldots, 2^{i+5} \cdot \ell_i - 1\}$ **do** play $(r_{t_i+j}, c^i_{j \mod \ell_j})$
  For $j \in \{2^{i+5} \cdot \ell_i, \ldots, 2^{i+5} \cdot \ell_i + \frac{n}{2^{i+1}}\}$ **do** play $(r_{t_i+j}, 1)$
Phase $i = 3\log\log(n)$
  For $j \in \{0, \ldots, n - 1\}$ **do** play $(r_{j+t_{3\log\log(n)}}, 1)$

Adaptive Algorithm VI

For $\ell = 1$ to $8n$ **do**
  $s_\ell = (r_\ell, b)$
While CC[$*$] not finished **do**
  let b' be a box that does not have all coupons
  While b' has not all coupons **do**
    $s_\ell = (r_\ell, b')$; $\ell = \ell + 1$

Fig. 1. Our algorithms

Lemma 1. *Let $k \geq 3$ be a constant. At the end of phase 1 every coupon is in $\Omega(\log n)$ many boxes with high probability.*

Lemma 2. *In the oblivious model, Algorithm I solves* CC[1] *in $\mathcal{O}(n \log\log n)$ steps with high probability.*

Proof. We have to show that there is a copy of every coupon in box 1 at the end of Algorithm V. In Phase 2 of the algorithm the contents of $kn \log\log n$ randomly chosen boxes are s copied into Box 1. From Lemma 1 we know that

a fixed coupon c is in $\ell \log n$ many boxes, w.h.p. Hence, the expected number of times that c is in one of the $kn \cdot \log(\log(n))$ boxes chosen in Phase 2 is at least $k \cdot \log(\log(n)) \cdot \ell \log n$. The result now follows from a simple application of Chernoff bounds. □

So far we have collected all coupons in Box 1. For CC[∗] it remains to be shown how to distribute the coupons from Box 1 to all boxes.

Lemma 3. *In the oblivious model, Algorithm II solves* CC[∗] *in* $\mathcal{O}(n \log \log n)$ *steps with high probability.*

Proof. Algorithm II (Fig. 1) executes a broadcast in the oblivious model. To simplify the presentation we assume that the algorithm starts in step $t = 1$. If Algorithm II is used directly after Algorithm I, we have to use s_{t+i} instead of s_i, where t is the running time of Algorithm I. Phase 1 of our algorithm distributes the coupons such that there are w.h.p. $n/2$ boxes having all coupons. Phase 2 is similar to the second phase of the broadcast algorithm from [11], where every uninformed node tries to get the message (vial pull) from a randomly chosen node. The following lemma follows from a slight adaptation of [11, Theorem 2.1] (Phase 3). Notice that in [11] only uninformed nodes try to pull the message whereas in our protocol (which may be considered a sequentialization) *all* nodes play (are being simulated), hence the n as upper index in the j-loop in Algorithm VI. □

2.2 Lower Bound

In this section we prove Theorem 2, a lower bound on the runtime for the oblivious model. Let $\mathcal{B}_i^t$ be the set of boxes that have coupon i at time step t. Note that $\mathcal{B}_i^t$ can be expressed recursively via $\mathcal{B}_i^0 := \{i\}$ and

$$\mathcal{B}_i^t := \begin{cases} \mathcal{B}_i^{t-1} & \text{if } d_t \notin \mathcal{B}_i^{t-1} \text{ and } r_t \notin \mathcal{B}_i^{t-1} \\ \mathcal{B}_i^{t-1} \cup \{r_t, d_t\} & \text{otherwise.} \end{cases} \tag{1}$$

Note that, as soon as $\mathcal{B}_i^t \cap \mathcal{B}_j^t \neq \emptyset$, the distribution of Coupon i and coupon j is no longer independent. For that reason the following lemma shows that after T steps there exists coupons that have only $O(\log n)^{4c}$ copies (for a constant c) in *disjoint* box sets. The existence of at least two coupons which have fewer than $(\log n)^{4c}$ copies in *disjoint* boxes implies that coupon collection is not completed. The disjointness also implies that CC[∗] is not completed.

Lemma 4. *Let* $c = 1/400$ *and* $T = cn \log \log n$. *With probability at least* $1 - 2 \cdot \exp(-n^{8/9})$ *there is a set* $\mathcal{C}''$ *of coupons with the following properties.*

1. $|\mathcal{C}''| \geq n^{8/9}$,
2. *each of the coupons in* $\mathcal{C}''$ *has fewer than* $(\log n)^{4c}$ *copies at the end of step* T, *and*
3. *all the copies of coupons in* $\mathcal{C}''$ *are in disjoint sets of boxes.*

The outline of the proof is as follows. Our goal is to find coupons that are in the set $\mathcal{C}''$. There are two possible ways to increase the number of copies of a coupon i in step t. The random r_t choice can hit a box with coupon i and box d_t does not have a copy of coupon i (random propagation), or d_t is a box with coupon i and box r_t does not have a copy of coupon i (deterministic propagation). In the beginning S is the set of coupons which are initially in a box that does not appear in too many deterministic choices. This set of coupons is called $\mathcal{S}$ (definition before Observation 5). Note that S can also be regarded as a set of boxes. We then eliminate coupons from $\mathcal{S}$ which have too many copies in S due to deterministic propagation (Lemma 7). This results in set $\mathcal{C}'$. After that we eliminate coupons in $\mathcal{C}'$ that have a random propagation (Definition 3(2)), or that have copies in boxes not in S due to a deterministic propagation. We also eliminate coupons from S that have copies in the same boxes.

Proof. Divide the interval $[0, T]$ into $2c \log\log(n)$ consecutive rounds of length $n/2$. Let us define $\mathcal{S}^\ell$ as the set of boxes which are deterministically chosen at most $4c \log\log(n)$ times within the time interval $[\ell \cdot n/2, (\ell + 1) \cdot n/2)$. Let $\mathcal{S} := \bigcap_{\ell=1}^{2c \log\log(n)} \mathcal{S}^\ell$. Note that every box in $\mathcal{S}$ is chosen at most $8c \log\log(n)$ times within any time interval $[t, t + n/2)$ with $1 \leq t \leq T - n/2$.

Observation 5. $|\mathcal{S}| \geq \frac{3}{4} \cdot n$. *(Proof omitted due to space limitations.)*

Let us assume that for every coupon $i \in S$ we have a set $\mathcal{Z}_i \subseteq \mathcal{S}$ of boxes and only boxes from Z_i are allowed to have a copy of coupon i (of course, $i \in Z_i$, Z_i will be defined later). In the following, we focus on the deterministic propagation of i within the set $\mathcal{Z}_i$. Our goal is to show that $|Z_i| = (\log n)^{4c}$ is sufficient. More precisely, we consider the following process.

Definition 1. *For $0 \leq t \leq T$ we define a set $\tilde{\mathcal{B}}_i^t \subset \mathcal{B}_i^t$ recursively as follows. Fix any set $\mathcal{Z}_i \subseteq \mathcal{S}$ with $i \in \mathcal{Z}_i$. Initially we have $\tilde{\mathcal{B}}_i^0 := \{i\}$ and*

$$\tilde{\mathcal{B}}_i^t := \begin{cases} \tilde{\mathcal{B}}_i^{t-1} & \text{if } d_t \notin \tilde{\mathcal{B}}_i^{t-1} \text{ or } |\tilde{\mathcal{B}}_i^{t-1}| = (\log n)^{4c} \\ \tilde{\mathcal{B}}_i^{t-1} \cup \{r_t\} & \text{if } d_t \in \tilde{\mathcal{B}}_i^{t-1} \wedge r_t \in \mathcal{Z}_i \end{cases} . \tag{2}$$

By comparing (2) and (1), one can see that indeed $\tilde{\mathcal{B}}_i^t \subseteq \mathcal{B}_i^t$ for all $1 \leq i \leq n$ and time steps $t \in \mathbb{N}$. Let $T_i(x)$ be the first time step such that $|\tilde{\mathcal{B}}_i^{T_i(x)}| = x$. We make the following important observation which follows directly from the definition of $\tilde{\mathcal{B}}_i^t$.

Observation 6. *The distribution of $\tilde{\mathcal{B}}_i^{T_i(x)} \setminus \{i\}$ uniform over all subsets of size $x - 1$ of $\mathcal{Z}_i \setminus \{i\}$.*

Notice that as soon as we fix a time step t we cannot assume that $\tilde{\mathcal{B}}_i^t \setminus \{i\}$ is a uniformly at random chosen subset of $\mathcal{S}$ of size $|\tilde{\mathcal{B}}_i^t| - 1$. To illustrate the dependencies, consider the following example. $s_1 = (1, 2)$ and $s_2 = (2, 3)$. At the end of step 2, coupon 1 is in boxes 1, 2 and 3. Hence $\tilde{\mathcal{B}}_1^2 = \{1, 2, 3\}$. On the other hand, $|\tilde{\mathcal{B}}_1^2| = 3$, $d_1 = 1$ and $d_2 = 2$ implies $r_1 = 2$.

First we consider a slight modification of the process $\tilde{\mathcal{B}}_i$, called $\tilde{\mathcal{C}}_i$, where we assume that all copies of coupon i are in randomly chosen boxes. We show that for $\tilde{\mathcal{C}}_i$ the probability that the number of copies of coupon i grows very fast is relatively small (Lemma 5). Then we relate the growth probability of $\tilde{\mathcal{C}}_i$ back to the probability for $\tilde{\mathcal{B}}_i$ (Lemma 6).

Definition 2. *Consider a process $\tilde{\mathcal{C}}_i$ that begins at an arbitrary step $t' \in [0, T]$. Fix any set $\mathcal{Z}_i \subseteq \mathcal{S}$ with $i \in \mathcal{Z}_i$. We assume that initially $\tilde{\mathcal{C}}_i^{t'} := \mathcal{D}$ with $\mathcal{D}$ being a uniformly at random chosen subset of $\mathcal{Z}_i$ with $|\mathcal{D}| = x - 1$. For $t \geq t' + 1$ we have $\tilde{\mathcal{C}}_i^t := \tilde{\mathcal{C}}_i^{t-1}$ if $d_t \notin \tilde{\mathcal{C}}_i^{t-1}$ or $|\tilde{\mathcal{C}}_i^{t-1}| = (\log n)^{4c}$, and $\tilde{\mathcal{C}}_i^t := \tilde{\mathcal{C}}_i^{t-1} \cup \{r_t\}$ if $d_t \in \tilde{\mathcal{C}}_i^{t-1} \wedge r_t \in \mathcal{Z}_i$*

The proofs of the following two lemmas are omitted to to space limitations.

Lemma 5. *Fix any subset $\mathcal{Z}_i \subseteq \mathcal{S}$ with $|\mathcal{Z}_i| \geq (5/8)n$ and $i \in \mathcal{Z}_i$. Let x be an arbitrary integer with $16c \log\log(n) \leq x \leq (\log n)^{4c}/15$ and let $\mathcal{D}$ be a randomly chosen subset of $\mathcal{Z}_i$ s.t. $i \in \mathcal{D}$ and $|\mathcal{D}| = x$. For any $t' \in [1, T - n/2]$ it holds that*

$$\mathbf{Pr}\left[|\tilde{\mathcal{C}}_i^{t'+n/2}| \geq 15x\right] \leq 2 \cdot \exp\left(-x/(5 \cdot (4c \cdot \log\log n)^2)\right).$$

where the probability is both over the random choice of $\mathcal{D}$ and $r_{t'+1}, \ldots, r_{t'+n/2}$.

Now we bound the behaviour of $\tilde{\mathcal{B}}_i$ instead of $\tilde{\mathcal{C}}_i$.

Lemma 6. *Fix any subset $\mathcal{Z}_i \subseteq \mathcal{S}$ with $|\mathcal{Z}_i| \geq (5/8)n$ and consider any coupon $i \in \mathcal{Z}_i$. Let x be an arbitrary integer with $16c \log\log(n) \leq x \leq (\log n)^{4c}/15$. Let $T_i(x)$ be the first time step with $|\tilde{\mathcal{B}}_i^{T_i(x)}| = x$. Then*

1. $\mathbf{Pr}\left[|\tilde{\mathcal{B}}_i^{T_i(x)+n/4}| \geq 15x\right] \leq (16c + 2) \cdot \log\log(n) \cdot 2 \exp\left(-\frac{x}{10 \cdot (4c \log\log n)^2}\right)$.
2. $\mathbf{Pr}\left[|\tilde{\mathcal{B}}_i^{\beta n/8}| \geq (\log n)^{4c}\right] \leq \frac{1}{2}$.

where $\beta := \log_{15}((\log n)^{4c}) - \log_{15}((\log\log n)^3) = \Theta(\log\log n)$.

The following lemma is similar to Lemma 4 but makes statements about $\tilde{\mathcal{B}}$ instead of $\mathcal{B}$. Finally, Lemma 8 shows that there are many coupons in C' for which $\tilde{\mathcal{B}}$ and $\mathcal{B}$ are the same. The proof is omitted due to space limitations.

Lemma 7. *With probability at least $1 - \exp(-\frac{n^{9/10}}{8})$, there exists a set $\mathcal{C}' \subseteq \mathcal{S}$ of coupons with $|\mathcal{C}'| = \frac{1}{4} \cdot n^{9/10}$ with the following properties: (i) for all $i \in \mathcal{C}'$, $|\tilde{\mathcal{B}}_i^T| \leq (\log n)^{4c}$, (ii) gor every $i, j \in \mathcal{C}'$ with $i \neq j$, the sets $\tilde{\mathcal{B}}_i^T$ and $\tilde{\mathcal{B}}_j^T$ are disjoint.*

Definition 3. *Let $\mathcal{Z}_i$ be defined as in the proof of Lemma 7. For coupon i, let us call a time step t i-*bad *if (1) $d_t \in \tilde{\mathcal{B}}_i^t$ and $r_t \notin \mathcal{Z}_i$, or (2) $d_t \notin \tilde{\mathcal{B}}_i^t$ and $r_t \in \tilde{\mathcal{B}}_i^t$. Otherwise the time step is called i-*good*. A coupon i is called* good *if there is no time step t which is i-bad.*

If a coupon i is good, then $\mathcal{B}_i^t = \tilde{\mathcal{B}}_i^t$ for all time steps $0 \leq t \leq T$. Our goal is to calculate a lower bound on the probability that a not too small proportion of the coupons in $\mathcal{I}$ of the previous lemma are good. The proof of the following lemma is omitted due to space limitations. Lemma 8 also completes the proof of Lemma 4 (and thereby Theorem 2).

Lemma 8. *Consider a set $\mathcal{C}'$ with the properties as defined in Lemma 7. Then with probability at least $1 - \exp(-n^{8/9})$ there is a subset $\mathcal{C}'' \subseteq \mathcal{C}'$, $|\mathcal{C}''| = n^{8/9}$, such that every coupon in $\mathcal{C}''$ is good.* □

3 The Clairvoyant Model

In this section we consider the most powerful model in which all random choices are revealed to the player in advance. To prove Theorem 3, we first consider the time for CC[1]. We show that for any random sequence $(r_1, \ldots, r_{n-1})$ there exists a deterministic strategy $(d_1, \ldots, d_{n-1})$ such that there is some box b having all coupons. Let $R = \bigcup_{i=1}^{n-1}\{r_i\}$ be the set of boxes appearing in the random sequence and $r = |R|$. For the algorithm we also define $R' = \{(r_i, i) \mid 1 \leq i < n;\ r_i \in R;\ r_i \notin \{r_{i+1}, \ldots, r_{n-1}\}\}$ to be the set of tuples (r_i, i) such that step i is the last step in which box r_i is chosen randomly. The box b is chosen arbitrarily s.t. $b \notin R$. Recall that $\mathcal{B} = \{1, \ldots, n\}$ denotes the set of all boxes. Denote by $U = \mathcal{B} \setminus (R \cup \{b\})$ the set of remaining boxes, and let $(u_1, u_2, \ldots, u_{n-1-r})$ be an arbitrary ordering of U. The algorithm may be found in Fig. 1.

Lemma 9. *In the clairvoyant model, Algorithm III solves* CC[1] *in $n-1$ steps.*

Proof. The Algorithm III (Fig. 1) works as follows. In a step i in which $r_i \in R'$ (the box appears for the last time in the random sequence) all coupons that are in r_i are copied into box b. In all other steps we chose a box for d_i which is never chosen randomly (hence it is in U) and copy its contents into box r_i. Since r_i is chosen again in a later step the coupons of d_i will be copied into b. There are exactly $n-1-r$ such steps so that every box in U can be used. This way b will have all coupons in the end (after $n-1$ steps). □

Given Lemma 9, we can assume that after step $n-1$ there is a box b that contains all coupons. Without loss of generality we assume that all boxes except b contain no coupons. Then Algorithm IV (Fig. 1) distributes the coupons to all boxes in $n-1$ steps.

Lemma 10. *In the clairvoyant model, Algorithm III and Algorithm IV together solve* CC[$*$] *in $2n-2$ steps.*

Proof. Algorithm IV works as follows. Whenever r_i is a box which does not contain any coupons we set $d_i = b$ such that r_i receives all coupons. Otherwise r_i is a box that contains all coupons. Then we set d_i to a box that does not contain any coupons and d_i contains all coupons, too. Using this strategy the number of boxes that do not contain all coupons decreases by 1 in every step. Hence, after step $2n-2$ each box contains all n coupons. □

4 The Adaptive Model

Again, the proof of Theorem 4 is split into two parts, one for CC[1] and one for CC[$*$]. We first consider the time for CC[1]. Algorithm V (Fig. 1) collects all coupons in box 1. We define ℓ_i as the number of coupons which are not in box 1 at the beginning of phase i. Define $c_1^i, \ldots, c_{\ell_i}^i$ to be the coupons which are not in box 1 at the beginning of step i. Let t_i be the first step of phase i and assume $t_0 = 0$. Algorithm V works as follows. The algorithm has $3\log\log(n)+1$ many phases. Each but the last phase is split into two halves (first and second For-loop). The first For-loop of Phase i ($0 \le i < 3\log\log(n) - 1$) is responsible for creating roughly 2^{i+2} copies (in random boxes) of each coupon that is not in box 1 at the beginning of the phase. The second half of the phase decreases the number of coupons not in box 1 by a factor of e^{-4}. The last phase finally ensures that box 1 will obtain all coupons.

Lemma 11. *In the adaptive model, Algorithm V solves* CC[1] *in* $\mathcal{O}(n)$ *steps with high probability.*

Proof. The correctness of the algorithm follows from Lemmas 12 and 13 in this section. The running time of the algorithm is $\left(\sum_{i=0}^{3\log\log(n)-1} 2^{i+5} \cdot \ell_i + \frac{n}{2^{i+1}}\right) + n$. From Lemma 12 we get that $\ell_i \le n/2^{4i}$, and we can upper-bound the sum by $\sum_{i=0}^{3\log\log(n)-1} \left(2^{i+5} \cdot \frac{n}{2^{4i}} + \frac{n}{2^{i+1}}\right) = \sum_{i=0}^{3\log\log(n)-1} \left(\frac{n}{2^{3i-5}} + \frac{n}{2^{i+1}}\right) = \mathcal{O}(n)$. □

Lemma 12. *Assume that at the beginning of phase* i, $0 \le i \le 3\log\log(n) - 1$, *all but* $n/2^{4i}$ *coupons have a copy in box* 1. *Then all but* $n/2^{4(i+1)}$ *coupons have a copy in box* 1 *at the end of phase* i *with a probability of at least* $1 - n^{-2}$.

Lemma 13. *Assume that at the beginning of Phase* $3\log\log n$ *all but* $n/2^{12\log\log n}$ *coupons have a copy in box* 1 *and that every coupon not in box* 1 *is in at least* $(\log n)^3$ *boxes. Then all coupons have a copy in box* 1 *by the end of phase* $3\log\log n$ *with a probability of at least* $1 - n^{-2}$.

The algorithm in Figure 1 distributes the coupons from box b (we can assume that b contains all coupons) to all other boxes.

Lemma 14. *In the adaptive model, Algorithm V and Algorithm VI solve* CC[$*$] *in* $\mathcal{O}(n)$, *steps with high probability.*

Proof. The correctness of Algorithm IV is immediate, it only remains to analyse the runtime of the algorithm. Let us first lower bound the number of boxes that receive all coupons within the first $8n$ steps. The probability that fewer than $n/2$ boxes receive all coupons within the first $8n$ steps is at most $\binom{n}{n/2} \cdot \left(\frac{1}{2}\right)^{8n} \le (2e)^{n/2} \cdot 2^{-8n} \le n^{-2}$. Given that at least $n/2$ boxes receive all coupons in the first $8n$ steps, the expected time for Algorithm IV to spend in the inner while loop is majorised by a geometric random variable with success probability $1/2$. Hence, the total runtime is majorised by the sum of $n/2$ geometric random variables with success probability $1/2$. Using a Chernoff bound, it follows that $\mathcal{O}(n)$ steps are sufficient with high probability. □

5 Number of Necessary Copies

The following lemma shows that, in order to solve the coupon collecting problem, many coupons must have at least $\Omega(\log n)$ copies in different boxes. The Lemma is general in the sense that it does not assume any of the suggested models. The proof is omitted due to space limitations.

Theorem 7. *In order to provide one vertex with all coupons, at least* $n - n^{1/c}$ *coupons must have copies in at least* $\log(n)/c + 2$ *boxes.*

References

1. Azar, Y., Broder, A.Z., Karlin, A.R., Upfal, E.: Balanced Allocations. SIAM Journal on Computing 29(1), 180–200 (1999)
2. Alistarh, D., Gilbert, S., Guerraoui, R., Zadimoghaddam, M.: How efficient can gossip be (On the cost of resilient information exchange). In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 115–126. Springer, Heidelberg (2010)
3. Berenbrink, P., Elsässer, R., Friedetzky, T.: Efficient randomised broadcasting in random regular networks with applications in peer-to-peer systems. In: Proc. of PODC 2008, pp. 155–164 (2008)
4. Berenbrink, P., Czyzowicz, J., Elsässer, R., Gasieniec, L.: Efficient information exchange in the random phone-call model (2010) (manuscript)
5. Chen, J., Pandurangan, G.: Optimal Gossip-Based Aggregate Computation. To appear in Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA (2010)
6. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. In: Proc. 6th ACM Symposium on Principles of Distributed Computing, PODC 1987, pp. 1–12 (1987)
7. Elsässer, R., Sauerwald, T.: The power of memory in randomized broadcasting. In: Proc. of SODA 2008, pp. 290–227 (2008)
8. Elsässer, R.: On the communication complexity of randomized broadcasting in random-like graphs. In: Proc. of SPAA 2006, pp. 148–157 (2006)
9. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized Broadcast in Networks. Random Structures and Algorithms 1(4), 447–460 (1990)
10. Fernandess, Y., Malkhi, D.: On collaborative content distribution using multi-message gossip. Journal of Parallel and Distributed Computing 12, 1232–1239 (2007)
11. Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumor spreading. In: Proc. of FOCS 2000, pp. 565–574 (2000)
12. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. of FOCS 2003, pp. 482–491 (2003)
13. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambdrige University Press, Cambdrige (1995)

Verifying Proofs in Constant Depth*

Olaf Beyersdorff[1], Samir Datta[2], Meena Mahajan[3],
Gido Scharfenberger-Fabian[4], Karteek Sreenivasaiah[3],
Michael Thomas[5], and Heribert Vollmer[1]

[1] Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany
[2] Chennai Mathematical Institute, India
[3] Institute of Mathematical Sciences, Chennai, India
[4] Institut für Mathematik und Informatik, Ernst-Moritz-Arndt-Universität Greifswald, Germany
[5] TWT GmbH, Neuhausen a. d. F., Germany

Abstract. In this paper we initiate the study of proof systems where verification of proofs proceeds by NC^0 circuits. We investigate the question which languages admit proof systems in this very restricted model. Formulated alternatively, we ask which languages can be enumerated by NC^0 functions. Our results show that the answer to this problem is not determined by the complexity of the language. On the one hand, we construct NC^0 proof systems for a variety of languages ranging from regular to NP-complete. On the other hand, we show by combinatorial methods that even easy regular languages such as Exact-OR do not admit NC^0 proof systems. We also present a general construction of NC^0 proof systems for regular languages with strongly connected NFA's.

1 Introduction

The notion of a proof system for a language L was introduced by Cook and Reckhow in their seminal paper [10] as a polynomial-time computable function f that has as its range exactly all strings of L. In this setting, pre-images of f are considered as proofs for elements $x \in L$. Finding such a proof might be difficult, but verifying the validity of a proof can be done efficiently. In the last decades, proof systems were deeply studied in the field of proof complexity and a rich body of results is known regarding the complexity of proofs for concrete proof systems (cf. [18] for a survey).

Recently, there has been great interest in understanding the power of proof systems that use stronger computational resources to verify proofs. In this direction, Pudlák [17] studies quantum proof systems, Cook and Krajíček [9] introduce proof systems that may use a limited amount of non-uniformity (see also [7, 8]), and Hirsch and Itsykson [14, 15] consider proof systems that verify proofs with the help of randomness. In this research, the original Cook-Reckhow

* Research supported by a DAAD/DST grant, DFG grant VO 630/6-2, and by grant N. 20517 from the John Templeton Foundation.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 84–95, 2011.

framework is generalized and exciting results are obtained about the strength and the limitations of theorem proving with respect to these powerful models.

In this work we take the opposite approach and ask for minimal resources that suffice to verify proofs. Our starting point is the observation that every polynomial-time computable proof system in the Cook-Reckhow model is efficiently simulated (*i.e.*, p-simulated) by a proof system where verification of proofs proceeds in AC^0. This immediately leads to the question whether even less powerful computational resources are sufficient. Our investigation focuses on NC^0 circuits—Boolean circuits of constant depth over NOT gates and bounded fan-in AND and OR gates—which constitute one of the weakest computational models in computational complexity. In a related approach, Goldwasser et al. [12] recently studied proof verification by NC^0 circuits in the context of interactive proof systems.

The restrictions imposed by the NC^0 model are so severe that a similar result as the mentioned one for AC^0 fails drastically. NC^0-computable proof systems are functions which shrink the input by at most a constant factor. Thus every language with an NC^0 proof system is computable in nonuniform nondeterministic linear time. We therefore concentrate on the question which languages admit NC^0 proof systems, *i.e.*, which languages can be *enumerated* by families of NC^0 circuits.

A related line of research studies NC^0-computable functions in a cryptographic context [4, 5, 11, 13, 16]. One of the main problems in this area is to construct pseudorandom generators which are computed by NC^0 circuits [4, 5, 11, 16]. This question asks for NC^0-computable functions for which the range is hard to distinguish from a uniform distribution. In contrast, we are looking here at the related, but possibly easier problem to understand which sets can appear at all as the range of NC^0-computable functions. We note that Cryan and Miltersen [11] exhibit an NC^0 computable function whose range is NP-complete. Thus, there are NP-complete languages that admit an NC^0-proof system.

Our results, however, indicate that the answer to the question of the existence of such a proof system does not correlate with the computational complexity of the target language. In our first contribution, we construct NC^0 proof systems for a variety of natural problems, including regular, NC^1-complete, and P-complete languages. In addition, we exhibit a general construction for NC^0 proof systems which works for all regular languages that are accepted by a strongly connected NFA. Our construction directly transforms this NFA into an NC^0 proof system.

Secondly, we demonstrate that there even exist regular languages which do not admit NC^0 proof systems. This implies that lower bound techniques which are used against restricted circuit classes (cf. [19, 20]) are not directly applicable to show lower bounds for NC^0 proof systems. The proof techniques we use are combinatorial arguments tailored towards our specific problems.

This paper is organized as follows. We start in Sect. 2 by defining the concept of NC^0 proof systems and make some initial observations. In Sect. 3 we construct NC^0 proof systems for several languages of different type. This is followed by Sect. 4 where we develop a lower bound technique for the depths of NC circuit

enumerations of several easy languages including Exact-OR and some threshold functions. In Sect. 5 we generalize some of the ideas for NC^0 proof systems from Sect. 3 to obtain proof systems for large classes of regular languages. Finally, we conclude in Sect. 6 with some discussion and future perspectives.

2 Definitions

A function $f\colon \{0,1\}^* \longrightarrow \{0,1\}$ is said to admit an NC^0 proof system if there exists a family of Boolean circuits (see, e.g., [19]) $(C_n)_{n\geq 1}$ satisfying the following conditions:

1. For all $n \geq 1$, $C_n\colon \{0,1\}^{m(n)} \to \{0,1\}^n$, where $m : \mathbb{N} \longrightarrow \mathbb{N}$.
2. For all n and for all words $x \in \{0,1\}^{m(n)}$, $C_n(x) \in f^{-1}(1)$.
3. For all $y \in f^{-1}(1) \cap \{0,1\}^n$, there is a word $x \in \{0,1\}^{m(n)}$ such that $C_n(x) = y$; we say that x is a *proof* of the word y in the pre-image of 1 under f.
4. For some constants c, d, each C_n has size n^c, depth d, and is built using AND, OR, and NOT gates of bounded (constant) fan-in.

That is, the circuit family has as its range exactly the set $f^{-1}(1)$.

A function $f\colon \{0,1\}^* \longrightarrow \{0,1\}$ is said to admit an AC^0 proof system if there exists a family of Boolean circuits $f^{-1}(1)$ as above, with the only difference that this time the circuits are allowed to use unbounded fan-in AND and OR gates. (Note that here applies the non-standard size bound: we require the circuit size to be polynomial in the output length, not input length.)

If the circuit family is *uniform*, then we say that the proof system is *uniform*. Here, a uniform circuit family is a family whose direct connection language, i.e., a language describing the structure (nodes, wires, gates types) of the circuits in the family, is decidable. If the direct connection language is decidable in polynomial time, then the family is said to be P-*uniform*. If the language is decidable in logarithmic time, then the family is said to be $\mathsf{DLOGTIME}$-*uniform*. (For more formal definitions, we refer the reader to [19].)

We remark that all lower bounds we will present in the sequel of this paper hold even for nonuniform proof systems, while all upper bounds will yield $\mathsf{DLOGTIME}$-uniform proof systems, unless explicitly stated otherwise.

For a language $L \subseteq \{0,1\}^*$, we say that L admits an NC^0 proof system, or that L is enumerable in NC^0, if its characteristic function χ_L admits such a proof system. In other words, there is an NC^0 circuit family which produces as output all the strings in L and no other strings. As before, if $C(x) = y$, then we view x as a *proof* that $y \in L$.

Since the circuit must always produce a string in L, we cannot construct such proof systems if a language has "gaps"; if for some n, $L \cap \{0,1\}^{=n} = \emptyset$, then we cannot define C_n. We therefore allow circuits that are "empty"; C_n is empty if and only if $L \cap \{0,1\}^{=n} = \emptyset$.

We observe that AC^0 proof systems do exist for every NP-set. In fact, a more general statement is true.

Proposition 1 (Folklore). *Every language in* NP *admits an* AC^0 *proof system. Every recursively enumerable language admits a constant-depth proof system.*

As mentioned already in the introduction, Cryan and Miltersen [11] exhibit an NP-complete language that admits even an NC^0-proof system. But it is quite easy to see that this is not the case for every NP-language. Indeed, as a consequence of the last condition of the definition above, we see that $m(n) \leq 2^d n \in O(n)$ and the circuits C_n are also of size $O(n)$; each bit of the output depends on $O(1)$ bits of the input proof. Thus if L has NC^0 proof systems, then strings in L have linear-sized proofs that are locally verifiable. This leads to the following observation, which will be considerably strengthened in Sect. 4.

Proposition 2. *There are non-trivial languages in* NP *that do not admit any* DLOGTIME*-uniform* NC^0 *proof system.*

3 Languages with NC^0 Proof Systems

In this section, we construct NC^0 proof systems for a variety of languages.

We start with an NC^1-complete language that admits an NC^0 proof system. The word problem for a finite monoid M with identity e is (membership in) the language: $\{\langle m_1, m_2, \ldots, m_n\rangle \in M^* : \prod_{i=1}^{n} m_i = e\}$. We assume here that for some constant c depending only on M, each element of M is described by a bit string of exactly c bits.

Proposition 3. *The word problem for finite groups admits an* NC^0 *proof system.*

Proof. We describe the circuit $C_n\colon \{0,1\}^{cn-c} \to \{0,1\}^{cn}$. (Since the word problem contains only words of lengths divisible by c, we produce circuits only for such lengths.) Given the encoding of a sequence $g_1, \ldots, g_{n-1}$, and assuming that $g_0 = g_n = e$, C_n produces the sequence $\langle h_1, \ldots, h_n\rangle$ where $h_i = g_{i-1}^{-1} g_i$. □

Corollary 1. *The parity function admits an* NC^0 *proof system.*

In proving Proposition 3, we used all the three group axioms: associativity, existence of an identity and existence of inverses. We can relax some of these and still get an NC^0 proof system. E.g. the *OR* operation is associative and has an identity, but all elements do not have an inverse. Yet we show that the language $L_{OR} = \{w = w_1 \ldots w_n \in \{0,1\}^* : \bigvee_{i=1}^{n} w_i = 1\}$ has an NC^0 proof system.

Proposition 4. *The language* L_{OR} *admits an* NC^0 *proof system.*

Proof. (Sketch) The circuit $C_n\colon \{0,1\}^{2n-1} \to \{0,1\}^n$ takes as input bit strings $a = a_1 \ldots a_n$ and $b = b_1 \ldots b_{n-1}$, and, assuming $b_0 = 0$, $b_n = 1$, outputs a sequence $w = w_1 \ldots w_n$ where, for $1 \leq i \leq n$,

$$w_i = \begin{cases} a_i & \text{if } (b_{i-1} \vee a_i) = b_i \\ 1 & \text{otherwise.} \end{cases}$$

□

We next consider another NC^1-complete problem viz. reachability in bounded width directed acyclic graphs. This example illustrates a proof system, which, for the lack of a better description, we refer to as "input altering proofs".

A layered graph with vertices arranged in layers from $0, 1, \ldots, L$ with exactly W vertices per layer (numbered from $0, \ldots, W-1$) and edges between vertices in layer i to $i+1$ for $i \in \{0, \ldots, L-1\}$ is a positive instance of reachability if and only if there is a directed path from vertex 0 at layer 0 to vertex 0 at layer L. A description of the graph consists of a layer by layer encoding of the edges as a bit vector. In other words it consists of a string $x = x^0 x^1 \ldots x^{L-1} \in (\{0,1\}^{W^2})^L$ where the x^i is indexed by $j, k \in \{0, \ldots, k-1\}$ and $x^i[j,k] = 1$ if and only if the j-th vertex on the i-th layer and the k-th vertex on the $(i+1)$-th layer share an edge. The language L_{BWDR} consists of those strings $x \in (\{0,1\}^{W^2})^L$ which describe a positive instance of reachability, for some $W \in O(1)$. Then we have:

Proposition 5. *L_{BWDR} admits an NC^0 proof system.*

Proof. (Sketch) The proof encodes the graph G and a path P. The circuit outputs the union of the edges in G and in P. □

The same idea can be used for addition and comparison. Consider the function $f_+ : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{n+1} \to \{0,1\}$ such that $f_+(a,b,s) = 1$ if and only if $A + B = S$ where a, b are the n-bit binary representations of the numbers A, B and s is the $(n+1)$-bit binary representation of S. Also consider the function $f_\leq : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ where $f_\leq(a,b) = 1 \iff A \leq B$, where a, b are the n-bit binary representations of numbers A, B.

Proposition 6. *f_+ and $f_\leq$ admit an NC^0 proof system.*

Proof. (Sketch) (f_+:) The proof consists of the binary representations of A, B, S and the sequence of carry bits generated in the addition. At any position, if the carry bit is inconsistent with those of A, B, S, then the corresponding bits of A and/or B are modified. ($f_\leq$:) The proof consists of four n-bit strings $\alpha, \alpha', \gamma, \beta$, with the intent that γ is the carry sequence for the sum of α, α' which yields β. Inconsistencies are locally corrected by altering α, β. □

We now consider a P-complete language, Grid Circuit Value. An instance consists of a planar circuit with vertices embedded in a square grid so that the circuit wires lie only along the grid edges and are directed to go only due east or due north. All possible wires are present. The gates can be arbitrary functions of the two inputs and two outputs. All inputs are present on the outer face of the circuit (i.e. on the southern and western boundaries). It is easy to see that the Grid Circuit Value Problem is P-complete. Using the strategy of locally correcting the input if the proof shows an inconsistency, we can show the following:

Proposition 7. *The Grid Circuit Value Problem admits an NC^0 proof system.*

Remark 1. As mentioned earlier, Cryan and Miltersen [11] show that an NP-complete language admits an NC^0 proof system. The language is just an encoding of 3-SAT: for each n, instances with n variables are encoded by an $M = 8\binom{n}{3}$

bit string, where each bit indicates whether the corresponding potential clause is present in the instance. A proof consists of an assignment to the propositional variables and a suggestion for a 3-CNF, which is modified by the proof system in order to be satisfied by the given assignment.

Next, we describe some generic constructions and closures. They are easy to see, but we state them explicitly for later use.

Lemma 1. *Let w be any fixed string, and let L be any language. Then L admits an NC^0 system if and only if $L \cdot \{w\}$ does.*

Lemma 2. *If $A, B \subseteq \{0,1\}^*$ admit NC^0 proof systems, then so does $A \cup B$.*

Proof. Use the proof systems C_A, C_B for A, B respectively, along with an extra bit which decides whether to retain the output of C_A or the output of C_B. □

Note that in the above proof, the depth of the circuit for $A \cup B$ is two more than the maximum depth of the circuits for A and B. Since union is associative, a union of k sets can be expressed as a binary tree of unions of depth $\lceil \log k \rceil$. Thus the union of k languages, each with an NC^0 proof system of depth d, has an NC^0 proof system of depth $d + 2\lceil \log_2 k \rceil$. In particular, we get the following nonuniform upper bounds.

Lemma 3. *Let $L \subseteq \{0,1\}^*$ have the property that there is a constant k such that for each n, $|L \cap \{0,1\}^n| \leq k$. That is, at each length, at most k strings of that length are in L. Then L admits a nonuniform NC^0 proof system.*

In certain cases, the complement of a language with an NC^0 proof system also has an NC^0 proof system. For example:

Lemma 4. *Let $L \subseteq \{0,1\}^*$ have the property that there is a constant k such that for each n, $|L \cap \{0,1\}^n| \geq 2^n - k$. That is, at each length, at most k strings of that length are not in L. Then L admits a nonuniform NC^0 proof system.*

Proof. (Sketch) We first take the proof circuit for L_{OR} and generalize it to exclude a fixed string y instead of 0^n. This can further be generalized to exclude a constant k many strings from $\{0,1\}^n$ giving the required circuit. □

Theorem 1. *Every language decidable in nonuniform NC^0 has a nonuniform NC^0 proof system.*

Proof. If the circuit C accepts a word w, then let D be the circuit extending C, which outputs the input if C accepts and otherwise outputs w. Then D enumerates the words accepted by C. □

4 Lower Bounds

We now consider languages, which do not admit NC^0 proof systems, some of them even regular,. At first we focus on non-constant lower bounds for the depth

required in order to enumerate these languages by circuits with binary gates. Later on we take the opposite perspective and ask, given a constant depth bound d, how large a fraction of a language can be enumerated by an NC^0 proof system of depth d. This fraction can turn out to be exponentially small. All our examples in this section are characterized be some counting feature.

4.1 Lower Bounds on Depth

We begin with our main concrete example of a non-NC^0-enumerable language.

Theorem 2. *The function Exact-OR^n on n bits, that evaluates to 1 if and only if exactly one of the input bits is 1, does not admit NC^0 proof systems.*

Proof. (Sketch) Suppose there is such a proof system, namely an NC^0-computable function $f : \{0,1\}^m \longrightarrow \{0,1\}^n$. Let $R_i \subseteq [m]$ be the proof bit positions that have a path to the ith output bit. For each i, there is at least one setting of the R_i bits that places a 1 in the ith bit of the output (producing the output string e_i). All extensions of this setting must produce e_i. Therefore $|f^{-1}(e_i)| \geq 2^{m-|R_i|}$. Let $c = \max_{i=1}^n |R_i|$; by assumption, $c \in O(1)$. Then for each $i \in [n]$, $|f^{-1}(e_i)| \geq 2^{m-c}$. But the $f^{-1}(e_i)$ partition $\{0,1\}^m$. Hence

$$2^m = \sum_{i=1}^{n} |f^{-1}(e_i)| \geq \sum_{i=1}^{n} 2^{m-c} = n2^{m-c}$$

Therefore $c \geq \log n$, so $\exists i \in [n] : |R_i| \geq \log n$, a contradiction. □

Generalising this proof technique, we derive below a criterion which implies non-constant lower bounds for the depth of an enumerating circuit family.

Theorem 3. *Let L be a language and $\ell, t \colon \mathbb{N} \to \mathbb{N}$ functions such that for each length n there are $t(n)$ distinct settings to subsets of $\ell(n)$ bits $x_{i_1}, \ldots, x_{i_{\ell(n)}}$ such that each of these partial configurations enforces a fixed value to each of the remaining bits. Then the depth of each circuit family that enumerates L is at least* $\log\log t(n) - \log \ell(n)$.

Using this theorem, we can show that several functions are not enumerable in constant depth.

Exact Counting. Consider the function Exact-Count$_k^n$ on n bits: it evaluates to 1 if and only if exactly k of the input bits are 1. For each length n there are exactly $\binom{n}{k}$ words in Exact-Count$_k^n$. And whenever k bits of a word are set to value 1, then all remaining bits are bound to take the value 0. So for Exact-Count$_k^n$ the parameters $t(n)$ and $\ell(n)$ defined in the theorem above take the values $\binom{n}{k}$ and k, respectively, which yields a lower bound of

$$d(n) = \log\log \binom{n}{k} - \log k \geq \log\log \left(\frac{n^k}{k^k}\right) - \log k = \log(\log n - \log k)$$

on the depth of an enumerating circuit family. For $k(n)$ sub-linear in n this gives an unbounded function, so in this case Exact-Count$_k^n$ does not admit an NC^0 proof system. Note that for a constant k this language is even regular.

$\neg Th^n_{k+1}$ and dually Th^n_{n-k} for sub-linear k. Let Th^n_a be the function that evaluates to 1, if and only if at least a of the n inputs are set to 1. The lower bounds for these languages are derived precisely by the same argument given above for Exact-Count^n_k. So they also yield the same set of parameters.

0^*1^* and iterations. First consider 0^*1^*, whose members consist of a (possibly empty) block of 0's followed by a (possibly empty) block of 1's. The $n+1$ length-n members of 0^*1^* are in 1-1 correspondence to the members of Exact-Count$^{n+1}_1$ via the NC^0 mapping $w_1 \dots w_n \longmapsto x_1 \dots x_{n+1}$, where $x_i := w_{i-1} \oplus w_i$, with the convention that $w_0 := 0$ and $w_{n+1} := 1$. Thus an NC^0 proof system of 0^*1^* would directly yield one for Exact-Count$^{n+1}_1$, which we have shown to be impossible. The parameters from the theorem are $\ell(n) = 2$ (two consecutive bits with different values or simply $w_1 = 1$ or $w_n = 0$) and $t(n) = n+1$. By the same argument, for sub-linear k, the languages consisting of either exactly or up to k alternating blocks of 0's and 1's do not admit NC^0 proof systems.

Open problem: Majority. The Majority language consists of those words which have at least as many 1's as 0's. Does Majority admit an NC^0 proof system?

4.2 List Enumerations

Consider a circuit $C\colon \{0,1\}^m \longrightarrow \{0,1\}^{tn}$. On input x, C can be thought of as producing a list $L(x)$ of t strings of length n. (An alternative view is that we allow t circuits, here merged into one, to enumerate words of length n.) We say that C *t-enumerates* L or is a *t-list proof system for* L if $\bigcup_x L(x) = L$. All along what we have been considering is $t = 1$.

For instance, every sparse language admits a nonuniform NC^0 polynomial-list proof system, as every word can be generated by a sub-circuit with constant output. So in particular, the regular languages Exact-Count^n_k for constant k are of this kind, though they do not have NC^0 proof systems. We observe below that any sub-language of Exact-Count$^n_1$ enumerated by a single circuit is small, and hence Exact-Count$^n_1$ **requires** $\Omega(n)$-lists. We will use this in Theorem 4 to prove a lower bound for the list length of the language of all permutation matrices.

Lemma 5. *Let L be a subset of Exact-Count^n_k that has an NC^0 proof system which is computed by a depth d circuit family. Then for each length n the set $L^{=n}$ of length n members of L has at most 2^{k2^d} elements.*

Proof. This follows directly from Theorem 3, replacing $t(n)$ by $|L^{=n}|$. □

A permutation matrix of order n is an $n \times n$ 0-1-matrix in which every row and every column contains exactly one 1. Lemma 5 gives the following:

Theorem 4. *If C is a depth d circuit that t-enumerates the set of all permutation matrices of order n, then t grows exponentially with n.*

The same idea also works for proving lower bounds on the list length of enumerations of matrices which encode all Hamiltonian cycles in a complete graph or all paths from 1 to n in K_n.

5 Proof Systems for Regular Languages

In this section, we describe some sufficient conditions under which regular languages have NC^0 proof systems. The regular languages we consider may not necessarily be over a binary alphabet, but we assume that a binary (letter-by-letter) encoding is output.

Our first sufficient condition abstracts the strategy used to show that OR has an NC^0 proof system. This strategy exploits the fact that there is a DFA for OR, where every useful state has a path to an "absorbing" final state.

Theorem 5. *Let L be a regular language accepted by an NFA $M = (Q, \Sigma, \delta, F, q_0)$. Let $F' \subseteq F$ denote the set of absorbing final states; that is, $F' = \{f \in F \mid \forall a \in \Sigma, \delta(f, a) = f\}$. Suppose M satisfies the following condition:*

> *For each $q \in Q$, if there is a path from q to some $f \in F$, then there is a path from q to some $f' \in F'$.*

Then L has an NC^0 proof system.

Proof. (Sketch.) The idea is to give a word x and the states of the NFA on some accepting run of M on x. Instead of giving the entire state sequence, only the states after every k steps are given. The NC^0 circuit checks, for each "block" of x, if the states before and after the block are consistent with x. If so, this part of x is output as is; otherwise, it is replaced by a string of length $\leq k$ that takes M from the state at the beginning of the block to an absorbing final state. □

Observe that the OR and the Exact-OR are both star-free languages but the complementations in the expression for OR are applied to the empty set, whereas those in Exact-OR are applied to non-empty sets. Based on this, we formulate and prove the following sufficient condition for a star-free regular language to have an NC^0 proof system.

Definition 1. *Strict star-free expressions over an alphabet Σ are exactly the expressions obtained as follows:*

1. *ϵ, a for each $a \in \Sigma$, $\Sigma^* = \bar{\emptyset}$ are strict star-free.*
2. *If r and s are strict star-free, so is $r \cdot s$.*
3. *If r and s are strict star-free, so is $r + s$.*

Theorem 6. *Let r be a strict star-free expression describing a language $L = L(r)$. Then L admits an NC^0 proof system.*

Proof. (Sketch) Strict star-free expressions can be written as sums of star-free sum-free terms. Use Theorem 5 for each such term, and finally use Lemma 2. □

Theorem 5 essentially characterizes functions like OR. On the other hand, the parity function, that has a NC^0 proof system, cannot be recognized by any DFA or NFA with an absorbing final state. The strategy used in constructing the proof system for parity exploits the fact that the underlying graph of the DFA for parity is strongly connected. In the following result, we abstract this property and prove that strong connectivity in an NFA recogniser is indeed sufficient for the language to admit an NC^0 proof system.

Theorem 7. *Let L be accepted by NFA $M = (Q, \Sigma, \delta, F, q_0)$. If the directed graph underlying M is strongly connected, then L admits an NC^0 proof system.*

Proof. (Sketch.) We use the term "walk" to denote a path that is not necessarily simple, and "closed walk" to denote a walk that begins and ends at the same vertex. The idea behind the NC^0 proof system we will construct here is as follows: We take as input a sequence of blocks of symbols $x^1, x^2, \ldots, x^k$, each of length ℓ and as proof, we take the sequence of states $q^1, q^2, \ldots, q^k$ that M reaches after each of these blocks, on some accepting run. Now we make the circuit verify at the end of each block whether that part of the proof is valid. If it is valid, then we output the block as is. Otherwise, if some x^i does not take M from q^{i-1} to q^i, then, we want to make our circuit output a string of length ℓ that indeed makes M go from q^{i-1} to q^i. So we make our circuit output a string of symbols which will first take M from q^{i-1} to q_0, then from q_0 to q^i. To ensure that the total length is indeed ℓ, we sandwich in between a string of symbols that takes M on a closed walk from q_0 to q_0. We thus need to formally prove that closed walks of the required length always exist, and that this can be done in NC^0.

Define the following set of non-negative integers:

$$L = \{\, \ell \mid \text{there is a closed walk through } q_0 \text{ of length exactly } \ell \,\}$$

Let g be the greatest common divisor of all the numbers in L. Though L is infinite, it has a finite subset L' whose gcd is g. Choose a set $S \subseteq Q$ as follows:

$$S = \{\, q \in Q \mid \text{there is a walk from } q_0 \text{ to } q \text{ whose length is } 0 \bmod g \,\}$$

Claim. For every $p \in Q$, $\exists \ell_p, r_p \in \{0, 1, \ldots, g-1\}$ such that

1. the length of every path from q_0 to p is $\equiv \ell_p \pmod g$;
2. the length of every path from p to q_0 is $\equiv r_p \pmod g$.

Now onwards, $\forall p \in Q$, by ℓ_p and r_p we mean the numbers as defined above.

Claim. For every $p \in S$, $\ell_p = r_p = 0$.

Claim. There is a constant c_0 such that for every $K \geq c_0$, there is a closed walk through q_0 of length exactly Kg.

Let $K = |Q|$. Now set $t = \lfloor \frac{K-1}{g} \rfloor$ and $\ell = t \cdot g$. Then for every $p \in S$, there is a path from q_0 to p of length $t'g$ on word $\alpha(p)$, and a path from p to q_0 of length $t''g$ on word $\beta(p)$, where $0 \leq t', t'' \leq t$. ($\alpha(p)$ and $\beta(p)$ are not necessarily unique. We can arbitrarily pick any such string.)

If for all accepting states $f \in F$, $\ell_f \not\equiv n \pmod g$, then $L^{=n} = \emptyset$, and the circuit C_n is empty. Otherwise, let $r = n \pmod g$. There is at least one final state f such that $\ell_f \equiv r \pmod g$. Thus there is at least one string of length $t'g + r$, with $0 \leq t' \leq t$, that takes M from q_0 to f. Putting these facts together, we can construct a proof that can be corrected in NC^0. □

Corollary 2. *For every p prime, the language $MOD_p = \{\, x \mid |x|_1 \equiv 1 \bmod p \,\}$ admits an NC^0 proof system.*

All the proof systems for regular languages in Section 3 are obtained by applying one of Theorems 5, 6, 7, in conjunction with a generic closure property.

6 Conclusion

In this paper we initiated a systematic study of the power of NC^0 proof systems. We obtained a number of upper and lower bounds, some for specific languages, some more generic. The main open question that arises from our investigation is a combinatorial characterization of all languages that admit NC^0 proof systems. Our generic results from Sect. 5 can be seen as a first step towards such a characterization for regular languages. We believe that further progress essentially depends on strengthening our lower bound techniques. In particular, we ask whether Majority admits an NC^0 proof system.

Agrawal's results on constant-depth isomorphisms [1] provide a possible tool to approach our main question: if we have an NC^0 isomorphism between two languages A and B, and B admits an NC^0 proof system, then so does A. The proofs for A are taken to be the proofs for B, then we simulate the proof system for B, and to the obtained word in B we apply the inverse of the reduction and enumerate an element from A.

In fact, our work seems to bear further interesting connections to recent examinations on isomorphism of complete sets for the class NP. This work was started in the nineties in a paper by Agrawal et al. [3] where it was shown that (1) every language complete for NP under AC^0 reductions is in fact already complete under (non-uniform) NC^0 reductions (this is called "gap theorem" in [3]), and (2) that all languages complete for NP under AC^0 reductions are (non-uniformly) AC^0 isomorphic (that is, the reduction is an AC^0 bijection). This was later improved to uniform AC^0 isomorphisms [1]. It follows from a result in [2] that this cannot be improved to P-uniform NC^0 isomorphisms. Using our results on proof systems, we obtain a very simple direct proof:

Proposition 8. *There are sets A and B that are NP-complete under NC^0 reductions but not NC^0 isomorphic.*

Proof. Let A be the NP-complete set from [11] that admits an NC^0 proof system, cf. Rem. 11. A is NP-complete under AC^0-reductions, hence by the gap theorem, under NC^0-reductions.

Let B be the disjoint union of A and Exact-OR from Sect. 4. Then B is complete for NP under NC^0 reductions because A reduces to B in NC^0.

If now A and B are NC^0-isomorphic, then we obtain an NC^0 proof system for B and from this, an NC^0 proof system for Exact-OR, a contradiction. □

Motivated by their investigation into NC^0 cryptography [4, 5], Applebaum et al. [6] investigate cryptography with constant input locality. As a related question we ask which languages can be proven by circuits that have the property that every input bit influences only constantly many output bits.

Acknowledgments. We thank Sebastian Müller (Prague) for interesting and helpful discussions on the topic of this paper.

References

1. Agrawal, M.: The isomorphism conjecture for constant depth reductions. Journal of Computer and System Sciences 77(1), 3–13 (2010)
2. Agrawal, M., Allender, E., Impagliazzo, R., Pitassi, T., Rudich, S.: Reducing the complexity of reductions. Computational Complexity 10(2), 117–138 (2001)
3. Agrawal, M., Allender, E., Rudich, S.: Reductions in circuit complexity: An isomorphism theorem and a gap theorem. J. Comput. Syst. Sci. 57(2), 127–143 (1998)
4. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. SIAM J. Comput. 36(4), 845–888 (2006)
5. Applebaum, B., Ishai, Y., Kushilevitz, E.: On pseudorandom generators with linear stretch in NC^0. Computational Complexity 17(1), 38–69 (2008)
6. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. J. Cryptology 22(4), 429–469 (2009)
7. Beyersdorff, O., Köbler, J., Müller, S.: Proof systems that take advice. Information and Computation 209(3), 320–332 (2011)
8. Beyersdorff, O., Müller, S.: A tight Karp-Lipton collapse result in bounded arithmetic. ACM Transactions on Computational Logic 11(4) (2010)
9. Cook, S.A., Krajíček, J.: Consequences of the provability of NP $\subseteq$ P/poly. The Journal of Symbolic Logic 72(4), 1353–1371 (2007)
10. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1), 36–50 (1979)
11. Cryan, M., Miltersen, P.B.: On pseudorandom generators in NC^0. In: Proc. 26th Symposium on Mathematical Foundations of Computer Science, pp. 272–284 (2001)
12. Goldwasser, S., Gutfreund, D., Healy, A., Kaufman, T., Rothblum, G.N.: Verifying and decoding in constant depth. In: Proc.39th ACM Symposium on Theory of Computing, pp. 440–449 (2007)
13. Håstad, J.: One-way permutations in NC^0. Inf. Process. Lett. 26(3), 153–155 (1987)
14. Hirsch, E.A.: Optimal Acceptors and Optimal Proof Systems. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 28–39. Springer, Heidelberg (2010)
15. Hirsch, E.A., Itsykson, D.: Hirsch and Dmitry Itsykson. On optimal heuristic randomized semidecision procedures, with application to proof complexity. In: Proc. 27th Symposium on Theoretical Aspects of Computer Science, pp. 453–464 (2010)
16. Mossel, E., Shpilka, A., Trevisan, L.: On ε-biased generators in NC^0. Random Struct. Algorithms 29(1), 56–81 (2006)
17. Pudlák, P.: Quantum deduction rules. Annals of Pure and Applied Logic 157(1), 16–29 (2009)
18. Segerlind, N.: The complexity of propositional proofs. Bulletin of Symbolic Logic 13(4), 417–481 (2007)
19. Vollmer, H.: Introduction to Circuit Complexity – A Uniform Approach. Texts in Theoretical Computer Science. Springer, Heidelberg (1999)
20. Wegener, I.: The Complexity of Boolean Functions. Wiley-Teubner series in computer science. B. G. Teubner & John Wiley, Stuttgart (1987)

The Complexity of the Cover Polynomials for Planar Graphs of Bounded Degree

Markus Bläser and Radu Curticapean

Saarland University, Department of Computer Science
{mblaeser,curticapean}@cs.uni-saarland.de

Abstract. The cover polynomials are bivariate graph polynomials that can be defined as weighted sums over all path-cycle covers of a graph. In [3], a dichotomy result for the cover polynomials was proven, establishing that their evaluation is $\#\mathbf{P}$-hard everywhere but at a finite set of points, where evaluation is in $\mathbf{FP}$. In this paper, we show that almost the same dichotomy holds when restricting the evaluation to planar graphs. We even provide hardness results for planar DAGs of bounded degree. For particular subclasses of planar graphs of bounded degree and for variants thereof, we also provide algorithms that allow for polynomial-time evaluation of the cover polynomials at certain new points by utilizing Valiant's holographic framework.

1 Introduction

Graph polynomials map directed or undirected graphs to polynomials in one or more variables, such that isomorphic graphs are mapped to the same polynomial. Probably the most famous graph polynomials are the chromatic polynomial and its generalization, the Tutte polynomial. By substituting values for the indeterminates, we can view graph polynomials as parameterized graph invariants. For instance, the chromatic polynomial χ evaluated at a natural number $k \in \mathbb{N}$ counts the number of proper k-colorings of a graph. Its bivariate generalization, the Tutte polynomial T, has many interpretations from different fields of combinatorics. For example, $T(G;1,1)$ is the number of spanning trees of a graph G, $T(G;1,2)$ is the number of spanning subgraphs, $T(G;2,0)$ is the number of acyclic orientations of G and so on, see [18] for an overview.

While the Tutte polynomial has been established for undirected graphs, the cover polynomial by Chung and Graham [9] and its geometric variant by D'Antona and Munarini [11] are analogues for the directed case. Both graph polynomials satisfy similar identities such as a contraction-deletion identity and a product rule. The cover polynomial has connections to rook polynomials and drop polynomials and it "interpolates" between the permanent and the determinant of a graph. Like many other graph polynomials, the cover polynomial is of interest because it combines a variety of combinatorial properties of a graph into one unifying framework.

1.1 Previous Results

For the chromatic polynomial χ, Linial [16] showed that the evaluation is $\#\mathbf{P}$-hard except for the points 0, 1, and 2, at which the evaluation is in $\mathbf{FP}$. Jaeger, Vertigan, and

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 96–107, 2011.

Welsh [15] generalized this by proving that the Tutte polynomial is $\#\mathbf{P}$-hard except along one hyperbola and at nine special points. For both versions of the cover polynomial, a similar dichotomy was achieved [3]: Evaluating the cover polynomial is $\#\mathbf{P}$-hard except for three and two points, respectively, where the problem is computable in polynomial time. Makowsky and Lotz [17] proved that the coloured Tutte polynomial by Bollobás and Riordan [7] is complete for Valiant's algebraic complexity class $\mathbf{VNP}$, and Goldberg and Jerrum [12] showed that the Tutte polynomial is inapproximable in large regions of the plane, see [4] for a similar result for the cover polynomial. For other graph polynomials, similar results have been achieved in the recent years, see e.g. [1,5,6,14]. Makowsky [18] conjectures that this is a general phenomenon, i.e., that every polynomial that is definable in monadic second order logic is $\#\mathbf{P}$-hard to evaluate almost everwhere, provided it has at least one hard point. Even stronger, he conjectures that the evaluation at any point can be reduced to the evaluation at (almost) any other point. Many of the graph polynomials studied in the literature are definable in monadic second order logic and satisfy this conjectured property.

On the positive side, the Tutte and the cover polynomials can be computed in time exponential in the number of vertices for general graphs [2] and in linear time for graphs of bounded tree-width [10].

It is an interesting question whether the problem becomes easier for restricted graph classes, e.g., planar graphs. Vertigan [20] has obtained a dichotomy for the Tutte polynomial for planar graphs and Goldberg and Jerrum [13] obtain some inapproximability results similar to the ones in [12]. The matching polynomials are the only other graph polynomials for which we are aware of hardness results for planar graphs, see e.g. [21].

1.2 Our Results

In this paper, we consider the complexity of evaluating the cover polynomial for planar graphs. We show that the cover polynomial is $\#\mathbf{P}$-hard to evaluate almost everywhere for planar graphs of bounded degree.[1] For the geometric cover polynomial, we obtain the same dichotomy as in [3] for general graphs. For the factorial cover polynomial, only the points $(n, -1)$ with $n \in \mathbb{N} \setminus \{0, 1\}$ remain unclassified. For the ease of presentation, we present all our results over $\mathbb{Q}$.

We complement the hardness results by providing positive results for slightly more restricted graph classes via holographic algorithms. As these graphs can have unbounded tree-width, the results of [10] do not apply. Our results are almost as tight as possible. More precisely, we show that evaluating the cover polynomials is $\#\mathbf{P}$-hard almost everywhere for planar graphs with maximum indegree 2 and maximum outdegree 3. We call such graphs 3-nice.

On the other hand, we show that some of the hard points for 3-nice graphs are easy for (non-necessarily planar) graphs whose indegrees and outdegrees are bounded by 2. For a few other points, we can only show hardness for 4-nice graphs. However, one of them can be computed in polynomial time for graphs with outdegree 3 whose *split graph* is planar (see Section 5 for a precise defintion).

[1] In this paper, "$\#\mathbf{P}$-hardness" refers to "$\#\mathbf{P}$-hardness under poly-time Turing reductions".

The hardness results for nice graphs even hold for directed acyclic graphs (DAGs). Furthermore, we introduce a novel horizontal reduction for the geometric cover polynomial that increases the maximum degree only by 1, as opposed to the polynomial factors incurred by the other reductions we are aware of.

Omitted proofs are given in the full version of this paper, which is available at arXiv.

1.3 The Cover Polynomials

Our results regard the factorial cover polynomial C_{fac} and the geometric cover polynomial C_{geo}, first defined in [9] and [11], respectively. To define these polynomials, we can use the concept of path-cycle covers, as defined below:

Definition 1. *Let $G = (V, E)$ be a digraph. A set $C \subseteq E$ is a* path-cycle cover *of G if C is a vertex-disjoint union of directed paths and cycles such that every $v \in V$ is contained in a path or cycle.*[2] *We denote the set of all path-cycle covers of G by $\mathcal{PC}[G]$.*

For $C \in \mathcal{PC}[G]$, the numbers of paths and cycles of C are denoted by $\rho(C)$ and $\sigma(C)$, respectively. C is a cycle cover *if $\rho(C) = 0$, and a* path cover *if $\sigma(C) = 0$. $\mathcal{C}[G]$ and $\mathcal{P}[G]$ denote the sets of cycle covers and path covers of G, respectively.*

The cover polynomials of a graph can be defined as weighted sums over its path-cycle covers, each such cover being weighted by a bivariate monomial. In the following, we denote the falling factorial by $x^{\underline{k}} := (x)(x-1)\dots(x-k+1)$.

Definition 2. *Given a digraph G, define C_{fac} and C_{geo} as follows:*

$$C_{\mathrm{fac}}(G; x, y) := \sum_{C \in \mathcal{PC}[G]} x^{\underline{\rho(C)}} y^{\sigma(C)} \qquad C_{\mathrm{geo}}(G; x, y) := \sum_{C \in \mathcal{PC}[G]} x^{\rho(C)} y^{\sigma(C)}.$$

On the y-axis, C_{fac} and C_{geo} coincide, since $0^{\rho(C)} = 0^{\underline{\rho(C)}}$ holds. We facilitate notation by writing $C(G; 0, y) := C_{\mathrm{fac}}(G; 0, y) = C_{\mathrm{geo}}(G; 0, y)$.

Remark 1. Let $c_G(i, j)$ denote the number of path-cycle covers $C \in \mathcal{PC}[G]$ that satisfy $\rho(C) = i$ and $\sigma(C) = j$. With this notation, we can write:

$$C_{\mathrm{fac}}(G; x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} c_G(i, j) x^{\underline{i}} y^j \qquad C_{\mathrm{geo}}(G; x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} c_G(i, j) x^i y^j.$$

This implies that, for any digraph G with adjacency matrix A, we have the identities $C_{\mathrm{fac}}(G; 1, 0) = \#HamPaths(G)$ and $C(G; 0, 1) = \mathrm{perm}(A)$. We define the counting problems $C_{\mathrm{fac}}(x, y)$ and $C_{\mathrm{geo}}(x, y)$, each mapping an input digraph G to the value $C_{\mathrm{fac}}(G; x, y)$ or $C_{\mathrm{geo}}(G; x, y)$, respectively.[3] Furthermore, we use the symbol C to

[2] Trivial paths (consisting of a single isolated vertex) are permitted in this definition.

[3] The symbols C_{fac} and C_{geo} may either refer to the graph polynomials or to the associated counting problems. We adopt the following standard notational conventions:

- $C_{\mathrm{fac}}(G)$ is the factorial cover polynomial of G.
- $C_{\mathrm{fac}}(G; x, y)$ is the value of $C_{\mathrm{fac}}(G)$, evaluated at the point (x, y).
- $C_{\mathrm{fac}}(x, y)$ is the weighted counting problem of computing $C_{\mathrm{fac}}(G; x, y)$ on input G.

make statements about both C_{fac} and C_{geo}: For example, when writing that $C(x, y)$ is $\#\mathbf{P}$-hard, we intend to express that both $C_{\mathrm{fac}}(x, y)$ and $C_{\mathrm{geo}}(x, y)$ are $\#\mathbf{P}$-hard.

A dichotomy result for the complexity of both polynomials for *general* digraphs has already been established in [3], where C_{fac} and C_{geo} are shown to be $\#\mathbf{P}$-hard everywhere except for the evaluations $C(0, 0)$, $C(0, -1)$ and $C_{\mathrm{fac}}(1, -1)$, which are proven to be in $\mathbf{FP}$. However, the constructions given in [3] are inherently non-planar and increase the maximum degree of graphs by a polynomial factor.

In the present paper, the dichotomy result for C_{geo} given in [3] is shown to hold also when the input graphs are required to be planar. Furthermore, we prove that C_{geo} is $\#\mathbf{P}$-hard for planar graphs of bounded degree at all points that are not on the y-axis. For C_{fac}, we obtain similar, but slightly weaker results. Our results are summarized in two theorems, establishing hardness for planar graphs of unbounded degree first:

Theorem 1. *Let $x, y \in \mathbb{Q}$. If $y \notin \{-1, 0\}$, then $C(0, y)$ is $\#\mathbf{P}$-hard for planar graphs. If $(x, y) \neq (0, 0)$ and $y \neq -1$, then $C_{\mathrm{fac}}(x, y)$ is $\#\mathbf{P}$-hard for planar graphs.*

In the next theorem, we consider more restricted graphs: We call a planar graph G *k-nice* if all of its vertices feature indegree ≤ 2 and outdegree $\leq k$. (The sum of all indegrees and the sum of all outdegrees is of course the same.) Throughout this paper, we denote by $\mathcal{F} := \{-3, -2, -1, 0\}$ the set of "bad" x-coordinates.

Theorem 2. *Let $x, y \in \mathbb{Q}$. For $x \notin \mathbb{N}_0$, $C_{\mathrm{fac}}(x, y)$ is $\#\mathbf{P}$-hard for 3-nice DAGs.*
If $x \neq 0$, $C_{\mathrm{geo}}(x, y)$ is $\#\mathbf{P}$-hard for 4-nice DAGs.
If $x \notin \mathcal{F}$, $C_{\mathrm{geo}}(x, y)$ is $\#\mathbf{P}$-hard even for 3-nice DAGs.

For C_{geo}, the only points not covered by either of the two theorems are $(0, 0)$ and $(0, -1)$. For C_{fac}, only the points $(0, 0)$ and $(n, -1)$ for $n \in \mathbb{N}_0$ remain uncovered. Recall that the problems $C(0, 0), C(0, -1)$ and $C_{\mathrm{fac}}(1, -1)$ are all in $\mathbf{FP}$.

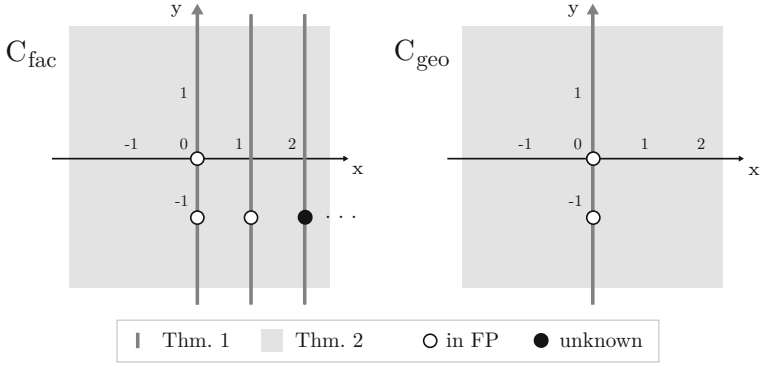


Fig. 1. Plots of the hardness results for the cover polynomials C_{fac} and C_{geo} on planar graphs. Every point is annotated with the strongest theorem that proves its hardness.

1.4 Matching Polynomials

Several results in this paper require $\#\mathbf{P}$-hardness of two matching-related graph polynomials for planar graphs. The hardness results we use are proven in [8] and [21] using Valiant's holographic framework [19].

Definition 3. *Let $G = (V, E)$ be an undirected graph. Denote by $\mathcal{M}[G]$ the set of all matchings of G. For $M \in \mathcal{M}[G]$, let $\operatorname{usat}(M)$ denote the set of unmatched vertices in M. Define the univariate graph polynomial $\operatorname{Match}(G)$ as follows:*

$$\operatorname{Match}(G; x) = \sum_{M \in \mathcal{M}[G]} x^{|\operatorname{usat}(M)|}$$

Let $G = (L \cup R, E)$ be an undirected bipartite graph. For $M \in \mathcal{M}[G]$, let $\operatorname{usat}_R(M)$ denote the set of unmatched R-vertices in M. Define $\operatorname{RMatch}(G)$ as follows:

$$\operatorname{RMatch}(G; x) = \sum_{M \in \mathcal{M}[G]} x^{|\operatorname{usat}_R(M)|}$$

For bipartite graphs, there is a simple connection between Match and RMatch.

Lemma 1. *For every bipartite undirected graph $G = (L \cup R, E)$, it holds that*

$$\operatorname{Match}(G; x) = x^{|L|-|R|} \operatorname{RMatch}(G; x^2)$$

Proof. Follows easily from the observation that, for every $M \in \mathcal{M}[G]$, we have the identity $|\operatorname{usat}(M)| = |L| - |R| + 2 \cdot |\operatorname{usat}_R(M)|$. □

We collect hardness results about Match and RMatch in the next lemma:

Lemma 2. *For planar 3-regular graphs, $\operatorname{Match}(\lambda)$ is $\#\mathbf{P}$-hard at all $\lambda \in \mathbb{C} \setminus \{0\}$. For planar bipartite $(2,3)$-regular graphs, $\operatorname{RMatch}(\lambda)$ is $\#\mathbf{P}$-hard at all $\lambda \in \mathbb{Q} \setminus \mathcal{F}$.*

Proof. The first claim follows from a subclaim in [8, Proof of Thm. 6.1]. The second claim can be derived from [21], where RMatch is named $\#\mathrm{BP}{-}\lambda\mathrm{Matchings}$. □

2 Horizontal Reductions

2.1 A Horizontal Reduction for C_{fac}

For a digraph G, let $G^{(k)}$ denote the graph obtained from G by adding k isolated vertices. It was observed in [9] that $C_{\mathrm{fac}}(G^{(k)}; x, y) = x^{\underline{k}} C_{\mathrm{fac}}(G; x - k, y)$. This yields:

Lemma 3. *Let $x, y \in \mathbb{Q}$. If $x \in \mathbb{Q} \setminus \mathbb{N}_0$, we have $C_{\mathrm{fac}}(x', y) \leq_{\mathrm{p}}^{\mathrm{T}} C_{\mathrm{fac}}(x, y)$ for all $x' \in \mathbb{Q}$. If $x \in \mathbb{N}_0$, we have $C_{\mathrm{fac}}(x', y) \leq_{\mathrm{p}} C_{\mathrm{fac}}(x, y)$ for all $x' \in \mathbb{N}_0$ with $x' \leq x$.*

Note that adding isolated vertices preserves planarity and does not increase the maximum degree of a graph. Furthermore, given oracle access to $C_{\mathrm{fac}}(x, y)$ for $x \in \mathbb{Q} \setminus \mathbb{N}_0$, we can actually interpolate the whole polynomial $C_{\mathrm{fac}}(G; \cdot, y)$.

2.2 A New Horizontal Reduction for C_{geo}

A horizontal reduction for C_{geo} is given in [3]. The graph transformation used in this reduction increases the maximum degree by a polynomial factor and transforms simple graphs to multigraphs. We introduce a more conservative transformation:

Definition 4. *Let $l \in \mathbb{N}_0$. Denote by $Q^l = (V, E)$ the graph with $V = \{q_1 \dots q_l\}$ and*

$$E = \{(q_i, q_{i+1}) \mid 1 \le i < l \wedge i \text{ odd}\} \cup \{(q_{i+1}, q_i) \mid 1 \le i < l \wedge i \text{ even}\}$$

Let G be a digraph. Denote by G^{*l} the graph obtained from G by attaching to each $v \in V(G)$ a fresh copy of Q^l, identifying the vertices v and q_1. We call the copy of Q^l at v the alternating path rooted at v *and denote it by Q_v.*

Note that for all $l \in \mathbb{N}_0$, the graph G^{*l} is $(k+1)$-nice if G is k-nice. The construction given in Def. 4 yields the following horizontal reduction for C_{geo}:

Lemma 4. *For all $x, x', y \in \mathbb{Q}$ with $x < 0$ and $x \neq -4$, $C_{\text{geo}}(x', y) \le_{\text{p}}^{\text{T}} C_{\text{geo}}(x, y)$.*

Proof. The set $\mathcal{PC}[G^{*l}]$ admits a partition into classes $\mathcal{E}_D$, for $D \in \mathcal{PC}[G]$, that are defined as follows: $C \in \mathcal{E}_D \Leftrightarrow C[V(G)] = D$. Every $C \in \mathcal{E}_D$ can be written as $C = D \cup A$, where D is the "inner" cover and A is a cover of the alternating paths.

Using this partition, we can rewrite $C_{\text{geo}}(G^{*l}; x, y)$ as a sum over $D \in \mathcal{PC}[G]$, where each D is weighted by an expression E_D that depends only on $\mathcal{E}_D$:

$$C_{\text{geo}}(G^{*l}; x, y) = \sum_{D \in \mathcal{PC}[G]} \underbrace{\sum_{C \in \mathcal{E}_D} x^{\rho(C)} y^{\sigma(C)}}_{=: E_D}$$

To compute E_D, we examine all valid covers of the alternating paths of G^{*l} that extend D to some $C \in \mathcal{E}_D$. Let D_I be the set of vertices with outdegree 1 in D, and D_O be the set of vertices with outdegree 0 in D. Note that $|D_O| = \rho(C)$, since every path in D has a unique vertex of outdegree 0. Since alternating paths rooted at different vertices can be covered independently of each other, it suffices to consider the possibilities to cover Q_v separately for each $v \in V(G)$, distinguishing between $v \in D_I$ and $v \in D_O$.

For $v \in D_I$, the first edge of Q_v must not be present in any $C \in \mathcal{E}_D$. For $v \in D_O$, the first edge of Q_v may or may not be present in any $C \in \mathcal{E}_D$. Defining

$$\mathcal{T}_l := \sum_{C \in \mathcal{PC}[Q^l],\ (q_1, q_2) \notin C} x^{\rho(C)} \quad \text{and} \quad \mathcal{P}_l := \sum_{C \in \mathcal{PC}[Q^l],\ (q_1, q_2) \in C} x^{\rho(C)},$$

it can be shown that

$$C_{\text{geo}}(G^{*l}; x, y) = \left(\frac{1}{x}\mathcal{T}_l\right)^n \cdot C_{\text{geo}}\left(G; x\left(1 + \frac{\mathcal{P}_l}{\mathcal{T}_l}\right), y\right)$$

In the full version, we prove that, for $x < 0$ with $x \neq -4$, we have $\mathcal{T}_l \neq 0$ and $\frac{\mathcal{P}_l}{\mathcal{T}_l}$ is strictly monotonically increasing in l. Thus, $C_{\text{geo}}(G^{*1}; x, y), \dots, C_{\text{geo}}(G^{*n}; x, y)$ are the values of n different points of $C_{\text{geo}}(G; \cdot, y)$. This allows us to interpolate the polynomial $C_{\text{geo}}(G; \cdot, y)$, proving $C_{\text{geo}}(x', y) \le_{\text{p}}^{\text{T}} C_{\text{geo}}(x, y)$ for all $x' \in \mathbb{Q}$. □

3 Proof of Thm. 1

3.1 A Previous Approach

To understand a previous hardness result from [3], a generalization of the cover polynomial, the so-called *weighted* cover polynomial, is needed:

Definition 5. *Given an edge-weighted digraph $G = (V, E, w)$, the* weight *of a cycle cover $C \in \mathcal{C}[G]$ is defined as $w(C) = \prod_{e \in C} w(e)$. Building upon this, the* weighted cover polynomial *C^w is then defined as $C^w(G; 0, y) = \sum_{C \in \mathcal{C}[G]} w(C) y^{\sigma(C)}$.*

Obviously, we have $C(0, y) \leq_{par} C^w(0, y)$. On the other hand, if restricted to graphs with a constant number of different weights, $C^w(0, y) \leq_p^T C(0, y)$ also holds, as proven via polynomial interpolation and thickening of edges in [3].

In [3], hardness of the problem $C^w(0, y)$ for $y \notin \{-1, 0\}$ is shown via vertical reductions that use *equality gadgets*. An equality gadget E is a weighted digraph with four special nodes u, v and u', v', as shown in Fig. 2. Given a graph G with edges $e = \{a, b\}$ and $e' = \{a', b'\}$, these edges can be replaced by a local copy of E by identifying a, b, a', b' with u, v, u', v' each. This yields a new graph $G_{e=e'}$, in which e and e' correspond to the paths highlighted in Fig. 2.

An important observation made in [3] is that in every $C \in \mathcal{C}[G_{e=e'}]$, the equality gadget either is in one of the good states shown in Fig. 2 or C has a partner C^- with $w(C) = -w(C^-)$. This enforces that cycle covers that include only one of the highlighted paths corresponding to e or e' cancel in the weighted cover polynomial. This result will be stated more precisely later.

The hardness proof in [3] is then continued by observing $\#\mathbf{P}$-hardness of $C(0, 1)$ and proving $C(0, 1) \leq_p^T C(0, y)$ via vertical reductions (using equality gadgets). This fails on the class of planar graphs since the vertical reductions do not preserve planarity.

3.2 Reducing Match to $C(0, y)$

In contrast to the method sketched before, our approach relies on the $\#\mathbf{P}$-hardness of $\mathrm{Match}(\lambda)$ for planar graphs at all $\lambda \in \mathbb{C} \setminus \{0\}$. We construct a planarity-preserving

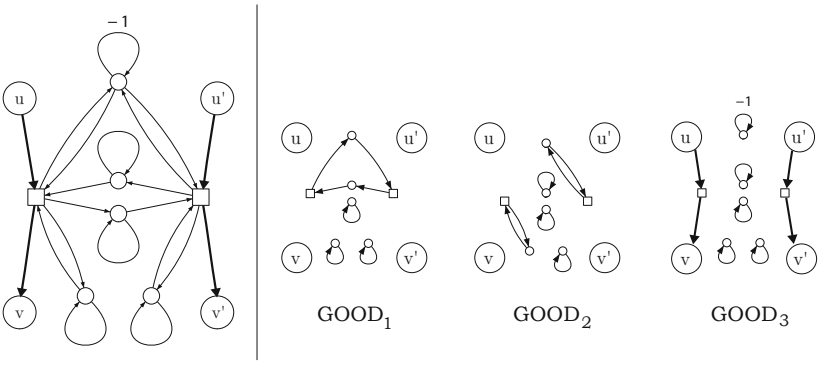


Fig. 2. (left) The equality gadget from [3]. The thick lines represent the paths that replace edges. (right) The good states of the equality gadget.

graph transformation T together with polynomial-time computable functions f and g such that for all graphs G with n vertices and m edges, we have

$$C^w(T(G); 0, y) = g(y, m, n) \cdot \mathrm{Match}(G; f(y)).$$

Hardness of $\mathrm{Match}(\lambda)$ for planar graphs at all $\lambda \in \mathbb{C} \setminus \{0\}$ then allows us to conclude hardness of $C(0, y)$ on planar graphs, provided that y satisfies $f(y) \neq 0$ and that $g(y, m, n) \neq 0$. These conditions will be fulfilled for all $y \notin \{-1, 0\}$.

Our construction consists of two graph transformations, namely G' and $T(G)$, of which G' can be considered as an intermediate step towards $T(G)$:

Definition 6. *Let $G = (V, E)$ be an undirected graph. The digraph $G' = (V, E')$ is defined by attaching a self-loop to each $v \in V$ and replacing every $e \in E$ with $e = \{u, v\}$ by the edge pair $\overrightarrow{e} = (u, v)$ and $\overleftarrow{e} = (v, u)$.*

The graph $T(G)$ is obtained from G' by connecting every edge pair $\overleftarrow{e}, \overrightarrow{e}$ to a fresh copy of the equality gadget from Fig. 2.

Both G' and $T(G)$ are planar if G is planar, since the transformations only add self-loops and perform local replacements of edges by planar gadgets.

We aim at isolating a particular set of cycle covers of G', namely that of *consistent* cycle covers $\mathcal{C}_{\mathrm{cons}}[G'] \subseteq \mathcal{C}[G']$, as defined below. Consistent cycle covers are useful, since there is a bijection between $\mathcal{C}_{\mathrm{cons}}[G']$ and $\mathcal{M}[G]$, as proven in Lemma 5.

Definition 7. *Let $G = (V, E)$ be undirected. We call $C \in \mathcal{C}[G']$* consistent *and write $C \in \mathcal{C}_{\mathrm{cons}}[G']$ iff for all edge pairs $\overleftarrow{e}, \overrightarrow{e} \in E(G')$, we have $\overleftarrow{e} \in C \Leftrightarrow \overrightarrow{e} \in C$.*

In the next lemma, we construct a bijection B that translates matchings $M \in \mathcal{M}[G]$ to consistent cycle covers $B(M) \in \mathcal{C}_{\mathrm{cons}}[G']$, and furthermore respects the structure of M in that $\sigma(B(M)) = |\mathrm{usat}(M)| + |M|$.

Lemma 5. *Let G be an undirected graph. Define $B : \mathcal{M}[G] \to \mathcal{C}_{\mathrm{cons}}[G']$ as follows: Given $M \in \mathcal{M}[G]$, attach a self-loop to every $v \in \mathrm{usat}(M)$ and replace every $e \in M$ by the edge pair $\overleftarrow{e}, \overrightarrow{e}$ to obtain $B(M)$. B is bijective and satisfies*

$$\forall M \in \mathcal{M}[G] : \sigma(B(M)) = |M| + |\mathrm{usat}(M)|. \tag{1}$$

Next, we analyze the equality gadget to show a relation between $\mathcal{C}_{\mathrm{cons}}[G']$ and $\mathcal{C}[T(G)]$. Similarly to the restriction of $\mathcal{C}[G']$ to *consistent* cycle covers, we now restrict $\mathcal{C}[T(G)]$ to the subset of so-called *"good"* cycle covers, which can be related to $\mathcal{C}_{\mathrm{cons}}[G']$.

Definition 8. *Let H be a digraph containing an equality gadget E as an induced subgraph and let $C \in \mathcal{C}[H]$. We say that E* is in a good state *if $C[E]$ is one of the graphs* GOOD_1, GOOD_2 *or* GOOD_3 *in Fig. 2. If all equality gadgets of C are in good states, we call C* good. *The set of good cycle covers is denoted by $\mathcal{C}_{\mathrm{good}}[H]$.*

As shown in [3, Lemma 5], equality gadgets ensure that the contributions of "bad" cycle covers cancel out in C^w. More precisely, their result states:

$$C^w(T(G); 0, y) = \sum_{C \in \mathcal{C}[T(G)]} w(C) y^{\sigma(C)} = \sum_{C \in \mathcal{C}_{\mathrm{good}}[T(G)]} w(C) y^{\sigma(C)}. \tag{2}$$

Consider some $C \in \mathcal{C}_{\text{good}}[T(G)]$: For every edge pair $\overleftarrow{e}, \overrightarrow{e}$, the corresponding equality gadget is in a good state and thus enforces that either both $\overleftarrow{e}, \overrightarrow{e} \in C$ (state GOOD_3) or both $\overleftarrow{e}, \overrightarrow{e} \notin C$ (states GOOD_2 and GOOD_3).[4] This allows us to relate good cycle covers in $T(G)$ to consistent cycle covers in G', and thus finally to matchings in G:

Definition 9. *Let G be an undirected graph, let $C \in \mathcal{C}_{\text{good}}[T(G)]$ and $C' \in \mathcal{C}_{\text{cons}}[G']$. We say that C* originates from *C' if C can be obtained from C' as follows:*

1. *Start with C'.*
2. *For every edge pair $\overleftarrow{e}, \overrightarrow{e}$ in G':*
   (a) *If both $\overleftarrow{e}, \overrightarrow{e} \in C'$, connect them by an equality gadget in state GOOD_3.*
   (b) *If both $\overleftarrow{e}, \overrightarrow{e} \notin C'$, connect them by an eq. gadget in GOOD_1 or GOOD_2.*

Let $M \in \mathcal{M}[G]$. We say that C originates from *M if C originates from $B(M)$.*

The set $\mathcal{C}_{\text{good}}[T(G)]$ can be partitioned into classes $\mathcal{O}[M]$, for $M \in \mathcal{M}[G]$, such that $C \in \mathcal{O}[M]$ iff C originates from M: Given any good C, the M it originates from is *uniquely* determined by the equality gadgets in state GOOD_3. In Lemma 6, we explicitly compute the contribution of each such matching M to $C^w(T(G))$, thus rewriting $C^w(T(G))$ in terms of $\text{Match}(G)$. The proof can be found in the full version.

Lemma 6. *For every graph G with n vertices and m edges, and every $y \in \mathbb{Q}$:*

$$C^w(T(G); 0, y) = g(y, m, n) \cdot \text{Match}(G; f(y)) \tag{3}$$

with $g(y,m,n) = (-1)^{\frac{n}{2}} y^{\frac{n}{2}+5m} \left(1 + \frac{1}{y}\right)^{m-\frac{n}{2}}$ and $f(y) = \sqrt{-(y+1)}$.

Using Lemma 6, we are now finally able to complete the proof of Thm. 1:

Proof (of Theorem 1). Under the condition that $g(y,m,n) \neq 0$, we can use (3) to compute $\text{Match}(G; f(y))$ from $C^w(T(G); 0, y)$. Thus, for $y \notin \{-1, 0\}$, we get:

$$\text{Match}(f(y)) \leq_p C^w(0, y) \leq_p^T C(0, y),$$

where the last reduction follows from interpolation and edge thickening.

From Lemma 2, we know that $\text{Match}(f(y))$ is $\#\mathbf{P}$-hard on planar graphs at all complex $f(y) \neq 0$, which is equivalent to $y \neq -1$. Since T preserves planarity, we conclude that $C(0, y)$ is also $\#\mathbf{P}$-hard for planar graphs, provided that the reduction chain works (which it does for $y \notin \{-1, 0\}$) *and* that $\text{Match}(f(y))$ is $\#\mathbf{P}$-hard (which it is for $y \neq -1$). Thus $C(0, y)$ is $\#\mathbf{P}$-hard for planar graphs at all $y \notin \{-1, 0\}$.

With Lemma 3, this result can be stretched along horizontal lines, yielding $\#\mathbf{P}$-hardness of $C_{\text{fac}}(x, y)$ at all $x, y \in \mathbb{Q}$ with $y \notin \{-1, 0\}$. Finally, hardness of $C_{\text{fac}}(x, 0)$ for $x \neq 0$ can be obtained with the horizontal reduction from Lemma 3, observing that $C_{\text{fac}}(G; 1, 0) = \#HamPaths(G)$, which is $\#\mathbf{P}$-hard for planar graphs. □

[4] To avoid awkward notation, we denote by $\overleftarrow{e}, \overrightarrow{e}$ the highlighted paths in Fig. 2 that represent the edges $\overleftarrow{e}, \overrightarrow{e}$ in the equality gadget.

4 Proof of Thm. 2

In this section, we show Thm. 2, which claims that $C_{\text{geo}}(x,y)$ is $\#\mathbf{P}$-hard at $x,y \in \mathbb{Q}$ with $x \neq 0$ for 4-nice DAGs, and even $\#\mathbf{P}$-hard for 3-nice DAGs if $x \notin \mathcal{F}$. Concerning C_{fac}, it is claimed that for $x \notin \mathbb{N}_0$, $C_{\text{fac}}(x,y)$ is $\#\mathbf{P}$-hard for 3-nice DAGs.

Since every DAG D is acyclic, every $C \in \mathcal{PC}[D]$ is a path cover, yielding the following identity, which naturally also holds for C_{fac} when adapted accordingly:

$$C_{\text{geo}}(D;x,y) = \sum_{C \in \mathcal{PC}[D]} x^{\rho(C)} y^{\sigma(C)} = \sum_{C \in \mathcal{P}[D]} x^{\rho(C)}. \tag{4}$$

Thus, Thm. 2 shows that in fact already a weighted sum over *path* covers is $\#\mathbf{P}$-hard, i.e., allowing cycles in the covers does not change the complexity of $C_{\text{geo}}(x,y)$.

We first prove a modified (and slightly weaker) version of Thm. 2:

Lemma 7. *For $x,y \in \mathbb{Q}$, $x \notin \mathcal{F}$, $C_{\text{geo}}(x,y)$ is $\#\mathbf{P}$-hard for 3-nice DAGs.*

Proof. We aim at reducing $\text{RMatch}(x) \leq_{\text{p}} C_{\text{geo}}(x,y)$. The former is $\#\mathbf{P}$-hard for $(2,3)$-regular bipartite planar graphs at $x \notin \mathcal{F}$, as stated in Lemma 2.

We start with a $(2,3)$-regular planar bipartite graph $G = (L \cup R, E)$, orientate all edges from L to R and call the resulting 3-nice DAG $\vec{G}$. By the general observation made in (4), we obtain $C_{\text{geo}}(\vec{G};x,y) = \sum_{C \in \mathcal{P}[\vec{G}]} x^{\rho(C)}$.

We first prove that $\mathcal{M}[G] \sim \mathcal{P}[\vec{G}]$: Let $f : \mathcal{M}[G] \to \mathcal{P}[\vec{G}]$ be the function that directs the edges in every $M \in \mathcal{M}[G]$ from L to R. This function clearly is injective. It is also surjective: Since $\vec{G}$ is obtained by orienting all edges in G from L to R, each path in $\vec{G}$ has maximum length 1. Thus, every path cover $C \in \mathcal{P}[\vec{G}]$ is a vertex-disjoint union of paths of maximum length 1, which in turn implies that $C = f(M)$ for the matching M obtained from C by ignoring all edge orientations.

Let M, C be such that $f(M) = C$. Obviously, we have $\rho(C) = |M| + |\text{usat}(M)|$, which implies $\rho(C) = \frac{n}{2} + \frac{|\text{usat}(M)|}{2}$. This allows us to write

$$C_{\text{geo}}(\vec{G};x,y) = \sum_{C \in \mathcal{P}[\vec{G}]} x^{\rho(C)} = \sqrt{x}^n \sum_{M \in \mathcal{M}[G]} \sqrt{x}^{|\text{usat}(M)|} = \sqrt{x}^n \cdot \text{Match}(G;\sqrt{x}).$$

Next, we apply Lemma 1 to obtain the identity

$$C_{\text{geo}}(\vec{G};x,y) = \sqrt{x}^n \cdot \text{Match}(G;\sqrt{x}) = \underbrace{\sqrt{x}^n \cdot \sqrt{x}^{|L|-|R|}}_{=x^{|L|}} \cdot \text{RMatch}(G;x). \tag{5}$$

Note that (5) also holds for $x < 0$: As it holds for all $x \geq 0$, it equates the polynomials $C_{\text{geo}}(\vec{G};x,y)$ and $x^{|L|} \cdot \text{RMatch}(G;x)$ at infinitely many (and thus at all) points. □

In Lemma 7, we cannot give hardness results for the set $\mathcal{F}$ on which RMatch is not shown to be $\#\mathbf{P}$-hard. However, horizontal reductions allow us to stretch the hardness results to points with x-coordinates from $\mathcal{F}$:

Corollary 1. *For $x,y \in \mathbb{Q}$ with $x \neq 0$, $C_{\text{geo}}(x,y)$ is $\#\mathbf{P}$-hard for 4-nice DAGs. For $x,y \in \mathbb{Q}$ with $x \notin \mathbb{N}_0$, $C_{\text{fac}}(x,y)$ is $\#\mathbf{P}$-hard for 3-nice DAGs.*

5 Positive Results

In this section, we approach the hardness results from the opposite side by providing positive results. These results build upon a simple relation between $C_{\text{geo}}(G)$ and $\text{RMatch}(S(G))$, where $S(G)$ denotes the *split graph* obtained from a digraph G as follows: Split each vertex $v \in V(G)$ into the vertices v^{out} and v^{in} and transform each $e \in E(G)$ of the form $e = (u, v)$ to the undirected edge $\{v^{\text{out}}, v^{\text{in}}\}$ in $E(S(G))$. Note that $S(G)$ is bipartite. Some results are summarized in Table 1.

Lemma 8. *Let G be a digraph and $S(G) = (L \cup R, E)$ be its split graph. The function S is a bijection $S : \mathcal{PC}[G] \to \mathcal{M}[S(G)]$ with the property*

$$\forall C \in \mathcal{PC}[G] : \rho(C) = |\text{usat}_R(S(C))|. \tag{6}$$

This especially implies that for all $x \in \mathbb{Q}$, we have $C_{\text{geo}}(G; x, 1) = \text{RMatch}(S(G); x)$.

A first positive result that can be shown using Lemma 8 is the following:

Lemma 9. *Let $x \in \mathbb{Q}$ and let G be a (non-necessarily planar) digraph of maximum in- and outdegree 2. Then $C(x, 1) \in \mathbf{FP}$.*

Next, given a digraph G, we aim at computing $C(G)$ by applying algorithms for planar graphs to $S(G)$. If $S(G)$ is planar, we call G *split-planar*. It should be emphasized that the class of split-planar graphs differs from that of planar graphs: The complete digraph of order 3 (with self-loops) is planar, but not split-planar. Likewise, there exists a split-planar orientation of $K_{3,3}$. More details can be found in the full version of this paper. It is easily checked that all hardness results proven in Thm. 2 carry over directly to split-planar graphs. Lemma 8 yields a positive result for C_{fac} on split-planar graphs:

Lemma 10. *Let $k \in \mathbb{N}_0$. Then $C_{\text{fac}}(G; k, 1)$ can be computed in time $O(n^k)$ for split-planar graphs G of order n.*

Finally, we use Lemma 8 together with a holographic algorithm first presented in [19] to obtain the following positive result:

Lemma 11. *Let $k \in \mathbb{N}_0$ and let $G = (V, E)$ be a split-planar digraph whose vertices all have indegree ≤ 2 and outdegree equal to k. Then we can compute the value $C_{\text{geo}}(G; -k, 1)$ in polynomial time.*

Lemma 11 supports an alternative formulation, noted in a similar form in [19]: Let $k \in \mathbb{N}$. If $G = (V, E)$ is split-planar and all $v \in V$ have indegree ≤ 2 and outdegree equal to 0 or $k+t_v(k+1)$, for $t_v \in \mathbb{N}_0$, then we can compute $C_{\text{geo}}(G; 1, 1) \bmod (k+1)$ in polynomial time. With $k = 1$, this implies that $C_{\text{geo}}(G; 1, 1) \bmod 2$ for split-planar G can be computed in poly-time if G has max. indegree 2 and only odd outdegrees.

Table 1. Positive results and corresponding hardness results in comparison

evaluation point	is in FP if...	is #P-hard if...
$C_{\text{geo}}(G; x, 1)$ at $x \in \mathbb{Q}$	in-, outdeg. of G are both ≤ 2 (Lemma 9)	G 3- resp. 4-nice (Thm. 2)
$C_{\text{fac}}(G; x, 1)$ at $x \in \mathbb{Q}$		G planar / 3-nice (Thm. 1, 2)
$C_{\text{fac}}(G; k, 1)$ at $k \in \mathbb{N}$	G is split-planar (Lemma 10)	G planar (Thm. 1)
$C_{\text{geo}}(G; -3, 1)$	G is split-pl. with indeg. ≤ 2 and outdeg. $= 3$ (Lemma 11)	G 4-nice (Thm. 2)

References

1. Averbouch, I., Godlin, B., Makowsky, J.A.: A most general edge elimination polynomial. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 31–42. Springer, Heidelberg (2008)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Computing the Tutte polynomial in vertex-exponential time. In: FOCS 2008, pp. 677–686 (2008)
3. Bläser, M., Dell, H.: Complexity of the cover polynomial. In: ICALP, pp. 801–812 (2007)
4. Bläser, M., Dell, H., Fouz, M.: Complexity and approximability of the cover polynomial. Computation Complexity (accepted)
5. Bläser, M., Dell, H., Makowsky, J.A.: Complexity of the bollobás-riordan polynomial. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 86–98. Springer, Heidelberg (2008)
6. Bläser, M., Hoffmann, C.: On the complexity of the interlace polynomial. In: Albers, S., Weil, P. (eds.) 25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008), Dagstuhl, Germany, pp. 97–108 (2008)
7. Bollobás, B., Riordan, O.: A Tutte polynomial for coloured graphs. Combinatorics, Probability and Computing 8(1-2), 45–93 (1999)
8. Cai, J.-y., Lu, P., Xia, M.: Holographic algorithms with matchgates capture precisely tractable planar #CSP. CoRR abs/1008.0683 (2010)
9. Chung, F.R.K., Graham, R.L.: On the cover polynomial of a digraph. J. Combin. Theory Ser. B 65, 273–290 (1995)
10. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics 108(1-2), 23–52 (2001)
11. D'Antona, O.M., Munarini, E.: The cycle-path indicator polynomial of a digraph. Advances in Applied Mathematics 25(1), 41–56 (2000)
12. Goldberg, L.A., Jerrum, M.: Inapproximability of the Tutte polynomial. Inform. Comput. 206(7), 908–929 (2008)
13. Goldberg, L.A., Jerrum, M.: Inapproximability of the Tutte polynomial of a planar graph. CoRR abs/0907.1724 (2009)
14. Hoffmann, C.: A most general edge elimination polynomial - thickening of edges. Fundam. Inform. 98(4), 373–378 (2010)
15. Jaeger, F., Vertigan, D.L., Welsh, D.J.A.: On the computational complexity of the Jones and Tutte polynomials. Math. Proc. of the Cambridge Phil. Society 108(1), 35–53 (1990)
16. Linial, N.: Hard enumeration problems in geometry and combinatorics. SIAM J. Algebraic Discrete Methods 7(2), 331–335 (1986)
17. Lotz, M., Makowsky, J.A.: On the algebraic complexity of some families of coloured Tutte polynomials. Advances in Applied Mathematics 32(1), 327–349 (2004)
18. Makowsky, J.A.: From a zoo to a zoology: Towards a general theory of graph polynomials. Theory of Computing Systems 43(3), 542–562 (2008)
19. Valiant, L.G.: Holographic algorithms (extended abstract). In: FOCS 2004: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 306–315. IEEE Computer Society, Washington, DC (2004)
20. Vertigan, D.L.: The computational complexity of Tutte invariants for planar graphs. SIAM J. Comput. 35(3), 690–712 (2005)
21. Xia, M., Zhang, P., Zhao, W.: Computational complexity of counting problems on 3-regular planar graphs. Theoretical Computer Science 384(1), 111–125 (2007)

Model Checking Coverability Graphs of Vector Addition Systems

Michel Blockelet and Sylvain Schmitz

LSV, ENS Cachan & CNRS, Cachan, France
mblockel@dptinfo.ens-cachan.fr, schmitz@lsv.ens-cachan.fr

Abstract. A large number of properties of a vector addition system—for instance coverability, boundedness, or regularity—can be decided using its coverability graph, by looking for some characteristic pattern. We propose to unify the known exponential-space upper bounds on the complexity of such problems on vector addition systems, by seeing them as instances of the model-checking problem for a suitable extension of computation tree logic, which allows to check for the existence of these patterns. This provides new insights into what constitutes a "coverability-like" property.

Keywords: Vector Addition Systems, CTL, Coverability Properties, Complexity.

1 Introduction

Vector addition systems (or equivalently Petri nets) are widely employed to reason about concurrent computations. Many decidable problems for vector addition systems are known to be ExpSpace-hard thanks to a proof originally due to Lipton [3]. Regarding complexity upper bounds, a key distinction arises between "reachability-like" problems on the one hand, for which no upper-bound is currently known in spite of continuous research on the subject [17, 15, 16], and "coverability-like" problems on the other hand, for which ExpSpace upper bounds have been derived after the work of Rackoff [18]. The latter class of problems is known to encompass many questions for the analysis of vector addition systems (prominently linear-time model-checking [10]), and related models of concurrency [e.g. 9, 13].

We promote in this paper a characterization of "coverability-like" properties as relying on the existence of some witness pattern in the *coverability graph* [14, 11] of the system—this graph provides a finite abstraction of the system's possible behaviours. This stance is backed up by several results [see e.g. 20, 8, 4] that rely on the same powerful technique: since the coverability graph is finite, the existence of a witness can be checked, yielding the decidability of the property at hand. As the coverability graph might have non primitive-recursive size [3], this technique comes however at a very high price—at least at first sight.

We show in this paper that a fragment of *existential computation tree logic* (ECTL) extended with Presburger constraints on paths enjoys a small model

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 108–119, 2011.

property when checked against runs in coverability graphs, and deduce an ExpSpace complexity upper bound for properties expressed in this fragment. These properties encompass many examples of properties testable in exponential space we found in the literature. We further believe the resulting formulæ to be quite natural and intuitive (they can express branching properties and the existence of ω-markings *directly*), and illustrate this point with several examples.

On the technical side, the proof of this small model property is in the line of similar results shown by Rackoff [18] for the coverability and the boundedness problems, and extended by Yen [22], Atig and Habermehl [1], Demri [5] to more complex properties. These extensions rely on rather terse, ad-hoc logical formalisms, which are checked against the actual runs of the system—it is tempting to blame the complexity of Yen's logical formalism for the issue found in his proof by Atig and Habermehl. Thus a major contribution of the paper is the key insight that what should be checked are runs in coverability graphs instead of actual runs, and that a reasonably standard logic based on CTL is perfectly usable to this end. In more details, we define a notion of VAS coverability graphs that will constitute the models of our logic (Sec. 2) and investigate their simulation relations; we define an extension of CTL using Presburger constraints on paths and atomic propositions testing for coverability (Sec. 3.1) before considering the decidability of VAS model-checking for some of its fragments (Sec. 3.2); we then consider a restricted fragment of *eventually increasing* formulæ and prove its VAS model-checking problem to be ExpSpace-complete (Sec. 4).

Notations. Let $\mathbb{Z}_\omega = \mathbb{Z} \uplus \{\omega\}$ be the set of integers completed with a limit element ω, which is larger than any finite z in $\mathbb{Z}$ and verifies $\omega + d = \omega$ for all d in $\mathbb{Z}$. Whenever working on vectors in $\mathbb{Z}_\omega^k$ for some k, we implicitly employ component-wise orderings. We consider throughout the paper rooted *labeled transition systems* (LTS) $\mathcal{S} = \langle S, \to, \ell, s_{\text{init}} \rangle$ where, for some $k \geq 1$, S is a set of states, ℓ is a state labeling function from S to $\mathbb{Z}_\omega^k$, s_{init} is the initial state in S, and $\to$ is a labeled transition relation included in $S \times \mathbb{Z}^k \times S$. In our developments we ignore labels and define the *size* $|\mathcal{S}|$ of a LTS $\mathcal{S}$ as the cardinality of $\to$, and for a set of vectors $\mathsf{V} \subseteq \mathbb{Z}_\omega^k$, $\|\mathsf{V}\| = \max_{\mathsf{v} \in \mathsf{V}, 1 \leq j \leq k, \mathsf{v}(j) < \omega} (0, \lceil \log_2(|\mathsf{v}(j)|) + 1 \rceil)$. An LTS is called *tree-shaped* if any state has at most one predecessor by $\to$, i.e. for all s, $|\{s' \in S \mid s' \to s\}| \leq 1$, and *path-shaped* if furthermore it has at most one successor by $\to$, i.e. for all s, $|\{s' \in S \mid s \to s'\}| \leq 1$. The omitted proofs can be found in the long version of this paper at `http://hal.archives-ouvertes.fr/hal-00600077/`.

2 Coverability Graphs

Let us first recall the definition of coverability graphs for vector addition systems and how they can be used to decide various properties.

Vector Addition Systems. A *k-dimensional vector addition system* (k-VAS) is a pair $\mathcal{S} = \langle \mathsf{V}, \mathsf{x}_0 \rangle$ where V is a finite set of transitions in $\mathbb{Z}^k$ and x_0 an initial marking in $\mathbb{N}^k$ [14]. Formally, we can define the *reachability graph* of such a

k-VAS as the (generally infinite) LTS $R(\mathcal{S}) = \langle \mathbb{N}^k, \to, id, \mathsf{x}_0 \rangle$ with states (also called *markings*) in $\mathbb{N}^k$, the identity id as state labeling function, and transitions labels in V s.t. $\mathsf{x} \xrightarrow{\mathsf{a}} \mathsf{x}'$ iff $\mathsf{x} + \mathsf{a} = \mathsf{x}'$ (note that it implies $\mathsf{x} + \mathsf{a} \geq \mathsf{0}$). In some proofs, we will consider *generalized VAS*, where x_0 can be taken from $\mathbb{Z}^k_\omega$. We consider several parameters for VAS size, as do Rosier and Yen [19]: the size of the binary encoding of the largest difference a vector from the transitions set can induce $\|\mathsf{V}\|$, the cardinal of the transition relation $|\mathsf{V}|$, and the dimension k.

Canonical Coverability Graph. Coverability graphs are finite abstractions of VAS reachability graphs. In order to remain finite, they employ markings over the complete space $(\mathbb{N} \uplus \{\omega\})^k$, noted $\mathbb{N}^k_\omega$. There are several possible definitions for coverability graphs, all based on the original Karp and Miller coverability tree construction [14]; here is one particular flavour, as found for instance in [20].

Given a LTS $\langle S, \to, \ell, s_{\text{init}} \rangle$ and given some $1 \leq j \leq k$, let us first define a *j-antecedent* of a pair (s, a) in $S \times \mathbb{Z}^k$ as a state s' satisfying

$$s_{\text{init}} \to^* s' \xrightarrow{w} s \wedge \ell(s') \leq \ell(s) + \mathsf{a} \wedge \ell(s')(j) < (\ell(s) + \mathsf{a})(j) \tag{1}$$

for some w in $(\mathbb{Z}^k)^*$. A j-antecedent witnesses the fact that, by repeating the sequence of transitions $w\mathsf{a}$ from s', we can obtain arbitrarily high values in coordinate j—which will be represented symbolically by an ω value in the coverability graph.

The *coverability tree* of a k-VAS $\mathcal{S} = \langle \mathsf{V}, \mathsf{x}_0 \rangle$ is a tree-shaped LTS $T(\mathcal{S}) = \langle S, \to, \ell, s_{\text{init}} \rangle$ with state labels in $\mathbb{N}^k_\omega$ and transition labels in V constructed by:

basis initially $S = \{s_{\text{init}}\}$ with label $\ell(s_{\text{init}}) = \mathsf{x}_0$ and s_{init} is flagged as unprocessed,

step for every unprocessed state s and every a in V
- if $\ell(s) + \mathsf{a} \not\geq \mathsf{0}$: do nothing, as a is not firable in $\ell(s)$,
- otherwise, let s' be a fresh state, update S to be $S \uplus \{s'\}$, add a transition $s \xrightarrow{\mathsf{a}} s'$, and set the label of s' in $\mathbb{N}^k_\omega$ to

$$\ell(s')(j) \stackrel{\text{def}}{=} \begin{cases} \omega & \text{if } \exists s'' \text{ a } j\text{-antecedent of } (s, \mathsf{a}) \\ (\ell(s) + \mathsf{a})(j) & \text{otherwise} \end{cases} \tag{2}$$

  If there does not exist any state s'' in S with $\ell(s') = \ell(s'')$ and $s_{\text{init}} \to^* s'' \to^* s$, flag s' as unprocessed; s' is otherwise a leaf of the tree.

The *canonical coverability graph* $C(\mathcal{S})$ of a k-VAS $\mathcal{S} = \langle \mathsf{V}, \mathsf{x}_0 \rangle$ is obtained by identifying identically-labeled states in $T(\mathcal{S})$, i.e. it is the quotient $C(\mathcal{S}) = T(\mathcal{S})/\equiv$ for the equivalence relation $s \equiv s'$ iff $\ell(s) = \ell(s')$ (see e.g. Fig. 1).

Examples of Coverability Properties. Coverability graphs allow to decide many properties on a k-VAS $\mathcal{S}$; for instance,

coverability given a marking x in $\mathbb{N}^k$, whether a marking $\mathsf{x}' \geq \mathsf{x}$ is reachable in $R(\mathcal{S})$—simply check whether a state s with $\ell(s) \geq \mathsf{x}$ is reachable in $C(\mathcal{S})$; for instance in Fig. 1 we see that $\langle 1, 5, 1 \rangle$ is coverable but $\langle 2, 1, 2 \rangle$ is not—,

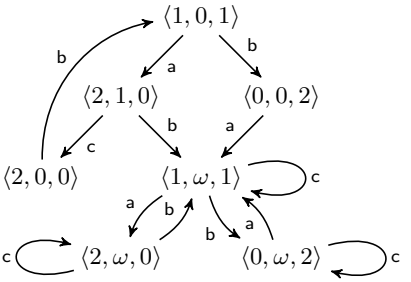


Fig. 1. The canonical coverability graph for the VAS $\mathcal{S} = \langle\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \langle 1, 0, 1\rangle\rangle$ with transitions $\mathsf{a} = \langle 1, 1, -1\rangle$, $\mathsf{b} = \langle -1, 0, 1\rangle$, and $\mathsf{c} = \langle 0, -1, 0\rangle$

boundedness whether the set of reachable markings in $R(\mathcal{S})$ is finite—this occurs iff no ω value appears in the label of any state of $C(\mathcal{S})$; for instance in Fig. 1 the VAS is not bounded—,

place boundedness given a coordinate $1 \leq j \leq k$, whether the set of values $\mathsf{x}(j)$ for all reachable x in $R(\mathcal{S})$ is finite—this occurs iff no ω value appears as $\ell(s)(j)$ for some state s of $C(\mathcal{S})$; for instance in Fig. 1, the second coordinate is unbounded but the other two are bounded,

language regularity whether the *language*, i.e. the set of labels w in V^* of transition sequences $s_0 \xrightarrow{w} s$ in $R(\mathcal{S})$, is regular—this occurs if no state s with a cycle $s \xrightarrow{\mathsf{a}_1 \cdots \mathsf{a}_n} s$ appears in $C(\mathcal{S})$ s.t. there exists $1 \leq j \leq k$, $\ell(s)(j) = \omega$ and $(\sum_{i=1}^{n} \mathsf{a}_i)(j) < 0$ [20, Thm. 5]; for instance in Fig. 1 we find one such cycle $\langle 1, \omega, 1\rangle \xrightarrow{\mathsf{c}} \langle 1, \omega, 1\rangle$, and indeed the language of this VAS yields $(\mathsf{ab})^n \mathsf{c}^{\leq n}$ when intersected with $(\mathsf{ab})^*\mathsf{c}^*$, and is therefore non-regular.

All these properties are decidable in exponential space; see [18] for the first two, and [5] for the last two. Observe that we were able to characterize each property by the existence of some witness in the canonical coverability graph; we shall see in Sec. 3 that we can easily express those properties in a modal logic, and later in Sec. 4 that the exponential space upper bound applies to all properties expressed in a fragment of this logic.

Partial Covers. In preparation of the technical developments of the following sections, we define structures related to the coverability graph that will serve as witnesses. The motivation is that later we will build small models for properties by induction on the dimension, thus it will be convenient to consider *partial* coverability graphs, which are "correct" only on the first j coordinates out of k.

Definition 2.1. *A* partial cover *for a generalized k-VAS $\langle \mathsf{V}, \mathsf{x}_0\rangle$ is an accessible LTS $\langle S, \to, \ell, s_{init}\rangle$ with transition labels in V s.t. $\ell(s_{init}) = \mathsf{x}_0$ and, if $s \xrightarrow{\mathsf{a}} s'$, then for all $1 \leq j \leq k$, either $\ell(s)(j) + \mathsf{a}(j) = \ell(s')(j)$, or $\ell(s)(j) < \omega$, $\ell(s')(j) = \omega$, and on every path $s_{init} = s_0 \xrightarrow{\mathsf{a}_1} \cdots \xrightarrow{\mathsf{a}_n} s_n = s$, there exists $0 \leq i \leq n$ s.t. s_i is a j-antecedent of (s, a) (see (1)).*

Thus a partial cover does not enforce positive values on the state labels, but guarantees transition labels to be compatible with state labels, and ω values to be introduced only when legal, i.e. when at least one j-antecedent exists on every path from the initial state. Partial covers can also be seen as LTS with j-antecedency relations—in addition to the transition relation—from a state with a newly introduced ω value to each of its j-antecedents. When constructing partial covers we will need to preserve the existence of at least one such j-antecedent.

With the example of Fig. 1, the system reduced to the initial marking

$$\langle 1, 0, 1\rangle \tag{3}$$

is a partial cover, the following are two more (path-shaped) partial covers of the same VAS:

$$\langle 1, 0, 1\rangle \xrightarrow{\mathsf{a}} \langle 2, 1, 0\rangle \xrightarrow{\mathsf{a}} \langle 3, 2, -1\rangle \tag{4}$$

$$\langle 1, 0, 1\rangle \xrightarrow{\mathsf{a}} \langle 2, 1, 0\rangle \xrightarrow{\mathsf{b}} \langle 1, \omega, 1\rangle \xrightarrow{\mathsf{c}} \langle 1, \omega, 1\rangle\ , \tag{5}$$

and this last one is not a partial cover, as we cannot introduce an ω value at this point:

$$\langle 1, 0, 1\rangle \xrightarrow{\mathsf{a}} \langle 2, \omega, 0\rangle\ . \tag{6}$$

Definition 2.2. *Let $0 \leq i \leq k$. A partial cover $\mathcal{C} = \langle S, \to, \ell, s_{init}\rangle$ is* i-admissible *if for all $1 \leq j \leq i$ and for all s in S, $0 \leq \ell(s)(j)$.*

Note that in particular the initial marking x_0 of a generalized k-VAS also needs to satisfy $\mathsf{x}_0(j) \geq 0$ for $1 \leq j \leq i$ in order for a i-admissible partial cover to even exist. Both the canonical coverability graph and the reachability graph of a k-VAS are k-admissible partial covers. Among the previous examples, (3) and (5) are 3-admissible, (4) is 2-admissible but not 3-admissible. Considering our examples of coverability properties, the LTS in (5) could be used as a witness of coverability of $\langle 1, 5, 1\rangle$, unboundedness in the second coordinate, and non regularity of the language.

Covering Simulations. Among all the k-admissible partial covers of a k-VAS $\mathcal{S}$ (Defs. 2.1 and 2.2), we find in particular its canonical coverability graph $C(\mathcal{S})$. All these k-admissible partial covers are in fact related to $C(\mathcal{S})$:

Definition 2.3. *Let $k \geq 1$ and $\mathcal{S}_1 = \langle S_1, \to_1, \ell_1, s_1\rangle$ and $\mathcal{S}_2 = \langle S_2, \to_2, \ell_2, s_2\rangle$ be two LTS. A* covering simulation *between $\mathcal{S}_1$ and $\mathcal{S}_2$ is a relation $R \subseteq S_1 \times S_2$ s.t.*

1. *$s_1\ R\ s_2$,*
2. *if $s\ R\ s'$, then*
   (a) *$\ell_1(s) \leq \ell_2(s')$ and*
   (b) *if $s \xrightarrow{\mathsf{a}}_1 q$ for some a in $\mathbb{Z}^k$ and q in S_1 then there exists q' in S_2 with $s' \xrightarrow{\mathsf{a}}_2 q'$ and $q\ R\ q'$.*

We say that $\mathcal{S}_2$ simulates *$\mathcal{S}_1$, noted $\mathcal{S}_1 \preccurlyeq \mathcal{S}_2$, if there exists a covering simulation R between $\mathcal{S}_1$ and $\mathcal{S}_2$.*

Lemma 2.4. *Let $\mathcal{S} = \langle \mathsf{V}, \mathsf{x}_0\rangle$ be a k-VAS and $\mathcal{C}$ a k-admissible partial cover of $\mathcal{S}$. Then $\mathcal{C} \preccurlyeq C(\mathcal{S})$.*

3 CTL Logics for Coverability Graphs

We first propose a very general logic based on CTL for model-checking coverability graphs (Sec. 3.1). Our purpose with this general logic is to motivate our choice of CTL fragment for the following sections: indeed, the full logic will turn out to be too powerful, and we will restrict ourselves to an existential fragment with a decidable model checking problem (Sec. 3.2).

3.1 An Extension of CTL

We define an extension $\mathrm{PrCTL}_{\geq}(\mathsf{U})$ of CTL specifically designed to express properties of VAS coverability graphs. It features

coverability constraints $\mu(j) \geq c$, where c is a constant in $\mathbb{N}_\omega$, as atomic formulæ, allowing to express that the label in the current state has value greater or equal to c in its jth coordinate. These extend the usual coverability constraints [see e.g. 6] by also allowing to express $\mu(j) = \omega$.

Presburger-refined temporal modalities U_ψ using Presburger formulæ ψ to constrain the allowed paths—this is similar to the regular modalities found for instance in [2], but what is constrained here is the *effect* of a transition sequence rather than its label.

Presburger Formulæ. We restrict our attention to quantifier-free Presburger (QFP) formulæ, since one such formula can be obtained from any Presburger formula at the expense of a worst-case triple exponential blowup [see e.g. 21, Thm. 2.1]. More precisely, given an infinite countable set of variables $\mathcal{X}$, a QFP formula ψ is defined through

$$\psi ::= \top \mid \neg\psi \mid \psi \vee \psi \mid \alpha, \quad \alpha ::= \tau \geq \tau \mid \tau \equiv_p \tau, \quad \tau ::= 0 \mid 1 \mid x \mid \tau + \tau \tag{7}$$

where x is a variable from $\mathcal{X}$ and $p \geq 2$.[1] Given a vector x of values in $\mathbb{Z}^k$ and a formula ψ with k free variables $x_1, \ldots, x_k$, we write $\psi(\mathsf{x})$ for the closed formula with $\mathsf{x}(j)$ substituted for x_j for each $1 \leq j \leq k$. Given a closed Presburger formula ψ, we write $\mathrm{PA} \models \psi$ if the formula is valid.

Syntax of PrCTL$_{\geq}$(U). Formally, fix some k in $\mathbb{N}$; a k-formula of $\mathrm{PrCTL}_{\geq}(\mathsf{U})$ is a term φ defined by the abstract syntax

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}(\varphi\ \mathsf{U}_\psi\ \varphi) \mid \mu(j) \geq c$$

where ψ denotes a QFP formula with k free variables, $1 \leq j \leq k$, and c is a constant in $\mathbb{N}_\omega$. Note that a k-formula is also a k'-formula for all $k' \geq k$. We can simulate the classical "next" modalities X (see the proof of Thm. 3.1). The classical, unrefined U modality can be defined by $\mathsf{E}(\varphi\ \mathsf{U}\ \varphi') \stackrel{\text{def}}{=} \mathsf{E}(\varphi\ \mathsf{U}_\top\ \varphi')$ using the $\top$ formula of QFP. We also define as usual $\mathsf{EF}_\psi\varphi \stackrel{\text{def}}{=} \mathsf{E}(\top\ \mathsf{U}_\psi\ \varphi)$, and the dualities $\bot \stackrel{\text{def}}{=} \neg\top$, $\varphi \wedge \varphi' \stackrel{\text{def}}{=} \neg((\neg\varphi) \vee (\neg\varphi'))$, $(\mu(j) < c) \stackrel{\text{def}}{=} \neg(\mu(j) \geq c)$, and $\mathsf{AG}_\psi\varphi \stackrel{\text{def}}{=} \neg\mathsf{EF}_\psi\neg\varphi$.

[1] We include the *divisibility* relations, which are required for quantifier elimination, with semantics $x \equiv_p y$ iff $\exists z. x + pz = y$ for all x, y in $\mathbb{Z}$ and $p \geq 2$.

Semantics of PrCTL$_{\geq}$(U). The models of PrCTL$_{\geq}$(U) formulæ are labeled transition systems $\langle S, \to, \ell, s_{\text{init}}\rangle$. Given a state s in S, write $\mathit{Paths}(s)$ for the set of *maximal paths* $\pi = s_0 \xrightarrow{\mathsf{a}_1} s_1 \xrightarrow{\mathsf{a}_2} \cdots$ starting in $s = s_0$ and where each a_i is in $\mathbb{Z}^k$ and each s_i in S. The path is either infinite with length $|\pi| = \omega$, or finite of form $\pi = s_0 \xrightarrow{\mathsf{a}_1} \cdots \xrightarrow{\mathsf{a}_n} s_n$ if s_n has no successor and then $|\pi| = n$. If $\mathsf{a}_1 \cdots \mathsf{a}_n$ is a sequence in $(\mathbb{Z}^k)^*$ (with $n = 0$ for the empty sequence), then its *effect* is $\Delta\mathsf{a}_1 \cdots \mathsf{a}_n = \sum_{i=1}^{n} \mathsf{a}_i$ in $\mathbb{Z}^k$.

A state s in S *satisfies* a PrCTL$_{\geq}$(U) formula φ, written $s \models \varphi$, in the following inductive cases:

$$
\begin{array}{ll}
s \models \top & \text{always,} \\
s \models \neg\varphi & \text{iff } s \not\models \varphi\ , \\
s \models \varphi_1 \vee \varphi_2 & \text{iff } s \models \varphi_1 \text{ or } s \models \varphi_2\ , \\
s \models \mathsf{E}(\varphi\ \mathsf{U}_{\psi}\ \varphi') & \text{iff } \exists \pi = s_0 \xrightarrow{\mathsf{a}_1} s_1 \xrightarrow{\mathsf{a}_2} \cdots \in \mathit{Paths}(s),\ \exists n \leq |\pi|, \\
 & \quad \text{PA} \models \psi(\Delta\mathsf{a}_1 \cdots \mathsf{a}_n),\ s_n \models \varphi',\ \text{and } \forall m < n,\ s_m \models \varphi, \\
s \models \mu(j) \geq c & \text{iff } \ell(s)(j) \geq c\ .
\end{array}
$$

As usual, a LTS $\mathcal{S}$ satisfies φ, written $\mathcal{S} \models \varphi$, if $s_{\text{init}} \models \varphi$. A k-VAS $\langle \mathsf{V}, \mathsf{x}_0\rangle$ *satisfies* a PrCTL$_{\geq}$(U) k-formula φ, written $\langle \mathsf{V}, \mathsf{x}_0\rangle \models \varphi$, if there exists a k-admissible partial cover $\mathcal{C}$ of $\langle V, \mathsf{x}_0\rangle$ such that $\mathcal{C} \models \varphi$. We will see later (Thm. 3.2) that for *existential* PrCTL$_{\geq}$(U) this boils down to model-checking the canonical coverability graph.

Examples of Formulæ. Consider once more the coverability properties of Sec. 2: the coverability problem for a marking x can be checked by model-checking the following formula against $\mathcal{C}(\mathcal{S})$:

$$\varphi_{\text{cov},\mathsf{x}} \stackrel{\text{def}}{=} \mathsf{EF} \bigwedge_{j=1}^{k} \mu(j) \geq \mathsf{x}(j)\ ; \tag{8}$$

unboundedness by

$$\varphi_{\text{unb}} \stackrel{\text{def}}{=} \mathsf{EF} \bigvee_{j=1}^{k} \mu(j) \geq \omega\ ; \tag{9}$$

place unboundedness in coordinate $1 \leq j \leq k$ by

$$\varphi_{\text{unb},j} \stackrel{\text{def}}{=} \mathsf{EF}\mu(j) \geq \omega\ ; \tag{10}$$

and non-regularity of the language by

$$\varphi_{\text{unreg}} \stackrel{\text{def}}{=} \mathsf{EF} \bigvee_{\substack{I \subseteq \{1,\ldots,k\} \\ I \neq \emptyset}} \bigvee_{I \subseteq J \subseteq \{1,\ldots,k\}} \left(\bigwedge_{j \in J} \mu(j) \geq \omega \wedge \mathsf{EF}_{\psi_{I,J}} \top \right) \tag{11}$$

where

$$\psi_{I,J}(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \bigwedge_{j \in I} x_j < 0 \wedge \bigwedge_{j \notin J} x_j \geq 0\ . \tag{12}$$

We can check that the 3-admissible partial cover (5) satisfies all these formulæ (setting $\mathsf{x} = \langle 1, 5, 1\rangle$ for (8)), thus our example VAS satisfies all these formulæ.

3.2 VAS Model Checking

We turn now to the *VAS model checking problem*: for a VAS $\mathcal{S} = \langle \mathsf{V}, \mathsf{x}_0\rangle$ and a $\mathrm{PrCTL}_{\geq}$ formula φ, does $\langle \mathsf{V}, \mathsf{x}_0\rangle$ satisfy φ?

Undecidability of PrCTL$_{\geq}$(U). When considering how general $\mathrm{PrCTL}_{\geq}$ is, its model-checking problem is rather unsurprisingly undecidable, even if restricted to EF modalities, i.e. for the $\mathrm{PrCTL}_{\geq}(\mathsf{F})$ fragment (the proof uses results by Esparza [6]):

Proposition 3.1. *The VAS model-checking problem for PrCTL$_{\geq}$(F) is undecidable.*

Decidability of PrECTL$_{\geq}$(U). The formulæ used in the proof of Thm. 3.1 employ alternation in a crucial way in order to encode the VAS containment problem, and a natural question is whether the *existential* fragment $\mathrm{PrECTL}_{\geq}(\mathsf{U})$, with syntax

$$\varphi ::= \top \mid \bot \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{E}(\varphi\ \mathsf{U}_{\psi}\ \varphi) \mid \mu(j) \geq c\ ,$$

is decidable. This is the case: it suffices to check whether the canonical coverability graph satisfies the formula, since by Thm. 2.4 it *simulates* any other k-admissible partial cover. This is one of the benefits of considering CTL fragments rather than ad-hoc logics: the standard toolkit of modal logic is readily applicable, like the connection between simulations and existential CTL:

Proposition 3.2. *The VAS model-checking problem for PrECTL$_{\geq}$(U) is decidable in nondeterministic polynomial time in $|C(\mathcal{S})|$ and $|\varphi|$.*

The decidability of VAS model-checking for $\mathrm{PrECTL}_{\geq}(\mathsf{U})$ is encouraging, but our decision procedure relies on the construction of the canonical coverability graph $C(\mathcal{S})$. As the latter can have non primitive-recursive size [3, who attribute the idea to Hack], this is not a very efficient algorithm: it yields an Ackermannian upper bound [7, Sec. VII.C] on the complexity of VAS model-checking for $\mathrm{PrECTL}_{\geq}(\mathsf{U})$. This is a ridiculously high upper bound, but we actually suspect the VAS model-checking problem for $\mathrm{PrECTL}_{\geq}(\mathsf{U})$ to be Ackermann-complete. On the one hand, Thm. 3.2 implies the $\mathrm{PrECTL}_{\geq}(\mathsf{U})$ problem to be in NPTime for fixed VAS; on the other hand, small extensions within existential fragments quickly lead to undecidability (e.g. when allowing $\mu(j) < c$ or G_{ψ}).

The remainder of the paper is dedicated to a fragment of $\mathrm{PrECTL}_{\geq}(\mathsf{F})$ for which we demonstrate a small model property and deduce decision procedures working in exponential space. Although we use techniques adapted from Rackoff and his successors, several points make these contributions stand out: the *simplicity* of the logic, its ability to express *branching* properties directly, and its intuitive semantics in terms of *coverability graphs*.

4 Eventually Increasing Formulæ

Let us consider the PrECTL$_{\geq}(\mathsf{F})$ fragment. We are going to introduce a semantic restriction to PrECTL$_{\geq}(\mathsf{F})$ formulæ, inspired by a similar restriction employed by Atig and Habermehl [1] to fix Yen [22]'s proof.

4.1 The PrECTL$_{\geq}(\mathsf{F})$ Fragment

Eventually Increasing Formulæ. We can restrict ourselves to finite tree-shaped models for PrECTL$_{\geq}(\mathsf{F})$ formulæ; such a model $\mathcal{C}$ has a root s and a number of leaves $s_1, \ldots, s_n$, each satisfying some *coverability constraint* (CC) subformula γ, of form

$$\gamma ::= \top \mid \bot \mid \gamma \wedge \gamma \mid \gamma \vee \gamma \mid \mu(j) \geq c \,, \tag{13}$$

where $1 \leq j \leq k$ and c is in $\mathbb{N}_\omega$. We call this model *increasing* if $\ell(s_i) \geq \ell(s)$ for all $1 \leq i \leq n$. A formula φ of PrECTL$_{\geq}(\mathsf{F})$ is *increasing* if all its tree-shaped models are increasing. An *eventually increasing formula* is a formula of form $\mathsf{EF}\varphi$ for some increasing formula φ. We denote the set of (eventually) increasing PrECTL$_{\geq}(\mathsf{F})$ formulæ by (e)iPrECTL$_{\geq}(\mathsf{F})$. All our example formulæ (8)–(11) are eventually increasing.

Such a semantic restriction naturally leads to the question: is it decidable whether a formula fits into the fragment? We first consider the related problem of PrECTL$_{\geq}(\mathsf{F})$ satisfiability: given a k-formula φ, does there exist a k-VAS $\langle \mathsf{V}, \mathsf{x}_0 \rangle$ s.t. $\langle \mathsf{V}, \mathsf{x}_0 \rangle \models \varphi$? It turns out that this satisfiability problem reduces to the satisfiability of its QFP subformulæ, which can be checked in NPTime:

Proposition 4.1. *The satisfiability problem for PrECTL$_{\geq}(\mathsf{F})$ is decidable in* NPTime.

Checking whether a formula is increasing is a bit more involved: we need to check whether the various QFP subformulæ ensure every possible model is increasing, which we do by constructing a (universal) Presburger formula:

Proposition 4.2. *Let φ be a k-formula of PrECTL$_{\geq}(\mathsf{F})$. Whether φ is a k-formula of iPrECTL$_{\geq}(\mathsf{F})$ is decidable in* NPTime.

4.2 Small Model Properties

The proof of the small model property for eiPrECTL$_{\geq}(\mathsf{F})$ formulæ follows the general design of Rackoff's proof: first a small model property on models with *bounded* values (Thm. 4.3) using results on the existence of small solutions for linear integer programming, and then a proof of existence of a small model in general by induction on the dimension (Thm. 4.4).

Bounded Models. Define as usual with Rackoff's approach an (i, r)-*bounded* LTS as an i-admissible one where no finite value on the first i coordinates is larger than r: for all s and every $1 \leq j \leq i$, $\ell(s)(j) \geq r$ implies $\ell(s)(j) = \omega$. If $i \leq k$, the *i-projection* of a formula φ is a formula $\varphi_{|_i}$ where every $\mu(j) \geq c$ term of φ with $j > i$ and $c < \omega$ has been replaced by $\top$. Adapting the proof of [18, Lem. 4.5] to our case, we obtain:

Lemma 4.3 (Small Models for Bounded LTS). *Let $\langle \mathsf{V}, \mathsf{x}_0 \rangle$ be a generalized k-VAS with $k > 1$, φ be a PrECTL$_{\geq}(\mathsf{F})$ formula, and $0 \leq i \leq k$ and $r \geq 0$. If there exists an (i,r)-bounded partial cover $\mathcal{C}$ of $\langle \mathsf{V}, \mathsf{x}_0 \rangle$ s.t. $\mathcal{C} \models \varphi_{|_i}$, then there exists a tree-shaped (i,r)-bounded partial cover $\mathcal{C}'$ of $\langle \mathsf{V}, \mathsf{x}_0 \rangle$ with $\mathcal{C}' \models \varphi_{|_i}$ and $|\mathcal{C}'| \leq (2^{\|\mathsf{V}\|} r)^{(k+|\varphi|)^d}$ for some constant d (independent of V, x_0, k, φ, i, and r).*

General Models. We prove now a general small model property for eiPrECTL$_{\geq}(\mathsf{F})$ formulæ. It borrows several elements from earlier research, prominently [18, Lem. 4.6 & 4.7], but also crucially the use of an increasing condition to allow "replaying" a model at a leaf [1]. Given $\mathsf{V} \subseteq (\mathbb{Z}_\omega)^k$, a k-coverability formula φ, and some $0 \leq i < k$, let

$$g(0) \stackrel{\text{def}}{=} (2^{\|\mathsf{V}\|} \cdot |\mathsf{V}|)^{(k+|\varphi|)^d}$$

$$g(i+1) \stackrel{\text{def}}{=} \left(2^{\|\mathsf{V}\|} \cdot (2^{\|\mathsf{V}\|} g(i) + |\varphi|)\right)^{(k+|\varphi|)^d} + 1 + g(i)$$

where d is the constant of Thm. 4.3. We finally obtain our small model property:

Lemma 4.4 (Small Model Property). *Let $\langle \mathsf{V}, \mathsf{x}_0 \rangle$ be a generalized k-VAS and $\varphi = \mathsf{EF}\varphi'$ be a k-eventually increasing formula. Let $\varphi_{|_i}$ be satisfiable. Then there exists a tree-shaped i-admissible partial cover of $\langle \mathsf{V}, \mathsf{x}_0 \rangle$ that models $\varphi_{|_i}$ and of size $\leq g(i)$.*

Lemma 4.4 results in a doubly exponential bound on the size of a k-admissible model for an eventually increasing formula, from which an ExpSpace algorithm can be designed, which is optimal considering the ExpSpace lower bound [3]:

Theorem 4.5 (Complexity of VAS model checking). *The VAS model-checking problem for eiPrECTL$_{\geq}(\mathsf{F})$ formulæ is* ExpSpace*-complete.*

Note that the different parameters on the size of the VAS and of the formula influence this complexity differently: for fixed k the obtained algorithm works in PSpace. A matching PSpace lower bound on the place coverability problem is given by Rosier and Yen [19], Coro. 3.1 for fixed $k \geq 4$.

Another interesting consequence of our bounds is that bounds for model checking vector addition systems *with states* (VASS) are easy to derive; for instance by encoding a k-VASS with state-space Q into a $(k + 2\lceil \log_2 |Q| \rceil)$-VAS: this is not as tight as the $(k+3)$-VAS encoding of Hopcroft and Pansiot [12], but allows to test in which control state we are in an eiPrECTL$_{\geq}(\mathsf{F})$ formula using coverability constraints $\mu(j) \geq 1$. Thus the number $|Q|$ of states only influences polynomially the complexity of VASS model checking for eiPrECTL$_{\geq}(\mathsf{F})$.

4.3 Related Work

The first attempt at unifying ExpSpace upper bounds on VAS problems was proposed by Yen [22]. This provided ExpSpace upper bounds for many problems

(boundedness, coverability, self-coverability, etc.; see Sec. 4 in [22]). For instance, we can consider the place boundedness problem: a path formula for it has to guess nondeterministically a sequence π of introductions of ω-values leading to the desired unboundedness of place j. Let Π be the set of repetition-free sequences π over $\{1,\dots,k\}\backslash\{j\}$ and $|\Pi| = n$; write $c(\pi)$ for the set of elements appearing in π and $\pi[i..k]$ for the factor of π between indices i and k (inclusive):

$$\exists \mu_1,\dots,\mu_{2n+2}, \exists \sigma_1,\dots,\sigma_{2n+2} (\mu_0 \xrightarrow{\sigma_1} \mu_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{2n+2}} \mu_{2n+2})$$
$$\wedge \bigvee_{\pi\in\Pi} \left(\mu_{2n+1}(j) < \mu_{2n+2}(j) \wedge \bigwedge_{i\notin c(\pi)} \mu_{2n+1}(i) \leq \mu_{2n+2}(i) \right)$$
$$\wedge \bigwedge_{m=1}^{|\pi|} \left(\mu_{2m-1}(\pi[m..m]) < \mu_{2m}(\pi[m..m]) \wedge \bigwedge_{i\notin c(\pi[1..m])} \mu_{2m-1}(i) < \mu_{2m}(i) \right).$$

The first main conjunct under the scope of the choice of π checks that an ω-value can appear in place j. The second main conjunct verifies the same for each element of π in sequence.

The proof of [22] was flawed, and corrected by Atig and Habermehl [1] who introduced the *increasing* restriction to Yen's logic to characterize formulæ for which the ExpSpace bound held. Nevertheless, this restriction meant that some of the bounds claimed by Yen did not hold any longer, for instance for the regularity problem, and the above formula for place unboundedness is another instance of a non-increasing formula. Demri [5] finally proposed to relax the class of models by considering "pseudo-runs" instead of actual runs, and provided a formal framework (*general unboundedness properties*) to express properties on such runs, allowing him to prove ExpSpace upper bounds for several open problems like place boundedness, regularity, strong promptness, etc. This is the most closely related approach.

We can express general unboundedness properties as $\mathrm{PrECTL}_{\geq}(\mathsf{F})$ formulæ, but not as eventually increasing ones, because these properties only enforce local increasing conditions instead of the global one we employed in this work. On the other hand many aspects of $\mathrm{eiPrECTL}_{\geq}(\mathsf{F})$ formulæ are beyond the reach of general unboundedness properties, since for instance we allow full Presburger arithmetic, and can nest EF_ψ modalities directly (general unboundedness properties would intersperse plain EF modalities between any two Presburger-refined modalities). This opens the question whether we could design a larger fragment of $\mathrm{PrECTL}_{\geq}(\mathsf{F})$ with an ExpSpace-easy VAS model-checking problem and capturing general unboundedness properties.

We believe $\mathrm{eiPrECTL}_{\geq}(\mathsf{F})$ formulæ to be much easier to write than general unboundedness properties; for instance for place unboundedness, one would also have to write explicitly all the different permutations on the order in which ω-values can be introduced in a general unboundedness property, instead of the straightforward formula (10).

References

1. Atig, M.F., Habermehl, P.: On Yen's path logic for Petri nets. Int. J. Fund. Comput. Sci. 22(4), 783–799 (2011)
2. Axelsson, R., Hague, M., Kreutzer, S., Lange, M., Latte, M.: Extended computation tree logic. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 67–81. Springer, Heidelberg (2010)
3. Cardoza, E., Lipton, R.J., Meyer, A.R.: Exponential space complete problems for Petri nets and commutative semigroups. In: STOC 1976, pp. 50–54. ACM Press, New York (1976)
4. Chambart, P., Finkel, A., Schmitz, S.: Forward analysis and model checking for trace bounded WSTS. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 49–68. Springer, Heidelberg (2011)
5. Demri, S.: On selective unboundedness of VASS. INFINITY 2010. Elec. Proc. In: Elec. Proc. in Theor. Comput. Sci., vol. 39, pp. 1–15 (2010)
6. Esparza, J.: Decidability of model checking for infinite-state concurrent systems. Acta Inf. 34(2), 85–107 (1997)
7. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson's Lemma. LICS 2011. IEEE, Los Alamitos (2011)
8. Finkel, A., Sangnier, A.: Reversal-bounded counter machines revisited. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 323–334. Springer, Heidelberg (2008)
9. Ganty, P., Majumdar, R., Rybalchenko, A.: Verifying liveness for asynchronous programs. In: POPL 2009, pp. 102–113. ACM Press, New York (2009)
10. Habermehl, P.: On the complexity of the linear-time μ-calculus for Petri nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 102–116. Springer, Heidelberg (1997)
11. Hack, M.: Decision problems for Petri nets and vector addition systems. Computation Structures Group Memo 95, Project MAC. MIT, Cambridge (1974)
12. Hopcroft, J., Pansiot, J.J.: On the reachability problem for 5-dimensional vector addition systems. Theor. Comput. Sci. 8(2), 135–159 (1979)
13. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 645–659. Springer, Heidelberg (2010)
14. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. 3(2), 147–195 (1969)
15. Kosaraju, S.R.: Decidability of reachability in vector addition systems. In: STOC 1982, pp. 267–281. ACM Press, New York (1982)
16. Leroux, J.: Vector addition system reachability problem: a short self-contained proof. In: POPL 2011, pp. 307–316. ACM Press, New York (2011)
17. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: STOC 1981, pp. 238–246. ACM Press, New York (1981)
18. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. 6(2), 223–231 (1978)
19. Rosier, L.E., Yen, H.C.: A multiparameter analysis of the boundedness problem for vector addition systems. J. Comput. Syst. Sci. 32(1), 105–135 (1986)
20. Valk, R., Vidal-Naquet, G.: Petri nets and regular languages. J. Comput. Syst. Sci. 23(3), 299–325 (1981)
21. Weispfenning, V.: The complexity of almost linear Diophantine problems. J. Symb. Comput. 10(5), 395–403 (1990)
22. Yen, H.C.: A unified approach for deciding the existence of certain Petri net paths. Inform. and Comput. 96(1), 119–137 (1992)

Hard Functions for Low-Degree Polynomials over Prime Fields

Andrej Bogdanov[1], Akinori Kawachi[2], and Hidetoki Tanaka[2]

[1] Department of Computer Science and Engineering,
The Chinese University of Hong Kong
[2] Department of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology

Abstract. In this paper, we present a new hardness amplification for low-degree polynomials over *prime fields*, namely, we prove that if some function is mildly hard to approximate by any low-degree polynomials then the sum of independent copies of the function is very hard to approximate by them. This result generalizes the XOR lemma for low-degree polynomials over the binary field given by Viola and Wigderson [22]. The main technical contribution is the analysis of the Gowers norm over prime fields. For the analysis, we discuss a generalized low-degree test, which we call the *Gowers test*, for polynomials over prime fields, which is a natural generalization of that over the binary field given by Alon, Kaufman, Krivelevich, Litsyn and Ron [2]. This Gowers test provides a new technique to analyze the Gowers norm over prime fields. Actually, the rejection probability of the Gowers test can be analyzed in the framework of Kaufman and Sudan [17]. However, our analysis is self-contained and quantitatively better. By using our argument, we also prove the hardness of modulo functions for low-degree polynomials over prime fields.

1 Introduction

Hardness amplification [23] is a method for turning a function that is somewhat hard to compute into one that is very hard to compute against a given class of adversaries. The existence of many objects in average-case complexity and cryptography, such as hard on average NP problems and one-way functions, rely on unproven assumptions. In many cases, hardness amplification allows us to prove that if weakly hard versions of such objects exist, then strongly hard ones exist as well.

In settings where complexity lower bounds are known, applications of hardness amplification are not so common. Nevertheless, the method can sometimes be used to turn unconditional weak lower bounds into strong ones. Viola and Wigderson [22] showed an XOR lemma that amplifies the hardness of functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$ against low-degree polynomials over finite fields. There are many examples of weakly hard functions for this class of adversaries. The result of Viola and Wigderson allows us to turn these into functions of related complexity that are very hard to approximate (in terms of approximation accuracy) by polynomials of the same degree.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 120–131, 2011.

Low-degree polynomials are fundamental objects in theoretical computer science, with applications in error-correcting codes, circuit complexity, probabilistically checkable proofs, and so on [18,20,3,9,8]. Applications often require the use of polynomials over fields larger than $\mathbb{F}_2$. In some cases results about polynomials over $\mathbb{F}_2$ can be easily extended to other finite fields, but in other cases different ideas are required for binary and non-binary fields.

For example, the "quadraticity test" of Gowers was first analyzed at large distances by Green and Tao [12] over non-binary fields. The extension over $\mathbb{F}_2$ by Samorodnitsky [19] required additional ideas. In the other direction, Alon, Kaufman, Krivelevich, Litsyn and Ron [2] gave an analysis of a low-degree test at small distances over $\mathbb{F}_2$. Kaufman and Ron [16] introduced substantial new ideas to generalize this test to other fields.

In this work, we generalize the XOR lemma of Viola and Wigderson [22] to arbitrary prime fields. Let $\mathbb{F}_q$ be a finite field of prime order q (identified with $\{0, ..., q-1\}$) and let $\delta(f,g) = \Pr_x[f(x) \neq g(x)]$ be the distance between f and g. In particular, we define $\delta_d(f) = \min_{p \text{ of degree } d} \delta(f,p)$, that is the distance between f and its nearest degree-d polynomial $p : \mathbb{F}_q^n \to \mathbb{F}_q$. (See Section 2 for precise definitions.) We then prove the following.

Theorem 1. *Let q be any prime number, and $f : \mathbb{F}_q^n \to \mathbb{F}_q$ be any function such that $\delta_d(f) \geq \frac{q}{(d+1)2^{d+1}}$. If*

$$t \geq \frac{q^2 \cdot (d+1) \cdot 2^{2d+3}}{3} \ln\left\{\left(\frac{q-1}{q}\right)\varepsilon^{-1}\right\},$$

then

$$\delta_d\left(f^{+t}\right) \geq \frac{q-1}{q} - \varepsilon,$$

where $f^{+t} : (\mathbb{F}_q^n)^t \to \mathbb{F}_q$ is the sum over $\mathbb{F}_q$ of t independent copies of f.

Since $\delta_d(f) \leq \frac{q-1}{q}$ for any function f (see Proposition 1), Theorem 1 allows us to construct functions that are arbitrarily close to having optimal hardness against degree-d polynomials over $\mathbb{F}_q$, by choosing $t = t(d,q,\varepsilon)$ sufficiently large.

Applying our argument, we obtain an explicit function that is very hard to approximate by polynomials of degree d:

Theorem 2. *Let $d \geq 0$ be any integer, q be any prime and m be any integer coprime to q, where $m < q$. Define $\mathsf{MOD}_m : \mathbb{F}_q^n \to \mathbb{Z}_m$ as $\mathsf{MOD}_m(x_1, ..., x_n) := x_1 + x_2 + \cdots + x_n \mod m$, where $+$ is the addition over $\mathbb{Z}$. Then, for any degree-d polynomial p,*

$$\delta(\mathsf{MOD}_m, p \bmod m) > \frac{m-1}{m} - \frac{m-1}{m} \exp\left(-\frac{1}{m^2 q} \cdot \left(\frac{q-1}{q}\right)^{d+1} \cdot \frac{n}{2^{d+2}}\right).$$

Hardness of modulo functions for low-degree polynomials for different settings of parameters has been studied in several works [1,6,13,7,22]. Directly applying our hardness amplification to a function $f(x) = x \mod m$, we would prove the hardness of another modulo function, similarly to Theorem 2. However, q and d are then forced to satisfy $\delta_d(f) \geq \frac{q}{(d+1)2^{d+1}}$.

Our proof. We generalize the proof of Viola and Wigderson [22] over $\mathbb{F}_2$. Their argument makes use of the *Gowers d-norm* $\|\cdot\|_{U^d}$ [10,11] (see Section 2 for the definition). Starting from a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ that is mildly far from degree-d polynomials over $\mathbb{F}_2$, Viola and Wigderson reason as follows: (1) From the low-degree tests analysis of Alon et al. [2], we know that if f is mildly far from degree-d polynomials, then $\|(-1)^f\|_{U^{d+1}}$ is bounded away from one. (2) By the multiplicativity of the Gowers norm, $\|(-1)^{f^{+t}}\|_{U^{d+1}} = \|(-1)^f\|_{U^{d+1}}^t$, so $\|(-1)^{f^{+t}}\|_{U^{d+1}}$ is close to zero for t sufficiently large. (3) For any polynomial p of degree d, $\|(-1)^{f^{+t}-p}\|_{U^1} \leq (\|(-1)^f\|_{U^{d+1}}^{2^{d+1}})^t$, so $\|(-1)^{f^{+t}-p}\|_{U^1}$ must be close to zero as well. The last quantity simply measures the correlation between f^{+t} and p, so p must be far from all degree-d polynomials over $\mathbb{F}_2$.

Step (2) of this analysis extends easily to prime fields; step (3) requires some additional but standard technical tools (see Lemma 2). However, step (1) relies on the analysis of the low-degree test of Alon et al., which was designed specifically for the binary field. Our main technical contribution is the extension of the analysis for this test (in fact, a slight variant of it) to arbitrary prime fields, described in Section 3. We believe that our presentation of this test is also simpler and more modular.

The test, which we call the Gowers test, works as follows: Given a function $f\colon \mathbb{F}_q^n \to \mathbb{F}_q$, choose a random set of points $x, y_1, \ldots, y_{d+1} \in \mathbb{F}_q^n$, and query f at all inputs of the form $x + a_1 y_1 + \cdots + a_{d+1} y_{d+1}$, where $(a_1, \ldots, a_{d+1})$ ranges over $\{0,1\}^{d+1}$. If the evaluations are consistent with a degree-d polynomial accept, otherwise reject.

Let us call the collection of queries $\{x + a_1 y_1 + \cdots + a_{d+1} y_{d+1} \colon (a_1, \ldots, a_{d+1}) \in \{0,1\}^{d+1}\}$ a *subcube* of $\mathbb{F}_q^n$. In the case $q = 2$, something special happens: With high probability, a subcube of $\mathbb{F}_q^n$ coincides with a rank $d+1$ affine subspace of $\mathbb{F}_q^n$. This fact plays a crucial property in the analysis of Bhattacharyya et al. [4], who obtain tight lower bounds (within a constant factor) on the rejection probability of the Gowers test over $\mathbb{F}_2$.

The low-degree test of Kaufman and Ron [16] over general fields also works by choosing a random affine subspace of appropriate dimension and checking that the restriction of f on this space is a polynomial of degree d. Their work suggests that the proper way to generalize the Gowers test to larger fields is by viewing it as a random subspace test, and not a random subcube test. However, we do not see how the Kaufman-Ron test can be used to argue hardness amplification. Unlike the Gowers test, their test does not seem to be naturally related to the Gowers norm or any other measure on functions that is multiplicative and bounds the correlation with degree-d polynomials, and so we cannot proceed with steps (2) and (3) of the Viola-Wigderson argument. Jutla, Patthak, Rudra, and Zuckerman [15] also proposed another low-degree test over prime fields, which can be viewed as a kind of random subspace tests. From a similar reason, we cannot apply their test to our analysis.

In Theorem 4 we show that if f is δ-far from a degree-d polynomial, then the Gowers test performs 2^{d+1} queries and rejects f with probability $\min\{\delta/q, 1/(d+1)2^{d+1}\}$. The Gowers test has higher query complexity than the

Kaufman-Ron test.[1] However, its rejection probability is closely related to the Gowers norm over $\mathbb{F}_q$ (see Lemma 3), and we can conclude the proof.

Our analysis of the Gowers test is a generalization of the linearity test analysis of Blum, Luby, and Rubinfeld [5]. Given a function $f\colon \mathbb{F}_q^n \to \mathbb{F}_q$ that the test accepts with high probability, they define a function $g\colon \mathbb{F}_q^n \to \mathbb{F}_q$ that is close to f, and then they argue that g must be linear. The linearity of g is proved using a self-reducibility argument, which relates evaluations of g at arbitrary inputs to evaluations at random inputs, where the identity $g(x)+g(y)=g(x+y)$ holds with high probability.

We proceed along the same lines: Given f, we define a function g that is close to f, and then argue that g must be a degree-d polynomial. To argue the second part, we use a self-reducibility argument that relates evaluations of g at arbitrary subcubes to evaluations at random subcubes. The main technical tool in the self-reduction argument is Claim 2, which to the best of our knowledge is a new identity about discrete derivatives in finite fields.

Actually, the rejection probability of the Gowers test can be analyzed in the framework of Kaufman and Sudan [17]. Their framework offers a unified analysis for general testing of algebraic functions, including the Gowers test. However, our analysis is self-contained and quantitatively better than their general one in the case where a given function is far from polynomials, which we need for the hardness amplification. Indeed, their analysis shows the rejection probability is at least approximately $1/2^{2d}$, but ours shows approximately $1/(d2^d)$ if the distance from degree-d polynomials is a constant.

The reason why we suppose prime fields in our results is that the characterization of polynomials used in the Gowers test makes sense only over prime fields (Theorem 3). We need to discover a new characterization of polynomials over non-prime fields connected to the Gowers norm for further generalization.

2 Preliminaries

Notions and notation. We begin with basic notions and notation. Let q be a prime number. We denote by $\mathbb{F}_q$, a finite field of prime order q, identified with the set $\mathbb{Z}_q := \{0, ..., q-1\}$. Let $\mathbb{F}_q^+$ be a set of non-zero elements in $\mathbb{F}_q$, namely, $\mathbb{F}_q \setminus \{0\}$. First, we define multivariate polynomials over $\mathbb{F}_q$.

Definition 1 (polynomial). *For an n-variate function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and an integer $d \geq 0$, if f can be written as*

$$f(x) = \sum_{\alpha \in \mathbb{F}_q^n, \sum_{i=1}^n \alpha_i \leq d} C_\alpha \prod_{j=1}^n x_j^{\alpha_j},$$

where each $C_\alpha \in \mathbb{F}_q$, then we call f a degree-d polynomial.

[1] The Kaufman-Ron test makes q^ℓ queries, where $\ell = \lceil (d+1)/(q-q/p) \rceil$ and $q = p^k$ for a prime p and integer k. While the rejection probability was analyzed by themselves, Haramaty, Shpilka, and Sudan [14] announced an optimal (up to constant factor) rejection probability about $\min\{\Omega(\delta_d(f)q^\ell), \Omega(1)\}$ of the test very recently.

For multivariate polynomials over prime fields $\mathbb{F}_q$, the so-called directional derivatives can be defined for well-known characterization of polynomials over $\mathbb{F}_q$.

Definition 2 (derivative). *Let G, H be any additive groups. For a function $f : G \to H$ and an element $y \in G$, a derivative of f on y, denoted by $\Delta_y f$, is defined as*

$$\Delta_y f(x) := f(x+y) - f(x).$$

A k-th derivative of f on vectors $y_1, \ldots, y_k \in G$ is recursively defined such that

$$\Delta_{y_1,\ldots,y_k} f(x) := \Delta_{y_1,\ldots,y_{k-1}} \left(\Delta_{y_k} f(x)\right).$$

The well-known characterization of degree-d polynomials over prime fields $\mathbb{F}_q$ with $(d+1)$-th derivatives is given by the following (folklore) theorem[2].

Theorem 3 (characterization of polynomials). *For a function $f : \mathbb{F}_q^n \to \mathbb{F}_q$, $\Delta_{y_1,\ldots,y_{d+1}} f(x) = 0$ for any $x, y_1, \ldots, y_{d+1} \in \mathbb{F}_q^n$ if and only if f is a degree-d polynomial.*

Note that this characterization does not hold over non-prime fields in general.

For functions $f, g : G \to H$, the distance between f and g is defined as $\delta(f,g) := \Pr_{x \in G}[f(x) \neq g(x)]$. The distance between a function f and the set of all the degree-d polynomials is defined as $\delta_d(f) := \min_{p \in \mathcal{P}_{d,n}} \delta(f,p)$, where $\mathcal{P}_{d,n}$ is the set of all degree-d n-variate polynomials. Note that the distance has an upper bound.

Proposition 1. *For any function $f : G \to H$ and any integer d, $\delta_d(f) \leq \frac{|H|-1}{|H|}$.*

Proof. Let $c \in H$ be the value that occurs most often in f. Then, $\delta_d(f) \leq \delta(f,c) \leq \frac{|H|-1}{|H|}$. □

Gowers uniformity. The Gowers norm is a measure for correlation between functions and low-degree polynomials over finite fields. This measure was originally introduced by Gowers [10,11] to give an alternative proof of Szemerédi's theorem. We call it the Gowers uniformity here.[3]

Definition 3. *For every function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and every integer $k \geq 0$, the degree-k Gowers uniformity $U_k(f)$ is defined as*

$$U_k(f) := \mathop{\mathbb{E}}_{x,y_1,\ldots,y_k \in \mathbb{F}_q^n} \left[\omega_q^{\Delta_{y_1,\ldots,y_k} f(x)}\right],$$

where $\omega_q := \exp(2\pi i/q)$ and $\mathbb{E}[\cdot]$ is the expectation.

Remark 1. If $k \geq 1$, the degree-k Gowers uniformity $U_k(f)$ is a non-negative real number, namely $U_k(f) = |U_k(f)|$.

[2] A proof of Theorem 3 appears in, e.g., Terence Tao's Weblog [21].

[3] The Gowers norm $\|F\|_{U^k}$ is defined as $(U_k(f))^{1/2^k}$, where $F(x) := \omega_q^{f(x)}$.

The Gowers uniformity has important properties.

Proposition 2. *For any function $f : \mathbb{F}_q^n \to \mathbb{F}_q$, the following statements hold:*

1. $|U_k(f)| \le \sqrt{U_{k+1}(f)}$ *for any integer* $k \ge 0$
2. $U_{d+1}(f-p) = U_{d+1}(f)$ *for any degree-*d *polynomial* $p : \mathbb{F}_q^n \to \mathbb{F}_q$
3. $U_k(f^{+t}) = (U_k(f))^t$ *for any integers* $k \ge 0$ *and* $t > 0$,

where $f^{+t} : \left(\mathbb{F}_q^n\right)^t \to \mathbb{F}_q$ *is the sum of* t *independent copies of* f *defined as*

$$f^{+t}(x^{(1)}, \ldots, x^{(t)}) := f(x^{(1)}) + f(x^{(2)}) + \cdots + f(x^{(t)})$$

for an integer $t > 0$.

These properties can be proven by the almost same argument as of the Gowers norm in [10,11], and so we omit the proofs.

3 Gowers Test

Next, we consider a low-degree test for polynomials, which we call the Gowers test. The Gowers test is derived from the characterization of polynomials given in Theorem 3.

Definition 4 (Gowers test). *The degree-*d *Gowers test for a function* $f : \mathbb{F}_q^n \to \mathbb{F}_q$, *denoted by* $\mathsf{GT}_d(f)$, *is the following procedure:*

1. *Pick* $x, y_1, \ldots, y_{d+1} \in \mathbb{F}_q^n$ *uniformly and independently at random;*
2. *Accept if and only if* $\Delta_{y_1,\ldots,y_{d+1}} f(x) = 0$.

We denote by $\rho_d(f)$, *the rejection probability of* $\mathsf{GT}_d(f)$.

By Theorem 3, if f has degree at most d, $\mathsf{GT}_d(f)$ accepts with probability 1. Our question is how large the rejection probability is in the case when f is not a degree-d polynomial. An answer to this question is given in the following theorem, which estimates the rejection probability of $\mathsf{GT}_d(f)$, that is $\rho_d(f)$.

Theorem 4. *Let f be any function $\mathbb{F}_q^n \to \mathbb{F}_q$. Then*

$$\rho_d(f) \ge \min\left\{\frac{\delta_d(f)}{q}, \frac{1}{(d+1)2^{d+1}}\right\}.$$

Proof. This proof is obtained from the following technical main lemma:

Lemma 1. *Let $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and $\epsilon < \frac{1}{(d+1)2^{d+1}}$. If $\mathsf{GT}_d(f)$ accepts with probability $1-\epsilon$, then $\delta_d(f) \le q\epsilon$.*

If $\rho_d(f) \ge 1/(d+1)2^{d+1}$, we are done. So, assume that $\rho_d(f) < 1/(d+1)2^{d+1}$. Let $\epsilon := \rho_d(f)$. By Lemma 1, $\delta_d(f) \le q\epsilon = q\rho_d(f)$. Then we obtain $\rho_d(f) \ge \delta_d(f)/q$. Hence, the theorem follows. □

We prove the main technical lemma below.

Proof (Proof of Lemma 7). Our proof of this lemma is a generalization of the linearity test analysis of Blum, Luby, and Rubinfeld [5], using ideas from the work of Alon et al. [2] to higher degree polynomials over $\mathbb{F}_2$. Namely, we construct a function g such that (1) $g(x) = 0$ for all but at most $q\epsilon$ fraction of inputs x and (2) $g(x) - f(x)$ is a degree-d polynomial. We define

$$g(x) = \text{the plurality value of } \Delta_{y_1,\ldots,y_{d+1}} f(x), \text{ where } y_1, \ldots, y_{d+1} \in \mathbb{F}_q^n,$$

where if the plurality value is not unique, we define $g(x)$ as an arbitrary value from the plurality ones. The property (1) is almost immediate: If $g(x) \neq 0$, it follows that $\Pr_{y_1,\ldots,y_{d+1}}[\Delta_{y_1,\ldots,y_{d+1}} f(x) \neq 0] \geq 1/q$, so if $g(x) \neq 0$ for more than a $q\epsilon$ fraction of xs, it would follow that the Gowers test rejects with probability more than $(q\epsilon)/q = \epsilon$, a contradiction.

We now prove the property (2). We begin by showing that for *all* x, $g(x)$ not only agrees with the plurality value of $\Delta_{y_1,\ldots,y_{d+1}} f(x)$, but in fact with a vast majority:

Claim 1. *For all* $x \in \mathbb{F}_q^n$, $\Pr_{y_1,\ldots,y_{d+1}}[g(x) = \Delta_{y_1,\ldots,y_{d+1}} f(x)] \geq 1 - (d+1)\epsilon$.

We will also make use of the following identity. For $a \in \{0,1\}^{d+1}$, let $|a| = a_1 + \cdots + a_{d+1} \bmod 2$.

Claim 2. *For all* $x, y_1, \ldots, y_{d+1}, z_1, \ldots, z_{d+1} \in \mathbb{F}_q^n$,

$$\Delta_{z_1,\ldots,z_{d+1}} f(x) = \sum_{a \in \{0,1\}^{d+1}} (-1)^{|a|} \Delta_{y_1 - a_1 z_1, \ldots, y_{d+1} - a_{d+1} z_{d+1}} f\left(x + \sum_{i=1}^{d+1} a_i z_i\right).$$

The proofs of Claims 1 and 2 will be given in the full version. We are now in a position to prove that $g - f$ is a polynomial of degree d. By Claim 1, we have that

$$\Pr_{y_1,\ldots,y_{d+1}}\left[g\left(x + \sum_{i=1}^{d+1} a_i z_i\right) \neq \Delta_{y_1 - a_1 z_1, \ldots, y_{d+1} - a_{d+1} z_{d+1}} f\left(x + \sum_{i=1}^{d+1} a_i z_i\right)\right] \leq (d+1)\epsilon$$

for all $x, z_1, \ldots, z_{d+1} \in \mathbb{F}_q^n$, and $a \in \{0,1\}^{d+1}$. Taking a union bound over all $a \in \{0,1\}^{d+1}$ it follows that

$$\Pr_{y_1,\ldots,y_{d+1}}\left[\exists a\colon g\left(x + \sum_{i=1}^{d+1} a_i z_i\right) \neq \Delta_{y_1 - a_1 z_1, \ldots, y_{d+1} - a_{d+1} z_{d+1}} f\left(x + \sum_{i=1}^{d+1} a_i z_i\right)\right] \leq 2^{d+1} \cdot (d+1)\epsilon < 1.$$

Therefore, there must exist values for $y_1, \ldots, y_{d+1}$ such that

$$g\left(x + \sum_{i=1}^{d+1} a_i z_i\right) = \Delta_{y_1 - a_1 z_1, \ldots, y_{d+1} - a_{d+1} z_{d+1}} f\left(x + \sum_{i=1}^{d+1} a_i z_i\right)$$

for all $x, z_1, \ldots, z_{d+1}$ in $\mathbb{F}_q^n$ and $a \in \{0,1\}^{d+1}$. But then

$$\begin{aligned}\Delta_{z_1,\ldots,z_{d+1}} g(x) &= \sum_{a\in\{0,1\}^{d+1}} (-1)^{|a|} g\left(x + \sum_{i=1}^{d+1} a_i z_i\right) \\ &= \sum_{a\in\{0,1\}^{d+1}} (-1)^{|a|} \Delta_{y_1 - a_1 z_1,\ldots,y_{d+1}-a_{d+1}z_{d+1}} f\left(x + \sum_{i=1}^{d+1} a_i z_i\right) \\ &= \Delta_{z_1,\ldots,z_{d+1}} f(x)\end{aligned}$$

by Claim 2, and so $\Delta_{z_1,\ldots,z_{d+1}}(f-g)(x) = 0$ for all $x, z_1, \ldots, z_{d+1}$, namely, $f-g$ is a degree-d polynomial. □

4 Hardness Amplification

Our goal is to construct a hard function for low-degree polynomials (in other words, a function far from low-degree polynomials) from a mildly hard function for low-degree polynomials (in other words, a function mildly far from low-degree polynomials). Recall that, for a function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and an integer $t > 0$, a function $f^{+t} : (\mathbb{F}_q^n)^t \to \mathbb{F}_q$ is defined as

$$f^{+t}(x^{(1)}, \ldots, x^{(t)}) := f(x^{(1)}) + f(x^{(2)}) + \cdots + f(x^{(t)}).$$

We prove that f^{+t} is very hard for low-degree polynomials if f is mildly hard for low-degree polynomials. Recall that $\delta_d(f) \le \frac{q-1}{q}$ for any function f. Hence our goal is to prove $\delta_d(f^{+t}) \ge \frac{q-1}{q} - \epsilon$ for some small ϵ.

Theorem 5. *Let f be any function such that $\delta_d(f) > \frac{q}{(d+1)2^{d+1}}$ and $t > 0$ be any integer. Then*

$$\delta_d\left(f^{+t}\right) > \frac{q-1}{q} - \frac{q-1}{q}\exp\left(-\frac{3t}{q^2\cdot(d+1)\cdot 2^{2d+3}}\right).$$

Note that our main theorem (Theorem 1) in Section 1 immediately follows from this theorem by choosing t appropriately.

Proof. We first state two lemmas, of which proofs will be shown in the full version, on relations between the distance from degree-d polynomials and the Gowers uniformity and between the Gowers uniformity and the rejection probability of the Gowers test.

Lemma 2 (distance to uniformity). *For any function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and any integer d,*

$$\delta_d(f) \ge \frac{q-1}{q} - \frac{q-1}{q} \mathop{\mathbb{E}}_{a\in\mathbb{F}_q^+}\left[\left(U_{d+1}(af)\right)^{1/2^{d+2}}\right].$$

Lemma 3 (uniformity to test). *For any function* $f : \mathbb{F}_q^n \to \mathbb{F}_q$ *and any integer* $d \geq 0$,

$$U_{d+1}(f) < 1 - \frac{3}{q^2}\rho_d(f).$$

Recall that $\rho_d(f)$ is the rejection probability of the Gowers test $\mathsf{GT}_d(f)$.

Note that the distance $\delta_d(f)$ is lower bounded by $\frac{q-1}{q}$ minus the term involved with $\mathbb{E}_{a\in\mathbb{F}_q^+}\left[(U_{d+1}(af))^{1/2^{d+2}}\right]$ in Lemma 2. One can easily see that the expectation is not required in the binary case, as in [22]. Hence, our analysis needs some technical tricks for the general case.

We can prove Theorem 5 using these two lemmas, Proposition 2 and Theorem 4. By Lemma 2 and the averaging principle, there is an $\alpha \in \mathbb{F}_q^+$ such that

$$\delta_d(f^{+t}) \geq \frac{q-1}{q} - \frac{q-1}{q}\left(U_{d+1}(\alpha f^{+t})\right)^{1/2^{d+2}}.$$

By the property of the Gowers uniformity (Proposition 2 (3)),

$$\left(U_{d+1}(\alpha f^{+t})\right)^{1/2^{d+2}} = \left(U_{d+1}(\alpha f)\right)^{t/2^{d+2}}.$$

Then, by Lemma 3,

$$\left(U_{d+1}(\alpha f)\right)^{t/2^{d+2}} < \left(1 - \frac{3}{q^2}\rho_d(f)\right)^{t/2^{d+2}} < \exp\left(-\frac{3}{q^2}\cdot\frac{t}{2^{d+2}}\rho_d(f)\right).$$

Note that $\rho_d(\alpha f) = \rho_d(f)$, since for all $x, y_1, \ldots, y_{d+1} \in \mathbb{F}_q$ and all $\alpha \in \mathbb{F}_q^+$, $\Delta_{y_1,\ldots,y_{d+1}}(\alpha f(x)) = 0$ if and only if $\Delta_{y_1,\ldots,y_{d+1}} f(x) = 0$. Combining this with Theorem 4 and the assumption that $\delta_d(f) > \frac{q}{(d+1)2^{d+1}}$, we obtain

$$\left(U_{d+1}(\alpha f^{+t})\right)^{1/2^{d+2}} < \exp\left(-\frac{3}{q^2}\cdot\frac{t}{2^{d+2}}\cdot\frac{1}{(d+1)2^{d+1}}\right).$$

Therefore

$$\delta_d\left(f^{+t}\right) > \frac{q-1}{q} - \frac{q-1}{q}\exp\left(-\frac{3t}{q^2\cdot(d+1)\cdot 2^{2d+3}}\right).$$

□

5 Hardness of MOD_m

$\mathsf{MOD}_m : \mathbb{F}_q^n \to \mathbb{Z}_m$ is defined as

$$\mathsf{MOD}_m(x) := x_1 + x_2 + \cdots + x_n \mod m,$$

where $1 < m < q$ and $+$ is the addition over $\mathbb{Z}$. In this section, we see the distance between MOD_m and low-degree polynomials.

Since the range of MOD_m is $\mathbb{Z}_m$, we define the distance δ_d between MOD_m and degree-d polynomials as follows:

$$\delta_d(\mathsf{MOD}_m) := \min_{p \in \mathcal{P}_{d,n}} \Pr_{x \in \mathbb{F}_q^n}[\mathsf{MOD}_m(x) \neq (p(x) \bmod m)].$$

Namely, we identify a standard polynomial (from $\mathbb{F}_q^n$ to $\mathbb{F}_q$) modulo m as a polynomial from $\mathbb{F}_q^n$ to $\mathbb{Z}_m$ here. Also, we modify the definition of the Gowers uniformity $U_d(f)$ for such functions $f : \mathbb{F}_q^n \to \mathbb{Z}_m$:

$$U_d(f) := \mathop{\mathbb{E}}_{x,y_1,\ldots,y_d \in \mathbb{F}_q^n} \left[\omega_m^{\Delta_{y_1,\ldots,y_d} f(x)}\right].$$

It is easy to see the same properties given in Proposition 2 hold for this definition as before.

Theorem 6. *Let $d \geq 0$ be any integer, q be any prime, and m be any integer coprime to q, where $m < q$. Then,*

$$\delta_d(\mathsf{MOD}_m) > \frac{m-1}{m} - \frac{m-1}{m} \exp\left(-\frac{1}{m^2 q} \cdot \left(\frac{q-1}{q}\right)^{d+1} \cdot \frac{n}{2^{d+2}}\right).$$

Proof. By the same reasoning of Lemma 2 and the averaging argument, there is an $\alpha \in \mathbb{F}_q^+$ such that

$$\delta_d(\mathsf{MOD}_m) \geq \frac{m-1}{m} - \frac{m-1}{m} \left(U_{d+1}(\alpha \mathsf{MOD}_m)\right)^{1/2^{d+2}}.$$

Let $f : \mathbb{F}_q \to \mathbb{Z}_m$ be a 1-variable function defined as $f(x) = x \bmod m$. Then, we have $U_{d+1}(\alpha \mathsf{MOD}_m)^{1/2^{d+2}} = U_{d+1}(\alpha f)^{n/2^{d+2}}$ by the same reasoning of Proposition 2 and Theorem 3. So, we now estimate an upper bound of $U_{d+1}(\alpha f)$ by using the following claim.

Claim 3. *For any function f, the following properties hold:*

1. *If $y_i = 0$ for some i, then $\omega_m^{\Delta_{y_1,\ldots,y_{d+1}} f} \equiv 1$.*
2. *If ω_m^f is not a constant function and $y_i \neq 0$ for all i, then $\omega_m^{\Delta_{y_1,\ldots,y_{d+1}} f}$ is not a constant function.*

Proof. We first show the property 1. By the symmetry of derivatives, we can suppose that $y_{d+1} = 0$ without loss of generality. Then, for any x,

$$\Delta_{y_1,\ldots,y_d,y_{d+1}} f(x) = \Delta_{y_1,\ldots,y_d}\left(f(x+0) - f(x)\right) = 0.$$

Thus, $\omega_m^{\Delta_{y_1,\ldots,y_{d+1}} f(x)} = 1$ for any x.

We next prove the property 2. We show the following statement: "If ω_m^f is not a constant, then $\omega_m^{\Delta_y f}$ is not a constant function for every nonzero $y \in \mathbb{F}_q^+$." Repeatedly applying this statement, we obtain the property 2.

We prove its contrapositive. Suppose $\omega_m^{\Delta_y f(x)}$ is a constant function for some nonzero $y \in \mathbb{F}_q$. Then it must be that for every $x \in \mathbb{F}_q$:

$$f((x+y) \bmod q) - f(x \bmod q) \equiv c \mod m.$$

Plugging in $x := x+y, x+2y, \ldots, x+(q-1)y$, we obtain

$$\begin{aligned} f((x+2y) \bmod q) - f((x+y) \bmod q) &\equiv c \mod m, \\ f((x+3y) \bmod q) - f((x+2y) \bmod q) &\equiv c \mod m, \\ &\vdots \\ f((x+qy) \bmod q) - f((x+(q-1)y) \bmod q) &\equiv c \mod m. \end{aligned}$$

If we add these equations, on the left hand side we obtain zero, and on the right hand side we obtain $qc \mod m$, which equals zero only if $c = 0$. If $c = 0$, then f is a constant function since we have $\Delta_y f(x) \equiv 0 \bmod m$. □

By this claim, we have for some nonzero $0 < \alpha' < m$

$$\begin{aligned} &\mathop{\mathbb{E}}_{x,y_1,\ldots,y_{d+1}} \left[\omega_m^{\alpha \Delta_{x,y_1,\ldots,y_{d+1}} f(x)}\right] \\ &\leq \frac{1}{q^{d+2}} \left\{ \left(q^{d+2} - (q-1)^{d+1} \cdot q\right) \cdot 1 + (q-1)^{d+1} \left| (q-1) \cdot 1 + \omega_m^{\alpha'} \right| \right\} \\ &< 1 - \frac{1}{m^2 q} \left(\frac{q-1}{q}\right)^{d+1}. \end{aligned}$$

From this estimation, the theorem immediately follows. □

Acknowledgments. Andrej Bogdanov's work was partially supported by Hong Kong RGC GRF grant 2150617. Akinori Kawachi's work was partially supported by Grant-in-Aid for Scientific Research (B) 21300002.

References

1. Alon, N., Beigel, R.: Lower bounds for approximations by low degree polynomials over $\mathbb{Z}_m$. In: Proceedings of the 16 th IEEE Conference on Computational Complexity, pp. 184–187 (2001)
2. Alon, N., Kaufman, T., Krivelevich, M., Litsyn, S., Ron, D.: Testing low-degree polynomials over GF(2). In: Proceedings of RANDOM-APPROX, pp. 188–199 (2003)
3. Babai, L., Fortnow, L., Lund, C.: Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity 1(1), 3–40 (1991)
4. Bhattacharyya, A., Kopparty, S., Schoenebeck, G., Sudan, M., Zuckerman, D.: Optimal testing of Reed-Muller codes. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (2010)
5. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/Correcting with Applications to Numerical Problems. Journal of Computer and System Sciences (3), 549–595 (1993)

6. Bourgain, J.: Estimation of certain exponential sums arising in complexity theory. Comptes Rendus Mathematique 340(9), 627–631 (2005)
7. Chattopadhyaym, A.: An improved bound on correlation between polynomials over $\mathbb{Z}_m$ and MOD_q. Technical Report TR06-107, Electronic Colloquium on Computational Complexity (2006)
8. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. Journal of the ACM 43(2), 268–292 (1996)
9. Gemmell, P., Lipton, R.J., Rubinfeld, R., Sudan, M., Wigderson, A.: Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the 23th Annual ACM Symposium on Theory of Computing, pp. 32–42 (1991)
10. Gowers, T.: A new proof of Szemerédi's theorem for arithmetic progressions of length four. Geometric and Functional Analysis 8(3), 529–551 (1998)
11. Gowers, T.: A new proof of Szemerédi's theorem. Geometric and Functional Analysis 11(3), 465–588 (2001)
12. Green, B., Tao, T.: An inverse theorem for the Gowers U^3 norm. Technical report, Mathematics ArXiv NT/0503014 (2005)
13. Green, F., Roy, A., Straubing, H.: Bounds on an exponential sum arising in Boolean circuit complexity. Comptes Rendus Mathematique 341(5), 279–282 (2005)
14. Haramaty, E., Shpilka, A., Sudan, M.: Optimal testing of multivariate polynomials over small prime fields. Technical Report TR11-059, Electronic Colloquium on Computational Complexity (2011)
15. Jutla, C.S., Patthak, A.C., Rudra, A., Zuckerman, D.: Testing low-degree polynomials over prime fields. Random Structures and Algorithms 35(2), 163–193 (2009)
16. Kaufman, T., Ron, D.: Testing polynomials over general fields. SIAM Journal on Computing 36(3), 779–802 (2006)
17. Kaufman, T., Sudan, M.: Algebraic property testing: the role of invariance. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 403–412 (2008)
18. Razborov, A.: Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. Mathematical notes of the Academy of Sciences of the USSR 41(4), 333–338 (1987)
19. Samorodnitsky, A.: Low degree tests at large distances. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 506–515 (2007)
20. Smolensky, R.: Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, pp. 77–82 (1987)
21. Tao, T.: Some notes on non classical polynomials in finite characteristic (2008), http://terrytao.wordpress.com/2008/11/13/some-notes-on-non-classical-polynomials-in-finite-characteristic/
22. Viola, E., Wigderson, A.: Norms, XOR lemmas, and lower bounds for polynomials and protocols. Theory of Computing 4(1), 137–168 (2008)
23. Yao, A.C.-C.: Theory and applications of trapdoor functions (extended abstract). In: Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, pp. 80–91 (1982)

Temporal Logics for Concurrent Recursive Programs: Satisfiability and Model Checking*

Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and Marc Zeitoun

LSV, ENS Cachan, CNRS & INRIA, France
{bollig,cyriac,gastin,zeitoun}@lsv.ens-cachan.fr

Abstract. We develop a general framework for the design of temporal logics for concurrent recursive programs. A program execution is modeled as a partial order with multiple nesting relations. To specify properties of executions, we consider any temporal logic whose modalities are definable in monadic second-order logic and that, in addition, allows PDL-like path expressions. This captures, in a unifying framework, a wide range of logics defined for ranked and unranked trees, nested words, and Mazurkiewicz traces that have been studied separately. We show that satisfiability and model checking are decidable in EXPTIME and 2EXPTIME, depending on the precise path modalities.

1 Introduction

It is widely acknowledged that linear-time temporal logic (LTL) [18] is a yardstick among the specification languages. It combines high expressiveness (equivalence to first-order logic) with a reasonable complexity of decision problems such as satisfiability and model checking. LTL has originally been considered for finite-state sequential programs. As real programs are often concurrent or rely on recursive procedures, LTL has been extended in two directions.

First, asynchronous finite-state programs (or Zielonka automata) [10] are a formal model of shared-memory systems and properly generalize finite-state sequential programs. Their executions are no longer sequential (i.e., totally ordered) but can be naturally modeled as graphs or partial orders. In the literature, these structures are known as *Mazurkiewicz traces*. They look back on a long list of now classic results that smoothly extend the purely sequential setting (e.g., expressive equivalence to first-order logic) [10,9].

Second, in an influential paper, Alur and Madhusudan extend the finite-state sequential model to *visibly pushdown automata* (VPA) [3]. VPA are a flexible model for recursive programs, where subroutines can be called and executed while the current thread is suspended. The execution of a VPA is still totally ordered. However, it comes with some extra information that relates a subroutine call with the corresponding return position, which gives rise to the notion of *nested words* [3]. Alur et al. recently defined versions of LTL towards this infinite-state setting [2,1] that can be considered as canonical counterparts of the classical logic introduced by Pnueli.

* Supported by ARCUS, DOTS (ANR-06-SETIN-003), and DIGITEO LoCoReP.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 132–144, 2011.

To model programs that involve both recursion and concurrency, one needs to mix both views. Most approaches to modeling concurrent recursive programs, however, reduce concurrency to interleaving and neglect a behavioral semantics that preserves independencies between program events [19,15,4]. A first model for concurrent recursive programs with partial-order semantics was considered in [5]. Executions of their *concurrent VPA* equip Mazurkiewicz traces with multiple nesting relations. Temporal logics have not been considered, though, and there is for now no canonical merge of the two existing approaches. It must be noted that satisfiability is undecidable when considering multiple nesting relations, even for simple logics. Yet, it becomes decidable if we restrict to system behaviors that can be executed within a bounded number of phase switches, a notion introduced in [15]. A phase switch consists of a transfer of control from one process to another. This allows for the discovery of many errors, since they typically manifest themselves after a few phase switches [19].

In this paper, we present linear-time temporal logics for concurrent recursive programs. A temporal logic is parametrized by a finite set of modalities that are definable in monadic second-order logic (cf. [12]). In addition, it provides path expressions similar to those from PDL [11] or XPath [17], which are orthogonal to the modalities. This general framework captures temporal logics considered in [2,1,8] when we restrict to one process, and it captures those considered in [9,12,13] when we go without recursion. Our decision procedures for the (bounded phase) satisfiability problem are optimal in all these special cases, but provide a unifying proof. They also apply to other structures such as ranked and unranked trees. We then use our logics for model checking. To do so, we provide a system model whose behavioral semantics preserves concurrency (unlike the models from [19,4,15]). The complexity upper bounds from satisfiability are preserved.

Outline. In Section 2, we introduce graphs, trees, and nested traces as our model of program executions. Section 3 provides a range of related temporal logics. Sections 4 and 5 address satisfiability and model checking, resp. The full version of this paper is available at: `http://hal.archives-ouvertes.fr/hal-00591139/`

2 Graphs, Nested Traces, and Trees

To model the behavior of distributed systems, we consider labeled graphs, each representing one single execution. A node of a graph is an event that can be observed during an execution. Its labeling reveals its type (e.g., procedure call, return, or internal) or some processes that are involved in its execution. Edges reflect causal dependencies: an edge (u, v) from node u to node v implies that u happens before v. A labeling of (u, v) may provide information about the kind of causality between u and v (e.g., successive events on some process).

Accordingly, we consider a *signature*, which is a pair $\mathcal{S} = (\Sigma, \Gamma)$ consisting of a finite set Σ of node labelings and a finite set Γ of edge labelings. Throughout the paper, we assume $|\Sigma| \geq 1$ and $|\Gamma| \geq 2$. An $\mathcal{S}$-*graph* is a structure $G = (V, \lambda, \nu)$ where V is a non-empty set of countably many *nodes*, $\lambda : V \to 2^{\Sigma}$ is the *node-labeling function*, and $\nu : (V \times V) \to 2^{\Gamma}$ is the *edge-labeling function*, with the

intuitive understanding that there is an edge between u and v iff $\nu(u,v) \neq \emptyset$. For $\sigma \in \Sigma$, $V_\sigma := \{u \in V \mid \sigma \in \lambda(u)\}$ denotes the set of nodes that are labeled with σ. Moreover, for $\gamma \in \Gamma$, $E_\gamma := \{(u,v) \in V \times V \mid \gamma \in \nu(u,v)\}$ denotes the set of edges with labeling γ. Then, $E := \bigcup_{\gamma \in \Gamma} E_\gamma$ is the set of all the edges. We require that the transitive closure E^+ of E is a well-founded (strict) partial order on V. We write $\prec^G$ or simply $\prec$ for E^+, and we write $\preceq^G$ or $\preceq$ for E^*.

Nested Traces. To model executions of concurrent recursive programs that communicate via shared variables, we introduce graphs with multiple nesting relations. We fix non-empty finite sets *Proc* and *Act*, and let $\mathit{Type} = \{\mathsf{call}, \mathsf{ret}, \mathsf{int}\}$. Then, $\Sigma = \mathit{Proc} \cup \mathit{Act} \cup \mathit{Type}$ is the set of node labelings. Its component *Type* indicates whether an event is a procedure *call*, a *return*, or an *internal* action. A nesting edge connects a procedure call with the corresponding return, and will be labeled by $\mathtt{cr} \in \Gamma$. In addition, we use $\mathtt{succ}_p \in \Gamma$ to label those edges that link successive events of process $p \in \mathit{Proc}$. Thus, $\Gamma = \{\mathtt{succ}_p \mid p \in \mathit{Proc}\} \cup \{\mathtt{cr}\}$. We obtain the signature $\mathcal{S} = (\Sigma, \Gamma)$. Formally, a *nested (Mazurkiewicz) trace* over *Proc* and *Act* is an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ such that the following hold:

- T1 $V = V_{\mathsf{call}} \uplus V_{\mathsf{ret}} \uplus V_{\mathsf{int}} = \biguplus_{a \in \mathit{Act}} V_a = \bigcup_{p \in \mathit{Proc}} V_p$
- T2 for all processes $p, q \in \mathit{Proc}$ with $p \neq q$, we have $V_p \cap V_q \subseteq V_{\mathsf{int}}$
- T3 for all $p \in \mathit{Proc}$, $E_{\mathtt{succ}_p}$ is the direct successor relation of a total order on V_p
- T4 $E_{\mathtt{cr}} \subseteq (V_{\mathsf{call}} \times V_{\mathsf{ret}}) \cap \bigcup_{p \in \mathit{Proc}} (V_p \times V_p)$
- T5 for all $(u,v), (u',v') \in E_{\mathtt{cr}}$, we have $u = u'$ iff $v = v'$
- T6 for all $p \in \mathit{Proc}$ and $u \in V_{\mathsf{call}} \cap V_p$ and $v' \in V_{\mathsf{ret}} \cap V_p$, if $u \prec v'$ then either there exists $v \preceq v'$ with $(u,v) \in E_{\mathtt{cr}}$ or there exists $u' \succeq u$ with $(u',v') \in E_{\mathtt{cr}}$

Intuitively, each event has exactly one type and one action and belongs to at least one process (T1), synchronizing events are always internal (T2), along any process the events are totally ordered (T3), a nesting edge is always between a call and a return of the same process (T4), and cr-edges restricted to any process are well nested (T5 and T6). Note that we may have unmatched calls or returns.

For $u \in V$, we let $\mathit{Proc}(u) = \lambda(u) \cap \mathit{Proc}$. When $|\mathit{Proc}| = 1$, then a nested trace is a nested word in the classical sense [3]. The set of nested traces over *Proc* and *Act* is denoted by $\mathit{Traces}(\mathit{Proc}, \mathit{Act})$. Figure 1 depicts a nested trace over $\mathit{Proc} = \{p, q\}$ and $\mathit{Act} = \{c, r, sv\}$. Action c denotes a call, r a return, and sv reveals some synchronization via a shared variable. Node labelings from *Proc* are given by the gray-shaded regions, i.e., sv-events involve both p and q. Edge labelings $\mathtt{succ}_p$ and $\mathtt{succ}_q$ are abbreviated by p and q, resp.

We introduce a restricted class of nested traces over *Proc* and *Act*. It is parametrized by an (existential) upper bound $k \geq 1$ on the number of *phases* that a trace needs to be executed. In each phase, return events belong to one dedicated process. Let us first introduce the notion of *linearization*. A linearization of a nested trace $G = (V, \lambda, \nu)$ is any structure $(V, \lambda, \leq)$ such that $\leq$ is a total order extending $\preceq$. Fig. 2 depicts a linearization of the nested trace from Fig. 1. We identify isomorphic structures so that a linearization can be considered as a word over 2^Σ. Note that, for every word $w \in (2^\Sigma)^*$, there is at most one (up to isomorphism) nested trace G such that w is a linearization of G [10].

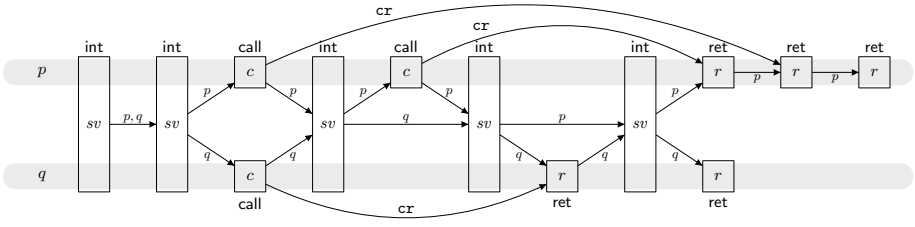


Fig. 1. A nested trace over $Proc = \{p, q\}$ and $Act = \{c, r, sv\}$

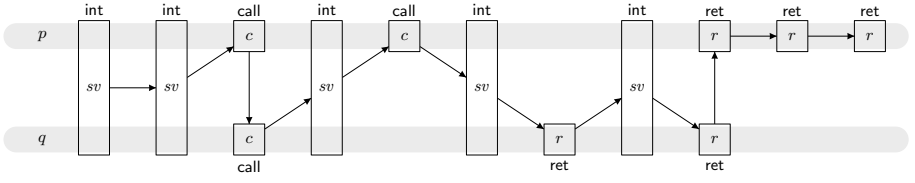


Fig. 2. A 2-phase linearization

For $k \geq 1$, a word $w \in (2^{\Sigma})^*$ is a *k-phase word* if it can be written as $w_1 \cdots w_k$ where, for all $i \in \{1, \ldots, k\}$, there is $p \in Proc$ such that for each letter a of w_i we have $\mathsf{ret} \in a$ implies $p \in a$. A nested trace is called a *k-phase nested trace* if at least one of its linearizations is a k-phase word. The set of k-phase nested traces over $Proc$ and Act is denoted by $Traces_k(Proc, Act)$. We denote by $Lin_k(G)$ the set of linearizations of nested trace G that are k-phase words. In particular, G is a k-phase nested trace iff $Lin_k(G) \neq \emptyset$. The nested trace from Fig. 1 is a 2-phase trace: its linearization from Fig. 2 schedules returns of q before all returns of p.

Ranked Trees. Let $\mathcal{S} = (\Sigma, \Gamma)$. An $\mathcal{S}$-*tree* is an $\mathcal{S}$-graph $t = (V, \lambda, \nu)$. We require that there is a "root" $u_0 \in V$ such that for all $u, v, v' \in V$ and $\gamma, \gamma' \in \Gamma$:

(i) $(u_0, u) \in E^*$, and $(v, u), (v', u) \in E$ implies $v = v'$
(ii) $(u, v), (u, v') \in E_\gamma$ implies $v = v'$, and $(u, v) \in E_\gamma \cap E_{\gamma'}$ implies $\gamma = \gamma'$

Note that Γ can be seen as a set of *directions*. Thus, $\Gamma = \{\mathtt{left}, \mathtt{right}\}$ yields binary trees. The set of all $\mathcal{S}$-trees is denoted $Trees(\mathcal{S})$.

Ordered Unranked Trees. Each node in an ordered unranked tree can have a potentially unbounded number of children, and the children of any node are totally ordered. Formally it is an $\mathcal{S}$-graph $t = (V, \lambda, \nu)$ over $\mathcal{S} = (\Sigma, \Gamma)$ where $\Gamma = \{\mathtt{child}, \mathtt{next}\}$. Again, there is a "root" $u_0 \in V$ such that for all $u, v, v' \in V$:

(i) $(u_0, u) \in E^*$ and $(u_0, u) \notin E_{\mathtt{next}}$
(ii) $(v, u), (v', u) \in E_{\mathtt{child}}$ implies $v = v'$, and $(v, u), (v', u) \in E_{\mathtt{next}}$ implies $v = v'$
(iii) $(u, v), (u, v') \in E_{\mathtt{next}}$ implies $v = v'$ and $(u, v) \in E_\gamma \cap E_{\gamma'}$ implies $\gamma = \gamma'$

(iv) $(u, v) \in E_{\texttt{child}}$ implies that there exists $v_0 \in V$ such that $(u, v_0) \in E_{\texttt{child}}$ and, $(u, v') \in E_{\texttt{child}}$ if and only if $(v_0, v') \in E^*_{\texttt{next}}$.

The set of all ordered unranked trees over $\mathcal{S}$ is denoted $o.u.Trees(\mathcal{S})$.

3 Temporal Logic

In this section, let $\mathcal{S} = (\Sigma, \Gamma)$ be any signature. We study temporal logics whose modalities are defined in the monadic second-order (MSO) logic over $\mathcal{S}$-graphs, which we recall in the following. We use $x, y, \ldots$ to denote first-order variables which vary over nodes of the graphs, and $X, Y, \ldots$ to denote second-order variables which vary over sets of nodes. The syntax of MSO($\mathcal{S}$) is given by the grammar $\varphi ::= \sigma(x) \mid \gamma(x, y) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$ where σ ranges over Σ, γ ranges over Γ, x and y are first-order variables, and X is a second-order variable. We use $\prec$, the transitive closure of the relations induced by Γ, freely as it can be expressed in MSO($\mathcal{S}$). For an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ and a formula $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ with free variables in $\{x_1, \ldots, x_n, X_1, \ldots, X_m\}$, we write $G \models \varphi(u_1, \ldots, u_n, U_1, \ldots, U_m)$ if φ is evaluated to true when interpreting the variables by $u_1, \ldots, u_n \in V$ and $U_1, \ldots, U_m \subseteq V$, respectively.

For $m \in \mathbb{N} = \{0, 1, 2, \ldots\}$, we call $\varphi \in \text{MSO}(\mathcal{S})$ an *m-ary modality* if its free variables consist of m set variables $X_1, \ldots, X_m$ and one first-order variable x.

A *temporal logic* over $\mathcal{S}$ is given by a triple $\mathcal{L} = (\mathcal{M}, arity, [\![-]\!])$ including a finite set $\mathcal{M}$ of *modality names*, a mapping $arity : \mathcal{M} \to \mathbb{N}$, and a mapping $[\![-]\!] : \mathcal{M} \to \text{MSO}(\mathcal{S})$ such that, for $M \in \mathcal{M}$ with $arity(M) = m$, $[\![M]\!]$ is an m-ary modality. Its syntax, i.e., the set of *formulas* $\varphi \in \text{Form}(\mathcal{L})$ is given by

$$\varphi \quad ::= \quad \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$$

$$\pi \quad ::= \quad ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$$

where σ ranges over Σ, M ranges over $\mathcal{M}$, and γ ranges over Γ. We call φ a *node formula* and π a *path formula* (or *path expression*). Their semantics wrt. an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$ is defined inductively: for subformulas φ, we obtain a set $[\![\varphi]\!]_G \subseteq V$, containing the nodes of G that satisfy φ. Accordingly, $[\![\pi]\!]_G \subseteq V \times V$ is the set of pairs of nodes linked with a path defined by π. Then, $\exists\pi$ is the set of nodes that admit a path following π. Formally, $[\![-]\!]_G$ is given in Fig. 3 where $\otimes \in \{\cup, \cap, \circ\}$ ($\circ$ denotes the product of two relations). We may write $G, u \models \varphi$ if $u \in [\![\varphi]\!]_G$ and $G, u, v \models \pi$ if $(u, v) \in [\![\pi]\!]_G$. We also use $\pi^+ := \pi \circ \pi^*$.

An *intersection free temporal logic* over $\mathcal{S}$ is defined as expected: path expressions do not contain subformulas of the form $\pi_1 \cap \pi_2$. Moreover, a *path-expression free temporal logic* does not contain formulas of the form $\exists\pi$.

Example 1. We consider the path-expression free temporal logic CTL over (Σ, Γ) (interpreted over (Σ, Γ)-trees) [7]. The modalities are $\mathcal{M} = \{\mathsf{EX}, \mathsf{EG}, \mathsf{EU}\}$ with EX and EG being unary and EU being binary. Node formula $\mathsf{EX}\varphi$ holds at a

$$\begin{aligned}
[\![\sigma]\!]_G &:= V_\sigma \qquad [\![\neg\varphi]\!]_G := V \setminus [\![\varphi]\!]_G \qquad [\![\varphi_1 \vee \varphi_2]\!]_G := [\![\varphi_1]\!]_G \cup [\![\varphi_2]\!]_G \\
[\![M(\varphi_1, \ldots, \varphi_m)]\!]_G &:= \{u \in V \mid G \models [\![M]\!](u, [\![\varphi_1]\!]_G, \ldots, [\![\varphi_m]\!]_G)\} \\
[\![\exists\pi]\!]_G &:= \{u \in V \mid \text{there is } v \in V \text{ such that } (u, v) \in [\![\pi]\!]_G\} \\
[\![?\varphi]\!]_G &:= \{(u, u) \mid u \in [\![\varphi]\!]_G\} \qquad [\![\gamma]\!]_G := E_\gamma \qquad [\![\gamma^{-1}]\!]_G := E_\gamma^{-1} \\
[\![\pi \otimes \tau]\!]_G &:= [\![\pi]\!]_G \otimes [\![\tau]\!]_G \qquad [\![\pi^*]\!]_G := [\![\pi]\!]_G^*
\end{aligned}$$

Fig. 3. Semantics of temporal logic

node if there is a child satisfying φ. Thus, $[\![\mathsf{EX}]\!](x, X) = \exists y\,(x \prec y \wedge y \in X)$ where $x \prec y := \bigvee_{\gamma \in \Gamma} \gamma(x, y)$. Formula $\mathsf{EG}\,\varphi$ means that there is an infinite path starting from the current node where φ always holds. Formula $\varphi\,\mathsf{EU}\,\psi$ means that there is a path starting from the current node satisfying φ until ψ:

$$\begin{aligned}
[\![\mathsf{EG}]\!](x, X) &= \exists Y\,(Y \subseteq X \wedge x \in Y \wedge \forall z\,(z \in Y \rightarrow \exists z'\,(z' \in Y \wedge z \prec z'))) \\
[\![\mathsf{EU}]\!](x, X_1, X_2) &= \exists z\,(x \preceq z \wedge z \in X_2 \wedge \forall y\,(x \preceq y \prec z \rightarrow y \in X_1))
\end{aligned}$$

Example 2. Our approach captures various logics over unranked trees (see [17] for an overview). E.g., the intersection free temporal logic $\mathcal{L}_0^-$ with no modalities over ordered unranked trees is precisely regular XPath [6].

Example 3. We give a property over nested traces using a path expression: $\varphi = \neg\exists(\mathtt{cr} \cap (?q \circ (\bigcup_{\gamma \in \Gamma} \gamma)^+ \circ ?(\mathsf{call} \wedge p) \circ (\bigcup_{\gamma \in \Gamma} \gamma)^+))$ means that process p is not allowed to call a new procedure when it is in the scope of an active procedure call from q. The first call node along q in Fig. 1 does not satisfy this property.

Example 4. We present a path-expression free temporal logic over nested traces, $\mathsf{NTrLTL} = (\mathcal{M}, \mathit{arity}, [\![-]\!])$. The unary modalities are $\{\mathsf{X}^{\mathtt{cr}}, \mathsf{Y}^{\mathtt{cr}}\} \cup \{\mathsf{X}_p, \mathsf{Y}_p \mid p \in \mathit{Proc}\}$. Intuitively, $\mathsf{X}_p\,\varphi$ means that φ holds at the next p-position and $\mathsf{X}^{\mathtt{cr}}$ claims that we are at a call position whose return position satisfies φ. The dual *past* modalities are Y_p and $\mathsf{Y}^{\mathtt{cr}}$. The semantics of the *future* modalities is given by

$$\begin{aligned}
[\![\mathsf{X}_p]\!](x, X) &= \exists y\,(p(y) \wedge x \prec y \wedge y \in X \wedge \forall z\,(p(z) \wedge x \prec z \rightarrow y \preceq z)) \\
[\![\mathsf{X}^{\mathtt{cr}}]\!](x, X) &= \exists y\,(\mathtt{cr}(x, y) \wedge y \in X)
\end{aligned}$$

The binary modalities are $\{\mathsf{AU}, \mathsf{AS}, \mathsf{EU}^s, \mathsf{ES}^s\}$. Here, $\varphi\,\mathsf{AU}\,\psi$ means that in the *partial order* G there is a future node satisfying ψ, and φ should hold on all nodes in between: $[\![\mathsf{AU}]\!](x, X_1, X_2) = \exists z\,(x \preceq z \wedge z \in X_2 \wedge \forall y\,(x \preceq y \prec z \rightarrow z \in X_1))$. Modality EU^s refers to the *summary* path in G, which may freely use $\mathtt{cr}$-edges. Formally, the semantics $[\![\mathsf{EU}^s]\!](x, X_1, X_2)$ is defined as

$$\begin{aligned}
\exists z \exists Y\,(z \in X_2 \wedge Y \subseteq X_1 \wedge \forall y\,(y \in Y \vee y = z) \rightarrow (y = x \vee \\
\exists y'\,(y' \in Y \wedge (\mathtt{cr}(y', y) \vee \textstyle\bigvee_{q \in \mathit{Proc}} \mathtt{succ}_q(y', y)))))
\end{aligned}$$

The modalities $\mathsf{AS}, \mathsf{ES}^s$ are the past-time counterparts of $\mathsf{AU}, \mathsf{EU}^s$. When we drop AU, AS and assume $|\mathit{Proc}| = 1$, our logic is precisely NWTL defined in [1].

4 Satisfiability: From Trees to Nested Traces

Consider any signature $\mathcal{S} = (\Sigma, \Gamma)$ and temporal logic $\mathcal{L}$ over $\mathcal{S}$. The following decision problem is well known.

Problem 5. Tree-Sat($\mathcal{L}$):
Given $\varphi \in \mathrm{Form}(\mathcal{L})$, are there $t \in \mathit{Trees}(\mathcal{S})$ and node u of t such that $t, u \models \varphi$?

Theorem 6 ([14,16,11,20]). *Let $\mathcal{L}_0$ be the temporal logic over $\mathcal{S}$ with $\mathcal{M} = \emptyset$. The problem* Tree-Sat($\mathcal{L}_0$) *is* 2ExpTime*-complete [14,16]. For the intersection free fragment $\mathcal{L}_0^-$, the problem* Tree-Sat($\mathcal{L}_0^-$) *is* ExpTime*-complete [11,20].*

We will extend these results to logics $\mathcal{L}$ and $\mathcal{L}^-$ including MSO modalities. For this, we need the notion of an *alternating 2-way tree automaton* (A2A) over $\mathcal{S} = (\Sigma, \Gamma)$ of index $r \in \mathbb{N}$, which is a tuple $\mathcal{A} = (Q, \delta, q_0, \mathit{Acc})$ where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $\mathit{Acc} : Q \to \mathbb{N}$ is a *parity acceptance condition* with $r = \max(\mathit{Acc}(Q))$, and $\delta : Q \times 2^\Sigma \times 2^D \to \mathcal{B}^+(D \times Q)$ is the transition function where $D = \Gamma \cup \{\mathtt{stay}, \mathtt{up}\}$ and $\mathcal{B}^+(D \times Q)$ is the set of positive boolean formulas over $D \times Q$. We only give an intuition of the semantics of A2A and refer to [20,14] for details. An A2A walks in an $\mathcal{S}$-tree $t = (V, \lambda, \nu)$. A configuration is a set of "threads" (q, u) where $q \in Q$ and $u \in V$ is the current node. For every thread (q, u), we have to choose some model $\{(d_1, q_1), \ldots, (d_n, q_n)\}$ of $\delta(q, \lambda(u), D')$ where D' is the set of directions available at u. Then, we replace (q, u) with n new threads (q_i, u_i) for $1 \leq i \leq n$ where u_i is obtained from u by following direction d_i (if $d_i = \mathtt{stay}$, then $u_i = u$). The parity acceptance condition has to be applied to all infinite paths when we consider the run as a tree, threads (q_i, u_i) being the children of (q, u). For $u \in V$, a *run* over (t, u) is a run that starts in the single configuration (q_0, u). The *semantics* $[\![\mathcal{A}]\!]_t$ contains all nodes u of t such that there is an accepting run of $\mathcal{A}$ over (t, u).

Theorem 7 ([21]). *Given an A2A $\mathcal{A}$ of index r with n states, one can check in time exponential in $n \cdot r$ if there is a tree t such that $[\![\mathcal{A}]\!]_t \neq \emptyset$.*

The main ingredient of the proof of Theorem 6 is the construction of an A2A from a given formula, whose existence is given by the following lemma. Using the lemma and Theorem 7, we can then extend Theorem 6 towards Theorem 9.

Lemma 8 ([14]). *Consider the temporal logic $\mathcal{L}_0$ over $\mathcal{S}$ with $\mathcal{M} = \emptyset$. For every formula $\varphi \in \mathrm{Form}(\mathcal{L}_0)$, we can construct an A2A $\mathcal{B}_\varphi$ over $\mathcal{S}$ of exponential size such that, for all $\mathcal{S}$-trees t, we have $[\![\varphi]\!]_t = [\![\mathcal{B}_\varphi]\!]_t$. Moreover, if $\varphi \in \mathrm{Form}(\mathcal{L}_0^-)$ is intersection free, then $\mathcal{B}_\varphi$ is of polynomial size.*

Theorem 9. *For any temporal logic $\mathcal{L}$,* Tree-Sat($\mathcal{L}$) *is* 2ExpTime*-complete. For the intersection free fragment $\mathcal{L}^-$,* Tree-Sat($\mathcal{L}^-$) *is* ExpTime*-complete.*

Proof. The lower bounds follow from Theorem 6. We show the upper bounds. Let φ be any $\mathcal{L}$ formula. Let $\mathrm{Subf}(\varphi)$ denote the set of subformulas of φ and let $\mathrm{top}(\xi)$ denote the topmost symbol of $\xi \in \mathrm{Subf}(\varphi)$ which could be $\exists$ or a modality

$M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$: below, we treat atomic propositions $\sigma \in \Sigma$, negation $\neg$, and disjunction $\vee$ as modalities of arities 0, 1, and 2 resp.

For each modality $M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$ of arity m, we define an MSO($\mathcal{S}$) formula ψ_M with free variables $X_0, \ldots, X_m$ by $\psi_M(X_0, X_1, \ldots, X_m) := \forall x\,(x \in X_0 \longleftrightarrow [\![M]\!](x, X_1, \ldots, X_m))$. Let $\mathcal{S}_m = (\Sigma \cup \{X_0, \ldots, X_m\}, \Gamma)$ so that the node labeling encodes the valuations of the free set variables as usual. By Rabin's theorem, there is a non-deterministic (N1A) tree automaton $\mathcal{A}_M$ recognizing all $\mathcal{S}_m$-trees satisfying ψ_M. Note that $\mathcal{A}_M$ for $M \in \Sigma \cup \{\neg, \vee\}$ has only one state.

Let $\exists\pi(\xi_1, \ldots, \xi_m) \in \mathrm{Subf}(\varphi)$ where $\xi_1, \ldots, \xi_m$ are the node formulas checked in path π. Replacing $\xi_1, \ldots, \xi_m$ by set variables $X_1, \ldots, X_m$ (or new predicates) we will construct using Lemma 8 an A2A $\mathcal{A}_{\exists\pi}$ accepting all $\mathcal{S}_m$-trees satisfying the "formula" $\psi_{\exists\pi}(X_0, X_1, \ldots, X_m) := \forall x\,(x \in X_0 \longleftrightarrow \exists\pi(X_1, \ldots, X_m))$. By Lemma 8, we can construct automata $\mathcal{B}_1$ and $\mathcal{B}_2$ for $\exists\pi(X_1, \ldots, X_m)$ and $\neg\exists\pi(X_1, \ldots, X_m)$, resp., which are $\mathcal{L}_0$ formulas. Let ι_1 and ι_2 be the initial states of $\mathcal{B}_1$ and $\mathcal{B}_2$. The automaton $\mathcal{A}_{\exists\pi}$ includes the disjoint union of $\mathcal{B}_1$ and $\mathcal{B}_2$ plus a new initial state ι and, for $\sigma \subseteq \Sigma \cup \{X_0, \ldots, X_m\}$ and $D' \subseteq D$, the transition $\delta(\iota, \sigma, D') = \bigwedge_{\gamma \in \Gamma}(\gamma, \iota) \wedge (\mathtt{stay}, \theta)$ where $\theta = \iota_1$ if $X_0 \in \sigma$, and $\theta = \iota_2$ otherwise. By Lemma 8, the size of $\mathcal{A}_{\exists\pi}$ is exponential (resp. polynomial) in the size of $\pi(X_1, \ldots, X_m)$ (resp. if this path expression is intersection free).

The final automaton $\mathcal{A}$ runs over $\mathcal{S}_\varphi$-trees t where $\mathcal{S}_\varphi = (\Sigma \cup \mathrm{Subf}(\varphi), \Gamma)$, i.e., the node labeling includes the (guessed) truth values for $\mathrm{Subf}(\varphi)$. To check that these guesses are correct, $\mathcal{A}$ runs an automaton $\mathcal{A}_\xi$ for each $\xi \in \mathrm{Subf}(\varphi)$.

For each $\xi_0 = M(\xi_1, \ldots, \xi_m) \in \mathrm{Subf}(\varphi)$ with $M \in \mathcal{M} \cup \Sigma \cup \{\neg, \vee\}$, we define an automaton $\mathcal{A}_{\xi_0}$ over $\mathcal{S}_\varphi$-trees by taking a copy of $\mathcal{A}_M$ which reads a label $\sigma \subseteq \Sigma \cup \mathrm{Subf}(\varphi)$ of t as if it was $\sigma \cap (\Sigma \cup \{\xi_0, \ldots, \xi_m\})$ with ξ_i further replaced by X_i. Similarly, for each $\xi_0 = \exists\pi(\xi_1, \ldots, \xi_m) \in \mathrm{Subf}(\varphi)$, we define an automaton $\mathcal{A}_{\xi_0}$ over $\mathcal{S}_\varphi$-trees by taking a copy of $\mathcal{A}_{\exists\pi}$ which reads a label $\sigma \subseteq \Sigma \cup \mathrm{Subf}(\varphi)$ of t as above.

Finally, $\mathcal{A}$ is the disjoint union of all $\mathcal{A}_\xi$ for $\xi \in \mathrm{Subf}(\varphi)$ together with a new initial state ι which starts all the automata $\mathcal{A}_\xi$ with the initial transitions $\delta(\iota, \sigma, D') = \bigwedge_{\xi \in \mathrm{Subf}(\varphi)}(\mathtt{stay}, \iota_\xi)$ for all $D' \subseteq D$. We can check that an $\mathcal{S}_\varphi$-tree $t = (V, \lambda, \nu)$ is accepted by $\mathcal{A}$ iff its projection $t' = (V, \lambda', \nu)$ on Σ is an $\mathcal{S}$-tree and for each node $u \in V$ we have $\lambda(u) \setminus \Sigma = \{\xi \in \mathrm{Subf}(\varphi) \mid t', u \models \xi\}$. Therefore, satisfiability of φ over $\mathcal{S}$-trees is reduced to emptiness of the conjunction of $\mathcal{A}$ with a two state automaton checking that $\varphi \in \lambda(u)$ for some node u of the tree.

The size of $\mathcal{A}$ is at most exponential (resp. polynomial) in the size of φ. Indeed, each $\mathcal{A}_\xi$ with $\mathrm{top}(\xi) \neq \exists$ is of constant size since the MSO modalities are fixed and not part of the input. If $\xi = \exists\pi(\xi_1, \ldots, \xi_m)$ then the size of $\mathcal{A}_\xi$ is exponential in $|\pi(X_1, \ldots, X_m)|$ (note that ξ_i is replaced by X_i so that its size does not influence the size of $\mathcal{A}_\xi$). Moreover, if π is intersection free then the size of $\mathcal{A}_\xi$ is polynomial in $|\pi(X_1, \ldots, X_m)|$. We deduce from Theorem 7 the 2EXPTIME upper bound for Tree-Sat($\mathcal{L}$) and the EXPTIME upper bound for Tree-Sat($\mathcal{L}^-$), the intersection free case. □

From Ordered Unranked Trees to Binary Trees. We recall that an ordered unranked tree can be encoded as a binary tree by removing the edges

$(u, v) \in E_{\texttt{child}}$ whenever v is not a first-child. Note that $E_{\texttt{child}}$ can be retrieved from the binary encoding by the path expression $\texttt{child} \circ \texttt{next}^*$. Hence any path expression over ordered unranked trees can be converted to a path expression over binary trees (with only a linear blowup in the size), and any MSO($\mathcal{S}$)-formula over ordered unranked trees can be translated to an MSO($\mathcal{S}$)-formula over binary trees. Thus, Theorem 9 holds for ordered unranked trees as well:

Problem 10. O-U-Tree-Sat($\mathcal{L}$):
Given $\varphi \in \mathrm{Form}(\mathcal{L})$, are there $t \in o.u.\mathit{Trees}(\mathcal{S})$ and node u such that $t, u \models \varphi$?

Theorem 11. O-U-Tree-Sat($\mathcal{L}$) *is* 2EXPTIME*-complete. For the intersection free fragment* $\mathcal{L}^-$*, the problem* O-U-Tree-Sat($\mathcal{L}^-$) *is* EXPTIME*-complete.*

The 2EXPTIME lower bound follows from [16]. The EXPTIME lower bound is inherited from regular XPath [6] (cf. Example 2).

From Nested Traces to Trees. Now, we transform a temporal logic over nested traces into a temporal logic over their tree encodings that "simulates" the original logic. This allows us to solve the following problem, which is parametrized by *Proc*, *Act*, $k \geq 1$, and a temporal logic $\mathcal{L}$ over the induced signature:

Problem 12. Nested-Trace-Sat($\mathcal{L}, k$): Given $\varphi \in \mathrm{Form}(\mathcal{L})$, are there $G \in \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$ and node u such that $G, u \models \varphi$?

Theorem 13. Nested-Trace-Sat($\mathcal{L}, k$) *is in* 2EXPTIME*. For the intersection free fragment* $\mathcal{L}^-$*,* Nested-Trace-Sat($\mathcal{L}^-, k$) *is* EXPTIME*-complete.*

The proof of Theorem 13 will be developed in the following. In order to exploit Theorem 9, we interpret a k-phase nested trace $G = (V, \lambda, \nu)$ in a (binary) $\mathcal{S}'$-tree (where $\mathcal{S}' := (\Sigma \uplus \{1, \ldots, k\}, \{\texttt{left}, \texttt{right}\})$) using the encoding from [15], extended to infinite trees. Actually, [15] does not consider nested traces but k-phase *words*. Therefore, we will use linearizations of nested traces. Let $w = (V, \lambda, \leq) \in \mathit{Lin}_k(G)$. By $\lessdot$, we denote the direct successor relation of $\leq$. Suppose that $V = \{u_0, u_1, u_2, \ldots\}$ and that $u_0 \lessdot u_1 \lessdot u_2 \lessdot \ldots$ is the corresponding total order. For $0 \leq i < |V|$, we let $\mathit{phase}_w(u_i) = \min\{j \in \{1, \ldots, k\} \mid \lambda(u_0) \ldots \lambda(u_i)$ is a j-phase word$\}$. Intuitively, this provides a "tight" factorization of w. We associate with w the $\mathcal{S}'$-tree $t_k^w = (V, \lambda', \nu')$ where the node labeling is given by $\lambda'(u_i) = \lambda(u_i) \cup \{\mathit{phase}_w(u_i)\}$ and the sets of edges are defined by $E'_{\texttt{right}} = E_{\texttt{cr}}$ and $E'_{\texttt{left}} = \lessdot \setminus \{(u, v) \in \lessdot \mid$ there is u' such that $(u', v) \in E_{\texttt{cr}}\}$. That is, the tree encoding is obtained from the linearization by adding the cr-edges as right children and removing the superfluous linear edges to return nodes having a matching call. Figure 4 depicts the tree t_2^w for the linearization w that was illustrated in Fig. 2. The edges removed from the linearization are shown in dotted lines. The newly added edges are labelled right. All $\Box$-nodes are phase 1 and the $\bigcirc$-nodes are phase 2.

By $\mathit{Trees}_k(\mathit{Proc}, \mathit{Act})$, we denote the set $\{t_k^w \mid w \in \mathit{Lin}_k(G)$ for some $G \in \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})\}$ of *valid* tree encodings. The following was proved in [15] for finite structures, and extends easily to infinite structures.

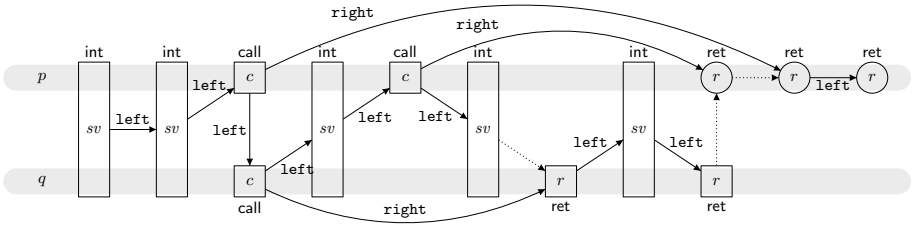


Fig. 4. The tree encoding of a 2-phase linearization

Lemma 14 ([15]). *There is a formula* $\mathsf{TreeEnc}_k \in \mathrm{MSO}(\mathcal{S}')$ *defining the set* $Trees_k(Proc, Act)$. *Also, there is* $\mathsf{less}_k(x, y) \in \mathrm{MSO}(\mathcal{S}')$ *such that for all* k*-phase words* $w = (V, \lambda, \leq)$ *and all* $u, v \in V$, *we have* $u < v$ *in* w *iff* $t_k^w \models \mathsf{less}_k(u, v)$.

Lemma 14 will be used to reduce nested-trace modalities to tree modalities in the proof of Theorem 13. We also need to deal with path expressions:

Lemma 15. *There exists a path expression* $\mathtt{succ}_{\leq k}$ *over* $\mathcal{S}'$ *such that, for all* k*-phase linearizations* $w = (V, \lambda, \leq)$, *we have* $[\![\mathtt{succ}_{\leq k}]\!]_{t_k^w} = \{(u, v) \in V^2 \mid u \lessdot v\}$. *Moreover, the length of* $\mathtt{succ}_{\leq k}$ *is exponential in* k.

Proof (of Theorem 13). Let $\mathcal{S} = (\Sigma, \Gamma)$ be the signature induced by *Proc* and *Act*, and let $\mathcal{L} = (\mathcal{M}, arity, [\![-]\!])$ be the considered temporal logic over nested traces. For $\mathcal{S}' = (\Sigma \uplus \{1, \ldots, k\}, \{\mathtt{left}, \mathtt{right}\})$, we define a new temporal logic $\mathcal{L}' = (\mathcal{M}', arity', [\![-]\!]')$ over $\mathcal{S}'$-trees and give an inductive, linear-time computable translation T of formulas over $\mathcal{L}$ to "equivalent" formulas over $\mathcal{L}'$. By "equivalent", we mean that for all $G \in Traces_k(Proc, Act)$ and all k-phase linearizations w of G, we have $[\![\varphi]\!]_G = [\![T(\varphi)]\!]_{t_k^w}$ for each node formula φ over $\mathcal{L}$ and $[\![\pi]\!]_G = [\![T(\pi)]\!]_{t_k^w}$ for each path formula π over $\mathcal{L}$.

We set $\mathcal{M}' = \mathcal{M} \cup \{\mathsf{Enc}\}$ where Enc is a new modality with $arity'(\mathsf{Enc}) = 0$ that characterizes valid tree encodings: the semantics $[\![\mathsf{Enc}]\!]'$ is given by the formula $\mathsf{TreeEnc}_k$ from Lemma 14. We also change the semantics of the modalities from $\mathcal{M}$: for each $M \in \mathcal{M}$, the new semantics $[\![M]\!]' \in \mathrm{MSO}(\mathcal{S}')$ is obtained from $[\![M]\!] \in \mathrm{MSO}(\mathcal{S})$ by replacing each occurrence of $\mathtt{cr}(x, y)$ by $\mathtt{right}(x, y)$ and each occurrence of $\mathtt{succ}_p(x, y)$ by $p(x) \wedge \mathsf{less}_k(x, y) \wedge p(y) \wedge \neg\exists z\, (\mathsf{less}_k(x, z) \wedge p(z) \wedge \mathsf{less}_k(z, y))$ where less_k is the formula from Lemma 14. Note that these transformations only depend on $\mathcal{L}$ and on k (which are not part of the input) and not on the formula for which we want to check satisfiability.

The translation T from formulas over $\mathcal{L}$ to "equivalent" formulas over $\mathcal{L}'$ is defined inductively for node formulas by $T(\sigma) = \sigma$, $T(M(\varphi_1, \ldots, \varphi_m)) = M(T(\varphi_1), \ldots, T(\varphi_m))$, etc., and for path formulas by $T(?\varphi) = ?T(\varphi)$, $T(\mathtt{cr}) = \mathtt{right}$, $T(\mathtt{cr}^{-1}) = \mathtt{right}^{-1}$, $T(\mathtt{succ}_p) = ?p \circ \mathtt{succ}_{\leq k} \circ (?\neg p \circ \mathtt{succ}_{\leq k})^* \circ ?p$, and $T(\mathtt{succ}_p^{-1}) = ?p \circ \mathtt{succ}_{\leq k}^{-1} \circ (?\neg p \circ \mathtt{succ}_{\leq k}^{-1})^* \circ ?p$. The other cases are straightforward. Here, $\mathtt{succ}_{\leq k}$ is defined in Lemma 15. Note that the transformation $T(\pi)$ of a path formula π is linear in $|\pi|$ since k is not part of the input.

Finally, a formula $\varphi \in \mathrm{Form}(\mathcal{L})$ is satisfiable over k-phase nested traces iff the formula $\mathsf{Enc} \wedge T(\varphi) \in \mathrm{Form}(\mathcal{L}')$ is satisfiable over $\mathcal{S}'$-trees. Using Theorem 9 we get the upper bounds stated in Theorem 13. To obtain the lower bound, one can show that NESTED-TRACE-SAT$(\mathcal{L}^-, k)$ is EXPTIME-hard. This is done by a reduction from the EXPTIME-complete logic NWTL [1] to the temporal logic $\mathcal{L}_0^-$ with no modalities and only one process. □

Remark 16. If k is given as part of the input, the above method for modalities does not work: the new semantics $[\![M]\!]'$ over trees depend on k and are no more fixed and independent of the input. However, if we consider the fragment $\mathcal{L}_0$ with no MSO modalities, we get a 3EXPTIME procedure even if k is part of the input since the length of the path expression $T(\pi)$ is linear in $|\pi|$ and exponential in k. Moreover, for the intersection free fragment $\mathcal{L}_0^-$, we get a 2EXPTIME procedure.

5 Model Checking

Our approach extends to model checking. We can define a model of concurrent recursive programs, called concurrent recursive Kripke structures (CRK), that generates nested traces. It is similar to the model from [5].

Definition 17. *A* concurrent recursive Kripke structure *(CRK) over finite sets Proc and Act is a tuple* $\mathcal{K} = ((S_p)_{p \in Proc}, \Delta, \iota)$. *The* S_p *are disjoint finite sets of* local states *(*S_p *containing the local states of process* p*). Given a set* $P \subseteq Proc$*, we let* $S_P := \prod_{p \in P} S_p$. *The tuple* $\iota \in S_{Proc}$ *is a global* initial state. *Finally,* Δ *provides the transitions, which are divided into four sets:* $\Delta = (\Delta_{\mathsf{call}}, \Delta^1_{\mathsf{ret}}, \Delta^2_{\mathsf{ret}}, \Delta_{\mathsf{int}})$ *where* $\Delta_{\mathsf{call}} \subseteq \bigcup_{p \in Proc}(S_p \times Act \times S_p)$, $\Delta^1_{\mathsf{ret}} \subseteq \bigcup_{p \in Proc}(S_p \times Act \times S_p)$, $\Delta^2_{\mathsf{ret}} \subseteq \bigcup_{p \in Proc}(S_p \times S_p \times Act \times S_p)$, *and* $\Delta_{\mathsf{int}} \subseteq \bigcup_{P \subseteq Proc}(S_P \times Act \times S_P)$.

Let $\mathcal{S} = \bigcup_{P \subseteq Proc} S_P$. For $s \in \mathcal{S}$ and $p \in Proc$, we let s_p be the p-th component of s (if it exists). A run of the CRK $\mathcal{K}$ is an $\mathcal{S}'$-graph $G = (V, \lambda, \nu)$ where $\mathcal{S}' = (\Sigma \uplus \biguplus_{p \in Proc} S_p, \Gamma)$ with $\Sigma = Proc \cup Act \cup Type$ and $\Gamma = \{\mathtt{cr}\} \cup \{\mathtt{succ}_p \mid p \in Proc\}$, and the following conditions hold:

- The graph G without the labelings from $\bigcup_{p \in Proc} S_p$ is a nested trace, i.e., $nt(G) := (V, \lambda', \nu)$, with $\lambda'(u) = \lambda(u) \cap \Sigma$, is contained in $Traces(Proc, Act)$.
- Every node u is labeled with one, and only one, state from S_p for each process $p \in Proc(u)$. This state is denoted $\rho(u)_p$. The label of a node u does not contain any state from S_p if $p \notin Proc(u)$. That is, for all $p \in Proc$ and all $u \in V$, $\lambda(u) \cap S_p = \{\rho(u)_p\}$ if $p \in \lambda(u)$, and $\lambda(u) \cap S_p = \emptyset$ otherwise. This defines a mapping $\rho : V \to \mathcal{S}$ by $\rho(u) = (\rho(u)_p)_{p \in Proc(u)}$.
- Let us determine another mapping $\rho^- : V \to \mathcal{S}$ as follows: for $u \in V$, we let $\rho^-(u) = (\rho^-(u)_p)_{p \in Proc(u)}$ where $\rho^-(u)_p = \rho(u')_p$ if $(u', u) \in E_{\mathtt{succ}_p}$, and $\rho^-(u)_p = \iota_p$ if there is no u' such that $(u', u) \in E_{\mathtt{succ}_p}$. The following hold, for every $u, u' \in V$ and $a \in Act$:
  - $(\rho^-(u), a, \rho(u)) \in \Delta_{\mathsf{call}}$ if $u \in V_{\mathsf{call}} \cap V_a$
  - $(\rho^-(u), a, \rho(u)) \in \Delta^1_{\mathsf{ret}}$ if $u \in V_{\mathsf{ret}} \cap V_a$ and there is no v with $(v, u) \in E_{\mathtt{cr}}$

- $(\rho(u'), \rho^-(u), a, \rho(u)) \in \Delta^2_{\mathsf{ret}}$ if $u \in V_{\mathsf{ret}} \cap V_a$ and $(u', u) \in E_{\mathsf{cr}}$
- $(\rho^-(u), a, \rho(u)) \in \Delta_{\mathsf{int}}$ if $u \in V_{\mathsf{int}} \cap V_a$

We are only interested in maximal runs. We say that a run G of a CRK $\mathcal{K}$ is *maximal* if G is not a strict prefix of another run of $\mathcal{K}$. The *language* $L(\mathcal{K})$ of $\mathcal{K}$ is the set $\{nt(G) \mid G \text{ is a maximal run of } \mathcal{K}\}$. By $L_k(\mathcal{K})$, we denote its restriction $L(\mathcal{K}) \cap \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$ to k-phase nested traces.

Let *Proc* and *Act* be non-empty finite sets inducing signature $\mathcal{S}$, let $k \geq 1$ and $\mathcal{L}$ be a temporal logic over $\mathcal{S}$. We consider the following decision problem.

Problem 18. Model-Checking$(\mathcal{L}, k)$: Given CRK $\mathcal{K}$ and $\varphi \in \mathrm{Form}(\mathcal{L})$, do we have $\mathcal{K} \models_k \varphi$, i.e., for all $G \in L_k(\mathcal{K})$, is there a node u of G such that $G, u \models \varphi$?

Theorem 19. *The problem* Model-Checking$(\mathcal{L}, k)$ *is in* 2EXPTIME. *For the intersection free fragment* $\mathcal{L}^-$, Model-Checking$(\mathcal{L}^-, k)$ *in* EXPTIME.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. Log. Meth. Comput. Sci. 4(4) (2008)
2. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM 56, 16:1–16:43 (2009)
4. Atig, M.F.: Global Model Checking of Ordered Multi-Pushdown Systems. In: FSTTCS 2010, vol. 8, pp. 216–227 (2010)
5. Bollig, B., Grindei, M.-L., Habermehl, P.: Realizability of concurrent recursive programs. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 410–424. Springer, Heidelberg (2009)
6. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: An Automata-Theoretic Approach to Regular XPath. In: Gardner, P., Geerts, F. (eds.) DBPL 2009. LNCS, vol. 5708, pp. 18–35. Springer, Heidelberg (2009)
7. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs, pp. 52–71 (1981)
8. Dax, C., Klaedtke, F.: Alternation elimination for automata over nested words. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 168–183. Springer, Heidelberg (2011)
9. Diekert, V., Gastin, P.: Pure future local temporal logics are expressively complete for Mazurkiewicz traces. Information and Computation 204(11), 1597–1619 (2006)
10. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific, Singapore (1995)
11. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18(2), 194–211 (1979)
12. Gastin, P., Kuske, D.: Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 222–236. Springer, Heidelberg (2003)
13. Gastin, P., Kuske, D.: Uniform satisfiability problem for local temporal logics over Mazurkiewicz traces. Information and Computation 208(7), 797–816 (2010)

14. Göller, S., Lohrey, M., Lutz, C.: PDL with intersection and converse: satisfiability and infinite-state model checking. J. Symb. Log. 74(1), 279–314 (2009)
15. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS 2007, pp. 161–170. IEEE Computer Society Press, Los Alamitos (2007)
16. Lange, M., Lutz, C.: 2-ExpTime lower bounds for Propositional Dynamic Logics with Intersection. J. Symb. Log. 70(5), 1072–1086 (2005)
17. Libkin, L.: Logics for Unranked Trees: An Overview. Log. Meth. Comput. Sci. 2(3) (2006)
18. Pnueli, A.: The temporal logic of programs. In: Proceedings of FOCS 1977, pp. 46–57. IEEE, Los Alamitos (1977)
19. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
20. Vardi, M.Y.: The taming of converse: Reasoning about two-way computations. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 413–423. Springer, Heidelberg (1985)
21. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)

The Reachability Problem for Vector Addition System with One Zero-Test*

Rémi Bonnet

LSV, ENS Cachan, CNRS
remi.bonnet@lsv.ens-cachan.fr

Abstract. We consider here a variation of Vector Addition Systems where one counter can be tested for zero, extending the reachability proof by Leroux for Vector Addition System to our model. This provides an alternate, and hopefully simpler to understand, proof of the reachability problem that was originally proved by Reinhardt.

1 Introduction

Context Petri Nets, Vector Addition Systems (VAS) and Vector Addition System with control states (VASS) are equivalent well known classes of counter systems for which the reachability problem is decidable ([11], [7], [10]). If we add to VAS the ability to test at least two counters for zero, one obtains a model equivalent to Minsky machines, for which all nontrivial properties are undecidable. The study of VAS with a *single* zero-test transition (VAS_0) began recently, and already a reasonable number of results are known for this model. Reinhardt [13] has shown that the reachability problem is decidable for VAS_0 (as well as for hierarchical zero-tests). Abdulla and Mayr have shown that the coverability problem is decidable in [1] by using the backward procedure of Well Structured Transition Systems [2]. The boundedness problem (whether the reachability set is finite), the termination and the reversal-boundedness problem (whether the counters can alternate infinitely often between the increasing and the decreasing modes) are all decidable by using a forward procedure, a finite but *non-complete* Karp and Miller tree provided by Finkel and Sangnier in [5]. The decidability of the place-boundedness problem, and more generally the possibility to compute a finite representation of the downward closure of the reachability set have been shown by Bonnet, Finkel, Leroux and Zeitoun in [4] using the notion of *productive sequences*.

The reachability problem. The decidability of reachability for VAS was originally solved by Mayr (1981, [11]) and Kosaraju (1982, [7]). Lambert later simplified these proofs (1992, [8]) while still using the same proof techniques. Recently, Leroux gave another way to prove this problem, by using Presburger invariants and productive sequences ([9], [10]).

* Supported by the Agence Nationale de la Recherche, AVERISS (grant ANR-06-SETIN-001).

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 145–157, 2011.

The history of the reachability problem for VAS_0 is shorter. The only proofs are the different versions of Reinhardt proof (original unpublished manuscript in 1995 [12], then published in 2008 [13]), which is based on showing that any expression representing a reachability problem can be put in a "normal form" for which satisfiability is easy to solve. However, the definition of the normal form is complex, and the proof of termination of the algorithm reducing any expression to the normal form is difficult to understand. Since this publication, some new results were found by reduction to reachability in VAS_0, for example decidability of minimal cost reachability in the Priced Timed Petri Nets of Abdulla and Mayr [1], and the decidability of reachability in a restricted class of pushdown counter automatas by Atig and Ganty [6].

Our contribution. We propose here an alternate proof of reachability in VAS_0, using the principles Leroux introduced in [10]. The similarity between our proof with Leroux' proof hopefully makes it easier to understand.

2 Preliminaries

Sets. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{Q}_{\geq 0}$ refers respectively to the sets of non-negative integers, integers, rationals and non-negative rationals. If X is a set, a vector of X^d is written $(x(1), ..., x(d))$ with the vector containing only 0's written 0^d. We define addition for $X, Y \subseteq \mathbb{Q}^d$ by $X + Y = \{x + y \mid x \in X, y \in Y\}$ and multiplication for $X \subseteq \mathbb{Q}^d$, $K \subseteq \mathbb{Q}$, $K * X = \{k * x \mid x \in X, k \in K\}$. $K * X$ will be shortened as KX when possible. We also define $k \star X$ ($k \in \mathbb{N}$) by $0 \star X = \{0^d\}$ and $(k+1) \star X = X + (k \star X)$ and we generalize this notation to $K \subseteq \mathbb{N}$ by $K \star X = \bigcup_{k \in K} (k \star X)$. A function f from $\mathbb{N}^d$ (resp. $\mathbb{Q}^d_{\geq 0}$) to $\mathbb{N}^{d'}$ (resp. $\mathbb{Q}^d_{\geq 0}$) is *linear* if $f(x + y) = f(x) + f(y)$ and for $k \in \mathbb{N}$ (resp. $k \in \mathbb{Q}_{\geq 0}$), $f(k * x) = k * f(x)$. We will allow ourselves to shorten the singleton $\{x\}$ as x when the risk of confusion is low. $X \subseteq \mathbb{Q}^d$ is a *vector space* if $\mathbb{Q}X \subseteq X$ and $X + X \subseteq X$. Finally, we define $\mathbb{N}^d_0 = \{0\} \times \mathbb{N}^{d-1}$.

A set $P \subseteq \mathbb{Q}^d$ is *periodic* if $P + P \subseteq P$. A set $X \subseteq \mathbb{N}^d$ is a *finitely generated periodic set* if there exists $\{x_1, \ldots, x_n\} \subseteq X$, $X = \mathbb{N}x_1 + \mathbb{N}x_2 + \cdots + \mathbb{N}x_n$. A semilinear set (also called Presburger set) is a finite union of sets $b_i + X_i$ where $b_i \in \mathbb{N}^d$ and $X_i \subseteq \mathbb{N}^d$ is a finitely generated periodic set.

Relations. A relation on X is a set $R \subseteq X \times X$. We will write $x\ R\ y$ to mean $(x, y) \in R$. Composition of relations on X is defined by $R \circ R' = \{(x, y) \in X \times X \mid \exists z \in X,\ (x, z) \in R \wedge (z, y) \in R'\}$. We shorten $R \circ R'$ as RR' when there is no ambiguousity. R^* is the transitive closure of R. For R a relation on X and $X' \subseteq X$, we define $R(X') = \{y \in X \mid \exists x \in X', (x, y) \in R\}$. A set $X' \subseteq X$ is a *R-forward invariant* if $R(X') \subseteq X'$. We define R^{-1} by $R^{-1} = \{(x, y) \in X \times X \mid (y, x) \in R\}$. A set $X' \subseteq X$ is a *R-backward invariant* if it is a R^{-1}-forward invariant. Similarly, for f a function from X to Y, we define $f(X') = \{y \in Y \mid \exists x \in X', y = f(x)\}$.

Words, Parikh Images. Given a set X, the set of words over X is written X^*. A word $w \in X^*$ is written $a_1 a_2 \dots a_n$ with $a_i \in X$ or optionally $\prod_{1 \leq i \leq n} a_i$. A language L is a subset of X^*. The concatenation of two words $w_1, w_2 \in X^*$ is written $w_1 w_2$ and we extend this notation to languages by $LL' = \{uv \mid u \in L, v \in L'\}$. $\mathbb{N}^X$ is the set of functions from X to $\mathbb{N}$. For $u \in X^*$, the *parikh image* $|u| \in \mathbb{N}^X$ is defined by $|u|(x) =$ 'number of x's in u'.

Orders, Well-orders. An ordering $\preceq$ on a set X is a transitive, reflexive and antisymmetric relation on X. The relation $\prec$ is defined by $x \prec y$ iff $x \preceq y$ and $x \neq y$. An element $x \in X$ is *minimal* if there exists no $x' \in X$, $x' \prec x$. $\preceq$ is a well-order on X if for all sequences $(x_i)_{i \in \mathbb{N}}$ with $x_i \in X$, there exists $i < j$ with $x_i \preceq x_j$. If X is well-ordered, then all subsets of X have a finite number of minimal elements. Common well-orders are $\leq$ on $\mathbb{N}$ and $\preceq$ on $X \times Y$ when X is well-ordered by $\preceq_X$, Y is well-ordered by $\preceq_Y$ and $(x, y) \preceq (x', y') \iff x \preceq_X x' \wedge y \preceq_Y y'$. Hence, if X is well-ordered by $\preceq$, X^d is also well-ordered by the component-wise ordering, that we will also write $\preceq$.

Word embedding, Higman lemma. If X is ordered by $\preceq$, we define $\preceq^{emb}$ (the *word embedding* order) on X^* by $a_i \dots a_n \preceq^{emb} b_1 \dots b_p$ if there exists a strictly increasing function φ from $\{1, \dots, n\}$ to $\{1, \dots, p\}$ such that $\forall i \in \{1, \dots, n\}$, $a_i \preceq b_{\varphi(i)}$. If $\preceq$ is a well-order on X, then $\preceq^{emb}$ is a well-order on X^* (Higman's lemma).

3 Vector Addition Systems with One Zero-Test

3.1 Transition Systems

Definition 1. *A* Labelled Transition System *(shortly: LTS) $\mathcal{S}$ is a tuple $\langle X, A, \rightarrow \rangle$ where X is the set of states, A is a set of transition labels and $\rightarrow \subseteq X \times A \times X$ is the transition relation.*

We write $x \xrightarrow{a} x'$ if $(x, a, x') \in \rightarrow$, and we extend this notation to words of A^* by $x \xrightarrow{\epsilon} x$ and $x \xrightarrow{uv} x'$ if there exists $x'' \in X$, $x \xrightarrow{u} x'' \xrightarrow{v} x'$. If $L \subseteq A^*$, we define $x \xrightarrow{L} y \iff \exists u \in L, x \xrightarrow{u} y$ and we shorten $x \xrightarrow{A^*} y$ as $x \xrightarrow{*} y$. A transition sequence $u \in A^*$ is said *fireable* from $x \in X$ if there exists $y \in X$ such that $x \xrightarrow{u} y$.

3.2 Vector Addition Systems

Definition 2. *A Vector Addition System (shortly: VAS) is a pair $\langle A, \delta \rangle$ where A is a set of transition labels and δ a function from A to $\mathbb{Z}^d$. d is called the dimension of the VAS.*

A Vector Addition System $\mathcal{V} = \langle A, \delta \rangle$ induces a transition system $ts(\mathcal{V}) = \langle \mathbb{N}^d, A, \rightarrow \rangle$ where $\rightarrow$ is defined by:

$$x \xrightarrow{a} y \iff y = x + \delta(a)$$

Reachability is already known to be decidable for VAS:

Theorem 1. *([11], [7], [10]) If X and Y are Presburger sets and $\mathcal{V}$ a VAS, one can decide whether $\{(x,y) \in X \times Y \mid x \xrightarrow{*}_{\mathcal{V}} y\}$ is empty.*

Definition 3. *A Vector Addition System with one zero-test (shortly: VAS$_0$) is a tuple $\langle A_z, \delta, a_z\rangle$ where (A_z, δ) is a VAS and $a_z \in A_z$ is the special zero-test transition.*

$\mathcal{V}_z = \langle A_z, \delta, a_z\rangle$ induces a transition system $ts(\mathcal{V}_z) = \langle \mathbb{N}^d, A_z, \rightarrow\rangle$ where $\rightarrow$ is defined by:

$$x \xrightarrow{a} y \iff y = x + \delta(a) \qquad a \neq a_z$$
$$x \xrightarrow{a_z} y \iff \begin{cases} y = x + \delta(a_z) \\ x(1) = 0 \end{cases}$$

The function δ is extended to parikh images by, for $v \in \mathbb{N}^{A_z}$, $\delta(v) = \sum_{a \in A_z} \delta(v(a))$ and to words by, for $u \in A_z^*$, $\delta(u) = \delta(|u|)$. This means that $x \xrightarrow{u} y \implies y = x + \delta(u)$.

A VAS$_0$ is partially monotonic (the proof is by an easy induction):

Proposition 1. *Let $x, y \in \mathbb{N}^d$ with $x \leq y$ and $x(1) = y(1)$. Then, if a transition sequence $u \in A_z^*$ is fireable from x, u is fireable from y.*

4 Proof Structure

Let us try to summarize the proof structure of [10], that we will mimic. The main idea is that if a relation has some properties, one can find a witness of non-reachability. These required properties are given by the notion of Petri set, which itself relies on the notions of polytope sets and Lambert sets, that generalizes linear and semilinear sets. After having given in section 4.1 the definitions of polytope, Lambert and Petri sets, we will recall in section 4.2 some tools from [10], and especially the result that if a relation is Petri, one can find a witness of non-reachability which is a Presburger forward invariant.

Now, to prove that our reachability relation is Petri, we have to show that each transition sequence (a run) can be associated a production relation, such that (1) the runs ordered by inclusion of their production relations is well-ordered and (2) these productions relations are polytope. With a few additionnal assumptions, this means the reachability relation can be written as a finite sum and union of productions relations (the relations associated to the minimal elements of the previously defined well-order) and can be shown to be Petri. The main difference between the proof presented in this paper and [10] is the definition of a new production relation, done in section 5, that is adapted to the semantics of the VAS$_0$. The remaining of the paper will show that these production relations enjoy the required properties, namely well-ordering in section 6 and that they are polytope in section 7. This will allow us to conclude in section 8.

Given the similarity between VAS and VAS_0, we will reuse a lot of Leroux' work. The later sections will focus on the changes between the two proofs, with proofs that are either non-critical or mostly unchanged from Leroux' paper available in the long version [3].

4.1 Polytope, Lambert and Petri Sets

A set $C \subseteq \mathbb{Q}^d$ is conic if it is periodic and $\mathbb{Q}_{\geq 0}C = C$. A conic set is finitely generated if there exists a finite set $\{c_1, \ldots, c_n\} \subseteq \mathbb{Q}$ such that $C = \mathbb{Q}_{\geq 0}c_1 + \ldots + \mathbb{Q}_{\geq 0}c_n$.

Definition 4. *([10], Definitions 4.1 and 4.6)*
A periodic set $P \subseteq \mathbb{N}^d$ is polytope if the conic set $\mathbb{Q}_{\geq 0}P$ is definable in $FO(\mathbb{Q}, +, \leq, 0, 1)$ (the first order logic on the specified symbols). A set $L \subseteq \mathbb{N}^d$ is Lambert if it is a finite union of sets $b_i + P_i$ where $b_i \in \mathbb{N}^d$ and $P_i \subseteq \mathbb{N}^d$ is a polytope periodic set.

The stability of Lambert sets will be of importance in the sequel. We have the following properties: (proofs of these statements are reasonably direct, and available in [3]):

Proposition 2. *Given $L \subseteq \mathbb{N}^{d_1}, L' \subseteq \mathbb{N}^{d_2}$ Lambert sets and $k \in \mathbb{N}$:*

1. *For $d_1 = d_2$, $L \cup L'$ is Lambert.*
2. *$L \times L'$ is Lambert.*
3. *For $d'_1 < d_1$, $\{x \in \mathbb{N}^{d'_1} \mid \exists y \in \mathbb{N}^{d_1 - d'_1},\ (x, y) \in L\}$ is Lambert.*
4. *For $d_1 = d_2$, $L + L'$ is Lambert.*
5. *$k \star L$ is Lambert.*
6. *$\mathbb{N} \star L$ is polytope (more generally Lambert).*
7. *If δ is a linear function, then $\delta(L)$ is Lambert.*

Definition 5. *([10], Definition 4.7)*
A set $X \subseteq \mathbb{N}^d$ is Petri if for all Presburger sets S, $S \cap X$ is Lambert.

4.2 Important Results from Leroux

We recall in this section a few important results from [10].

For a set $X \subseteq \mathbb{Q}^d$, the closure of X, written $\overline{X}$ is defined by:

$$\overline{X} = \{l \mid \forall \tau > 0,\ \exists x \in X,\ max_i(x - l)(i) < \tau \wedge max_i(l - x)(i) < \tau\}$$

We have this useful characterization of polytope sets, that we will use to show that our production relation is polytope:

Theorem 2. *([10], Theorem 3.5)*
A periodic set $P \subseteq \mathbb{N}^d$ is polytope if and only if the conic set $\overline{(\mathbb{Q}_{\geq 0}P) \cap V}$ is finitely generated for every vector space $V \subseteq \mathbb{Q}^d$

The second theorem needed is the one motivating Petri sets. A Petri relation admits witnesses of non-reachability:

Theorem 3. *([10], Theorem 6.1)*
Let R be a reflexive relation over $\mathbb{N}^d$ such that R^* is Petri. Let $X, Y \subseteq \mathbb{N}^d$ be two Presburger sets such that $R^* \cap (X \times Y)$ is empty. There exists a partition of $\mathbb{N}^d$ into a Presburger R-forward invariant that contains X and a Presburger R-backward invariant that contains Y.

And finally, we will also need to use that the reachability relation of a VAS is already known to be Petri:

Theorem 4. *([10], Theorem 9.1)*
The reachability relation of a Vector Addition System is a Petri relation.

Since, given a VAS, we can add counters that increase each time a transition is fired, we can extend this result to include the parikh image of transition sequences:

Corollary 1. *Let $\mathcal{V} = \langle A, \delta \rangle$ be a VAS. Then, $\{(x, v, y) \in \mathbb{N}^d \times \mathbb{N}^A \times \mathbb{N}^d \mid \exists u,\ x \xrightarrow{u}_{\mathcal{V}} y \wedge |u| = v\}$ is a Petri set.*

5 Production Relations

For all the remaining sections, we will fix a VAS_0 $\mathcal{V}_z = \langle A_z, \delta, a_z \rangle$ of dimension d. We consider the set $A = A_z \backslash \{a_z\}$ and $\mathcal{V} = \langle A, \delta_{|A} \rangle$ the restriction of $\mathcal{V}_z$ to its non-a_z transitions. We have $\xrightarrow{*}$ (or $\xrightarrow{A_z^*}$) the transition relation of $\mathcal{V}_z$ and $\xrightarrow{A^*}$ the transition relation of $\mathcal{V}$.

A run μ of $\mathcal{V}_z$ is a sequence $m_0.a_1.m_1.a_2 \dots a_n.m_n$ alternating markings $m_i \in \mathbb{N}^d$ and actions $a_i \in A_z$ such that for all $1 \leq i \leq n$, $m_{i-1} \xrightarrow{a_i} m_i$. m_0 is called the *source* of μ, written $src(\mu)$ and m_n is called the *target* of μ, written $tgt(\mu)$. A run ρ of $\mathcal{V}_z$ is also a run of $\mathcal{V}$ if a_z doesn't appear in ρ.

We recall the definitions of the productions relations for a VAS of [10], adapted to our case by restricting the relation to runs that don't use the zero-test.

- For a marking $m \in \mathbb{N}^d$, $\xrightarrow[\mathcal{V},m]{} \subseteq \mathbb{N}^d \times \mathbb{N}^d$ is defined by:

$$x \xrightarrow[\mathcal{V},m]{} y \iff \exists u \in A^*,\ m + x \xrightarrow{u} m + y$$

- For a run $\rho = m_0.a_1.m_1 \dots a_n.m_n$ of $\mathcal{V}$, $\xrightarrow[\rho]{}$ is defined by:

$$\xrightarrow[\rho]{} = \xrightarrow[\mathcal{V},m_0]{} \circ \xrightarrow[\mathcal{V},m_1]{} \circ \cdots \xrightarrow[\mathcal{V},m_n]{}$$

We also define the production relation $\xrightarrow[\mathcal{V}_z,m]{} \subseteq \mathbb{N}^d \times \mathbb{N}^d$ of a marking $m \in \mathbb{N}_0^d$ inside $\mathcal{V}_z$ by:

$$x \xrightarrow[\mathcal{V}_z,m]{} y \iff \begin{cases} \exists u \in A_z^*,\ m + x \xrightarrow{u} m + y \\ x(1) = y(1) = 0 \end{cases}$$

To extend the definition of a production relation to a run μ of $\mathcal{V}_z$, we consider the decomposition of $\mu = \rho_0.a_z.\rho_1 \dots a_z.\rho_p$ such that forall $1 \leq i \leq p$, ρ_i is a run of $\mathcal{V}$. In that case, we define the production relation of μ by:

$$\overrightarrow{\mu} = \overrightarrow{\rho_0} \circ \overrightarrow{\mathcal{V}_z, tgt(\rho_0)} \circ \overrightarrow{\rho_1} \circ \cdots \circ \overrightarrow{\mathcal{V}_z, tgt(\rho_{p-1})} \circ \overrightarrow{\rho_p}$$

Proposition 3. *For $m \in \mathbb{N}^d$, $m' \in \mathbb{N}_0^d$ and μ a run of $\mathcal{V}_z$ (a run $\mathcal{V}$ being a special case), $\overrightarrow{\mathcal{V}, m}$, $\overrightarrow{\mathcal{V}_z, m'}$ and $\overrightarrow{\mu}$ are periodic.*

Proof: The result is easy for $\overrightarrow{\mathcal{V}, m}$ and $\overrightarrow{\mathcal{V}_z, m'}$. We conclude by the fact that the composition of periodic relations is periodic. □

One can prove by a simple induction on the length of μ (available in [3]) the following statement:

Proposition 4. *For a run μ of $\mathcal{V}_z$, we have:*

$$(src(\mu), tgt(\mu)) + \overrightarrow{\mu} \subseteq \xrightarrow{*}$$

6 Well-Orderings of Production Relations

For two runs μ, μ', let us define $\preceq$ by:

$$\mu \preceq \mu' \iff (src(\mu'), tgt(\mu')) + \overrightarrow{\mu'} \subseteq (src(\mu), tgt(\mu)) + \overrightarrow{\mu}$$

Our aim is to show that $\preceq$ is a well-order. To do that, we define the order $\trianglelefteq$ on runs of $\mathcal{V}_z$ in the following way:

- For $\rho = m_0.a_1.m_1 \dots a_p.m_p$ and $\rho' = m'_0.a'_1.m'_1 \dots a'_q.m'_q$ runs of $\mathcal{V}$ ($a_i, a'_i \in A$), we have $\rho \trianglelefteq \rho'$ if (same definition as in [10]):
  - $m_0 \leq m'_0$ and $m_p \leq m'_q$
  - $\prod_{1 \leq i \leq p}(a_i, m_i) \leq^{emb} \prod_{1 \leq i \leq q}(a'_i, m'_i)$
    with $(a, m) \leq (a', m') \iff a = a' \wedge m \leq m'$
- For $\mu = \rho_0.a_z.\rho_1 \dots a_z.\rho_p$ and $\mu' = \rho'_0.a_z.\rho'_1 \dots a_z.\rho'_q$ runs of $\mathcal{V}_z$ (with ρ_i, ρ'_i runs of $\mathcal{V}$), we have $\mu \trianglelefteq \mu'$ if:
  - $\rho_0 \trianglelefteq \rho'_0$ and $\rho_p \trianglelefteq \rho'_q$
  - $\prod_{1 \leq i \leq p} \rho_i \trianglelefteq^{emb} \prod_{1 \leq i \leq q} \rho'_i$

Two applications of Higman's lemma gives us the following result:

Proposition 5. *The order $\trianglelefteq$ is well.*

Now, we only need to prove the following:

Proposition 6. *For μ, μ' runs of $\mathcal{V}_z$, we have:*

$$\mu \trianglelefteq \mu' \implies \mu \preceq \mu'$$

Proof Sketch: The full proof is available in [3]. [10] already contains the result for runs without the zero-test.

The idea is that our run can be decomposed in the following way, where $\varphi_{i,j}$ refers to "suppressed" sequences, and ρ_i'' are greater than ρ_i for $\trianglelefteq$.

$$\prod_{1\leq k\leq q} \rho_k' = \rho_0'' \left(\prod_{1\leq j\leq n_0} \varphi_{0,j} \right) \rho_1'' \cdots \left(\prod_{1\leq j\leq n_{p-1}} \varphi_{p-1,j} \right) \rho_p''$$

Now, the outline of the proof is to base ourselves on Leroux' result for runs without zero-tests, and to show that the productions of suppressed sequences are included in $\overrightarrow{\nu_z, tgt(\rho_i)}$ where ρ_i is the part of the run before the suppressed sequence. □

We can now combine propositions 5 and 6 to get:

Theorem 5. *$\preceq$ is a well-order on runs of $\mathcal{V}_z$.*

7 Polytopie of the Production Relation

Showing the polytopie of $\overrightarrow{\mu}$ follows (again!) [10]. The main difference is that while the production relations of Leroux were approximated by a finite automata where the edges represented transitions of the system, our approximation will have its edges representing runs of the VAS $\mathcal{V}$.

First, we note that the relation $\overrightarrow{\mu}$ is a finite composition of relations $\overrightarrow{\nu,m}$ (for $m \in \mathbb{N}^d$) and $\overrightarrow{\nu_z,m}$ (for $m \in \mathbb{N}_0^d$). To show that $\overrightarrow{\mu}$ is polytope, we recall two results from [10] regarding production relations:

Lemma 1. *([10], Lemma 8.2)*
If R and R' are two polytope periodic relations, then $R \circ R'$ is a polytope periodic relation.

Theorem 6. *([10], Theorem 8.1)*
For $m \in \mathbb{N}^d$, $\overrightarrow{\nu,m}$ is polytope.

These two results mean we only need to prove that $\overrightarrow{\nu_z,m}$ is a polytope periodic relation for $m \in \mathbb{N}_0^d$.

Proposition 7. *For $m \in \mathbb{N}_0^d$, $\overrightarrow{\nu_z,m}$ is polytope.*

Proof: Theorem 2 shows that $\overrightarrow{\nu_z,m}$ is polytope if and only if the following conic space is finitely generated for every vector space $V \subseteq \mathbb{Q}^d \times \mathbb{Q}^d$:

$$\overline{(\mathbb{Q}_{\geq 0}\ \overrightarrow{\nu_z,m}) \cap V} = \overline{\mathbb{Q}_{\geq 0}(\ \overrightarrow{\nu_z,m} \cap V)}$$

Let us define $V_0 = (\mathbb{N}_0^d \times \mathbb{N}_0^d) \cap V$. We will re-use the idea of Leroux' intraproductions, but by restricting them to $\mathbb{N}_0^d$. Let $Q_{m,V} = \{y \in \mathbb{N}_0^d \mid \exists (x,z) \in (m,m) + V_0,\ x \xrightarrow{*} y \xrightarrow{*} z\}$ and $I_{m,V} \subseteq \{1,\dots,d\}$ by $i \in I_{m,V} \iff \{q(i) \mid q \in Q_{m,V}\}$ is infinite. Note that $1 \notin I_{m,V}$, as for all $q \in Q_{m,V}$, $q(1) = 0$. An *intraproduction* for (m, V_0) is a triple (r, x, s) such that $x \in \mathbb{N}_0^d$ and $(r,s) \in V_0$ with:

$$r \xrightarrow[\nu_z,m]{} x \xrightarrow[\nu_z,m]{} s$$

An intraproduction is *total* if $x(i) > 0$ for every $i \in I_{m,V}$. The following lemma can be proved exactly as Lemma 8.3 of [10] (a precise proof is available in [3]):

Lemma 2. *There exists a total intraproduction for* (m, V_0).

Now we define $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, ordered by $x < \infty$ for every $x \in \mathbb{N}$. Given a finite set $I \subseteq \{1,\dots,d\}$ and a marking $m \in \mathbb{N}^d$, we denote by m^I the vector of $\mathbb{N}_\infty^d$ defined by $m^I(i) = \infty$ if $i \in I$ and $m^I(i) = m(i)$ otherwise. We also define the order $\leq_\infty$ by $x \leq_\infty y$ if for all i, $y(i) = \infty$ or $x(i) = y(i)$ (equivalently there exists $I \subseteq \{1,\dots,d\}$, $x^I = y$). For a relation $\to$, and $(x,y) \in \mathbb{N}_\infty^d$. We define $x \to x'$ if there exists $(m,m') \in \mathbb{N}^d$, $m \leq_\infty x$ and $m' \leq_\infty x'$ with $m \to m'$.

Let $Q = \{q^{I_{m,V}} \mid q \in Q_{m,V}\}$ and $\mathcal{G}$ the complete directed graph with nodes Q whose edges from q to q' are labeled by (q,q'). For $w \in (Q \times Q)^*$, we define $TProd(w) \subseteq \mathbb{N}^{A_z}$ by:

$$\begin{aligned} TProd(\varepsilon) &= \{0^{A_z}\} \\ TProd((q,q')) &= \left\{ |u| \mid \exists (x,x') \in \mathbb{N}_0^d \times \mathbb{N}_0^d,\ \begin{array}{l} x \leq_\infty q,\ x' \leq_\infty q', \\ u \in a_z A^* \cup A^*, \\ x \xrightarrow{u} x' \end{array} \right\} \\ TProd(uv) &= TProd(u) + TProd(v) \end{aligned}$$

We define the periodic relation $R_{m,V}$ on V_0 by $r\ R_{m,V}\ s$ if:

1. $r(i) = s(i) = 0$ for every $i \notin I_{m,V}$
2. there exists a cycle labelled by w in $\mathcal{G}$ on the state $m^{I_{m,V}}$ and $v \in TProd(w)$ such that $r + \delta(v) = s$.

Lemma 3. *The periodic relation* $R_{m,V}$ *is polytope.*

Proof: First, let's show that $TProd((q,q'))$ is Lambert for every $(q,q') \in Q \times Q$. We define $X_1 = \{(x',y) \in \mathbb{N}_0^d \times \mathbb{N}_0^d \mid \exists x \leq_\infty q,\ x \xrightarrow{a_z} x' \wedge y \leq_\infty q'\}$ and $X_2 = \{(x,y) \in \mathbb{N}_0^d \times \mathbb{N}_0^d \mid x \leq_\infty q \wedge y \leq_\infty q'\}$ which are Presburger sets. Because, $Y = \{(x',v,y) \in \mathbb{N}^d \times \mathbb{N}^A \times \mathbb{N}^d \mid \exists u \in A^*,\ x' \xrightarrow{u} y \wedge |u| = v\}$ is a Petri set (corollary 1), $Y_1 = Y \cap (X_1 \times \mathbb{N}^A \times \mathbb{N}^d)$ and $Y_2 = Y \cap (X_2 \times \mathbb{N}^A \times \mathbb{N}^d)$ are Lambert sets, and by projection (proposition 2), $TProd((q,q')) = \big(|a_z| + \{u \mid \exists (x,y) \in \mathbb{N}^d \times \mathbb{N}^d,\ (x,u,y) \in Y_1\}\big) \cup \{u \mid \exists (x,y) \in \mathbb{N}^d,\ (x,u,y) \in Y_2\}$ is Lambert.

Let $P \subseteq \mathbb{N}^{Q\times Q}$ be the Parikh image of the language L made of words labelling cycles in $\mathcal{G}$ on the state $m^{I_{m,V}}$. L is a language recognized by a finite automaton, hence P is a Presburger set.

Now, let's show that $R'_{m,V} = \{TProd(w) \mid w \in L\}$ is a Lambert set. We have:

$$R'_{m,V} = \left\{ \sum_{a\in Q\times Q} v(a) \star TProd(a) \mid v \in P \right\}$$

P is Presburger, hence there exists $(d_i)_{1\leq i\leq p}$, $(e_{i,j})_{1\leq i\leq p, 1\leq j\leq n_i}$ with $d_i, e_{i,j} \in \mathbb{N}^{Q\times Q}$ and $P = \bigcup_i d_i + \Sigma_j \mathbb{N} e_{i,j}$. This gives:

$$\begin{aligned} R'_{m,V} &= \bigcup_{1\leq i\leq p} \bigcup_{v\in\mathbb{N}^p} \sum_{1\leq j\leq n_i} \sum_{a\in Q\times Q} (d_i + v(j) * e_{i,j})(a) \star TProd(a) \\ &= \bigcup_i \sum_a d_i(a) \star TProd(a) + \bigcup_i \sum_j \bigcup_{k\in\mathbb{N}} \sum_a (k * e_{i,j})(a) \star TProd(a) \\ &= \bigcup_i \sum_a d_i(a) \star TProd(a) + \bigcup_i \sum_j \mathbb{N} \star \left(\sum_a e_{i,j}(a) \star TProd(a) \right) \end{aligned}$$

For all $a \in Q\times Q$, we have seen that $TProd(a)$ is Lambert. So because Lambert sets are stable by addition, union and $\mathbb{N}\star$, (proposition 2), $R'_{m,V}$ is Lambert.

We define $V_{I_{m,V}} = \{x \in \mathbb{N}^d \mid \forall i \notin I_{m,V},\ x(i) = 0\}$ and $R''_{m,V} = \{(r, r+\delta(x)) \mid r \in V_{I_{m,V}} \wedge x \in R'_{m,V}\} = \{(r,r) \mid r \in V_{I_{m,V}}\} + \{0\}^d \times \delta(R'_{m,V})$. By proposition 2, we have $R''_{m,V}$ built from $R'_{m,V}$ by the image through a linear function and the sum with a Presburger set, which means $R''_{m,V}$ is Lambert. But, $R''_{m,V}$ is periodic, which means $R''_{m,V} = \mathbb{N}\star R''_{m,V}$ is polytope. Finally, as proposition 2 gives us that polytope sets are stable by intersection with vector spaces, $R_{m,V} = R''_{m,V} \cap V$ is polytope. □

We will now show that our graph $\mathcal{G}$ is an accurate representation of the reachability relation:

Lemma 4. *Let w be the label of a path in $\mathcal{G}$ from $m_1^{I_{m,V}}$ to $m_2^{I_{m,V}}$ and $v \in TProd(w)$. Then, there exists $u \in A_z^*$ with $|u| = v$ and $(x,y) \in \mathbb{N}_0^d \times \mathbb{N}_0^d$, $x \leq_\infty m_1^{I_{m,V}}$ and $y \leq_\infty m_2^{I_{m,V}}$ such that $x \xrightarrow{u} y$.*

Proof: We show this by induction on the length of w. Let $w = w_0(q,q')$ where w_0 is a path from $m_1^{I_{m,V}}$ to $m_3^{I_{m,V}}$ and (q,q') is an edge from $m_3^{I_{m,V}}$ to $m_2^{I_{m,V}}$ and $v \in TProd(w_0(q,q'))$. This means there exists $v_1 \in TProd(w_0)$, $v_2 \in TProd(q,q')$ such that $v = v_1 + v_2$. By induction hypothesis, there exists $u_1 \in \mathbb{N}_0^d \times \mathbb{N}_0^d$, $x'_0 \leq_\infty m_1^{I_{m,V}}$ and $y'_0 \leq_\infty m_3^{I_{m,V}}$ such that $x'_0 \xrightarrow{u_1} y'_0$ and $|u_1| = v_1$.

By definition of $TProd((q,q'))$, as $v_2 \in TProd((q,q'))$, there exists $x'_1 \leq m_3^{I_{m,V}}$, $y'_1 \leq_\infty m_2^{I_{m,V}}$ and $u_2 \in a_z A^* \cup A^*$ such that $x'_1 \xrightarrow{u_2} y'_1$ and $|u_2| = v_2$. Let $z = max(y'_0, x'_1)$. We have $z(1) = y'_0(1) = x'_1(1) = m_3(1) = 0$, which gives us:

$$x'_0 + (z - y'_0) \xrightarrow{u_1} z \xrightarrow{u_2} y'_1 + (z - x'_1)$$

As $z^{I_{m,V}} = {y'_0}^{I_{m,V}} = {x'_1}^{I_{m,V}} = m_3^{I_{m,V}}$, we have $(z - y'_0) \leq_\infty 0^{I_{m,V}}$ and $(z - x'_1) \leq_\infty 0^{I_{m,V}}$, which allows us to define $x = x'_0 + (z - y'_0) \leq_\infty m_1^{I_{m,V}}$ and $y = y'_1 + (z - x'_1) \leq_\infty m_2^{I_{m,V}}$. $u = u_1 u_2$ completes the result. □

We now show a lemma for the other direction:

Lemma 5. *Let $(m_1, m_2) \in Q_{m,V} \times Q_{m,V}$ with $u \in A_z^*$ such that $m_1 \xrightarrow{u} m_2$. There exists $w \in (Q \times Q)^*$ label of a path from $m_1^{I_{m,V}}$ to $m_2^{I_{m,V}}$ such that $|u| \in TProd(w)$.*

Proof: Let $u = u_1 a_z u_2 \dots a_z u_n$ with $u_i \in A^*$. We define $(x_i)_{1 \leq i \leq n}$, $x_i \in \mathbb{N}_0^d$ by:

$$m \xrightarrow{u_1} x_1 \xrightarrow{a_z u_2} x_2 \xrightarrow{a_z u_3} x_3 \cdots \xrightarrow{a_z u_n} x_n = m_2$$

We have for all i, $x_i \in \mathbb{N}_0^d$, which leads to $|u_1| \in TProd((m_1^{I_{m,V}}, x_1^{I_{m,V}}))$ and for all $i \in \{1, \dots, n-1\}$, $|a_z u_n| \in TProd((x_i^{I_{m,V}}, x_{i+1}^{I_{m,V}}))$. Hence, we can define $w = (m_1^{I_{m,V}}, x_1^{I_{m,V}})(x_1^{I_{m,V}}, x_2^{I_{m,V}}) \dots (x_{n-1}^{I_{m,V}}, m_2^{I_{m,V}})$ and we have $|u| \in TProd(w)$. □

Thanks to lemmas 4 and 5, we can now prove the following lemma exactly in the same way as Lemma 8.5 of [10] (full proof in [3])

Lemma 6. $\overline{\mathbb{Q}_{\geq 0} R_{m,V}} = \overline{\mathbb{Q}_{\geq 0}(\overrightarrow{\nu_{z,m}} \cap V_0)}$

By lemma 3, $R_{m,V}$ is polytope, hence $\overline{\mathbb{Q}_{\geq 0} R_{m,V}}$ is finitely generated. We have proven proposition 7. □

Finally, as $\overrightarrow{\mu}$ is a finite composition of elements of the form $\overrightarrow{\nu_{,m}}$ and $\overrightarrow{\nu_{z,m}}$, we have proven the following result:

Theorem 7. *If μ is a run of $\mathcal{V}_z$, then $\overrightarrow{\mu}$ is polytope.*

8 Decidability of Reachability

We have now all the results necessary to show the following:

Theorem 8. $\xrightarrow{*}$ *is a Petri relation.*

Proof Sketch: Similarly as in Theorem 9.1 of [10], one can show thanks to proposition 4 and theorem 5 that for any $(m, n) \in \mathbb{N}^d \times \mathbb{N}^d$ and $P \subseteq \mathbb{N}^d$ finitely generated periodic set, there exists a finite set B of runs of $\mathcal{V}_z$ such that:

$$\xrightarrow{*} \cap ((m, n) + P) = \bigcup_{\mu \in B} (src(\mu), tgt(\mu)) + (\overrightarrow{\mu} \cap P)$$

Then, proposition 5 allows to conclude that $\xrightarrow{*}$ is Petri. The full proof is available in [3]. □

Because $\left(\xrightarrow{a_z A^* \cup A^*}\right)^* = \xrightarrow{A_z^*}$, we can now apply theorem 3 and get:

Proposition 8. *If X and Y are two Presburger sets such that $\xrightarrow{A_z^*} \cap (X \times Y) = \emptyset$, then there exists a Presburger $\xrightarrow{a_z A^* \cup A^*}$-forward invariant X' with $X' \cap Y = \emptyset$.*

Now that we have shown the existence of such an invariant, we only need to show that we are able to test whether a given set is an invariant:

Proposition 9. *Whether a Presburger set X is a $\xrightarrow{a_z A^* \cup A^*}$-forward invariant is decidable.*

Proof: X is a forward invariant for $\xrightarrow{a_z A^* \cup A^*}$ if and only if $\xrightarrow{a_z} (X) \subseteq X$ and $\xrightarrow{A^*} (X) \subseteq X$. Because $\xrightarrow{a_z} (X)$ is a Presburger set, the first condition is decidable as the inclusion of Presburger sets, and the second reduces to deciding whether $\xrightarrow{A^*} \cap (X \times \mathbb{N}^d \backslash X)$ is empty, which is an instance of the reachability problem in a VAS (Theorem 1). □

By the propositions 8 and 9, reachability is co-semidecidable by enumerating forward invariants, and as semidecidability is clear, we conclude:

Theorem 9. *Reachability in VAS_0 is decidable.*

Acknowledgements. I would like to thank my advisor, Alain Finkel, for his numerous proof-readings and advices. Thanks also to Jerome Leroux for many helpful discussions and for pointing a critical mistake in a previous version and to the anonymous referees for their comments.

References

1. Abdulla, P., Mayr, R.: Minimal cost reachability/coverability in priced timed petri nets. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 348–363. Springer, Heidelberg (2009)
2. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: LICS 1996, pp. 313–321 (1996)
3. Bonnet, R.: The reachability problem for vector addition systems with one zero-test. Research Report LSV-11-11, Laboratoire Spécification et Vérification, ENS Cachan, France, pages 19 (May 2011)
4. Bonnet, R., Finkel, A., Leroux, J., Zeitoun, M.: Place-boundedness for vector addition systems with one zero-test. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2010, Leibniz International Proceedings in Informatics, Chennai, India, vol. 8, pp. 192–203. Leibniz-Zentrum für Informatik (December 2010)
5. Finkel, A., Sangnier, A.: Mixing coverability and reachability to analyze VASS with one zero-test. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 394–406. Springer, Heidelberg (2010)
6. Ganty, P., Atig, M.F.: Approximating petri net reachability along context-free traces. CoRR, abs/1105.1657 (2011)
7. Kosaraju, S.R.: Decidability of reachability in vector addition systems (preliminary version). In: STOC 1982, pp. 267–281. ACM, New York (1982)

8. Lambert, J.L.: A structure to decide reachability in petri nets. Theoretical Computer Science 99(1), 79 (1992)
9. Leroux, J.: The general vector addition system reachability problem by presburger inductive invariants. In: Symposium on Logic in Computer Science, pp. 4–13 (2009)
10. Leroux, J.: Vector addition system reachability problem: a short self-contained proof. SIGPLAN Not. 46, 307–316 (2011)
11. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: STOC 1981: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, pp. 238–246. ACM, New York (1981)
12. Reinhardt, K.: Reachability in Petri Nets with inhibitor arcs (1995), http://www-ti.informatik.uni-tuebingen.de/~reinhard/abstracts.html (unpublished)
13. Reinhardt, K.: Reachability in Petri Nets with inhibitor arcs. Electr. Notes Theor. Comput. Sci. 223, 239–264 (2008)

The Bounded Search Tree Algorithm for the Closest String Problem Has Quadratic Smoothed Complexity

Christina Boucher

Department of Computer Science and Engineering
University of California, San Diego
cboucher@eng.ucsd.edu

Abstract. Given a set S of n strings, each of length ℓ, and a nonnegative value d, we define a *center string* as a string of length ℓ that has Hamming distance at most d from each string in S. The Closest String problem aims to determine whether there exists a center string for a given set of strings S and input parameters n, ℓ, and d. When n is relatively large with respect to ℓ then the basic majority algorithm solves the Closest String problem efficiently, and the problem can also be solved efficiently when either n, ℓ or d is reasonably small [12]. Hence, the only case for which there is no known efficient algorithm is when n is between $\log \ell / \log \log \ell$ and $\log \ell$. Using smoothed analysis, we prove that such Closest String instances can be solved efficiently by the $O(n\ell + nd \cdot d^d)$-time algorithm by Gramm *et al.* [13]. In particular, we show that for any given Closest String instance I, the expected running time of this algorithm on a small perturbation of I is $O\left(n\ell + nd \cdot d^{2+o(1)}\right)$.

1 Introduction

Finding similar regions in multiple DNA, RNA, or protein sequences plays an important role in many applications, including universal PCR primer design [7,16,19,26], genetic probe design [16], antisense drug design [6,16], finding transcription factor binding sites in genomic data [28], determining an unbiased consensus of a protein family [4], and motif recognition [16,24,25]. The Closest String problem formalizes this task of finding a common pattern in an input set of strings and can be defined as follows:

Input: A set of n length-ℓ strings $S = \{s_1, \ldots, s_n\}$ over a finite alphabet Σ and a nonnegative integer d.
Question: Find a string s of length ℓ, where the Hamming distance from s to any string in s_i is at most d.

We refer to s as the *center string* and let $d(x, y)$ be the Hamming distance between strings x and y. The optimization version of this problem tries to minimize the parameter d.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 158–169, 2011.

The Closest String problem was first introduced and studied in the context of bioinformatics by Lanctot *et al.* [16]. Frances and Litman [11] showed the problem to be NP-complete even for the special case when the input contains only binary strings, implying there is unlikely to be a polynomial-time algorithm for solving this problem unless P = NP. Since its introduction, efficient approximation algorithms and exact heuristics for the Closest String problem have been thoroughly considered [9,10,13,16,17,21]. Most recently, Hufsky *et al.* [14] introduced a data reduction techniques that allows instances that do not have a solution and can be filtered out and incorporate this preprocessing step into the algorithm of Gramm *et al.* [13].

One approach to investigating the computational intractability of the Closest String problem is to consider its parameterized complexity, which aims to classify computationally hard problems according to their inherent difficulty with respect to a subset of the input parameters. If it is solvable by an algorithm whose running time is polynomial in the input size and exponential in parameters that typically remain small then it can still be considered tractable in some practical sense. A problem φ is said to be *fixed-parameter tractable* with respect to parameter k if there exists an algorithm that solves φ in $f(k) \cdot n^{O(1)}$ time, where f is a function of k that is independent of n [8]. Gramm *et al.* [13] proved that Closest String is fixed-parameter tractable with respect to the parameter d by giving a $O(n\ell + nd \cdot d^d)$-time algorithm that is based on the bounded search tree paradigm.

It has been previously shown that when the number of strings is significantly large with respect to ℓ (namely, whenever $2^n > \ell$) then the basic majority algorithm, which returns a string that contains the majority symbol at each position with ties broken arbitrarily, works well in practice. Also, there exist efficient solutions for the Closest String problem when either n, ℓ or d are reasonably small [12]. The only case for which there is no known efficient algorithm is when n is between $\log \ell / \log \log \ell$ and $\log \ell$. Ma and Sun [21] state: "The instances with d in this range seem to be the hardest instances of the closest string problem. However, because the fixed-parameter algorithm has polynomial (although with high degree) running time on these instances, a proof for the hardness of these instances seem to be difficult too."

We initiate the study of the smoothed complexity of a slightly modified version of the algorithm by Gramm *et al.* [13], and demonstrate that more careful analysis of the algorithm of Gramm *et al.* [13] reveals that it is efficient for the "hardest" Closest String instances where n is between $\log \ell / \log \log \ell$ and $\log \ell$. Our analysis gives an analytical reason as to why this algorithm performs well in practice. We introduce a perturbation model for the Closest String problem, and prove that the expected size of the search tree of the algorithm of Gramm *et al.* [13] on these smoothed instances is at most $d^{2+o(1)}$, hence resolving an open problem that was suggested by Gramm *et al.* [13], and Ma and Sun [21].

1.1 Related Work

Gramm *et al.* [13] proved that the Closest String problem is fixed-parameter tractable when parameterized by n, and when parameterized by d. More recently, Ma and Sun gave an $O(n|\Sigma|^{O(d)})$-time algorithm, which is a fixed-parameter algorithm in parameters d and Σ [21]. Chen *et al.* [5], Wang and Zhu [29], and Zhao and Zhang [30] improved upon the fixed-parameter tractable result of Ma and Sun [21]. Lokshtanov *et al.* [18] gave a lower bound for the time complexity for the Closest String problem with respect to d. Another approach to investigate the tractability of this NP-complete problem is to consider how well the Closest String problem can be approximated in polynomial-time. Lanctot *et al.* [16] gave a polynomial time algorithm that achieves a $\frac{4}{3}+o(1)$ approximation guarantee. Li *et al.* [17], Andoni *et al.* [1] and Ma and Sun [21] each proved PTAS results for this problem.

Smoothed analysis was introduced as an intermediate measure between worst-case and average-case analysis and is used to explain the phenomena that many algorithms with detrimental worst-case analysis efficiently find good solutions in practice. It works by showing that the worst-case instances are fragile to small change; slightly perturbing a worst-case instance destroys the property of it being worst-case [27]. The smoothed complexity of other string and sequence problems has been considered by Andoni and Krauthgamer [2], Banderier *et al.* [3], Manthey and Reischuk [23], and Ma [20]. Andoni and Krauthgamer [2] studied the smoothed complexity of sequence alignment by the use of a novel model of edit distance; their results demonstrate the efficiency of several tools used for sequence alignment, most notably PatternHunter [22]. Manthey and Reischuk gave several results considering the smoothed analysis of binary search trees [23]. Ma demonstrated that a simple greedy algorithm runs efficiently in practice for Shortest Common Superstring [20], a problem that has applications to string compression and DNA sequence assembly.

1.2 Preliminaries

Let s be a string over the alphabet Σ. We denote the length of s as $|s|$, and the jth letter of s as $s[j]$. Hence, $s = s[1]s[2]\dots s[|s|]$. It will be convenient to consider a set of strings $S = \{s_1, \dots, s_n\}$, each of which has length ℓ, as a $n \times \ell$ matrix. Then we refer to the ith *column* as the vector $c_i = [s_1(i), \dots, s_n(i)]^T$ in the matrix representation of S.

We refer to a *majority string* for S as the length-ℓ string containing the letter that occurs most often at each position; this string is not necessarily unique. The following fact, which is easily proved, is used in Section 3.

Fact 1. *Let $I = (S, d)$ be a* Closest String *instance and s_{maj} be any majority string for S then $d(s^*, s_{maj}) \leq 2d$ for any center string s^* for S.*

Given functions f and g of a natural number variable x, the notation $f \asymp g$ $(x \to \infty)$ is used to express that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1$$

and f is an asymptotic estimation of g (for relatively large values of x). The following asymptotic estimation is used in our analysis.

Fact 2. *For fixed $j > 0$ the following asymptotic estimation exists:*

$$\sum_{i=0}^{2d} \binom{2i+j}{i} \left(\frac{\ell-1}{\ell}\right)^i \left(\frac{1}{\ell}\right)^{i+j} \asymp \left(\frac{1}{\ell-1}\right)^{2d}.$$

Given a Closest String instance $I = (S, d)$ that has at least one center string we can assume, without loss of generality, that 0^ℓ is a center string; any instance that has a center string can be transformed to an equivalent instance where 0^ℓ is a center string [12]. Hence, for the remainder of this paper we assume that any instance that has a center string, has 0^ℓ is a center string.

2 Bounded Search Tree Algorithm

The following algorithm, due to Gramm *et al.* [13], applies a well-known bounded search tree paradigm to prove that the Closest String problem can be solved in linear time when parameterized by d.

Bounded Search Tree Algorithm
Input: A Closest String instance $I = (S, d)$, a candidate string s, and a parameter Δd.
Output: A center string s if it exists, and "not found" otherwise.
If $\Delta d < 0$, then return "Not found"
Choose $i \in \{1, \ldots, n\}$ such that $d(s, s_i) > d$.
  $\mathcal{P} = \{p | s[p] \neq s_i[p]\}$;
  Choose any $\mathcal{P}' \subseteq \mathcal{P}$ with $|\mathcal{P}'| = d + 1$.
  For each position $p \in \mathcal{P}'$
    Let $s(p) = s_i(p)$
    s_{ret} = **Bounded Search Tree Algorithm** (s, $\Delta d - 1$)
    If $s_{ret} \neq$ "not found ", then return s_{ret}
Return "not found"

The parameter Δd is initialized to be equal to d. Since every recursive call decreases Δd by one and the algorithm halts when $\Delta d < 0$, the search tree has height at most d. At each recursive step if the candidate string s is not a center string then it is *augmented* at one position as follows: a string s_i is chosen uniformly at random from the set of strings that have distance greater than d from s, and s is changed so that it is equal to s_i at one of the positions where s and s_i disagree. This yields an upper bound of $(d+1)^d$ on the search tree size.

Gramm *et al.* [13] initialize the candidate string s to be a string from S chosen uniformly at random. We consider a slight modification where the candidate string is initialized to be a majority string. As stated in Fact 1, any majority string has distance at most $2d$ from the center string. The analysis of Gramm *et*

al. [13] concerning the running time and correctness of the algorithm holds for this modification, and yields a worst-case of $O(n\ell+nd\cdot d^{2d})$. Hence, the following theorem is a trivial extension to the worst-case analysis by Gramm *et al.* [13] and will be used in our smoothed analysis of *Bounded Search Tree Algorithm*.

Theorem 1. *"Bounded Search Tree Algorithm" solves the* Closest String *problem in* $O(n\ell + nd \cdot d^{2d})$*-time.*

3 Smoothed Analysis

3.1 Pertubation of Closest String Instances

Our model applies to problems defined on strings. It is parameterized by a probability p, where $0 \leq p \leq 1$ and is defined as follows. Given a length-ℓ string $s[1]s[2]\ldots s[\ell]$, each element is selected (independently) with probability p. Let x be the number of selected elements (on average $x = p\ell$). Replace each of the x positions with a symbol chosen (uniformly at random) from the set of $|\Sigma|$ symbols from Σ. Given a Closest String instance $I = (S, d)$, we obtain a *perturbed instance* of I by perturbing each string in S with probability $p > 0$ as previously described. We denote a perturbed instance as $I' = (S', d')$, where $d' = d$ and S' contains the perturbed strings of S.

This perturbation model has the effect of naturally adding noise to input. Note that the pertubation model may have the affect of converting an instance that has a center string to one that does not, however, this remains a valid model for smoothed analysis. For example, the model used by Spielman and Teng allows pertubation from a feasible linear program to a non-feasible linear program [27].

3.2 Good Columns and Simple Instances

In this subsection we classify the instances for which the *Bounded Search Tree Algorithm* performs efficiently, and bound the probability that an instance has this classification. This classification is used in our smoothed analysis.

Definition 1. *Let* $I = (S, d)$ *be a* Closest String *instance, and* S_{maj} *be the set of majority strings for* S*. We define* S *as* simple *if any string in* S_{maj} *has Hamming distance at most* d *from all strings in* S*.*

Closest String instances that are simple have the property that *Bounded Search Tree Algorithm* halts immediately with a center string. In the remainder of this subsection we aim to bound the probability that an instance is simple. The next definitions are used to simplify the discussion of the analysis that bounds this probability.

Recall our assumption that any instance that contains a center string has 0^ℓ as a center string. Given an instance $I = (S, d)$ with a center string, we refer to a column of S as *good* if it contains more zeros than nonzeros and thus, guarantees that the majority symbol is equal to the center string at that position; all other columns are *bad*.

Lemma 1. *Let $I' = (S', d')$ be the perturbed instance of $I = (S, d)$ with probability p. Then the probability that I' is not simple is at least $1 - \left(1 - (q(1-q))^{n/2-1}\right)^{\ell}$ and at most $1 - \left(1 - \binom{n}{n/2+1}(q(1-q))^{n/2+1}\right)^{\ell}$, where $0 \leq q \leq \frac{d}{\ell}(1-2p) + p$.*

Proof. Let $s'_i \in S'$, s^* be a closest string for S, and q denote the probability that $s'_i[j] = 0$ for some $0 \leq j \leq \ell$. It follows that

$$q = \frac{d(s_i, s^*)}{\ell}(1-p) + \left(1 - \frac{d(s_i, s^*)}{\ell}\right)p = \frac{d(s_i, s^*)}{\ell}(1-2p) + p,$$

which is at most $\frac{d}{\ell}(1-2p) + p$. We first calculate the probability that a column is good when n is odd. Let $X_{i,j}$ be a binary random variable that is equal to 1 if s_i is equal to the value of the center string (*i.e.* equal to 0) at the jth position. For a given column j we let the number of zeros be $X_j = \sum_i X_{i,j}$.

$$\begin{aligned}\Pr[X_j \geq \lfloor n/2 \rfloor + 1] &= 1 - \Pr[X_j \leq \lfloor n/2 \rfloor] \\ &= 1 - (1-q)^n \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} \left(\frac{q}{1-q}\right)^i\end{aligned}$$

We focus on bounding $(1-q)^n \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} \left(\frac{q}{1-q}\right)^i$. Note that $\binom{n}{i}$ is unimodal, peaking when i is equal to $\lfloor n/2 \rfloor$ when $0 \leq i \leq \lfloor n/2 \rfloor$. We have that:

$$S = \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} \left(\frac{q}{1-q}\right)^i \geq \binom{n}{\lfloor n/2 \rfloor - 1} \left(\frac{q}{1-q}\right)^{\lfloor n/2 \rfloor - 1}$$

and similarly,

$$\begin{aligned}S = \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} \left(\frac{q}{1-q}\right)^i &\leq \binom{n}{\lfloor n/2 \rfloor} \sum_{i=0}^{\lfloor n/2 \rfloor} \left(\frac{q}{1-q}\right)^i && \text{[unimodality]} \\ &\leq \binom{n}{\lfloor n/2 \rfloor} \left(\frac{q}{1-q}\right)^{\lfloor n/2 \rfloor} && \text{[geometric series]}\end{aligned}$$

The sum S is equal to the first term up to a small multiplicative error and therefore, we obtain the following:

$$\Pr[X_j > \lfloor n/2 \rfloor] \approx 1 - q^{\lfloor n/2 \rfloor}(1-q)^{\lfloor n/2 \rfloor} \binom{n}{\lfloor n/2 \rfloor}$$

Therefore, we obtain the following bounds:

$$1 - \binom{n}{n/2-1}(q(1-q))^{n/2-1} \leq \Pr[X_j > \lfloor n/2 \rfloor] \leq 1 - (q(1-q))^{n/2+1}$$

Using the previous inequality we can bound the probability that I' is *not* simple by determining the probability that I' contains at least one bad column. Thus, we get

$$1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell} \leq \Pr[I' \text{ is not simple}]$$

and

$$\Pr[I' \text{ is not simple}] \leq 1 - \left(1 - \binom{n}{n/2-1}(q(1-q))^{n/2+1}\right)^{\ell}.$$

Similarly, these bounds exist for the case when n is even. □

3.3 Smoothed Height of the Bounded Search Tree

As previously discussed, there are $O(d^d)$ possible paths in a bounded search tree, denoted as $\mathcal{T}$, corresponding to the solutions traversed by the *Bounded Search Tree Algorithm*. We now bound the size of $\mathcal{T}$ for perturbed instances.

Let P_i be the indicator variable describing whether the ith path in $\mathcal{T}$ results in a center string, *i.e.* $P_i = 1$ if the ith path leads to a center string and $P_i = 0$ otherwise. The algorithm halts when $P_i = 1$. Let $\mathcal{P}_{\mathcal{P}_\backslash=\infty}$ be the number of paths considered until $P_i = 1$ and the algorithm halts.

Lemma 2. *Let $I' = (S', d')$ be a perturbed instance with probability $0 < p \leq \frac{1}{2}$. If $I' = (S', d')$ is not a simple instance, then for sufficiently large ℓ, constant $c > 0$, and when n is between $\log \ell / \log\log \ell$ and $\log \ell$, we have:*

$$\Pr[\mathcal{P} \geq d^{dpc}] \leq \frac{1}{d^{dpc}}.$$

Proof. If $I' = (S', d')$ does not have a center string then *Bounded Search Tree Algorithm* will always return false; otherwise, $P_i = 1$ with some probability. It follows that the expected number of paths that need to be considered is $1/\Pr[P_i = 1]$.

We now calculate $\Pr[P_i = 1]$. If the candidate string s is not equal to 0^ℓ (*i.e.* a center string) then there exists at least one position of s that can be augmented so that $d(s, 0^\ell)$ decreases by one; the probability of this occurring is at least $1/\ell$. Let $Y_k \in \{0, 1, \ldots, d'\}$ be the random variable that corresponds to the Hamming distance between s and 0^ℓ, where k is the number of recursive iterations of the algorithm. The process $Y_0, Y_1, Y_2, \ldots$ is a Markov chain with a barrier at state d' and contains varying time and state dependent transfer probabilities. This process is overly complicated and we instead analyze the following Markov chain: $Z_0, Z_1, \ldots$, where Z_k is the random variable that is equal to the state number after k recursive steps and there exists infinitely many states. Initially, this Markov chain is started like the stochastic process above, *i.e.* $Z_0 = Y_0$. We let $Z_{k+1} = Y_k - 1$ if the process decreases the Hamming distance between s and 0^ℓ by one; otherwise $Z_{k+1} = Y_{k+1}$. After the algorithm halts, we continue with the same transfer probabilities. We can show by induction on k that $Y_k \leq Z_k$ for all k and therefore, $\Pr[P_i = 1]$ is at least $\Pr[\exists t \leq d : Z_t = 0]$.

We made the assumption that S' contains only one center string, however, this assumption is not needed – the random walk may find another center string while

not in the terminating state but this possibility only increases the probability that the algorithm terminates.

Given that the Markov chain starts in state k it can reach a halting state in at least k steps by making transitions through the states $k-1, k-2, \ldots, 1, 0$. The probability of this happening is $(1/\ell)^k$. We now incorporate the possibility that several steps in the"wrong" direction are made in the analysis; "wrong" steps refer to when the candidate string is altered so that the distance between the candidate string and the center string *increases*. Suppose w steps in the Markov chain are taken in wrong direction then $k+w$ steps are needed in the "correct" direction, and therefore, the halting state can be reached in $2w+k$ steps. Let $q(w,k)$ be the probability that $Z_{2w+k} = 0$ such that the halting state is not reached in any earlier set, under the condition that the Markov chain started in state k. More formally,

$$q(w,k) = \Pr[Z_{2w+k} = 0 \text{ and } Z_\alpha > 0 \;\forall\; \alpha < 2w+k \mid Z_0 = k].$$

Clearly $q(0,k) = (1/\ell)^k$, and in the general case $q(w,k)$ is $((\ell-1)/\ell)^w(1/\ell)^{w+k}$ times the number of ways of arranging w wrong steps and $w+k$ correct steps such that the sequence starts in state k, ends in the halting state and does not reach this state before the last step.

By applying the ballot theorem [15] we can deduce that there are $\binom{2w+k}{w}\frac{w}{2w+k}$ possible arrangements of these w wrong steps and $w+k$ correct steps, and the above probability is at least

$$\binom{2w+k}{w}\frac{w}{2w+k}\left(\frac{\ell-1}{\ell}\right)^w\frac{1}{\ell^{w+k}}.$$

This expression is not defined in the case $w = k = 0$, however, it is equal to 1 in this case.

The probability that $P_i = 1$ at the ith path is dependent on the starting position of the Markov chain, which is equal to the number of bad columns. Let X_{bad} be the number of bad columns in S', which is at most $2d$ (by Fact 1). Hence, we get the following:

$$\begin{aligned}
\Pr[P_i = 1] &\geq \sum_{k=1}^{2d}\Pr[X_{bad} = k]\sum_{w=0}^{\frac{1}{2}(i-k)} q(w,k)\\
&\geq (\Pr[X_{bad} \leq 2d] - \Pr[X_{bad} = 0])\sum_{k=1}^{2d}\sum_{w=0}^{\frac{1}{2}(i-k)} q(w,k)\\
&\geq \Pr[I' \text{ is not simple}]\sum_{k=1}^{2d}\sum_{w=0}^{\frac{1}{2}(i-k)} q(w,k)\\
&\geq \left(1-\left(1-(q(1-q))^{n/2+1}\right)^\ell\right)\sum_{k=1}^{2d}\sum_{w=0}^{\frac{1}{2}(i-k)} q(w,k) \qquad \text{[Lemma 1}
\end{aligned}$$

We now aim to find a bound on $q(w,k)$.

$$\begin{aligned}
\Pr[P_i = 1] &\geq \left(1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell}\right) \sum_{w=0}^{\frac{1}{2}(i-1)} \sum_{k=1}^{2d} q(w,k) \\
&\asymp \left(1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell}\right) \sum_{w=0}^{\frac{1}{2}(i-1)} \left(\frac{1}{1-\ell}\right)^{2d+1} \qquad \text{[Fact 2]} \\
&\geq \left(1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell}\right) \frac{1 - \left(\frac{1}{\ell-1}\right)^{id+1}}{1 - \left(\frac{1}{\ell-1}\right)^{2d+1}}
\end{aligned}$$

Hence, for sufficiently large ℓ we have $\Pr[P_i = 1] = 1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell}$ and it follows that:

$$E[\mathcal{P}] = \frac{1}{1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell}}$$

and by Markov inequality recall that for any $c > 0$

$$\begin{aligned}
\Pr[\mathcal{P} \geq d^{dcp}] &\leq \frac{1}{d^{dcp}\left(1 - \left(1 - (q(1-q))^{n/2+1}\right)^{\ell}\right)} \\
&\leq \frac{1}{d^{dcp}\left(1 - \exp(-\ell/(q(1-q))^{n/2+1})\right)}.
\end{aligned}$$

Hence, $\Pr[\mathcal{P} \geq d^{dcp}]$ is equal to $\frac{1}{d^{dcp}}$ for significantly large ℓ and when n is between $\log \ell / \log\log \ell$ and $\log \ell$. □

The following is our main theorem which provides an upper bound on expected number of paths that need to be considered before a center string is found. An important aspect about this result is the small perturbation probability require in comparison to the instance size; the expected number of positions to change in each string is $O(\log \ell)$.

Theorem 2. *For some small $\epsilon > 0$ and perturbation probability $0 \leq p \leq \log \ell / \ell$, the expected running time of "Bounded Search Tree Algorithm" on the perturbed instances is $O(n\ell + nd \cdot d^{2+\epsilon})$ when n is between $\log \ell / \log\log \ell$ and $\log \ell$, and ℓ is sufficiently large.*

Proof. Let $\mathcal{P}'$ be the number of paths considered until a center string is found for a perturbed instance. There are $O(d^{2d})$ possible paths in the search tree corresponding to *Bounded Search Tree Algorithm*. The size of the bounded search tree is equal to zero for simple instances and therefore, we are only required to consider instances that are not simple. For instances that are not simple but

satisfy the conditions of Lemma 2, we use this lemma with $p \leq \frac{2+\epsilon}{dc}$, where $c > 0$, to bound the size of the search tree. Lemma 1, which describes the probability that an instance is not simple, is also used in the following analysis.

$$E[\mathcal{P}] \leq d^{dcp} \cdot \Pr[I' \text{ is not simple}] + \sum_{i=dcp}^{2d} d^i \Pr[\mathcal{P} \geq d^i]$$

$$\leq d^{2+\epsilon} \left(1 - \left(1 - \binom{n}{n/2-1} (q(q-1))^{n/2-1}\right)^{\ell}\right) + \sum_{i=2+\epsilon}^{2d} d^i \Pr[\mathcal{P} \geq d^i]$$

For sufficiently large ℓ and when n is between $\log \ell / \log\log \ell$ and $\log \ell$, we get:

$$E[\mathcal{P}] \leq d^{2+\epsilon} + \sum_{i=2+\epsilon}^{2d} d^i \Pr[\mathcal{P} \geq d^i]$$

Therefore, we have $E[\mathcal{P}] \leq d^{2+\epsilon} + d - 2 - \epsilon$. The expected size of the search tree is at most $o(1) + d^{2+\epsilon}$. It follows form the analysis of Gramm *et al.* [13] that demonstrated each recursive step takes time $O(nd)$ and the preprocessing time takes $O(n\ell)$, that *Bounded Search Tree Algorithm* has expected running time of $O(n\ell + nd \cdot d^{2+\epsilon})$. □

We note that we require ℓ to be sufficiently large, however, this restriction is not significant since we require $\ell \geq 10$. As previously mentioned, the problem can be solved efficiently when ℓ is relatively small (*i.e.* $\ell \leq 10$), even the trivial algorithm that tries all $|\Sigma|^{\ell}$ can be used for these instances.

Acknowledgements. The author would like to thank Professor Bin Ma for his discussions and insights concerning the results presented in this paper and Professor Ming Li for suggesting this area of study. The author is supported by NSERC Postdoctoral Fellowship, NSERC Grant OGP0046506, NSERC Grant OGP0048487, Canada Research Chair program, MITACS, and Premier's Discovery Award.

References

1. Andoni, A., Indyk, P., Patrascu, M.: On the optimality of the dimensionality reduction method. In: Proc. of FOCS, pp. 449–456 (2006)
2. Andoni, A., Krauthgamer, R.: The smoothed complexity of edit distance. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 357–369. Springer, Heidelberg (2008)
3. Banderier, C., Beier, R., Mehlhorn, K.: Smoothed analysis of three combinatorial problems. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 198–207. Springer, Heidelberg (2003)

4. Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus strings. In: Hein, J., Apostolico, A. (eds.) CPM 1997. LNCS, vol. 1264, pp. 247–261. Springer, Heidelberg (1997)
5. Chen, Z.-Z., Ma, B., Wang, L.: A three-string approach to the closest string problem. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 449–458. Springer, Heidelberg (2010)
6. Deng, X., Li, G., Li, Z., Ma, B., Wang, L.: Genetic design of drugs without side-effects. SIAM Journal on Computing 32(4), 1073–1090 (2003)
7. Dopazo, J., Rodríguez, A., Sáiz, J.C., Sobrino, F.: Design of primers for PCR amplification of highly variable genomes. Computer Applications in the Biosciences 9, 123–125 (1993)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Fellows, M.R., Gramm, J., Niedermeier, R.: On the parameterized intractability of CLOSEST SUBSTRING and related problems. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 262–273. Springer, Heidelberg (2002)
10. Fellows, M.R., Gramm, J., Niedermeier, R.: On the parameterized intractability of motif search problems. Combinatorica 26, 141–167 (2006)
11. Frances, M., Litman, A.: On covering problems of codes. Theory of Computing Systems 30(2), 113–119 (1997)
12. Gramm, J., Niedermeier, R., Rossmanith, P.: Exact solutions for CLOSEST STRING and related problems. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, p. 441. Springer, Heidelberg (2001)
13. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica 37(1), 25–42 (2003)
14. Hufsky, F., Kuchenbecker, L., Jahn, K., Stoye, J., Böcker, S.: Swiftly computing center strings. BMC Bioinformatics 12(106) (2011)
15. Konstantopoulos, T.: Ballot theorems revisited. Statistics and Probability Letters 24, 331–338 (1995)
16. Lanctot, J.K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Information and Computation 185(1), 41–55 (2003)
17. Li, M., Ma, B., Wang, L.: Finding similar regions in many strings. Journal of Computer and System Sciences 65(1), 73–96 (2002)
18. Lokshtanov, D., Marx, D., Saurabh, S.: Slightly superexponential parameterized problems. In: Proc. of the 22nd SODA, pp. 760–776 (2011)
19. Lucas, K., Busch, M., Össinger, S., Thompson, J.A.: An improved microcomputer program for finding gene- and gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. Computer Applications in the Biosciences 7, 525–529 (1991)
20. Ma, B.: Why greedy works for shortest common superstring problem. In: Proc.of CPM, pp. 244–254 (2008)
21. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. SIAM Journal on Computing 39, 1432–1443 (2009)
22. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics 18(3), 440–445 (2002)
23. Manthey, B., Reischuk, R.: Smoothed analysis of binary search trees. Theoretical Computer Science 3(378), 292–315 (2007)
24. Pavesi, G., Mauri, G., Pesole, G.: An algorithm for finding signals of unknown length in DNA sequences. Bioinformatics 214, S207–S214 (2001)

25. Pevzner, P., Sze, S.: Combinatorial approaches to finding subtle signals in DNA strings. In: Proc.of 8th ISMB, pp. 269–278 (2000)
26. Proutski, V., Holme, E.C.: Primer master: A new program for the design and analyiss of PCR primers. Computer Applications in the Biosciences 12, 253–255 (1996)
27. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: why the simplex algorithm ususally takes polynomial time. Journal of the ACM 51, 296–305 (2004)
28. Tompa, M., et al.: Assessing computational tools for the discovery of transcription factor binding sites. Nature 23(1), 137–144 (2005)
29. Wang, L., Zhu, B.: Efficient algorithms for the closest string and distinguishing string selection problems. In: Proc. of 3rd FAW, vol. 270, pp. 261–270 (2009)
30. Zhao, R., Zhang, N.: A more efficient closest string algorithm. In: Proc. of 2nd BICoB, pp. 210–215 (2010)

Solving Analytic Differential Equations in Polynomial Time over Unbounded Domains

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

[1] Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France
Olivier.Bournez@lix.polytechnique.fr
[2] CEDMES/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal
dgraca@ualg.pt
[3] SQIG /Instituto de Telecomunicações, Lisbon, Portugal
[4] Ecole Normale Supérieure de Lyon, France
Amaury.Pouly@ens-lyon.fr

Abstract. In this paper we consider the computational complexity of solving initial-value problems defined with analytic ordinary differential equations (ODEs) over *unbounded* domains of $\mathbb{R}^n$ and $\mathbb{C}^n$, under the Computable Analysis setting. We show that the solution can be computed in polynomial time over its maximal interval of definition, provided it satisfies a very generous bound on its growth, and that the function admits an analytic extension to the complex plane.

1 Introduction

We consider the following initial-value problem defined by an ODE

$$\begin{cases} \dot{x}(t) = f(x(t)) \\ x(0) = x_0 \end{cases} \tag{1}$$

where f is defined on some (possibly unbounded) domain.

In this paper we show that if $f : \mathbb{R}^n \to \mathbb{R}^n$ admits an analytic extension to $\mathbb{C}^n$ and $x : \mathbb{R} \to \mathbb{R}^n$ admits an analytic extension to $\mathbb{C}$ and both satisfy a very generous assumption concerning their growth, the solution of (1) can be computed in polynomial time from f and x_0 over $\mathbb{R}$. The notion of computability we use is that of Ko [1]. Actually, our constructions also work when considering solutions over $\mathbb{C}$ and assuming $f : \mathbb{C}^n \to \mathbb{C}^n$ analytic. Notice that, as it is well known, Equation (1) covers the case of an ODE of type $\dot{x} = f(t, x)$, as this latter case can be reduced to (1) by using a new variable x_{n+1} satisfying $x'_{n+1} = 1$.

Motivation 1 & Digression: Analog models of computation. We obtained our results by trying to understand whether analog continuous-time models of computation do satisfy (some variant) of the Church-Turing thesis: since such systems can usually be described by particular classes of ordinary differential equations (ODEs), understanding whether these models can compute more than Turing machines is equivalent to understanding whether ODEs can always be simulated by Turing machines.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 170–181, 2011.

For example, the most well known example of analog model of computation is the General Purpose Analog Computer (GPAC) introduced by Claude Shannon in [2] as the idealization of an analog computer, the Differential Analyzer [3]. Shannon worked as an operator early in his career on these machines.

As it can be proved [2,4] that any GPAC can be described by an ordinary differential equation of the form of (1) with f componentwise polynomial, proving that the GPAC does satisfy the Church-Turing thesis is equivalent to proving that solutions of such an ODE are always computable. It has been proved only recently that this holds [5], [6]. Hence, the GPAC does satisfy the Church-Turing thesis. Notice that computability of solutions doesn't hold for general f [7], since uncomputability results can be obtained when the system is "ill behaved" (e.g. non-unique solutions in [7]). These kind of phenomena does not appear in models physically inspired by real machines like the GPAC.

Here, we are dealing with the next step. Do analog models like the GPAC satisfy the *effective* (in the sense of computable complexity) version of Church Turing thesis: all (sufficiently powerful) "*reasonable*" models of computation with "*reasonable*" measure of time are polynomially equivalent. In other words, we want to understand whether analog systems can provably (not) compute faster than usual classical digital models like Turing machines.

Taking time variable t as a measure of time (which is the most natural measure), to prove that the GPAC cannot compute more than Turing machines would require to prove that solutions of ODE (1) are always computable (in the classical sense) in a time polynomial in t, for f (componentwise) polynomial.

We here don't get exactly this result: for f componentwise polynomial, corresponding to GPACs, y is clearly analytic. We have to suppose furthermore that y admits an analytic extension to $\mathbb{C}$. Although this case is stronger than when y is real analytic (it is well known that analyticity on the complex plane implies analyticity over the real line, but that the converse direction does not hold), we believe that our results are interesting on their own and provide a promising step towards the case of the real line.

Motivation 2: Recursive analysis. The results obtained in this paper turn out to be new and not known in a recursive analysis or classical computability or complexity perspective: see related work section.

Being able to compute efficiently solutions of general ordinary differential equations is clearly of interest. Observe that all usual methods for numerical integrations (including Euler's method, Runge Kutta's methods, ...) do not provide the value of $x(t)$ in a time polynomial in t, whereas our algorithm does for analytic functions which satisfy our hypothesis. Actually, as all these numerical methods falls in the general theory of n-order methods for some n, it is possible to use this theory (developed for example in [8]) to prove that none of them produces the value of $x(t)$ in a time polynomial in t. This has already been observed in [9] which claims to overcome this limitation for some classes of functions by using methods of order n with n depending on t, but without a full proof. We do not use this idea but prove that it is indeed possible to produce $x(t)$ in a time polynomial in t.

2 Related Work and Discussions

Typically the ODE (1) is considered over a subset of $\mathbb{R}^n$. It is well-known in mathematics that its solution exists whenever f is continuous (Peano's existence theorem), and is unique whenever f is Lipschitz (Picard or Cauchy-Lipschitz's theorem).

Considering computability, it is well-known that solutions of (1) are computable (in the sense of computable analysis) provided f is Lipschitz. To prove this result one can basically implement an algorithm which simulates Picard's classical method of successive approximations used in the proof of the fundamental existence-uniqueness theorem for (1), which assumes the existence of a Lipschitz condition (see e.g. [10]).

However, assuming f to be Lipschitz often restricts in practice f to be C^1 and defined on a bounded domain, or to have, at most, linear growth on an unbounded domain, which is not really a very satisfactory result.

To avoid the limitations pointed out above, in [5] the authors introduced the idea of effectively locally Lipschitz functions (functions which are locally Lipschitz and for which the local Lipschitz constants can be computed) and showed that if f is effectively Lipschitz, then the solution of (1) is computable over the maximal interval in which the solution is defined. Another related result can be found in [11] where the author proves computability of solutions of (1) in unbounded domains without requiring the use of Lipschitz constants. However Ruohonen requires a very restrictive bound on the growth of f.

In general, if f is continuous, Peano's existence theorem guarantees that at least a solution exists for (1). The problem is that the condition that f is continuous does not guarantee a unique solution. In [6] the authors show that the solution of (1) is computable in its maximal interval of definition if f is continuous and the solution of (1) is unique.

But what about *computational complexity*? The procedure presented in [6] relies on an exhaustive search and, as the authors mention (p. 11): "Of course, the resulting algorithms are highly inefficient in practice".

In the book [1] several interesting results are proved. For instance it is shown (Theorem 7.3) that there are (continuous) polynomial-time computable functions f such that (1) has a unique solution, but which can have arbitrarily high complexity. However this result follows because we do not require that f satisfies a Lipschitz condition. If f satisfies a Lipschitz condition and is polynomial-time computable, then an analysis of Euler's algorithm shows that, on a bounded domain, the solution for (1) is computable in polynomial space. It is also shown that if f satisfies a weak form of the Lipschitz condition and is polynomial-time computable, the solution to (1) is polynomial-time computable iff $P = PSPACE$ (again on a bounded domain). In [12] this result is extended and the author shows that initial-value problems given by a polynomial-time computable, Lipschitz continuous function can have a polynomial-space complete solution.

So it seems that the solution to (1) cannot be computed in polynomial time for Lipschitz functions in general. But what if we require more conditions on f? In particular, if we require f to be analytic, what is the computational complexity

of the solution? Will it be polynomial-time? This question cannot be answered by analyzing classical methods for solving ODEs (e.g. Euler's algorithm), since they do not use the assumption of analyticity. Instead, other techniques which explicitly use this assumption must be used.

Restricting to analytic functions is natural as this is indeed a natural class of functions, and as it is sometimes observed that functions coming from our physical world, are mostly analytic functions.

In [13], [14] the authors show that, *locally*, the solution is polynomial-time computable if f is (complex) analytic. However, Müller's construction relies on the highly non-constructive Heine-Borel theorem. This makes this results less convincing because although it guarantees the solution can be computed in polynomial-time, it gives no algorithm to compute it. Also it gives no insight on what happens on a broader domain, e.g. $\mathbb{C}^n$.

In this paper we study computability of (1) when f is analytic. Instead of taking analytic functions f defined over $\mathbb{R}^n$, our results will be for analytic functions with extensions to $\mathbb{C}^n$ (also known as holomorphic functions). The reasons of taking $\mathbb{C}^n$ and not $\mathbb{R}^n$ are twofold.

First, $\mathbb{C}^n$ is a broader domain than $\mathbb{R}^n$ and it is natural to generalize the results there. When the time variable is defined in the real line, existence and uniqueness results for ODEs defined over $\mathbb{R}^n$ are translated in the same way for ODEs on $\mathbb{C}^n$ [15], [16], as well as the results we prove here.

Second, some of our results rely on the use of the Cauchy integral formula, which assumes analytic functions over $\mathbb{C}^n$ which is a stricter condition than being real analytic (holomorphic functions, when restricted to $\mathbb{R}^n$, always originate analytic functions, but analytic functions over $\mathbb{R}^n$ may not have an holomorphic extension defined on the whole complex set $\mathbb{C}^n$). Therefore our results, in the case of $\mathbb{R}^n$, are not enough to capture the full class of analytic functions (over $\mathbb{R}^n$) but are still strong enough to capture ODEs defined with most of the "usual" functions: e^x, $\sin, \cos$, polynomials, etc. It would be interesting to know if these results can be fully extended to analytic functions defined over $\mathbb{R}^n$, but we have not yet obtained any result on this topic. Rather, we see the complex case as a preliminary approach for getting closer to the real case. Indeed, knowing that f is complex analytic is a stronger condition than only knowing that f is real analytic, which gives us more tools to work with, namely the Cauchy integral formula.

Organization of the paper. The organization of the paper is as follows. Section 3 presents background material about Computable Analysis, which will be needed in Section 4 to state the main result. We then proceed in the following sections with its proof.

3 Computable Analysis

Computable Analysis is an extension of the classical theory of computation to sets other than $\mathbb{N}$ due to Turing [17], Grzegorczyk [18], and Lacombe [19]. Several equivalent approaches can be used (proof of equivalence can be found [20])

for considering computability over $\mathbb{R}^n$: using Type-2 machines [20], using oracle Turing machines [1], or using modulus of continuity [21], [1], among other approaches.

In this paper we will use the approach of Ko in [1], based on oracle Turing machines. The idea underlying [1] to compute over $\mathbb{R}^n$ is to encode each element $x \in \mathbb{R}^n$ as a Cauchy sequence of "simple rationals" with a known "simple" rate of convergence. In [1] Ko uses sequences of *dyadic rational numbers*, i.e. rationals of the type $\frac{m}{2^n}$ for some $m \in \mathbb{Z}$ and $n \in \mathbb{N}$. Then a sequence of dyadic rational numbers $\{d_n/2^n\}_{n\in\mathbb{N}}$ represents a real number x if $|x - d_n/2^n| \leq 2^{-n}$ for all $n \in \mathbb{N}$. It is easy to represent points of $\mathbb{R}^k$ using dyadic sequences (use k sequences of dyadic rationals, each coding a component of x). Since $\mathbb{C} \simeq \mathbb{R}^2$, this approach can be used to compute with elements of $\mathbb{C}$. Note that what defines a sequence of dyadic rational numbers $\{d_n/2^n\}_{n\in\mathbb{N}}$ is the sequence $\{d_n\}_{\mathbb{N}}$, which is nothing more than a function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = d_n$. Therefore one can define the notion of computable point of $\mathbb{R}$: it is a point which can be coded by a sequence $\{d_n/2^n\}_{\mathbb{N}}$ such that the function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = d_n$ is computable. By other words, a computable point is a point for which we can compute an arbitrary rational approximation in finite time. Similarly one can define computable points of $\mathbb{R}^k$ and $\mathbb{C}^k$. Ko also deals with complexity: a point x is polynomial-time computable if one can compute a dyadic rational which approaches x with precision $\leq 2^{-n}$ in time polynomial in n.

Having worked with computability of points of $\mathbb{R}^n$ and $\mathbb{C}^n$, one can also define computability of functions $f : \mathbb{R}^k \to \mathbb{R}^j$ and $g : \mathbb{C}^k \to \mathbb{C}^j$. In essence, a function f is computable if there is some oracle Turing machine that, using as oracles functions which encode the argument x of f and as input a number $n \in \mathbb{N}$, it can compute in finite time a rational approaching $f(x)$ with precision 2^{-n}. Similarly, if this rational approximation can be computed in time polynomial in n, we say that f is polynomial-time computable. Precise details of this discussion can be found in [1].

4 Main Result

Let $f : \mathbb{C}^d \to \mathbb{C}^d$ be an analytic function on $\mathbb{C}^d$ and $t \in \mathbb{R}, x_0 \in \mathbb{C}^d$. We are interested in computing the solution of the initial-value problem

$$\begin{cases} \dot{x}(t) = f(x(t)) \\ x(t_0) = x_0. \end{cases} \tag{2}$$

It is well-known that if f is analytic then (2) has a unique solution which is analytic on its maximum life interval. We are interested in obtaining sufficient conditions that guarantee $x(t)$ to be polynomial-time computable.

4.1 Necessary Condition: Poly-boundedness

We first observe an easy necessary condition: if $x(t)$ is polynomial-time computable, then $x(t)$ cannot grow too fast, as a Turing machine cannot write more than t symbols in time t. Formally, we introduce the following concept.

Definition 1 (Poly-bounded function). *A function $f : \mathbb{C}^d \to \mathbb{C}^{d'}$ is* poly-bounded *(or p-*poly-bounded*) iff there is a polynomial p such that*

$$\forall x \in \mathbb{C}^d \setminus \{0\}, \|f(x)\| \leqslant 2^{p(\log_2 \lceil \|x\| \rceil)}. \tag{3}$$

Without loss of generality, we can assume that p is an increasing function on $\mathbb{R}_0^+$ (replace each coefficient in polynomial p by its absolute value if needed). We then get the following theorem:

Theorem 1. *If $f : \mathbb{C}^d \to \mathbb{C}^{d'}$ is polynomial-time computable, then f is poly-bounded.*

4.2 Sufficient Condition: Our Main Result

Our main result can be formulated as follows:

Theorem 2 (Main result). *Let $x(t)$ be the solution of the initial-value problem (2). Assume that*

- *f is analytic and polynomial-time computable on $\mathbb{C}^d$;*
- *x_0 is a polynomial-time computable vector of complex numbers*
- *t_0 is a polynomial-time computable real number*
- *function $x(t)$ admits an analytic extension to $\mathbb{C}$ and is poly-bounded over $\mathbb{C}$*

then the function $x(t)$ is polynomial-time computable.

Actually, we can even say more – the transformation is effective, if one adds the hypothesis that f is also poly-bounded.

Theorem 3 (Main result: Effective version). *Fix a polynomial p. Keep the same hypothesis of Theorem 2, but in addition, restrict to functions f that are p-poly-bounded.*

Then the transformation is effective and even polynomial-time computable: the functional that maps f, x_0, t_0, and t to function $x(t)$ is polynomial-time computable.

Remark 1. From Theorem 1, even if f is not assumed poly-bounded, we know it is p-poly-bounded for some p, as it is assumed polynomial-time computable. However, the problem is that we cannot compute in general such polynomial p from f, and hence we have to restrict Theorem 3 to functions f with given p.

The whole idea behind the proof of above theorem is to compute the solution of (2) in polynomial time in some fixed neighborhood of x_0, using Picard's classical method of successive approximations. From this solution we can compute the coefficients of its Taylor series expansion, which allow us to compute the solution on its maximal interval of definition using the hypothesis of poly-boundedness. All the construction can be done in polynomial time. A sketch of proof is presented in the following two sections.

4.3 Extension

Theorem 2 requires a strong condition on the solution: it needs to be analytic over $\mathbb{C}$. This can be too much of a requirement since even a simple function like $\frac{1}{1+x^2}$ doesn't satisfy our hypothesis. However at the expense of a small trick one can extend this result to functions having a finite number of poles over $\mathbb{C}$.

Theorem 4. *Keep the same hypothesis as in Theorem 2 except that x is assumed analytic over $U = \mathbb{C}^d \backslash \{a_1, \ldots, a_n\}$ where $a_1, \ldots, a_n$ are **poles** of order $k_1, \ldots, k_n$ of x. Assume that $y(z) = x(z) \prod_{i=1}^{n} (z - a_i)^{k_i}$ is poly-bounded and that the a_i are polynomial-time computable. Then x is polynomial-time computable over U.*

Proof (Sketch). The idea is that if x has a pole of order k on a then $(z-a)^k x(z)$ has a removable singularity. By repeating this trick for every pole, one can build a new function which is analytic over $\mathbb{C}$. Furthermore this function is still a solution of a IVP. To compute the initial function from the new one, it is sufficient to divide by a polynomial, which doesn't change the complexity.

5 On Analytic Functions

We first need to state some basic facts about analytic functions in order to be convinced that the complexity of computing an analytic function is the same as the complexity of computing the coefficients of its Taylor series. This is the purpose of the current section.

5.1 From the Function to the Taylor Series

The following theorem is known.

Theorem 5 ([22], [23]). *If f is complex analytic and polynomial-time computable on a neighborhood of x_0, where x_0 is a polynomial-time computable complex number, then the sequence of its Taylor series coefficients at x_0 is polynomial-time computable.*

This holds for one and multi-dimensional functions. We will actually use the following variant of the theorem, obtained by observing that if f is analytic on $\mathbb{C}^d$, then f is analytic on a neighborhood of x_0 and if f is polynomial-time computable on $\mathbb{C}^d$, then f is polynomial-time computable on a neighborhood of x_0, and that the proof of [23] is rather effective.

Theorem 6. *If f is analytic on $\mathbb{C}^d$ and polynomial-time computable on $\mathbb{C}^d$, then the sequence of coefficients $\{a_\alpha\}_\alpha$ of its Taylor series at x_0, where x_0 is a polynomial-time computable complex number, is polynomial-time computable.*

Fix a polynomial p, and restrict to functions f p-poly-bounded: The functional that maps f, x_0, and α to the corresponding coefficient a_α is polynomial-time computable.

5.2 From the Taylor Series to the Function

Theorem 6 is important because it allows us to go from the function to its coefficients. But it is only interesting if we can have the converse, that is if we can go from the coefficients to the function.

The next theorem gives sufficient conditions so that this can happen. A similar theorem is already proved in [23] for the case of a polynomial-time computable function on a compact set. However, since we consider functions defined on unbounded sets over $\mathbb{C}^d$, this requires a different proof.

Theorem 7. *Suppose $f : \mathbb{C}^d \to \mathbb{C}$ is analytic and poly-bounded on $\mathbb{C}^d$ and that the sequence $\{a_\alpha\}$ of its Taylor series at x_0, where x_0 is a polynomial-time computable complex number, is polynomial-time computable. Then f is polynomial-time computable on $\mathbb{C}^d$.*

Even if we can't pinpoint a polynomial p satisfying a poly-boundedness condition for f, the mere knowledge that f is poly-bounded allows us to conclude that f can be computed in polynomial time, by using the previous proof. In this case, we do not know a precise polynomial bound on the time complexity for computing f, but we do know that such bound exists.

6 Proof of Main Result

6.1 The Special Case of Integration

We first state a basic result for the case of integration: observe that integration can be considered as a very specific case of our general theorem.

Theorem 8. *If f is analytic, poly-bounded on $\mathbb{C}$, polynomial-time computable, and x_0 is a polynomial-time computable complex number, then*

$$g(x) = \int_{\gamma_x} f(z)dz \quad \textit{where} \quad \gamma_x = \begin{cases} [0,1] \to & \mathbb{C} \\ t & \mapsto (1-t)x_0 + tx \end{cases}$$

is analytic, poly-bounded and polynomial-time computable on $\mathbb{C}$.

Moreover, if one fixes a polynomial p and considers only functions f which are p-poly-bounded, then the transformation is effective and even polynomial-time computable: the functional that maps f, x_0 and x to $g(x)$ is polynomial-time computable

We remark that the previous theorem implies that the transformation which computes $g(x) = \int_0^x f(z)dz$ for $x \in \mathbb{R}$ is also computable. Again, we can go to the version where we don't have explicit knowledge of the polynomial which yields poly-boundedness for f.

6.2 On Lipschitz Constants

We will need a result about analytic functions (mainly derived from multi-dimensional Cauchy integral formula) that are poly-bounded.

Proposition 1. *If $f : \mathbb{C}^d \to \mathbb{C}^e$ is analytic and p-poly-bounded then for each $R > 0$ there is a $K(R) > 0$ such that*

$$\forall x, y, \|x\|, \|y\| \leqslant R \Rightarrow \|f(x) - f(y)\| \leqslant K(R)\|x - y\|$$

with

$$K(R) \leqslant 2^{q(\log_2 \lceil R \rceil)}$$

where $q(x) = p(2 + 4x) + A_d$ and A_d is a polynomial-time computable constant in d.

6.3 Proof of Theorem 3

We can now present the proof of Theorem 3. Theorem 2 is clearly a corollary of it, forgetting effectivity.

We can assume, without loss of generality, that $t_0 = 0$ and $x_0 = 0$. Consider the following operator

$$W(u)(t) = \int_0^t f(u(\xi))d\xi.$$

Because z is a solution of (2) we easily have

$$W(z)(t) = \int_0^t f(z(\xi))d\xi = z(0) + \int_0^t \dot{z}(\xi)d\xi = z(t)$$

Thus z is a fixed point of W. Now consider the following sequence of functions

$$\begin{cases} z_0(t) = 0 \\ z_{n+1} = W(z_n). \end{cases}$$

Obviously z_0 is analytic. Furthermore, one can easily show by induction (using Theorem 8) that for all $n \in \mathbb{N}$, z_n is analytic and polynomial-time computable. More importantly, one can compute effectively $z_n(t)$ in polynomial time in n. Indeed, it is just the iteration of the constructive part of Theorem 8.

Now the crucial idea is that z_n uniformly converges to z but only on a (really small) compact set near 0. Using this result we will use Theorem 5 to extract the coefficients of z and by using the hypothesis on the boundedness of z we will obtain z.

First of all, we need a uniform bound of z_n (in n). We already know, by hypothesis, that

$$\|z(t)\| \leqslant 2^{p(\log_2 \lceil |t| \rceil)}.$$

Now apply Proposition 1 to f. Let s be a polynomial such that f is s-poly-bounded and let q be the polynomial of Corollary 1 such that

$$\forall R > 0, \forall x, y \in \mathbb{C}^d, \|x\|, \|y\| \leqslant R \Rightarrow \|f(y) - f(x)\| \leq K(R)\|x - y\| \quad (4)$$

where $K(R) = 2^{q(\log_2 \lceil R \rceil)}$. Let $M = 2^{p(0)}$,$R = 2M$, $T = \frac{1}{2K(R)}$ so that

$$|t| \leqslant 1 \Rightarrow \|z(t)\| \leqslant M \quad (5)$$

We will show by induction that

$$|t| \leqslant T \Rightarrow \|z_n(t) - z(t)\| \leqslant 2^{-n}M. \tag{6}$$

This is trivial for $n = 0$ because $z_0(t) = 0$ so if $|t| \leqslant T$ then $|t| \leqslant 1$ (we assume, without loss of generality, $p(0) \geq 0$ which implies $R \geq 2$, and $q(0) \geq 1$, which implies $T \leqslant 1$) and, by (5) $\|z_0(t) - z(t)\| = \|z(t)\| \leqslant M$.

For $n > 0$, suppose that $|x| \leqslant T$. Then

$$\|z_{n+1}(x) - z(x)\| \leqslant \int_0^1 \|f(z_n(tx)) - f(z(tx))\| x dt$$

But now recall that:

- $\|z(x)\| \leqslant M \leqslant R = 2M$ by definition
- $\|z_n(x)\| \leqslant \|z(x)\| + \|z_n(x) - z(x)\| \leqslant M + 2^{-n}M \leqslant 2M = R$

So we can apply (4) and obtain

$$\begin{aligned}\|z_{n+1}(x) - z(x)\| &\leqslant \int_0^1 K(R)\|z_n(tx) - z(tx)\| x dt \\ &\leqslant 2^{-n-1}M.\end{aligned}$$

Now that we have (6), the problem is easy because we can uniformly approximate z on $\overline{B(0,T)}$ with an arbitrary precision which is exponential on the number of steps. To put it differently, we proved that z is polynomial-time computable on $\overline{B(0,T)}$. Notice that in $\overline{B(0,T)}$ both z and z_n are bounded by $2M$, which can be computed in polynomial time from p. Hence z and z_n are poly-bounded by the same (constant) polynomial on $\overline{B(0,T)}$ (the behavior outside this interval is irrelevant for our considerations) and from Theorem 8, the same polynomial time bound can be used to compute all the z_n and z, thus avoiding the potential problem of having increasing (polynomial) time with n (e.g. doubling in each increment of n) which could yield to overall computing time more than polynomial.

We can now use a mix of Theorem 5 and Theorem 6 to get the fact that we can compute the Taylor series of z in 0 in polynomial time (indeed, we only need to know how to compute z on a open ball around 0). And now, applying Theorem 7, we know that z is polynomial-time computable because we know by hypothesis that it is poly-bounded.

Furthermore, the whole process is polynomial-time computable because we gave explicit bounds on everything and then it is just a matter of iterating a function and applying two operations on Taylor series at the end.

7 Conclusion

In this paper we have studied the computational complexity of solving initial-value problems involving analytic ordinary differential equations (ODEs). We

gave special importance to solutions defined on unbounded domains, where the traditional assumption of numerical analysis – Lipschitz condition for the function defining the ODE – is no longer valid, making the analysis of the system non-trivial.

We have shown that if the solution has a bound on its growth – poly-boundedness – then the solution of the initial-value problem can be computed in polynomial time as long as f in (2) admits an analytic extension to $\mathbb{C}^d$.

Although the poly-boundedness condition is very generous and encompasses "usual" ODEs, it would be interesting to know if we can substitute the poly-boundedness condition by a more natural one. Note that some kind of assumption over the polynomial differential equations must be used, since their solutions can be, for example, a function of the type

$$2^{2^{\cdots^{2^x}}}$$

(see e.g. [24]) which is not poly-bounded and hence not polynomial-time computable by Corollary 1.

A topic for further work concerns the computational complexity of solving partial differential equations. This is quite interesting since research from Mills et al. suggest that from a complexity point of view, the EAC mentioned in the introduction may beat the Turing machine. It would be a significative hallmark for the EAC if one could decide theoretically if the EAC may or may not have super-Turing power for certain tasks, from a computational complexity perspective. However this problem seems to be quite difficult due to the lack of theoretical tools which might help us to settle the question. For instance, despite huge efforts from the scientific community, no existence-uniqueness theorem is known for partial differential equations, even for certain subsets like Navier-Stokes equations.

Acknowledgments. This work has been partially supported by the INRIA program "Équipe Associée" Comput$\mathbb{R}$. O. Bournez and A. Pouly were supported by ANR project SHAMAN. D. Graça was partially supported by *Fundação para a Ciência e a Tecnologia* and EU FEDER POCTI/POCI via SQIG - Instituto de Telecomunicações.

References

1. Ko, K.I.: Computational Complexity of Real Functions. Birkhäuser, Basel (1991)
2. Shannon, C.E.: Mathematical theory of the differential analyzer. J. Math. Phys. 20, 337–354 (1941)
3. Bush, V.: The differential analyzer. A new machine for solving differential equations. J. Franklin Inst. 212, 447–488 (1931)
4. Graça, D.S., Costa, J.F.: Analog computers and recursive functions over the reals. J. Complexity 19(5), 644–664 (2003)
5. Graça, D., Zhong, N., Buescu, J.: Computability, noncomputability and undecidability of maximal intervals of IVPs. Trans. Amer. Math. Soc. 361(6), 2913–2927 (2009)

6. Collins, P., Graça, D.S.: Effective computability of solutions of differential inclusions — the ten thousand monkeys approach. Journal of Universal Computer Science 15(6), 1162–1185 (2009)
7. Pour-El, M.B., Richards, J.I.: A computable ordinary differential equation which possesses no computable solution. Ann. Math. Logic 17, 61–90 (1979)
8. Demailly, J.-P.: Analyse Numérique et Equations Différentielles. Presses Universitaires de Grenoble (1991)
9. Smith, W.D.: Church's thesis meets the N-body problem. Applied Mathematics and Computation 178(1), 154–183 (2006)
10. Perko, L.: Differential Equations and Dynamical Systems, 3rd edn. Springer, Heidelberg (2001)
11. Ruohonen, K.: An effective Cauchy-Peano existence theorem for unique solutions. Internat. J. Found. Comput. Sci. 7(2), 151–160 (1996)
12. Kawamura, A.: Lipschitz continuous ordinary differential equations are polynomial-space complete. In: 2009 24th Annual IEEE Conference on Computational Complexity, pp. 149–160. IEEE, Los Alamitos (2009)
13. Müller, N., Moiske, B.: Solving initial value problems in polynomial time. In: Proc. 22 JAIIO - PANEL 1993, Part 2, pp. 283–293 (1993)
14. Müller, N.T., Korovina, M.V.: Making big steps in trajectories. Electr. Proc. Theoret. Comput. Sci. 24, 106–119 (2010)
15. Birkhoff, G., Rota, G.C.: Ordinary Differential Equations, 4th edn. John Wiley, Chichester (1989)
16. Coddington, E.A., Levinson, N.: Theory of Ordinary Differential Equations. McGraw-Hill, New York (1955)
17. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. (Ser.2–42), 230–265 (1936)
18. Grzegorczyk, A.: On the definitions of computable real continuous functions. Fund. Math. 44, 61–71 (1957)
19. Lacombe, D.: Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III. C. R. Acad. Sci. Paris 241, 151–153 (1955)
20. Weihrauch, K.: Computable Analysis: an Introduction. Springer, Heidelberg (2000)
21. Pour-El, M.B., Richards, J.I.: Computability in Analysis and Physics. Springer, Heidelberg (1989)
22. Ko, K.I., Friedman, H.: Computational complexity of real functions. Theoret. Comput. Sci. 20, 323–352 (1982)
23. Müller, N.T.: Uniform computational complexity of taylor series. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 435–444. Springer, Heidelberg (1987)
24. Graça, D.S., Campagnolo, M.L., Buescu, J.: Computability with polynomial differential equations. Adv. Appl. Math. 40(3), 330–349 (2008)

Pattern-Guided Data Anonymization and Clustering

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. A matrix M over a fixed alphabet is k-anonymous if every row in M has at least $k-1$ identical copies in M. Making a matrix k-anonymous by replacing a minimum number of entries with an additional $\star$-symbol (called "suppressing entries") is known to be NP-hard. This task arises in the context of privacy-preserving publishing. We propose and analyze the computational complexity of an enhanced anonymization model where the user of the k-anonymized data may additionally "guide" the selection of the candidate matrix entries to be suppressed. The basic idea is to express this by means of "pattern vectors" which are part of the input. This can also be interpreted as a sort of clustering process. It is motivated by the observation that the "value" of matrix entries may significantly differ, and losing one (by suppression) may be more harmful than losing the other, which again may very much depend on the intended use of the anonymized data. We show that already very basic special cases of our new model lead to NP-hard problems while others allow for (fixed-parameter) tractability results.

1 Introduction

The notion of k-anonymity is a basic concept in privacy-preserving data publishing [9]. An $n \times m$-matrix M—called "table" in database theory—over a fixed alphabet is called k-anonymous if for every row r in M there are at least $k-1$ further rows in M that are identical with r. The intuitive idea motivating this notion is that if each row in M contains data about a distinct person, and if M is k-anonymous, then it is hard to identify the data row corresponding to some specific individual [14]. Clearly matrices are, in general, not k-anonymous for any $k \geq 2$. It is NP-hard to make a given matrix k-anonymous by suppressing a minimum number of entries [2,11], that is, by replacing a minimum number of matrix entries with the $\star$-symbol. However, in the classical scenario it remains unspecified whether certain entries are less harmful to suppress

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

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 182–193, 2011.

than others.[1] Here, we present a simple combinatorial model that allows the user of the anonymized data to specify, as part of the input, which row entries (respectively, which combinations of row entries) may be suppressed in order to achieve k-anonymity. Studying the computational complexity, we identify both tractable and intractable cases of the underlying combinatorial problem which allows for user-specified "anonymization patterns".

Sweeney [13], who pioneered the notion of k-anonymity, pointed out that in the context of k-anonymization it is desirable to guide the process of entry suppression. We convert this idea into a formal model where the end-user of anonymized data specifies a number of pattern vectors from $\{\square, \star\}^m$, where m is the number of columns of the underlying matrix. A pattern vector $v \in \{\square, \star\}^m$ is associated with a set of matrix rows fulfilling the following condition: If the i^{th} pattern vector entry is a $\square$-symbol, then all rows associated with this pattern must have identical symbols at this position; they may differ in other positions. The corresponding minimization problem, which we refer to as Pattern Clustering, is to find a mapping of matrix rows to pattern vectors such that sanitizing[2] the rows according to their mapped pattern vectors makes the matrix k-anonymous with a minimum number of suppressions. Refer to Section 2 for the formal model and a simple example.

Related Work. Data anonymization is an active area of research with a considerable amount of published work. See, for example, the recent survey by Fung et al. [9]. Note that there are some weaknesses of the k-anonymity concept and it is well-known that it does not always assure privacy [7,9]. Typically, k-anonymity is most useful where there is a single release of data from a single publisher. However, k-anonymity provides a basic, clear, and easy to handle mathematical concept of privacy and related topics. Our research perhaps is most closely related to the recent work of Aggarwal et al. [1] who proposed a new model of data anonymization based on clustering. While developing several polynomial-time approximation algorithms, their modeling idea roughly is to cluster the matrix rows and then to publish the "cluster centers"; importantly, it is required that each cluster contains at least k rows, which corresponds to the k-anonymity concept. The fundamental difference to our model is that we allow to prespecify cluster centers by the user of anonymized data whereas in Aggarwal et al.'s model the end-user of the anonymized data has no influence on selecting which entries to suppress. Indeed, our Pattern Clustering model may be interpreted as a form of clustering via anonymization whereas Aggarwal et al. perform anonymization via clustering.

Our Results. We formally define a simple model of user-specified data anonymization based on the concepts of k-anonymity and pattern vectors. The central

[1] For instance, suppose that M contains data about patients used in medical research, where each row corresponds to a patient and each column is an attribute of the patient. Then an attribute like blood pressure is—typically, but not always—more useful to preserve than, say, hair color.

[2] That is, suppressing all row positions where the corresponding pattern vector has a $\star$-symbol.

Table 1. The computational complexity of Pattern Clustering with respect to various parameters

k	m	n	$\|\Sigma\|$	s	t	p
NP-hard for $k = 1$ ($\|\Sigma\| = 2$, $s = \infty$)	NP-hard for $m = 4$	FPT	NP-hard for $\|\Sigma\| = 2$ ($k = 1$, $s = \infty$)	XP	FPT	XP

combinatorial problem is called Pattern Clustering and it is shown to be NP-hard for every $k \geq 1$ and matrix alphabet size $|\Sigma| = 2$. It is also shown to be NP-hard for matrices containing only four columns. In contrast, Pattern Clustering is fixed-parameter tractable (FPT) for the parameters n (the number of matrix rows) and t (the number of different matrix rows). Moreover, it can be solved in polynomial time for a constant number p of given pattern vectors (in other words, Pattern Clustering is in the parameterized complexity class XP for the parameter p). Membership in XP also holds for the parameter number s of suppressions. See Table 1 for a list of our results with respect to single parameterizations. Clearly, several of our findings in Table 1 suggest investigations in the spirit of multivariate algorithmics [8,12], that is, the study of combined parameters. Here, the following results are known: Pattern Clustering is fixed-parameter tractable for the combined parameters $(m, |\Sigma|)$, and (s, p) (due to upper bound arguments using t) whereas the parameterized complexity status is open for the combined parameters (m, p) and (m, k).

Due to the lack of space some proofs and some further details are deferred to a full version of this paper.

2 Preliminaries and Basic Model

As mentioned in the introductory section, the main motivation for our new model is—in contrast to standard k-anonymization models—to let the end-user influence the data sanitization process by selecting—to some extent—how matrix entries may be suppressed. We now formally define a model that captures this intuitive notion. To this end, it is helpful to interpret a matrix simply as a multiset of rows, as we do in the next definition.

Definition 1. *Let $M \in \Sigma^{n \times m}$ be a matrix over the finite alphabet Σ. Then $\mathrm{R}(M)$ is the multiset of all the rows in M.*

The heart of our pattern-guided anonymization model lies in a function that "consistently" maps input matrix rows to some given pattern vectors. This is described in the following definition, where we use $v[i]$ and $x[i]$ to refer to the i^{th} vector and row entry, respectively.

Definition 2. *Let Σ be a finite alphabet and let $M \in \Sigma^{n \times m}$ and $P \in \{\square, \star\}^{p \times m}$ be two matrices. A function $\varphi : \mathrm{R}(M) \to \mathrm{R}(P)$ is* consistent *if for all $x, y \in \mathrm{R}(M)$ with $v := \varphi(x) = \varphi(y)$, and for all $1 \leq i \leq m$: $v[i] = \square \Rightarrow x[i] = y[i]$.*

Our cost measure that shall be minimized is the number of suppressed matrix entries. First, we define the cost of a pattern vector in the natural way.

Definition 3. *The* cost $c(v)$ *of vector* $v \in \{\square, \star\}^m$ *is the number of its* $\star$*-symbols.*

We now define the cost of a mapping.

Definition 4. *Let* $M \in \Sigma^{n \times m}$ *and* $P \in \{\square, \star\}^{p \times m}$ *be two matrices and let* $\varphi : \mathrm{R}(M) \to \mathrm{R}(P)$ *be a mapping from the rows of* M *to the pattern vectors of* P*. Let* $\#(v) := |\{x : x \in \mathrm{R}(M) \wedge \varphi(x) = v\}|$*. Then, the* cost *of* φ *is* $\sum_{v \in P} c(v) \cdot \#(v)$*.*

Next we define the concept of k-anonymity for functions.

Definition 5. *A function* $f : A \to B$ *is* k-anonymous *if for each* $a \in A$ *with* $f(a) = b$ *it holds that* $|\{x : x \in A \wedge f(x) = b\}| \geq k$*.*

Finally we are ready to define the central computational problem (formulated in its decision version) of this work.

Pattern Clustering
Input: A matrix $M \in \Sigma^{n \times m}$, a "pattern mask" $P \in \{\square, \star\}^{p \times m}$, and two positive integers s and k.
Question: Is there a consistent and k-anonymous function φ mapping the rows of M to the pattern vectors of P, with cost at most s?

Figure 1 provides a simple example to illustrate and further motivate our above definition of Pattern Clustering.

We use the following notation in the rest of the paper. The mapping φ (see Definition 4) plays a central role in the definition of Pattern Clustering. We often talk about it implicitly when saying that a row is mapped to a pattern vector. Moreover, we speak about assigning a $\square$-symbol of a pattern vector v to a symbol x which means that every row mapped to v has an x at the position of the $\square$-symbol.

3 Intractability Results

In this section, we show that Pattern Clustering is NP-complete even in very restricted cases. The membership in NP is easy to see: Guessing a mapping φ of the rows from M to pattern vectors from P, it is easy to verify in polynomial time that φ is consistent, k-anonymous, and has cost at most s. In the following, we provide two polynomial-time many-one reductions: One from CNF-Satisfiability to show NP-hardness for the unweighted variant (that is, $s = \infty$) with $k = 1$ and $|\Sigma| = 2$; and a second reduction from Set Cover to show NP-hardness for $m = 4$.

Before doing the reductions we show how to get rid of big alphabets. The structural properties of Pattern Clustering allow us to replace any alphabet with a binary alphabet, by encoding the alphabet in binary.[3]

[3] As consequence of the binarization, the question whether Pattern Clustering is fixed-parameter tractable with respect to the combined parameter $(p, |\Sigma|)$ is equivalent to the question whether Pattern Clustering is fixed-parameter tractable with respect to p alone.

~~name~~	hair color	disease	age
~~Alice~~	blond	asthma	40-60
~~Bob~~	blond	asthma	40-60
~~Clara~~	blond	laziness	20-30
~~Doris~~	brown	laziness	20-30
~~Emil~~	blond	asthma	20-30
~~Frank~~	brown	laziness	20-30
~~George~~	blond	asthma	40-60

standard k-anonymity, $k = 2$, minimize s:

blond	asthma	40-60	Alice, Bob, George
blond	$\star$	20-30	Emil, Clara
brown	laziness	20-30	Doris, Frank

Pattern Clustering, $k = 2$, minimize s:

$(\star, \square, \square)$	$\star$	asthma	40-60	Alice, Bob
$(\star, \square, \square)$	$\star$	laziness	20-30	Clara, Doris, Frank
$(\star, \square, \star)$	$\star$	asthma	$\star$	George, Emil

Fig. 1. Comparing Pattern Clustering with standard k-anonymization. Consider an extract from a medical database (table on the left). For privacy reasons, one only wants to publish k-anonymous data for $k \geq 2$. Clearly, the first step is to remove the identifier column "name". Using standard k-anonymity, one may end up with the database on the top right. Indeed, this is the 2-anonymous database with the fewest suppressions, requiring only two suppressions. Unfortunately, researchers cannot use it to deduce a correlation between age and disease, because the disease of Emil and Clara is suppressed. In our new model, researchers may specify their particular interests by providing the three shown pattern vectors—in this example they specify that they do not care about hair color. Thus, using Pattern Clustering one may get the database down right. It is one of the databases with fewest suppressions which, at the same time, "respects the interests of the scientists". In contrast to the result for classical k-anonymity, the scientists can see that only young people have the disease "laziness".

Lemma 1. *Let $I = (M, P, s, k)$ be an instance of* Pattern Clustering *with M being a matrix over the alphabet Σ. Then there is an equivalent instance $I' = (M', P', s', k)$ such that M' is a matrix over a binary alphabet Σ' and the number of columns m' of M' is $\lceil \log(|\Sigma|) \rceil$ times the number of columns m of M.*

Proof. Given the instance $I = (M, P, s, k)$, construct $I' = (M', P', s', k)$ as follows. Assign to each symbol in Σ a unique integer from $\{0, 1, \ldots, |\Sigma| - 1\}$. Each column will be replaced with $\lceil \log(|\Sigma|) \rceil$ columns. The corresponding columns are used to binary encode (filling up with zeros on the left) the identifier of the original symbol. The pattern vectors are extended analogously: Each $\star$- (respectively $\square$-) symbol is replaced by $\lceil \log(|\Sigma|) \rceil$ many consecutive $\star$- (respectively $\square$-) symbols. The new cost-bound s' is $\lceil \log(|\Sigma|) \rceil$ times the old cost-bound s. It is not hard to see that the new instance is equivalent to the original one. $\square$

In both reductions to follow we need unique entries in the input matrix M. For ease of notation we introduce the $\triangle$-symbol with an unusual semantics. Each occurrence of a $\triangle$-symbol stands for a *different* unique symbol in the alphabet Σ. One could informally state this as "$\triangle \neq \triangle$".

Now we present our first NP-hardness result.

Theorem 1. Pattern Clustering *is NP-complete, even if $k = 1$, $|\Sigma| = 2$, and $s = \infty$.*

Proof (Sketch). We provide a polynomial-time many-to-one reduction from the NP-complete CNF-Satisfiability. In the following, let (X, C) be a CNF-Satisfiability instance with $X := \{x_1, \ldots, x_n\}$ being the set of variables and

matrix:

1	1	△	$(x_1 \vee x_2)$
0	1	△	$(\neg x_1 \vee x_2)$
△	0	0	$(\neg x_2 \vee \neg x_3)$
1	0	1	$(x_1 \vee \neg x_2 \vee x_3)$
1	0	0	$(x_1 \vee \neg x_2 \vee \neg x_3)$
0	0	0	$(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

pattern vectors:
$(\square, \star, \star)$
$(\star, \square, \star)$
$(\star, \star, \square)$

Fig. 2. Example for the reduction from CNF-Satisfiability to Pattern Clustering. Given is the following formula: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$. Applying the reduction, we get a Pattern Clustering instance as illustrated. Note that each row in the table represents $n+1$ rows differing only in the △-positions. It is quite easy to see that every solution, e.g. mapping 1 0 1 and 1 0 0 to $(1, \star, \star)$, 1 1 △ and 0 1 △ to $(\star, 1, \star)$, and △ 0 0 and 0 0 0 to $(\star, \star, 0)$, corresponds to the satisfying assignment $x_1 = 1, x_2 = 1, x_3 = 0$.

$C := \{c_1, \ldots, c_m\}$ being the set of clauses. We construct an equivalent Pattern Clustering instance (M, P, s, k) with $k = 1$ and $s = \infty$ as described in the following (see Figure 2 for an example).

The columns of the input matrix M correspond to the variables. Blocks of rows of M correspond to the clauses: For each clause we have $n+1$ rows. The alphabet Σ contains 1, 0, and the unique symbols △. Summarizing, M is an $m(n+1) \times n$ matrix over an alphabet with at most $m \cdot (n+1) \cdot n$ symbols.

In the following we describe the rows of M. Recall that there are $n+1$ rows for each clause. Let $e_{i,j}[z]$ denote the entry in the z^{th} column of the j^{th} row for the i^{th} clause.

- If c_i contains variable x_z as a positive literal, then $\forall 1 \leq j \leq n+1 : e_{i,j}[z] = 1$.
- If c_i contains variable x_z as a negative literal, then $\forall 1 \leq j \leq n+1 : e_{i,j}[z] = 0$.
- Otherwise, $\forall 1 \leq j \leq n+1 : e_{i,j}[z] = \triangle$.

For each variable $x_i \in X$ we have a pattern vector v_i where all entries are $\star$-symbols aside from the entry in the i^{th} position. The idea is that the assignment of the $\square$-symbol from the pattern vector v_i corresponds to the assignment of the variable x_i.

The theorem now follows from Lemma 1. □

After proving NP-completeness for constant values of k and $|\Sigma|$, we show that Pattern Clustering is intractable even for a constant number of columns.

Theorem 2. Pattern Clustering *is NP-complete, even for* $m = 4$.

Proof (Sketch). We show the hardness by giving a polynomial-time many-one reduction from the NP-hard Set Cover problem.

Set Cover
Input: A set family $\mathcal{F} = \{F_1, \ldots, F_{|\mathcal{F}|}\}$ over a universe $U = \{u_1, \ldots, u_{|U|}\}$, and a positive integer h.
Question: Is there a set cover $\mathcal{F}' \subseteq \mathcal{F}$ of size at most h such that $\bigcup_{F \in \mathcal{F}'} F = U$?

We first describe the main idea of the reduction. Each element $u \in U$ is represented by $|\mathcal{F}|+1$ rows in M. All these $|\mathcal{F}|+1$ rows corresponding to one element u can be mapped to one pattern vector $v \in P$. By setting k to $|\mathcal{F}|$, we allow at most one of these rows to be mapped to another, cheaper pattern vector. The construction guarantees that if one or more rows are mapped to any of these cheaper pattern vectors, then the elements represented by these rows are contained together in at least one $F \in \mathcal{F}$. If $|U|$ rows (one row for each element $u \in U$) can be mapped to h cheaper pattern vectors, then this assignment denotes a set cover of size h in the given Set Cover-instance and vice versa.

Next, we describe the construction in detail. Given an instance $(\mathcal{F}, U, h)$ of Set Cover, we construct a Pattern Clustering-instance (M, P, s, k) as follows.

For each element $u_i \in U$ add a set $R_i^U = \{r_{i,0}^U, \ldots, r_{i,|\mathcal{F}|}^U\}$ of $|\mathcal{F}|+1$ rows to the input matrix M and one pattern vector $v_i^U = (\star, \star, \square, \star)$ to the pattern mask P. We call v_i^U an *expensive* pattern vector since its cost $c(v_i^U)$ is three. Set $r_{i,0}^U :=$ $\boxed{\triangle}\boxed{\triangle}\boxed{i}\boxed{\triangle}$. If $u_i \in F_j$, $1 \le j \le |\mathcal{F}|$, then set $r_{i,j}^U :=$ $\boxed{j}\boxed{j}\boxed{i}\boxed{\triangle}$, else set $r_{i,j}^U :=$ $\boxed{\triangle}\boxed{\triangle}\boxed{i}\boxed{\triangle}$. Further, let $R_i^F := \{r_{i,j}^U \mid u_i \in F_j\}$. All rows of R_i^U coincide in the third column, and so they can all be mapped together to the pattern vector v_i^U. Intuitively, the first row $r_{i,0}^U$ in R_i^U is a dummy row that has to get mapped to the expensive pattern vector v_i^U. By setting $k := |\mathcal{F}|$, we ensure that at least $|\mathcal{F}|-1$ other rows of R_i^U get mapped to the same expensive pattern vector. Hence, at most one row of R_i^U can be mapped to a cheaper pattern vector. The construction ensures that this one row is an element of R_i^F.

Now we specify the cheaper pattern vectors: Add h pattern vectors $v_1^{\mathcal{F}}, \ldots, v_h^{\mathcal{F}}$ of the form $(\square, \square, \star, \star)$. We call these pattern vectors *cheap* as they need one suppression less than the expensive pattern vectors. The idea is that each of these pattern vectors $v_i^{\mathcal{F}}$ corresponds to one set in a set cover. Note that there are $|U|$ rows of $R_1^U, \ldots, R_{|U|}^U$ that can be mapped to the h cheap pattern vectors (one row of each R_i^U). Since all the rows mapped to a pattern vector $v_i^{\mathcal{F}}$ have to coincide in the first two columns, the only possible candidates are the rows belonging to the sets R_i^F. If there are $|U|$ rows fulfilling these requirements, then the h different assigned numbers in the pattern vectors $v_1^{\mathcal{F}}, \ldots, v_h^{\mathcal{F}}$ denote the set cover $\mathcal{F}'$ in the Set Cover-instance.

To ensure that at least k rows are mapped to each pattern vector from $\{v_1^{\mathcal{F}}, \ldots, v_h^{\mathcal{F}}\}$, add for each $F_i \in \mathcal{F}$ a set $R_i^{\mathcal{F}} = \{r_{i,1}^{\mathcal{F}}, \ldots, r_{i,|\mathcal{F}|+1}^{\mathcal{F}}\}$ of $|\mathcal{F}|+1$ rows with $r_{i,j}^{\mathcal{F}} :=$ $\boxed{i}\boxed{i}\boxed{X}\boxed{X}$[4], $1 \le j \le |\mathcal{F}|+1$, to the input matrix M. Since not all these rows can be mapped to some pattern vector in $\{v_1^{\mathcal{F}}, \ldots, v_h^{\mathcal{F}}\}$, add a pattern vector v^X of the form $(\star, \star, \square, \square)$ to the pattern mask P. The rows $r_{i,j}^{\mathcal{F}}$ can be mapped to v^X or to one of $v_1^{\mathcal{F}}, \ldots, v_h^{\mathcal{F}}$. Since all these pattern vectors are cheap, the only constraint on the mapping of these rows is that each pattern vector $v^X, v_1^{\mathcal{F}}, \ldots, v_h^{\mathcal{F}}$ contains at least k rows. We finish the construction by setting $s := 2 \cdot |U| + 3 \cdot |U| \cdot |\mathcal{F}| + 2 \cdot (|\mathcal{F}|+1) \cdot |\mathcal{F}|$. □

[4] Here, we use X as a fixed symbol which is not used in M elsewhere.

Dom et al. [6] showed that Set Cover parameterized by $(h, |U|)$ does not admit a problem kernel of size $(|U|+h)^{O(1)}$ unless an unexpected complexity-theoretic collapse occurs, namely the polynomial-time hierarchy collapses to the third level. Given a problem instance I with parameter x, a *problem kernel* is an equivalent polynomial-time computable instance I' with parameter $x' \leq x$ such that the size of I' is upper-bounded by some function g only depending on x [3,10]; $g(x)$ is called the size of the problem kernel.

Bodlaender et al. [4] introduced a refined concept of parameterized reduction (called polynomial time and parameter transformation) that allows to transfer such hardness results to new problems. Indeed, the reduction above is such a parameterized reduction. The parameters h and $|U|$ are transformed to m and p as follows: $m = 4$ and $p = h + |U| + 1$. Hence Pattern Clustering does not admit a problem kernel of size $(m+p)^{O(1)}$.

Corollary 1. Pattern Clustering *parameterized by* (m, p) *has no problem kernel of polynomial size unless coNP* $\subseteq$ *NP/poly.*

4 Tractable Cases

In the previous section, we showed computational intractability for various special cases of Pattern Clustering. Here, we complement these hardness results by presenting some tractable cases. To this end, we consider several parameterizations of Pattern Clustering with respect to natural parameters and reasonable combinations thereof. Since Pattern Clustering allows the user to specify pattern vectors to influence the solution structure, the number of pattern vectors p seems to be one of the most natural problem-specific parameters. It is quite reasonable to assume that there are instances with a small amount of pattern vectors, for instance, when the user wants a clustering with few but huge clusters. We start with a general observation on the solution structure of Pattern Clustering instances. To this end, we introduce the concept of row types. A *row type* is a string from Σ^m. We say that a set of rows in the matrix has a certain row type if they are identical. In this sense, the number t of different matrix rows is the number of row types. The following lemma says that without loss of generality one may assume that at most t many pattern vectors are used in the solution.

Lemma 2. *Let* (M, P, k, s) *be a* YES*-instance of* Pattern Clustering. *If* M *has* t *row types, then there exists a mapping* φ*, which is solution for* (M, P, k, s)*, whose codomain contains at most* t *elements.*

We now take up the question of whether the problem is still intractable (in form of NP-hardness) when p is a constant.

Parameters p and t. We describe a fixed-parameter algorithm for Pattern Clustering with respect to the combined parameter (p, t). This algorithm can be interpreted as an XP-algorithm for Pattern Clustering parameterized by p, that is, it has polynomial running time for constant values of p.

Theorem 3. Pattern Clustering *can be solved in* $O(t^{\min(t,p)} \cdot 2^p \cdot (\min(t,p) \cdot t \cdot m + t^3 \log(t)) + n \cdot m)$ *time.*

Proof. As preprocessing we have to compute the input row types in $O(n \cdot m)$ time (by constructing a trie on the rows [5]). Our algorithm works in three phases:

1. For each pattern vector v, determine whether it is *used* in the solution, that is, whether v occurs in the codomain of the mapping.
2. For each pattern vector v that is used in the solution determine which input row types may contain rows that are mapped to v in the solution by guessing a representative element. In the following we call these input row types *preimage types* of the pattern vector v.
3. For each pattern vector v that is used in the solution determine how many rows from each preimage type are mapped to v in the solution.

Due to Lemma 2, Phase 1 can be realized by branching on all $\sum_{i=1}^{\min(t,p)} \binom{p}{i} \leq 2^p$ possibilities.

Phase 2 is realized by guessing for each pattern vector a prototype, that is, a row that is mapped to this vector in the solution. Clearly, knowing one preimage of the mapping for each pattern vector is sufficient to compute which input row types may contain rows that are mapped to the vector. Guessing the prototypes and computing the preimage types can be done in $O(t^p \cdot m)$ time.

In Phase 3, we have the following situation. We have t input row types and $p' \leq t$ pattern vectors that are used in the solution. In the following the set of input row types is represented by $T_{\text{in}} := \{1, \ldots, t\}$ whereas the set of pattern vectors is represented by $T_{\text{out}} := \{1, \ldots, p'\}$. For each pair consisting of an input row type R and a pattern vector v we already know whether rows from R may be mapped to v in the solution. Let $a : T_{\text{in}} \times T_{\text{out}} \to \{0, 1\}$ be a function expressing this information. Furthermore, let ω_i with $i \in T_{\text{out}}$ denote the cost of the i^{th} pattern vector and let n_j with $j \in T_{\text{in}}$ denote the number of rows in the j^{th} input row type. A consistent and k-anonymous mapping that has cost at most s and respects the preimage types (determined in Phase 2) corresponds to a solution of the Row Assignment [5] problem which is defined as follows.

Row Assignment
Input: Nonnegative integers k, s, $\omega_1, \ldots, \omega_{p'}$ and $n_1, \ldots, n_t$ with $\sum_{i=1}^{t} n_i = n$, 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

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

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

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

$$\sum_{i=1}^{t} \sum_{j=1}^{p'} g(i,j) \cdot \omega_j \leq s \tag{4}$$

The mapping is represented by the function g. Inequality (1) ensures that for each pattern vector v only rows from its preimage types are mapped to v. Inequality (2) ensures that the mapping is k-anonymous. Equation (3) ensures that each row is mapped to one pattern vector. Inequality (4) ensures that the costs of the mapping are at most s. Row Assignment can be solved in $O(t^3 \cdot \log(t))$ time [5, Lemma 2] and computing the function a takes $O(\min(t,p) \cdot t \cdot m)$ time. □

We showed fixed-parameter tractability for Pattern Clustering with respect to the combined parameter (t,p) by an algorithm with polynomial running time for constant values p. We leave it open whether there also exists an algorithm where the degree of the polynomial is independent of p, that is, whether Pattern Clustering is fixed-parameter tractable for parameter p. Next, we develop a fixed-parameter algorithm for the parameter t. This is mainly a classification result because its running time is impractical.

Corollary 2. Pattern Clustering *is fixed-parameter tractable with respect to the parameter t as well as with respect to the parameter n.*

Proof (Sketch). When $p \leq t$, we use the algorithm from Theorem 3 without any modification. The corresponding running time is $O(t^t \cdot 2^t \cdot (p \cdot t \cdot m + t^3 \log(t)) + n \cdot m)$. For the other case we show a refined realization of Phase 1 of the algorithm from Theorem 3.

In Phase 1 we determine a set P' of pattern vectors that are used in the solution, meaning that the codomain of the mapping function is P'. Due to Lemma 2 we know that w.l.o.g. $|P'| \leq t$. In Theorem 3 we simply tried all size-at-most-t subsets of P. Here, we show that for guessing P' we only have to take into account a relatively small subset $P^* \subseteq P$ with $|P^*| \leq g(t)$ with g being a function which depends only on t.

Consider a pattern vector v of the unknown P'. In Phase 2 of the algorithm, we determine the preimage types, that is, the set of input row types that may contain rows that are mapped to v in the solution. Assume that the preimage types for all pattern vectors from P' are fixed. To determine which concrete pattern vector corresponds to a set of preimage types, we only have to take into account the t cheapest "compatible" pattern vectors, where compatible means that all rows of these preimage types coincide at the □-symbol positions. By definition, there exist at most 2^t many different sets of preimage types. Thus, keeping for each set of preimage types the t cheapest pattern vectors and removing the rest results in a set P^* of size $2^t \cdot t$.

Hence, when $p > t$, we realize Phase 1 by computing P^* as described above and branch on all subsets $P' \subseteq P^*$ of size at most t. This can be done in $O\binom{2^t \cdot t}{t} \leq O(2^{t^2} t^t)$ time. As preprocessing we have to compute the input row types in $O(n \cdot m)$ time. Altogether, we can solve Pattern Clustering in $O(t^t \cdot (2^{t^2} t^t) \cdot (t^2 \cdot m + t^3 \log(t)) + n \cdot m)$ time. Clearly, since $t \leq n$, our result also holds for the parameter n. □

Combined parameters. As a corollary of Theorem 3 we show fixed-parameter tractability for some interesting combined parameters. All results rely on the fact that one can bound the number t of input row types from above by a function only depending on the respective combined parameter.

Corollary 3. Pattern Clustering *is fixed-parameter tractable with respect to the combined parameters* $(|\Sigma|, m)$ *and* (p, s).

Proof. As for the parameter $(|\Sigma|, m)$, Σ^m is an upper bound for the number t of input row types.

As for the parameter (p, s): in the following, we call rows that are mapped to pattern vectors with at least one $\star$-symbol *costly rows* and their corresponding row types *costly row types*. Analogously, rows that are mapped to pattern vectors without $\star$-symbols are called *costless rows* and their row types *costless row types*. Clearly, every input row type is costly or costless (or both). There are at most s costly rows and, hence, at most s costly row types. Furthermore, the number of pattern vectors without $\star$-symbols is at most p. Since no two costless rows from different input row types can be mapped to the same pattern vector, the number of costless row types is also at most p. Hence, in a YES-instance the number t of input row types is at most $s + p$. □

Corollary 4. Pattern Clustering *is in XP with respect to the parameter* s.

Proof. Using the definitions of costly and costless from Corollary 3 we give a simple algorithm that shows membership in XP. The first step is to guess, from $\sum_{i=0}^{s} \binom{n}{i}$ possibilities, the rows which are costly. The second step is to guess, from $\sum_{i=0}^{s} \binom{p}{i}$ possibilities, the pattern vectors (that contain $\star$-symbols) which are used in the solution. Then, guess the mapping between at most s rows and at most s pattern vectors and check whether it is consistent and k-anonymous. In the last step, the costless rows are greedily mapped to pattern vectors without $\star$-symbols. □

5 Conclusion

We initiated the investigation of a new variant of k-anonymity for ensuring "user-guided data privacy and clustering". The corresponding NP-hard combinatorial problem Pattern Clustering has a number of tractable and intractable special cases; our results are listed in Table 1 in the introduction. Several open questions remain. For example, the parameterized complexity of the problem for the parameters "number s of suppressions" and the combined parameters (m, p) and (m, k), where m is the number of columns and p is the number of pattern vectors, are all open. A particularly interesting question is whether Pattern Clustering for parameter p is fixed-parameter tractable or W[1]-hard. An equally interesting open question is whether Pattern Clustering becomes tractable for $m = 2$—note that we have shown it NP-hard for $m = 4$ and, based on some preliminary results, conjecture it to be NP-hard for $m = 3$. Finally,

it seems worth investigating Pattern Clustering also from the viewpoint of polynomial-time approximation.

Summarizing, we believe that Pattern Clustering is a well-motivated combinatorial problem relevant for data anonymization and data clustering; it deserves further investigation.

Acknowledgements. We are grateful to the anonymous referees of the *MFCS-2011* for helping to improve this work by spotting some flaws and providing the idea behind Corollary 4.

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. 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)
3. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
4. Bodlaender, H.L., Thomassé, S., Yeo, A.: Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical Report UU-CS-2008-030, Department of Information and Computing Sciences, Utrecht University (2008)
5. Bredereck, R., Nichterlein, A., Niedermeier, R., Philip, G.: The effect of homogeneity on the complexity of k-anonymity. In: Proc. 18th FCT. LNCS, Springer, Heidelberg (2011)
6. 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)
7. Domingo-Ferrer, J., Torra, V.: A critique of k-anonymity and some of its enhancements. In: Proc. 3rd ARES, pp. 990–993. IEEE Computer Society, Los Alamitos (2008)
8. Fellows, M.R.: Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 2–10. Springer, Heidelberg (2009)
9. 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), 14:1–14:14 (2010)
10. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
11. Meyerson, A., Williams, R.: On the complexity of optimal k-anonymity. In: Proc. 23rd PODS, pp. 223–228. ACM, New York (2004)
12. Niedermeier, R.: Reflections on multivariate algorithmics and problem parameterization. In: Proc. 27th STACS. LIPIcs, vol. 5, pp. 17–32. IBFI Dagstuhl (2010)
13. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. IJUFKS 10(5), 571–588 (2002)
14. Sweeney, L.: k-anonymity: A model for protecting privacy. IJUFKS 10(5), 557–570 (2002)

Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete

Stanislav Böhm[1,⋆] and Stefan Göller[2]

[1] Technical University of Ostrava, Department of Computer Science, Czech Republic
[2] Universität Bremen, Institut für Informatik, Germany

Abstract. We prove that deciding language equivalence of deterministic real-time one-counter automata is NL-complete, in stark contrast to the inclusion problem which is known to be undecidable. This yields a subclass of deterministic pushdown automata for which the precise complexity of the equivalence problem can be determined. Moreover, we show that deciding regularity is NL-complete as well.

1 Introduction

In formal language theory two of the most fundamental decision problems are to decide whether two languages are equivalent (*language equivalence*) or whether one language is a subset of another (*language inclusion*). It is well-known that already deciding if a context-free language is universal is undecidable.

In recent years, subclasses of context-free languages have been studied for which equivalence or even inclusion becomes decidable.

The most prominent such subclass is the class of deterministic context-free languages (however inclusion remains undecidable). A groundbreaking result by Sénizergues states the decidability of language equivalence of *deterministic pushdown automata (DPDA)* [20], see also [21]. In 2002 Stirling showed that DPDA language equivalence is in fact primitive recursive [23]. Probably due to its intricacy this fundamental problem has not attracted too much research in the past ten years. We emphasize that for DPDA language equivalence there is still a remarkably huge complexity gap ranging from a primitive recursive upper bound to P-hardness (which straightforwardly follows from P-hardness of the emptiness problem). To the best of the authors' knowledge, the same phenomenon holds if the DPDA are restricted to be *real-time* [18], i.e. ε-transitions are not present. However, for finite-turn DPDA a coNP upper bound is known [22]. For *simple DPDA* (which are single state and real-time DPDA) language equivalence is decidable in polynomial time [9], whereas language inclusion is still undecidable [6]. For *deterministic one-counter automata (DOCA)*, which are DPDA over a singleton stack alphabet plus a bottom stack symbol, language equivalence was shown decidable in time $2^{O(\sqrt{n \log n})}$ [24]. By a simple analysis of the proof in [24] a PSPACE upper bound can be derived for this problem.

⋆ S. Böhm has been supported by the Czech Ministry of Education, project No. 1M0567 and GAČR P202/11/0340.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 194–205, 2011.

The goal of this paper is to make a step towards understanding (a special case of) of the equivalence problem of DPDA better. We analyze a syntactic restriction of DPDA, namely *deterministic real-time one-counter automata (ROCA)*, which are real-time DOCA. ROCA satisfy the following points: (i) the automaton model is simple, (ii) it is powerful enough to capture relevant non-regular languages such as e.g. the set of well-matched parenthesis or $\{a^n b^n \mid n \geq 0\}$, (iii) its language is not defined modulo some predetermined stack behavior of the automata (i.e. as it is the case for visibly pushdown automata [1] or more general approaches as in [4,17]), and (iv) tight complexity bounds can be obtained for the equivalence problem. Although points (i) and (ii) have a subjective touch, the authors are not aware of any subclass of the context-free languages that satisfy all of the four mentioned points. We remark that for ROCA language inclusion remains undecidable.

Contributions. The main result of this paper is that language equivalence of ROCA is NL-complete, hence closing the gap from PSPACE [24] (which holds even for DOCA) to NL (hardness for NL is inherited from the emptiness problem for deterministic finite automata). As a second result we prove that deciding *regularity* of ROCA, i.e. deciding if the language of a given ROCA is regular, is NL-complete as well. The previously best known upper bound for this problem (as for DOCA) is a time bound of $2^{O(\sqrt{n \log n})}$ [24] (from which one can also derive a PSPACE upper bound).

Used techniques. For our NL upper bound for language equivalence of ROCA, we prove that if two ROCA are inequivalent, then they can already be distinguished by a word of polynomial length. To show this, we use an established approach that can be summarized as the "*belt technique*" that has already been used in [14,12,11,3] in the context of (bi)simulation equivalence checking of one-counter automata. More specifically, we use an approach from [11,3] that can be formulated as follows: There is a small set INC of *incompatible configurations* which two configurations necessarily have to have the same shortest distance to provided they are language equivalent — moreover, in case two configurations both have the same finite distance to INC, they must either both have small counter values or they lie in one of polynomially many so called *belts*. To prove the existence of polynomially long distinguishing words, in case two ROCA are not language equivalent, we carefully investigate how paths through such belts can look like.

Related work. Deterministic one-counter automata (DOCA) were introduced by Valiant and Paterson in [24], where the above-mentioned time upper bound for language equivalence was proven. Polynomial time algorithms for language equivalence and inclusion for strict subclasses of ROCA were provided in [7,8]. In [2,5] polynomial time learning algorithms were presented for ROCA. Simulation and bisimulation problems on one-counter automata were studied in [3,12,11,14,15,13].

Organization. Our paper is organized as follows. Section 2 contains definitions. Our main result is stated in Section 3 and proven in Section 4. Regularity of ROCA is proven NL-complete in Section 5. We conclude in Section 6.

Remark: In [2,19] it is stated that language equivalence of DOCA can be decided in polynomial time. Unfortunately, the proofs provided in [2,19] were not exact enough to be verified and raise several questions which are unanswered to date.

2 Definitions

By $\mathbb{Z}$ we denote the integers and by $\mathbb{N} = \{0, 1, \ldots\}$ we denote the naturals. For two integers $i, j \in \mathbb{Z}$ we define the interval $[i, j] = \{i, i+1, \ldots, j\}$ and $[j] = [1, j]$. The *sign* function $\text{sgn} : \mathbb{N} \to \{0, 1\}$ is defined as $\text{sgn}(n) = 1$ if $n > 0$ and $\text{sgn}(n) = 0$ if $n = 0$, for each $n \in \mathbb{N}$. For a word w over a finite alphabet Σ we denote by $|w|$ the *length* of w. By ε we denote the *empty word*. By Σ^* we denote the set of finite words over Σ, by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ the set of *non-empty words*, and for each $\ell \geq 0$ we define $\Sigma^{\leq \ell} = \{w \in \Sigma^* : |w| \leq \ell\}$.

A *deterministic and complete transition system* is a tuple $\mathcal{T} = (S, \Sigma, \{\xrightarrow{a} \mid a \in \Sigma\}, F)$, where S is a set of *states*, Σ is a *finite alphabet*, for each $a \in \Sigma$ we have that $\xrightarrow{a} \subseteq S \times S$ is a set of transitions, where for each $s \in S$ there is precisely one $t \in S$ such that $s \xrightarrow{a} t$, and $F \subseteq S$ is a set of *final states*. We extend $\xrightarrow{w}$ to words $w \in \Sigma^*$ inductively as expected, $\xrightarrow{\varepsilon} = \{(s, s) \mid s \in S\}$ and $\xrightarrow{wa} = \{(s, t) \mid \exists u \in S : s \xrightarrow{w} u \xrightarrow{a} t\}$, where $w \in \Sigma^*$ and $a \in \Sigma$. We also write $\longrightarrow$ for $\bigcup_{a \in \Sigma} \xrightarrow{a}$. For each subset $U \subseteq S$ we write $s \xrightarrow{w} U$ (resp. $s \to^* U$) if $s \xrightarrow{w} u$ (resp. $s \to^* u$) for some $u \in U$. For each state $s \in S$ we define the *language up to length* ℓ of s as $L_\ell(s) = \{w \in \Sigma^{\leq \ell} \mid s \xrightarrow{w} t, t \in F\}$ and the *language* of s as $L(s) = \bigcup_{\ell \in \mathbb{N}} L_\ell(s)$. We write $s \equiv_\ell t$ whenever $L_\ell(s) = L_\ell(t)$ and $s \equiv t$ whenever $L(s) = L(t)$. So note that we have $s \equiv_0 t$ if and only if either $s, t \in F$ or $s, t \notin F$. We call a word $w \in \Sigma^*$ a *(distinguishing) witness* for states s and t if $s \xrightarrow{w} s'$ and $t \xrightarrow{w} t'$ with $s' \not\equiv_0 t'$. A *mininal witness* is a witness of minimal length among all witnesses.

A *deterministic real-time one-counter automaton (ROCA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *control states*, Σ is a *finite alphabet*, $\delta : Q \times \Sigma \times \{0, 1\} \to Q \times \{-1, 0, 1\}$ is a *transition function* that satisfies $\delta(Q \times \Sigma \times \{0\}) \subseteq Q \times \{0, 1\}$ (i.e. no decrement is allowed when the counter is zero), $q_0 \in Q$ is an *initial control state*, $F \subseteq Q$ is a set of *final control states*. If the initial state q_0 of $\mathcal{A}$ is not relevant, we just write $\mathcal{A} = (Q, \Sigma, \delta, F)$. A *configuration* of $\mathcal{A}$ is a pair $(q, n) \in Q \times \mathbb{N}$ that we also abbreviate by $q(n)$. Each ROCA $\mathcal{A}$ defines a deterministic transition system $\mathcal{T}(\mathcal{A}) = (Q \times \mathbb{N}, \Sigma, \{\xrightarrow{a} \mid a \in \Sigma\}, F \times \mathbb{N})$, where $q(n) \xrightarrow{a} q'(n + j)$ whenever $\delta(q, a, \text{sgn}(n)) = (q', j)$. We define $L(\mathcal{A}) = L(q_0(0))$. In this paper, we are mainly interested in the following decision problem.

LANGUAGE EQUIVALENCE OF ROCA	
INPUT:	Two ROCA $\mathcal{A}$ and $\mathcal{A}'$.
QUESTION:	$L(\mathcal{A}) = L(\mathcal{A}')$?

Interestingly, inclusion between ROCA is undecidable. Valiant and Paterson were already aware of this without providing a proof [24].

Proposition 1 (Simple consequence of [16]). *Given two ROCA $\mathcal{A}$ and $\mathcal{A}'$, deciding whether $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ holds, is undecidable.*

It is worth noting that it is also a consequence of [16] that language equivalence of *nondeterministic* (real-time) one-counter automata is undecidable.

3 NL-Completeness of Equivalence of ROCA

Instead of considering language equivalence of *two* ROCA, we can simply take their disjoint union and ask whether two configurations of it are language equivalent. Therefore let us fix for the rest of this and the next section some ROCA $\mathcal{A} = (Q, \Sigma, \delta, F)$ and two control states $p_{\text{init}}, q_{\text{init}} \in Q$ for which we wish to decide if $p_{\text{init}}(0) \equiv q_{\text{init}}(0)$.

Lemma 2. *We have $p_{init}(0) \equiv q_{init}(0)$ if and only if $p_{init}(0) \equiv_\ell q_{init}(0)$, where ℓ is polynomially bounded in $|Q|$.*

In Section 4 we prove Lemma 2. We now use it to derive NL-completeness of language equivalence of ROCA.

Theorem 3. *Language equivalence of ROCA is* NL*-complete.*

Proof. The NL lower bound already holds for the emptiness problem for deterministic finite automata. For the upper bound, we apply Lemma 2 and store in logarithmic space a pair of configurations (the two counter values are stored in binary) for which we check inequivalence in an on-the-fly fashion: We repeatedly guess a symbol $a \in \Sigma$ and update the pair of configurations by applying the transition function on both of them synchronously. If the current pair is not $\equiv_0$-related, then the initial pair of configurations is inequivalent and if such a guessing is not possible then the initial pair of configurations has to be equivalent by Lemma 2. Hence inequivalence is NL. Since NL is closed under complement [10] the theorem follows. □

4 Polynomially Long Distinguishing Witnesses Suffice

Before we prove Lemma 2, we introduce some notions that allow us to get a better visual intuition of what minimal distinguishing witnesses can look like.

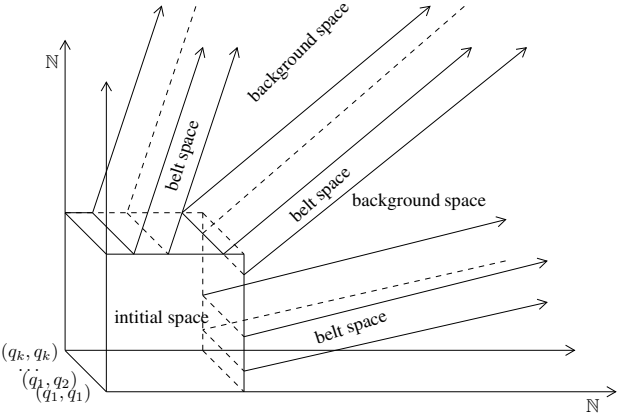


Fig. 1. The 3D space

For the rest of the paper, it will sometimes be more convenient to identify each pair of configurations $\langle p(m), q(n)\rangle$ by the point $\langle m, n, (p, q)\rangle$ in the 3D space $\mathbb{N} \times \mathbb{N} \times (Q \times Q)$, where the first two dimensions represent the two counter values and the third dimension $Q \times Q$ corresponds to the pair of control states. We will partition the 3D space into an *initial space*, *belt space* and *background space* as exemplarily depicted in Figure 1. The size of the initial space and the thickness and the number of belts will be polynomially bounded in $|Q|$.

We remark that the *belt technique* in the context of one-counter automata has already successfully been used in [14,12,11,3]. Moreover, we remark that our concrete way of partitioning the 3D space was already present in [11,3].

To each pair of configurations $\langle p(m), q(n)\rangle$ and each word $w = a_1 \cdots a_\ell \in \Sigma^*$ we can assign a unique sequence $\mathsf{Comp}(p(m), q(n), w) = \pi_0 \cdots \pi_\ell$ of 3D points that we call the *computation*, formally $\pi_0 = \langle m, n, (p, q)\rangle$ and if $\pi_i = \langle m_i, n_i, (p_i, q_i)\rangle$ for each $i \in [0, \ell]$, then in the transition system $\mathcal{T}(\mathcal{A})$ we have $p_{i-1}(m_{i-1}) \xrightarrow{a_i} p_i(m_i)$ and $q_{i-1}(n_{i-1}) \xrightarrow{a_i} q_i(n_i)$ for each $i \in [1, \ell]$. Hence $\mathsf{Comp}(p(m), q(n), w)$ can be seen as a path through the 3D space. The *counter effect of* π is defined as $(m_\ell - m_0, n_\ell - n_0) \in \mathbb{Z} \times \mathbb{Z}$. A *factor* of π is a sequence $\pi_i \pi_{i+1} \cdots \pi_j$ for some $0 \leq i \leq j \leq \ell$.

The overall proof strategy for Lemma 2 will be to show that for every *minimal (distinguishing) witness* w for $p_{\text{init}}(0)$ and $q_{\text{init}}(0)$ the path $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w)$ has the following property: It can stay in the initial space, it can be inside each belt space but only polynomially many steps consecutively, but once it is in the background space it terminates surely after polynomially many steps. This implies that the overall length of $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w)$ is polynomially bounded.

In Section 4.1 we (re-)investigate an important set INC of configurations and discuss in Section 4.2 that two configurations that have the same finite shortest distance to INC necessarily must lie in the initial space or in the belt space. In Section 4.3 we finally prove that any minimal witness has the above mentioned behavior in the 3D space, thus implying Lemma 2. For the rest of this section, let $k = |Q|$ denote the number of control states of the ROCA $\mathcal{A}$ that we fixed for this section.

4.1 The Underlying DFA and Incompatible Configurations

We start with the observation what happens in the transition system $\mathcal{T}(\mathcal{A})$ if the counter value is very big: It behaves for a long time just like a deterministic finite automaton (DFA). We will call this DFA the *underlying DFA* of $\mathcal{A}$. We can partition the set of configurations of $\mathcal{A}$ into two sets: Those configurations that are not equivalent to all states of the underlying finite system up to words of length at most k and the rest. By analyzing the reachability to the former set, we establish the partition of the 3D space in the next section.

We remark that below the notion *underlying DFA*, the set INC with its useful property stated in Lemma 4 and the distance function dist were already present in [11,3].

The *underlying DFA* of $\mathcal{A}$ is $\mathcal{F} = (Q, \Sigma, \{\xrightarrow{a} \mid a \in \Sigma\}, F)$, where $q \xrightarrow{a} q'$ if and only if $\delta(q, a, 1) \in \{q'\} \times \{-1, 0, 1\}$.

Observe that on the one hand we write Q to denote the set of control states of $\mathcal{A}$ and on the other hand we denote by Q the states of the DFA $\mathcal{F}$. Recall that $k = |Q|$. For each $q \in Q$ we write $L_k(q)$ to denote the language of $\mathcal{F}$ up to length at most k in case q is the initial state. Also note that in $\mathcal{F}$ we have that $\equiv_k$ coincides with $\equiv_{k-1}$.

Define the set INC as those configurations of $\mathcal{T}(\mathcal{A})$ that are incompatible (not k-equivalent) to *all* states in $\mathcal{F}$, formally

$$\mathsf{INC} \quad = \quad \{p(m) \in Q \times \mathbb{N} \mid \forall q \in Q : L_k(p(m)) \neq L_k(q)\}.$$

Remark 4. If $p(m) \in \mathsf{INC}$, then $m < k$.

The main motivation to study the set INC is due to the following lemma.

Lemma 5. *Assume* $p(m) \not\longrightarrow^* \mathsf{INC}$, *and* $q(n) \not\longrightarrow^* \mathsf{INC}$. *Then* $p(m) \equiv q(n)$ *if and only if* $p(m) \equiv_k q(n)$.

Proof. The "only if"-direction is trivial. For the "if"-direction, assume by contradiction $p(m) \not\equiv q(n)$ but $p(m) \equiv_k q(n)$. Let ℓ be minimal such that $p(m) \not\equiv_\ell q(n)$. Note that $\ell > k$. Thus, there is some word $u \in \Sigma^{\ell-k}$ with $p(m) \xrightarrow{u} p'(m')$ and $q(n) \xrightarrow{u} q'(n')$ where $p'(m') \not\equiv_k q'(n')$ but $p'(m') \equiv_{k-1} q'(n')$. Since by assumption $p'(m'), q'(n') \notin \mathsf{INC}$, there are $s, t \in Q$ such that $s \equiv_k p'(m')$ and $t \equiv_k q'(n')$ and hence $s \equiv_{k-1} t$. Recall that in $\mathcal{F}$ we have that $\equiv_{k-1}$ coincides with $\equiv_k$ and hence $s \equiv_k t$. Altogether we obtain $p'(m') \equiv_k s \equiv_k t \equiv_k q'(n')$, contradicting $p'(m') \not\equiv_k q'(n')$. □

Next, let us define the distance to the set INC for each configuration $p(m)$. We define

$$\mathsf{dist}(p(m)) \quad = \quad \min\left\{|w| : p(m) \xrightarrow{w} \mathsf{INC}\right\}.$$

By convention we put $\min \emptyset = \omega$. Note that $p(m) \equiv q(n)$ implies $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n))$.

4.2 Partitioning the 3D Space into Initial Space, Belt Space and Background Space

Let us formally define belts, see also [14,12,11,3]. Let $\alpha, \beta \geq 1$ be relatively prime. The *belt of thickness* d *and slope* $\frac{\alpha}{\beta}$ consists of those pairs $(m, n) \in \mathbb{N} \times \mathbb{N}$ that satisfy $|\alpha \cdot m - \beta \cdot n| \leq d$. An example of a belt is depicted in Figure 2.

Similarly as in [24] we say that two integers m and n are (γ, d)-*rationally related* if there are $\alpha, \beta \in [1, \gamma]$ that are relatively prime such that (m, n) is inside the belt of thickness d and of slope $\frac{\alpha}{\beta}$.

We call $w = a_1 \cdots a_n \in A^+$ $(n \geq 1)$ a *simple cycle* from $p(m)$ if the corresponding unique computation $p_0(m_0) \xrightarrow{a_1} p_1(m_1) \cdots \xrightarrow{a_n} p_n(m_n)$ (i.e. $p_0(m_0) = p(m)$) satisfies $p_0 = p_n$ and $p_i \neq p_j$ for all $i, j \in [1, n]$ with $i \neq j$. In case $n_0 > n_m$ we call $n_0 - n_m$ the *counter loss* of w from $p(m)$.

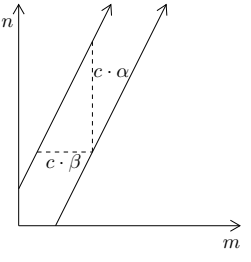


Fig. 2. A belt

The next lemma from [3] states that minimal words from configurations to INC can be chosen in a certain normal form: One first executes a polynomially long prefix, then repeatedly some most effective simple cycle (i.e. a simple cycle where the quotient of counter loss and length is maximal), and finally some polynomially long suffix.

Lemma 6 (Lemma 10 in [3]). *There is some polynomial* poly_0 *such that if* $p(m) \longrightarrow^* \mathsf{INC}$ *then already for some word* $u = u_1(u_2)^r u_3$ *(with* $r \geq 0$*) we have* $p(m) \xrightarrow{u} \mathsf{INC}$, *where (i)* $|u| = \mathsf{dist}(p(m))$, *(ii)* $|u_1 u_3| \leq \mathsf{poly}_0(k)$, *and (iii)* $|u_2| \leq k$, *and (iv) either* $u_2 = \varepsilon$ *or* u_2 *is a simple cycle of counter loss from* $[1, k]$.

The following lemma from [3] allows us to partition the 3D space.

Lemma 7 (Points 3. and 4. of Lemma 11 in [3]). *There are polynomials* poly_1 *and* poly_2 *s.t. if* $\max\{m,n\} > \mathsf{poly}_2(k)$ *and* $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n)) < \omega$*, then* (m,n)

(1) lies in a unique belt of thickness $\mathsf{poly}_1(k)$ *and slope* $\frac{\alpha}{\beta}$*, where* $\alpha,\beta \in [1,k^2]$ *and*
(2) is not neighbor to any point (m',n') *inside a different belt of thickness* $\mathsf{poly}_1(k)$ *and slope* $\frac{\alpha'}{\beta'}$ *with* $\alpha',\beta' \in [1,k^2]$*, i.e.* $\min\{|m-m'|,|n-n'|\} \geq 2$.

We now partition $\mathbb{N}\times\mathbb{N}\times(Q\times Q)$ into the following three subspaces, cf. Figure 1:

- *initial space*: All points $\langle m,n,(p,q)\rangle$ such that $m,n \leq \mathsf{poly}_2(k)$.
- *belt space*: All points $\langle m,n,(p,q)\rangle$ outside the initial space such that m and n are $(k^2,\mathsf{poly}_1(k))$-rationally related: By Lemma 7 the belt in which (m,n) lies is uniquely determined.
- *background space:* All remaining points.

4.3 Bounding the Minimal Witness

In this section we demonstrate the core of the proof of Lemma 2: any minimal witness w for $\langle p_{\text{init}}(0), q_{\text{init}}(0)\rangle$ is polynomially bounded in k. For the rest of this section we will assume that $p_{\text{init}}(0) \not\equiv q_{\text{init}}(0)$ and that w is a minimal witness for them.

Recall that $k = |Q|$. Our first lemma tells us once the minimal witness enters the background space at some point $\langle m,n,(p,q)\rangle$ then its remaining suffix is bounded by $k\cdot(\max\{m,n\}+1)+\mathsf{poly}_0(k)$.

Lemma 8. *For each point* $\langle m,n,(p,q)\rangle$ *in the background space we have* $p(m)\equiv q(n)$ *if and only if* $p(m)\equiv_\ell q(n)$*, where* $\ell \leq k\cdot(\max\{m,n\}+1)+\mathsf{poly}_0(k)$.

Proof. The "only if"-direction is trivial. For the "if"-direction assume $p(m) \not\equiv q(n)$. Since $\langle m,n,(p,q)\rangle$ is in the background space we cannot have $\mathsf{dist}(p(m)) = \mathsf{dist}(q(n)) < \omega$ by Point (1) of Lemma 7. In case $\mathsf{dist}(p(n)) = \mathsf{dist}(q(n)) = \omega$, then already for $\ell = k$ we have $p(m) \not\equiv_\ell q(n)$ by Lemma 5. So it remains to consider the case $\mathsf{dist}(p(m)) < \mathsf{dist}(q(n))$ without loss of generality, in particular $\mathsf{dist}(p(m)) < \omega$. Let u be a minimal word such that $p(m) \xrightarrow{u} p'(m')$ for some $p'(m') \in \mathsf{INC}$, note that if $q(n) \xrightarrow{u} q'(n')$, then $p'(m') \not\equiv_k q'(n')$. By applying Lemma 6, we can choose $u = u_1(u_2)^r u_3$ for some $r \geq 0$ such that (i) $|u| = \mathsf{dist}(p(m))$, (ii) $|u_1u_3| \leq \mathsf{poly}_0(k)$, (iii) $|u_2| \leq k$ and (iv) either $u_2 = \varepsilon$ or u_2 is a simple cycle of counter loss from $[1,k]$. This implies that already for $\ell = |u|+k \leq k\cdot m+\mathsf{poly}_0(k)+k$ we have $p(m) \not\equiv_\ell q(n)$. □

With this lemma one now observes that in case $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w)$ enters the background space after polynomially many steps, then the whole computation is polynomially bounded (the two counters are initialized with zero and by a polynomially bounded computation we can only obtain polynomially large counter values).

Thus, it suffices to focus on the longest prefix w_1 of w such that $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$ enters the background space for at most one point (i.e. if at all, then the last one). Thus, $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$ entirely stays inside the initial space or the belt space (except for the last point possibly). For the rest of this section will show that the length of w_1 is polynomially bounded in k.

First observe that if $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$ does not leave the initial space, then $|w_1|$ is trivially polynomially bounded since the size of the initial space is polynomially bounded by definition. So for the rest of this section assume that $\mathsf{Comp}(p(0), q(0), w_1)$ enters at least one belt.

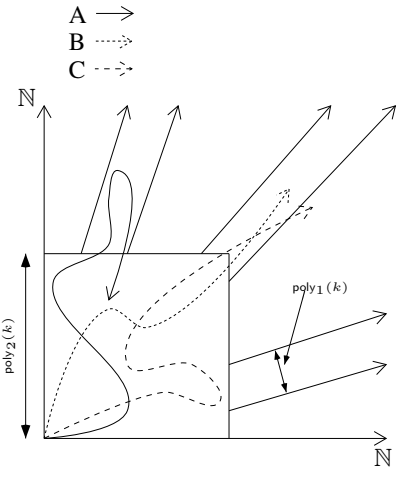


Fig. 3. Possible belt visits

In the following, whenever we talk about a belt we mean its points *outside* the 2D projection of the initial space. Recall that we made the initial space sufficiently large such that there are no intersections between belts and one cannot switch from one belt to another in one step (recall Point (2) of Lemma 7). Let us fix a computation π. A *belt visit* (with respect to some belt B) is a maximal factor of π whose points are all entirely in B. It is clear that $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$ can contain at most polynomially many belt visits. The following cases for belt visits can now be distinguished (a 2D projection of these cases is depicted in Figure 3):

- **Case A:** The initial space is visited immediately after the belt visit.
- **Case B:** The belt visit ends in the belt.
- **Case C:** The background space is visited immediately after the belt visit.

The goal of this section is to prove the following lemma.

Lemma 9. *Every belt visit of* $\mathsf{Comp}(p_{init}(0), q_{init}(0), w_1)$ *is polynomially bounded in* k.

First, we need some more notation. Let $\alpha, \beta \in [1, k^2]$ be relatively prime. We assume $\alpha \geq \beta$, i.e. $\frac{\alpha}{\beta} \geq 1$. The case when $\alpha < \beta$ can be proven analogously.

Points $\langle p(m), q(n)\rangle$ and $\langle p'(m'), q'(n')\rangle$ are $\frac{\alpha}{\beta}$-*related* if $p = p'$, $q = q'$, and $\alpha \cdot m - \beta \cdot n = \alpha \cdot m' - \beta \cdot n'$. Roughly speaking, they are $\frac{\alpha}{\beta}$-related if their control states coincide and they lie on a line with slope $\frac{\alpha}{\beta}$. An $\frac{\alpha}{\beta}$-*repetition* is a computation $\pi_0\pi_1 \cdots \pi_\ell$ such that π_0 and π_ℓ are $\frac{\alpha}{\beta}$-related. Figure 4 shows an example of an $\frac{\alpha}{\beta}$-repetition that lies inside some belt (these are the $\frac{\alpha}{\beta}$-repetitions we will be interested in).

Before we handle the cases **A**, **B**, and **C**, let us fix a belt B with slope $\frac{\alpha}{\beta}$. We will make use of the following claim.

Claim*: There is a polynomial poly_3 such that for each sequence of points $\langle p_0(m_0), q_0(n_0)\rangle \cdots \langle p_h(m_h), q_h(n_h)\rangle$ in B with $h = \mathsf{poly}_3(k)$ and $m_i = m_{i-1} + 1$ for each $i \in [h]$, there are two indices $0 \leq i < i' \leq h$ such that $\langle p_i(m_i), q_i(n_i)\rangle$ and $\langle p_{i'}(m_{i'}), q_{i'}(n_{i'})\rangle$ are $\frac{\alpha}{\beta}$-related.

Proof. Define $d_j = \alpha \cdot m_j - \beta \cdot n_j$ for each $j \in [0, h]$. Since the thickness of B is $\mathsf{poly}_1(k)$, there are at most polynomially many different values for d_j. Hence (for sufficiently large h) by the pigeonhole principle we can find two points $\langle p_i(m_i), q_i(m_i)\rangle$ and $\langle p_{i'}(m_{i'}), q_{i'}(n_{i'})\rangle$ such that $p_i = p_{i'}$, $q_i = q_{i'}$, and $d_i = d_{i'}$. □

Let us now analyze the possible belt visit cases **A**, **B**, and **C**. Note that cases **B** and **C** can occur only in the last belt visit of $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$. For the rest of this section let us fix some B-belt visit $\pi = \pi_0\pi_1 \cdots \pi_z$ of $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$, where $\pi_i = \langle p_i(m_i), q_i(n_i)\rangle$ for each $i \in [0, z]$.

Case A: The intuition behind this case is the following: Consider a long belt visit returning to the initial space. Then we can find two $\frac{\alpha}{\beta}$-repetitions that are factors of π, one going up and one going down with inverse counter effects. We can cut them out and obtain a shorter computation.

Let us assume that the length of π is sufficiently large such that there is some point π_h on π for the following arguments to work. Define for each suitable $m \in \mathbb{N}$

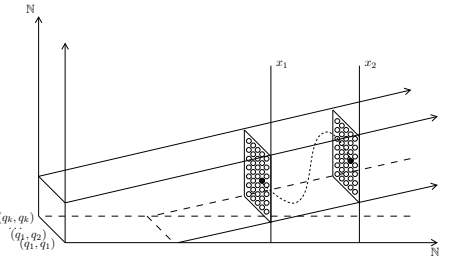


Fig. 4. $\frac{\alpha}{\beta}$-repetition inside a belt

$$\mathcal{L}(m) = \max\{i \mid m = m_i, i \in [0, h]\} \quad \text{and} \quad \mathcal{R}(m) = \min\{i \mid m = m_i, i \in [h, z]\}.$$

Recall that $\mathsf{poly}_2(k)$ was the height and width of the initial space. By a similar pigeonhole argument as the proof of Claim* there are H and J (since m_h is sufficiently large) such that (i) $\mathsf{poly}_2(k) < H < J < m_h$, (ii) Points π_c and $\pi_{c'}$ are $\frac{\alpha}{\beta}$-related where $c = \mathcal{L}(H)$ and $c' = \mathcal{L}(J)$ and (iii) Points π_d and $\pi_{d'}$ are $\frac{\alpha}{\beta}$-related where $d = \mathcal{R}(H)$ and $d' = \mathcal{R}(J)$. Note that the pair of counter effects from π_c to $\pi_{c'}$ and from $\pi_{d'}$ to π_d add up to $(0, 0)$ componentwise. One can now split up the computation π into $\pi_0 \xrightarrow{\gamma_1} \pi_c \xrightarrow{\gamma_2} \pi_{c'} \xrightarrow{\gamma_3} \pi_{d'} \xrightarrow{\gamma_4} \pi_d \xrightarrow{\gamma_5} \pi_z$. Note that by construction we have $m_i \geq J$ for each $i \in [c', d']$. Since $\alpha \geq \beta$ we can safely cut out the computations $\pi_c \xrightarrow{\gamma_2} \pi_{c'}$ and $\pi_{d'} \xrightarrow{\gamma_4} \pi_d$ and obtain the computation $\pi_0 \xrightarrow{\gamma_1} \pi_c \xrightarrow{\gamma_3} \pi_d \xrightarrow{\gamma_5} \pi_z$. In $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$ we can replace π by this computation and can hence obtain a shorter witness. However, this contradicts minimality of w.

Case B: Let us assume that the belt visit ends in the belt. Since we are considering a computation of a witness we have $p_z(m_z) \not\equiv_0 q_z(n_z)$ for some $m_z, n_z \geq 1$. Thus, $p_z(m) \not\equiv_0 q_z(n)$ for each $m, n \geq 1$. Let us assume π stays in the belt sufficiently long for the following argument to work. By the pigeonhole principle there are i and j with $0 \leq i < j \leq z$ and $j - i \leq k^2$ such that $p_i = p_j$ and $q_i = q_j$. We can assume that m_i, n_i, m_j, and n_j are sufficiently large that we can cut out the computation between π_i and π_j without reaching zero in the rest of the computation. We obtain a shorter computation ending in a point with pair of control states (p_z, q_z), hence contradicting minimality of w.

Case C: Let us assume that the first point after executing π lies in the background space, say in some point $\langle \widehat{p}(\widehat{m}), \widehat{q}(\widehat{n})\rangle$. In other words $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$ ends in $\langle \widehat{p}(\widehat{m}), \widehat{q}(\widehat{n})\rangle$ and π is the last belt visit of $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w_1)$.

First let us consider the case when there is a factor of π that goes "leftward" (and hence necessarily "downward") in the belt for too long. Formally we mean that there

is some sufficiently large polynomial poly_4 such that π contains a factor whose counter effect (d_1, d_2) satisfies $d_1 \leq -\mathsf{poly}_4(k)$ or $d_2 \leq -\mathsf{poly}_4(k)$ and the following argument can be realized: There is some point π_h whose counter values are both sufficiently large to which we can apply the same arguments as in Case A and thus obtain a shorter computation, contradicting minimality of w.

Thus, we can assume that for every factor of π with the counter effect (d_1, d_2) we have $d_1, d_2 > -\mathsf{poly}_4(k)$. One can now prove the existence of some polynomial $\mathsf{poly}_5(k)$ for the following arguments to work. In case $\widehat{m} \leq \mathsf{poly}_5(k)$, then $\widehat{n}$ is polynomially bounded and hence π is polynomially bounded.

In case $\widehat{m} > \mathsf{poly}_5(k)$, we do not directly contradict minimality of w but we show the existence of some polynomially bounded computation π' that distinguishes $p_{\text{init}}(0)$ and $q_{\text{init}}(0)$. We distinguish the following subcases: **C1:** $\mathsf{dist}(\widehat{p}(\widehat{m})) < \omega$, **C2:** $\mathsf{dist}(\widehat{q}(\widehat{n})) < \omega$ and **C3:** $\mathsf{dist}(\widehat{p}(\widehat{m})) = \mathsf{dist}(\widehat{q}(\widehat{n})) = \omega$.

C1: We note that from $\langle \widehat{p}(\widehat{m}), \widehat{q}(\widehat{n}) \rangle$ we do not care how π exactly looks like. However, we will prove that one can obtain such a polynomially bounded π' by repeatedly cutting out (polynomially long) $\frac{\alpha}{\beta}$-repetitions from π with the invariant that after each cutting-out the resulting computation can be extended in one step to a background point whose first configuration *still* has finite distance to INC.[1]
By assumption $\mathsf{dist}(\widehat{p}(\widehat{m})) < \omega$, so let u be a minimal word such that $\widehat{p}(\widehat{m}) \xrightarrow{u} \mathsf{INC}$. By Lemma 6 and since $\widehat{m}$ is sufficiently large, we can choose u as $u = u_1(u_2)^r u_3$ for some $r \geq 0$, where $|u_1 u_3| \leq \mathsf{poly}_0(k)$, $|u_2| \leq k$, and u_2 is a simple cycle of counter loss $d \in [1, k]$. This implies $\widehat{p}(\widehat{m} - jd) \xrightarrow{u_1(u_2)^{r-j}u_3} \mathsf{INC}$ for each $j \in [r]$.

Define $\lambda(m) = \max\{i \mid m_i = m, i \in [0, z]\}$ for each $m \in [\mathsf{poly}_2(k) + 1, \widehat{m} - 1]$. We note that $\lambda(m+1) - \lambda(m)$ is polynomially bounded for each $m, m+1 \in [\mathsf{poly}_2(k) + 1, \widehat{m} - 1]$ since the negative counter effect of each factor of π is polynomially bounded by assumption. Since $\widehat{m}$ assumed to be sufficiently large we can apply Claim* on polynomially many disjoint factors (each of length $\mathsf{poly}_3(k)$) of $\varphi = \pi_{\lambda(\mathsf{poly}_2(k)+1)} \cdots \pi_{\lambda(\widehat{m}-1)}$ and find an $\frac{\alpha}{\beta}$-repetition on each such factor. Each of these disjoint factors of length $\mathsf{poly}_3(k)$ of φ corresponds to a factor of π that also has only polynomial length, and so do the $\frac{\alpha}{\beta}$-repetitions of these factors. Among these $\frac{\alpha}{\beta}$-repetitions (interpreted as factors of π) we can pick out d all having the same counter effect, say (f, g); in particular $\frac{f}{g} = \frac{\alpha}{\beta}$. When cutting out precisely these d factors from $\mathsf{Comp}(p_{\text{init}}(0), q_{\text{init}}(0), w)$ it enters the background space at point $\langle \widehat{p}(\widehat{m} - df), \widehat{q}(\widehat{n} - dg) \rangle$ for the first time. By $\widehat{p}(\widehat{m} - jd) \xrightarrow{u_1(u_2)^{r-j}u_3} \mathsf{INC}$ for each $j \in [r]$ we have that $\widehat{p}(\widehat{m} - df)$ can reach INC. We can apply this cutting-out process repeatedly until the first point that enters the background space, say $\langle \widehat{p}(\widehat{m} - \Delta), \widehat{q}(\widehat{n} - \Delta') \rangle$, satisfies $\widehat{m} - \Delta \leq \mathsf{poly}_5(k)$.
C2: This case is symmetric to case C1.
C3: Since $\mathsf{dist}(\widehat{p}(\widehat{m})) = \mathsf{dist}(\widehat{q}(\widehat{n})) = \omega$ and $\widehat{p}(\widehat{m}) \not\equiv \widehat{q}(\widehat{n})$ we know from Lemma 5 that already some $u \in \Sigma^{\leq k}$ distinguishes $\widehat{p}(\widehat{m})$ and $\widehat{q}(\widehat{n})$. So as in Case B, if π is

[1] We note that we have to require that after the cutting-out the first configuration of the earliest point that is in the background space *must* still have finite distance to INC, for otherwise both configurations could have infinite distance to INC and could be language equivalent.

sufficiently long inside the belt, we can cut out a factor of repeated control state pairs and obtain a shorter witness for $p_{\text{init}}(0)$ and $q_{\text{init}}(0)$, thus contradicting minimality of w.

5 Regularity is NL-Complete

Theorem 10. *Regularity of ROCA, i.e. given a ROCA $\mathcal{A}$ deciding if $L(\mathcal{A})$ is regular, is* NL*-complete.*

Proof. For the *upper bound*, let us fix a ROCA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with $k = |Q|$. Recall the definition of the set INC and for each configuration $p(m)$ its shortest distance $\mathsf{dist}(p(m))$ to INC. We make use of the following characterizations, in analogy to [3]. The following statements are equivalent: (1) $L(q_0(0))$ is *not* regular, (2) for all $d \in \mathbb{N}$ there is some configuration $q(n)$ with $q_0(0) \longrightarrow^* q(n) \longrightarrow^* \mathsf{INC}$ and $d \leq \mathsf{dist}(q(n)) < \omega$, (3) there exists some $q \in Q$ such that $q_0(0) \longrightarrow^* q(2k) \longrightarrow^* \mathsf{INC}$.

For an NL upper bound note that, given a configuration $q(n)$ where n is in unary, deciding if $q(n) \in \mathsf{INC}$ can be done in NL, since $q(n) \in \mathsf{INC}$ if and only if for all $r \in Q$ there is some $w_r \in \Sigma^{\leq k}$ that distinguishes $q(n)$ and the state r of $\mathcal{A}$'s underlying DFA. Second, deciding condition (3) is in NL as well, since the length of such a witnessing path is polynomially bounded. Hence deciding regularity of $\mathcal{A}$ is in NL.

For the *lower bound*, we give a logspace reduction from the emptiness problem for DFA. One can compute in logspace from a given DFA $\mathcal{F}$ a ROCA $\mathcal{A}$ such that $L(\mathcal{A}) = \{a^n\$w\$b^n \mid w \in L(\mathcal{F})\}$. Hence $L(\mathcal{A})$ is regular (in particular empty) if and only if $L(\mathcal{F}) = \emptyset$. □

6 Conclusion

In this paper we have shown that language equivalence and regularity of ROCA is NL-complete. Using the idea of considering the reachability status of configurations to INC, we can extend our result to prove that it is NL-complete to decide language equivalence of a ROCA and a simple DOCA or to decide regularity of a simple DOCA. A *simple DOCA* is a ROCA that allows spontaneous counter resets (ε-moves) from $p(m)$ to $q(0)$ for some control state q but necessarily *for all* $m \geq 1$: In such configurations $p(m)$ with $m \geq 1$ one can only reset the counter and not read any symbols. We note that simple DOCA and DOCA are equi-expressive but DOCA are exponentially more succinct. The precise complexity of equivalence of DOCA is left for future work.

Acknowledgments. We thank Géraud Sénizergues and Etsuji Tomita for discussions.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. of STOC, pp. 202–211. ACM, New York (2004)
2. Berman, P., Roos, R.: Learning One-Counter Languages in Polynomial Time (Extended Abstract). In: Proc. of FOCS, pp. 61–67. IEEE, Los Alamitos (1987)
3. Böhm, S., Göller, S., Jančar, P.: Bisimilarity of one-counter processes is PSPACE-complete. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 177–191. Springer, Heidelberg (2010)

4. Caucal, D.: Synchronization of Pushdown Automata. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 120–132. Springer, Heidelberg (2006)
5. Fahmy, A.F., Roos, R.S.: Efficient Learning of Real Time One-Counter Automata. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS, vol. 997, pp. 25–40. Springer, Heidelberg (1995)
6. Friedman, E.P.: The inclusion problem for simple languages. Theor. Comput. Sci. 1(4), 297–316 (1976)
7. Higuchi, K., Wakatsuki, M., Tomita, E.: A polynomial-time algorithm for checking the inclusion for real-time deterministic restricted one-counter automata which accept by final state. IEICE Trans. Information and Systems E78-D, 939–950 (1995)
8. Higuchi, K., Wakatsuki, M., Tomita, E.: A polynomial-time algorithm for checking the inclusion for real-time deterministic restricted one-counter automata which accept by accept mode. IEICE Trans. Information and Systems E81-D, 1–11 (1998)
9. Hirshfeld, Y., Jerrum, M., Moller, F.: A Polynomial Algorithm for Deciding Bisimilarity of Normed Context-Free Processes. Theor. Comput. Sci. 158(1&2), 143–159 (1996)
10. Immerman, N.: Nondeterministic Space is Closed Under Complementation. SIAM J. Comput. 17(5), 935–938 (1988)
11. Jančar, P.: Decidability of bisimilarity for one-counter processes. Information Computation 158(1), 1–17 (2000)
12. Jančar, P., Kučera, A., Moller, F.: Simulation and bisimulation over one-counter processes. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 334–345. Springer, Heidelberg (2000)
13. Jančar, P., Kučera, A., Moller, F., Sawa, Z.: DP lower bounds for equivalence-checking and model-checking of one-counter automata. Inf. Comput. 188(1), 1–19 (2004)
14. Jančar, P., Moller, F., Sawa, Z.: Simulation Problems for One-Counter Machines. In: Bartosek, M., Tel, G., Pavelka, J. (eds.) SOFSEM 1999. LNCS, vol. 1725, pp. 404–413. Springer, Heidelberg (1999)
15. Mayr, R.: Undecidability of Weak Bisimulation Equivalence for 1-Counter Processes. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 570–583. Springer, Heidelberg (2003)
16. Minsky, M.L.: Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. Annals of Mathematics. Second Series 74, 437–455 (1961)
17. Nowotka, D., Srba, J.: Height-Deterministic Pushdown Automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 125–134. Springer, Heidelberg (2007)
18. Oyamaguchi, M.: The equivalence problem for real-time DPDAs. J. ACM 34, 731–760 (1987)
19. Roos, R.: Deciding Equivalence of Deterministic One-Counter Automata in Polynomial Time with Applications to Learning. PhD thesis, The Pennsylvania State University (1988)
20. Sénizergues, G.: L(A)=L(B)? decidability results from complete formal systems. Theor. Comput. Sci. 251(1-2), 1–166 (2001)
21. Sénizergues, G.: L(A)=L(B)? A simplified decidability proof. Theor. Comput. Sci. 281(1-2), 555–608 (2002)
22. Sénizergues, G.: The Equivalence Problem for t-Turn DPDA Is Co-NP. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 478–489. Springer, Heidelberg (2003)
23. Stirling, C.: Deciding DPDA Equivalence Is Primitive Recursive. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 821–832. Springer, Heidelberg (2002)
24. Valiant, L.G., Paterson, M.: Deterministic one-counter automata. J. Comput. Syst. Sci. 10(3), 340–350 (1975)

Energy and Mean-Payoff Parity Markov Decision Processes*

Krishnendu Chatterjee[1] and Laurent Doyen[2]

[1] IST Austria (Institute of Science and Technology Austria)
[2] LSV, ENS Cachan & CNRS, France

Abstract. We consider Markov Decision Processes (MDPs) with mean-payoff parity and energy parity objectives. In system design, the parity objective is used to encode ω-regular specifications, while the mean-payoff and energy objectives can be used to model quantitative resource constraints. The energy condition requires that the resource level never drops below 0, and the mean-payoff condition requires that the limit-average value of the resource consumption is within a threshold. While these two (energy and mean-payoff) classical conditions are equivalent for two-player games, we show that they differ for MDPs. We show that the problem of deciding whether a state is almost-sure winning (i.e., winning with probability 1) in energy parity MDPs is in NP $\cap$ coNP, while for mean-payoff parity MDPs, the problem is solvable in polynomial time.

1 Introduction

Markov decision processes (MDPs) are a standard model for systems that exhibit both stochastic and nondeterministic behaviour. The nondeterminism represents the freedom of choice of control actions, while the probabilities describe the uncertainty in the response of the system to control actions. The control problem for MDPs asks whether there exists a strategy (or policy) to select control actions in order to achieve a given goal with a certain probability. MDPs have been used in several areas such as planning, probabilistic reactive programs, verification and synthesis of (concurrent) probabilistic systems [12,22,1].

The control problem may specify a goal as a set of desired traces (such as ω-regular specifications), or as a quantitative optimization objective for a payoff function defined on the traces of the MDP. Typically, discounted-payoff and mean-payoff functions have been studied [15]. Recently, the energy objectives (corresponding to total-payoff functions) have been considered in the design of resource-constrained embedded systems [3,7,20] such as power-limited systems, as well as in queueing processes, and gambling models (see also [4] and references therein). The energy objective requires that the sum of the rewards be always nonnegative along a trace. Energy objective can be expressed in the setting of boundaryless one-counter MDPs [4]. In the case of MDPs, achieving energy objective with probability 1 is equivalent to achieving energy objective in the stronger setting of a two-player game where the probabilistic choices are

* This work was partially supported by FWF NFN Grant S11407-N23 (RiSE) and a Microsoft faculty fellowship.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 206–218, 2011.

replaced by adversarial choice. This is because if a trace ρ violates the energy condition in the game, then a finite prefix of ρ would have a negative energy, and this finite prefix has positive probability in the MDP. Note that in the case of two-player games, the energy objective is equivalent to enforce nonnegative mean-payoff value [3,5].

In this paper, we consider MDPs equipped with the combination of a parity objective (which is a canonical way to express the ω-regular conditions [21]), and a quantitative objective specified as either mean-payoff or energy condition. Special cases of the parity objective include reachability and fairness objectives such as Büchi and coBüchi conditions. Such combination of quantitative and qualitative objectives is crucial in the design of reactive systems with both resource constraints and functional requirements [6,11,3,2]. For example, Kucera and Stražovský consider the combination of PCTL with mean-payoff objectives for MDPs and present an EXPTIME algorithm [19]. In the case of energy parity condition, it can also be viewed as a natural extension of boundaryless one-counter MDPs with fairness conditions.

Consider the MDP in Fig. 1, with the objective to visit the Büchi state q_2 infinitely often, while maintaining the energy level (i.e., the sum of the transition weights) nonnegative. A winning strategy from q_0 would loop 20 times on q_0 to accumulate energy and then it can afford to reach the probabilistic state from which the Büchi state is reached with probability $\frac{1}{2}$ and cost 20. If the Büchi state is not reached immediately, then the strategy needs to recharge 10 units of energy and try again. This strategy uses memory and it is also winning with probability 1 for the nonnegative mean-payoff Büchi objective. In general however, the energy and mean-payoff parity objectives do not coincide (see later the example in Fig. 2). In particular, the memory requirement for energy parity objective is finite (at most exponential) while it may be infinite for mean-payoff parity.

We study the computational complexity of the problem of deciding if there exists a strategy to achieve energy parity objective, or mean-payoff parity objective with probability 1 (i.e., almost-surely). We provide the following bounds for these problems.

1. For energy parity MDPs, we show that the problem is in NP $\cap$ coNP, and present a pseudo-polynomial time algorithm. Since parity games polynomially reduce to two-player energy games [18,3,5], and thus to energy MDPs, the problem for almost-sure energy parity MDPs is at least as hard as solving two-player parity games.
2. For mean-payoff parity MDPs, we show that the problem is solvable in polynomial time (and thus PTIME-complete).

We refer to [12,16,9] for importance of the computation of almost-sure winning set related to robust solutions (independence of precise transition probabilities) and the more general quantitative problem. The computation of the almost-sure winning set in MDPs typically relies either on the end-component analysis, or analysis of attractors and sub-MDPs. Our results for mean-payoff parity objectives rely on the end-component analysis, but in a more refined way than the standard analysis, to obtain a polynomial-time algorithm. Our proof combines techniques for mean-payoff and parity objectives to produce infinite-memory strategy witnesses, which is necessary in general. We present an algorithm that iterates successively over even priorities $2i$ and computes almost-sure

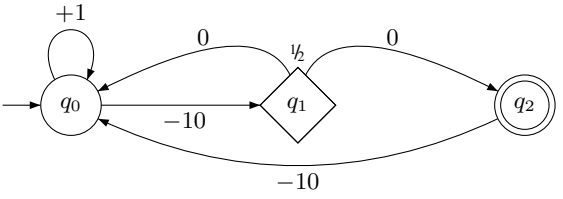


Fig. 1. An energy Büchi MDP. The player-1 states are q_0, q_2, and the probabilistic state is q_1.

winning end-components with the even priority $2i$ as the best priority. The problem of positive mean-payoff objectives and parity objectives has been considered independently in [17].

For energy parity MDPs the end-component based analysis towards polynomial-time algorithm does not work since solving energy parity MDPs is at least as hard as solving two-player parity games. Instead, for energy parity MDPs, we present a quadratic reduction to two-player energy Büchi games which are in NP ∩ coNP and solvable in pseudo-polynomial time [7].

From our results, it follows that for energy parity MDPs, strategies with finite memory are sufficient (linear in the number of states times the value of the largest weight), while infinite memory may be necessary for mean-payoff parity MDPs. The details of the proofs can be found in [8], as well as the solution for disjunction of mean-payoff parity and energy parity objectives. An interesting open question is to extend the results of this paper from MDPs to two-player stochastic games.

2 Definitions

Probability distributions. A *probability distribution* over a finite set A is a function $\kappa : A \to [0,1]$ such that $\sum_{a \in A} \kappa(a) = 1$. The *support* of κ is the set $\mathsf{Supp}(\kappa) = \{a \in A \mid \kappa(a) > 0\}$. We denote by $\mathcal{D}(A)$ the set of probability distributions on A.

Markov Decision Processes. A *Markov Decision Process* (MDP) $M = (Q, E, \delta)$ consists of a finite set Q of states partitioned into *player*-1 *states* Q_1 and *probabilistic states* Q_P (i.e., $Q = Q_1 \cup Q_P$ and $Q_1 \cap Q_P = \emptyset$), 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 $(q, q') \in E$, and a probabilistic transition function $\delta : Q_P \to \mathcal{D}(Q)$ such that for all $q \in Q_P$ and $q' \in Q$, we have $(q, q') \in E$ iff $\delta(q)(q') > 0$. We often write $\delta(q, q')$ for $\delta(q)(q')$. For a state $q \in Q$, we denote by $E(q) = \{q' \in Q \mid (q, q') \in E\}$ the set of possible successors of q.

End-components and Markov chains. A set $U \subseteq Q$ is *δ-closed* if for all $q \in U \cap Q_P$ we have $\mathsf{Supp}(\delta(q)) \subseteq U$. The sub-MDP induced by a δ-closed set U is $M \restriction U = (U, E \cap (U \times U), \delta)$. Note that $M \restriction U$ is an MDP if for all $q \in U$ there exists $q' \in U$ such that $(q, q') \in E$. A *Markov chain* is a special case of MDP where $Q_1 = \emptyset$. A *closed recurrent set* for a Markov chain is a δ-closed set $U \subseteq Q$ which is strongly connected. End-components in MDPs play a role equivalent to closed recurrent sets in Markov chains. Given an MDP $M = (Q, E, \delta)$ with partition (Q_1, Q_P), a set $U \subseteq Q$

of states is an *end-component* if U is δ-closed and the sub-MDP $M \restriction U$ is strongly connected [12,13]. We denote by $\mathcal{E}(M)$ the set of end-components of an MDP M.

Plays. An MDP can be viewed as the arena of a game played for infinitely many rounds from a state $q_0 \in Q$ as follows. If the game is in a player-1 state q, then player 1 chooses the successor state in the set $E(q)$; otherwise the game is in a probabilistic state q, and the successor is chosen according to the probability distribution $\delta(q)$. This game results in a *play* from q_0, i.e., an infinite path $\rho = q_0 q_1 \dots$ such that $(q_i, q_{i+1}) \in E$ for all $i \geq 0$. The prefix of length n of ρ is denoted by $\rho(n) = q_0 \dots q_n$, the last state of $\rho(n)$ is $\mathsf{Last}(\rho(n)) = q_n$. We write Ω for the set of all plays.

Strategies. A *strategy* (for player 1) is a function $\sigma : Q^* Q_1 \to \mathcal{D}(Q)$ such that for all $\rho \in Q^*$, $q \in Q_1$, and $q' \in Q$, if $\sigma(\rho \cdot q)(q') > 0$, then $(q, q') \in E$. We denote by Σ the set of all strategies. An *outcome* of σ from q_0 is a play $q_0 q_1 \dots$ where $q_{i+1} \in \mathsf{Supp}(\sigma(q_0 \dots q_i))$ for all $i \geq 0$ such that $q_i \in Q_1$. Strategies that do not use randomization are called pure. A player-1 strategy σ is *pure* if for all $\rho \in Q^*$ and $q \in Q_1$, there is a state $q' \in Q$ such that $\sigma(\rho \cdot q)(q') = 1$.

Outcomes and measures. Once a starting state $q \in Q$ and a strategy $\sigma \in \Sigma$ are fixed, the outcome of the game is a random walk ω_q^σ for which the probabilities of every *event* $\mathcal{A} \subseteq \Omega$, which is a measurable set of plays, are uniquely defined [22]. For a state $q \in Q$ and an event $\mathcal{A} \subseteq \Omega$, we denote by $\mathbb{P}_q^\sigma(\mathcal{A})$ the probability that a play belongs to $\mathcal{A}$ if the game starts from the state q and player 1 follows the strategy σ. For a measurable function $f : \Omega \to \mathbb{R}$ we denote by $\mathbb{E}_q^\sigma[f]$ the *expectation* of the function f under the probability measure $\mathbb{P}_q^\sigma(\cdot)$.

Finite-memory strategies. A strategy uses *finite-memory* if it can be encoded by a deterministic transducer $\langle \mathsf{Mem}, m_0, \alpha_u, \alpha_n \rangle$ where Mem is a finite set (the memory of the strategy), $m_0 \in \mathsf{Mem}$ is the initial memory value, $\alpha_u : \mathsf{Mem} \times Q \to \mathsf{Mem}$ is an update function, and $\alpha_n : \mathsf{Mem} \times Q_1 \to \mathcal{D}(Q)$ is a next-move function. The *size* of the strategy is the number $|\mathsf{Mem}|$ of memory values. If the game is in a player-1 state q, and m is the current memory value, then the strategy chooses the next state q' according to the probability distribution $\alpha_n(m, q)$, and the memory is updated to $\alpha_u(m, q)$. Formally, $\langle \mathsf{Mem}, m_0, \alpha_u, \alpha_n \rangle$ defines the strategy σ such that $\sigma(\rho \cdot q) = \alpha_n(\hat{\alpha}_u(m_0, \rho), q)$ for all $\rho \in Q^*$ and $q \in Q_1$, where $\hat{\alpha}_u$ extends α_u to sequences of states as expected. A strategy is *memoryless* if $|\mathsf{Mem}| = 1$. For a finite-memory strategy σ, let M_σ be the Markov chain obtained as the product of M with the transducer defining σ, where $(\langle m, q \rangle, \langle m', q' \rangle)$ is an edge in M_σ if $m' = \alpha_u(m, q)$ and either $q \in Q_1$ and $q' \in \mathsf{Supp}(\alpha_n(m, q))$, or $q \in Q_P$ and $(q, q') \in E$.

Two-player games. A *two-player game* is a graph $G = (Q, E)$ with the same assumptions as for MDP, except that the partition of Q is denoted (Q_1, Q_2) where Q_2 is the set of *player*-2 *states*. The notions of play, strategies (in particular strategies for player 2), and outcome are analogous to the case of MDP [7].

Objectives. An *objective* for an MDP M (or game G) is a set $\phi \subseteq \Omega$ of infinite paths. Let $p : Q \to \mathbb{N}$ be a *priority function* and $w : E \to \mathbb{Z}$ be a *weight function* where positive numbers represent rewards. We denote by W the largest weight (in absolute value) according to w. The *energy level* of a prefix $\gamma = q_0 q_1 \dots q_n$ of a play is

$\mathsf{EL}(w,\gamma) = \sum_{i=0}^{n-1} w(q_i, q_{i+1})$, and the *mean-payoff value*[1] of a play $\rho = q_0 q_1 \dots$ is $\mathsf{MP}(w,\rho) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{EL}(w,\rho(n))$. In the sequel, when the weight function w is clear from the context we omit it and simply write $\mathsf{EL}(\gamma)$ and $\mathsf{MP}(\rho)$. We denote by $\mathsf{Inf}(\rho)$ the set of states that occur infinitely often in ρ, and we consider the following objectives:

- *Parity objectives.* The *parity* objective $\mathsf{Parity}(p) = \{\rho \in \Omega \mid \min\{p(q) \mid q \in \mathsf{Inf}(\rho)\} \text{ is even } \}$ requires that the minimum priority visited infinitely often be even. The special cases of *Büchi* and *coBüchi* objectives correspond to the case with two priorities, $p : Q \to \{0,1\}$ and $p : Q \to \{1,2\}$ respectively.
- *Energy objectives.* Given an initial credit $c_0 \in \mathbb{N}$, the *energy* objective $\mathsf{PosEnergy}(c_0) = \{\rho \in \Omega \mid \forall n \geq 0 : c_0 + \mathsf{EL}(\rho(n)) \geq 0\}$ requires that the energy level be always positive.
- *Mean-payoff objectives.* Given a threshold $\nu \in \mathbb{Q}$, the *mean-payoff* objective $\mathsf{MeanPayoff}^{\geq\nu} = \{\rho \in \Omega \mid \mathsf{MP}(\rho) \geq \nu\}$ (resp. $\mathsf{MeanPayoff}^{>\nu} = \{\rho \in \Omega \mid \mathsf{MP}(\rho) > \nu\}$) requires that the mean-payoff value be at least ν (resp. strictly greater than ν).
- *Combined objectives.* The *energy parity* objective $\mathsf{Parity}(p) \cap \mathsf{PosEnergy}(c_0)$ and the *mean-payoff parity* objective $\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{\sim\nu}$ (for $\sim \in \{\geq, >\}$) combine the requirements of parity and energy (resp., mean-payoff) objectives.

Almost-sure winning strategies. For MDPs, we say that a player-1 strategy σ is *almost-sure winning* in a state q for an objective ϕ if $\mathbb{P}_q^\sigma(\phi) = 1$. For two-player games, we say that a player-1 strategy σ is *winning* in a state q for an objective ϕ if all outcomes of σ starting in q belong to ϕ. For energy objectives with unspecified initial credit, we also say that a strategy is (almost-sure) winning if it is (almost-sure) winning for *some* finite initial credit.

Decision problems. We are interested in the following problems. Given an MDP M with weight function w and priority function p, and a state q_0,

- the *energy parity problem* asks whether there exists a finite initial credit $c_0 \in \mathbb{N}$ and an almost-sure winning strategy for the energy parity objective from q_0 with initial credit c_0. We are also interested in computing the *minimum initial credit* in q_0 which is the least value of initial credit for which there exists an almost-sure winning strategy for player 1 in q_0. A strategy for player 1 is *optimal* in q_0 if it is winning from q_0 with the minimum initial credit;
- the *mean-payoff parity problem* asks whether there exists an almost-sure winning strategy for the mean-payoff parity objective with threshold 0 from q_0. Note that it is not restrictive to consider mean-payoff objectives with threshold 0 because for $\sim \in \{\geq, >\}$, we have $\mathsf{MP}(w,\rho) \sim \nu$ iff $\mathsf{MP}(w-\nu,\rho) \sim 0$, where $w - \nu$ is the weight function that assigns $w(e) - \nu$ to each edge $e \in E$.

The two-player game version of these problems is defined analogously [7]. It is known that the initial credit problem for two-player energy games [6,3], as well as two-player parity games [14] can be solved in NP $\cap$ coNP because memoryless strategies

[1] The results of this paper hold for the definition of mean-payoff value using $\limsup$ instead of $\liminf$.

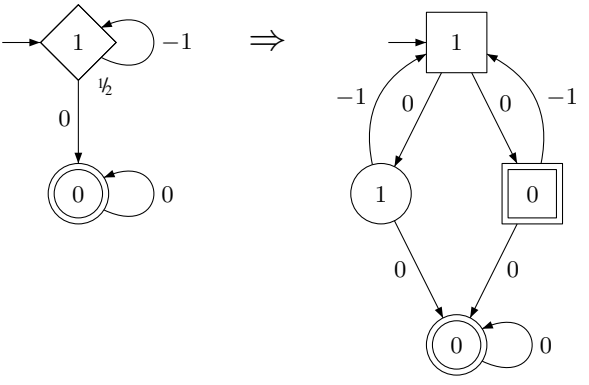


Fig. 2. The gadget construction is wrong for mean-payoff parity MDPs. Player 1 is almost-sure winning for mean-payoff Büchi in the MDP (on the left) but player 1 is losing in the two-player game (on the right) because player 2 (box-player) can force a negative-energy cycle.

are sufficient to win. Moreover, parity games reduce in polynomial time to mean-payoff games [18], which are log-space equivalent to energy games [3,5]. It is a long-standing open question to know if a polynomial-time algorithm exists for these problems. Finally, energy parity games and mean-payoff parity games are solvable in NP ∩ coNP although winning strategies may require exponential and infinite memory respectively, even in one-player games (and thus also in MDPs) [11,7].

The decision problem for MDPs with parity objective, as well as with mean-payoff objective, can be solved in polynomial time [15,12,9,13]. However, the problem is in NP ∩ coNP for MDPs with energy objective because an MDP with energy objective is equivalent to a two-player energy game (where the probabilistic states are controlled by player 2). Indeed (1) a winning strategy in the game is trivially almost-sure winning in the MDP, and (2) if an almost-sure winning strategy σ in the MDP was not winning in the game, then for all initial credit c_0 there would exist an outcome ρ of σ such that $c_0 + \mathsf{EL}(\rho(i)) < 0$ for some position $i \geq 0$. The prefix $\rho(i)$ has a positive probability in the MDP, in contradiction with the fact that σ is almost-sure winning. As a consequence, solving MDP with energy objectives is at least as hard as solving parity games.

In this paper, we show that the decision problem for MDPs with energy parity objective is in NP ∩ coNP, which is the best conceivable upper bound unless parity games can be solved in P. And for MDPs with mean-payoff parity objective, we show that the decision problem can be solved in polynomial time. The problem for MDPs with mean-payoff parity objectives under expectation semantics was considered in [10], whereas our semantics (threshold semantics) is different (we require the set of paths that satisfy the mean-payoff threshold has probability 1 rather than the expected value satisfy threshold).

The MDP in Fig. 2 on the left, which is essentially a Markov chain, is an example where the mean-payoff parity condition is satisfied almost-surely, while the energy parity condition is not, no matter the value of the initial credit. For initial credit c_0, the energy will drop below 0 with positive probability, namely $\frac{1}{2^{c_0+1}}$.

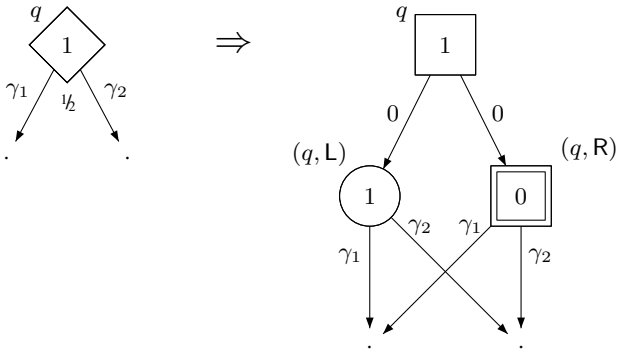


Fig. 3. Gadget for probabilistic states in energy Büchi MDP. Diamonds are probabilistic states, circles are player 1 states, and boxes are player 2 states.

End-component lemma. We now present an important lemma about end-components from [12,13] that we use in the proofs of our result. It states that for arbitrary strategies (memoryless or not), with probability 1 the set of states visited infinitely often along a play is an end-component. This lemma allows us to derive conclusions on the (infinite) set of plays in an MDP by analyzing the (finite) set of end-components in the MDP.

Lemma 1. *[12,13] Given an MDP M, for all states $q \in Q$ and all strategies $\sigma \in \Sigma$, we have $\mathbb{P}_q^\sigma(\{\omega \mid \mathsf{Inf}(\omega) \in \mathcal{E}(M)\}) = 1$.*

3 MDPs with Energy Parity Objectives

We show that energy parity MDPs can be solved in NP ∩ coNP, using a reduction to two-player energy Büchi games. Our reduction also preserves the value of the minimum initial credit. Therefore, we obtain a pseudo-polynomial algorithm for this problem, which also computes the minimum initial credit. Moreover, we show that the memory requirement for almost-sure winning strategies is at most $2{\cdot}|Q|{\cdot}W$, which is essentially optimal[2].

We first establish the results for the special case of energy Büchi MDPs. We present a reduction of the energy Büchi problem for MDPs to the energy Büchi problem for two-player games. The result then follows from the fact that the latter problem is in NP ∩ coNP and solvable in pseudo-polynomial time [7].

Given an MDP M, we can assume without loss of generality that every probabilistic state has priority 1, and has two outgoing transitions with probability ½ each [23, Section 6]. We construct a two-player game G by replacing every probabilistic state of M by a gadget as in Fig. 3. The probabilistic states q of M are mapped to player-2 states in G with two successors (q, L) and (q, R). Intuitively, player 2 chooses (q, L) to check whether player 1 can enforce the Büchi condition almost-surely. This is the case if player 1 can reach a Büchi state (with priority 0) infinitely often when he controls

[2] Example 1 in [7] shows that memory of size $2{\cdot}(|Q|-1){\cdot}W+1$ may be necessary.

the probabilistic states (otherwise, no Büchi state is ever visited, and since $(\cdot, \mathsf{L})$ states have priority 1, the Büchi condition is not realized in G). And player 2 chooses (q, R) to check that the energy condition is satisfied. If player 2 can exhaust the energy level in G, then the corresponding play prefix has positive probability in M. Note that (q, R) has priority 0 and thus cannot be used by player 2 to spoil the Büchi condition.

Formally, given $M = (Q, E, \delta)$ with partition (Q_1, Q_P) of Q, we construct a game $G = (Q', E')$ with partition (Q'_1, Q'_P) where $Q'_1 = Q_1 \cup (Q_P \times \{\mathsf{L}\})$ and $Q'_2 = Q_P \cup (Q_P \times \{\mathsf{R}\})$, see also Fig. 3. The states in Q' that are already in Q get the same priority as in M, the states $(\cdot, \mathsf{L})$ have priority 1, and the states $(\cdot, \mathsf{R})$ have priority 0. The set E' contains the following edges:

- all edges $(q, q') \in E$ such that $q \in Q_1$;
- edges $(q, (q, d))$, $((q, d), q')$ for all $q \in Q_P$, $d \in \{\mathsf{L}, \mathsf{R}\}$, and $q' \in \mathsf{Supp}(\delta(q))$.

The edges (q, q') and $((q, d), q')$ in E' get the same weight as (q, q') in M, and all edges $(q, (q, d))$ get weight 0.

Lemma 2. *Given an MDP M with energy Büchi objective, we can construct in linear time a two-player game G with energy Büchi objective such that for all states q_0 in M, there exists an almost-sure winning strategy from q_0 in M if and only if there exists a winning strategy from q_0 in G (with the same initial credit).*

Note that the reduction presented in the proof of Lemma 2 would not work for mean-payoff Büchi MDPs. Consider the MDP on Fig. 2 for which the gadget-based reduction to two-player games is shown on the right. The game is losing for player 1 both for energy and mean-payoff parity, simply because player 2 can always choose to loop through the box states, thus realizing a negative energy and mean-payoff value (no matter the initial credit). However player 1 is almost-sure winning in the mean-payoff parity MDP (on the left in Fig. 2).

While the reduction in the proof of Lemma 2 gives a game with $n' = |Q_1| + 3 \cdot |Q_P|$ states, the structure of the gadgets (see Fig. 3) is such that the energy level is independent of which of the transitions $(q, (q, \mathsf{L}))$ or $(q, (q, \mathsf{R}))$ is taken. Since from the result of [7, Lemma 8] and its proof, it follows that the memory updates in winning strategies for energy Büchi games can be done according to the energy level of the play prefix, it follows that the memory bound of $2 \cdot n \cdot W$ can be transfered to almost-sure winning strategies in Energy Büchi MDPs, where $n = |\mathsf{Win} \cap Q_1|$ is the number of player-1 almost-sure winning states. Also, the pseudo-polynomial algorithm for solving two-player energy Büchi games can be used for MDPs, with the same $O(|E| \cdot |Q|^5 \cdot W)$ complexity [7, Table 1] .

Using Lemma 2, we solve energy parity MDPs by a reduction to energy Büchi MDPs. The key idea of the reduction is that if player 1 has an almost-sure winning strategy for the energy parity objective, then player 1 can choose an even priority $2i$ and decide to satisfy the energy objective along with satisfying that priority $2i$ is visited infinitely often, and priorities less than $2i$ are visited finitely often.

W.l.o.g. we assume that player-1 states and probabilistic states alternate, i.e. $E(q) \subseteq Q_1$ for all $q \in Q_P$, and $E(q) \subseteq Q_P$ for all $q \in Q_1$. The reduction is then as follows.

Given an MDP $M = (Q, E, \delta)$ with a priority function $p : Q \to \mathbb{N}$ and a weight function $w : E \to \mathbb{Z}$, we construct $\langle M', p', w' \rangle$ as follows. M' is the MDP $M = (Q', E', \delta')$ where:

- $Q' = Q \cup (Q \times \{0, 2, \ldots, 2r\}) \cup \{\mathsf{sink}\}$ where $2r$ is the largest even priority of a state in Q. Intuitively, a state $(q, i) \in Q'$ corresponds to the state q of M from which player 1 will ensure to visit priority i (which is even) infinitely often, and never visit priority smaller than i;
- E' contains $E \cup \{(\mathsf{sink}, \mathsf{sink})\}$ and the following edges. For each probabilistic state $q \in Q_P$, for $i = 0, 2, \ldots, 2r$,
  - (a) if $p(q') \geq i$ for all $q' \in E(q)$, then $((q, i), (q', i)) \in E'$ for all $q' \in E(q)$,
  - (b) otherwise, $((q, i), \mathsf{sink}) \in E'$.

  For each player 1 state $q \in Q_1$, for each $q' \in E(q)$, for $i = 0, 2, \ldots, 2r$,
  - (a) $(q, \mathsf{sink}) \in E'$ and $((q, i), \mathsf{sink}) \in E'$, and
  - (b) if $p(q') \geq i$, then $(q, (q', i)) \in E'$ and $((q, i), (q', i)) \in E'$.

The partition (Q'_1, Q'_P) of Q' is defined by $Q'_1 = Q_1 \cup (Q_1 \times \{0, 2, \ldots, 2r\}) \cup \{\mathsf{sink}\}$ and $Q'_P = Q' \setminus Q'_1$. The weight of the edges (q, q'), $(q, (q', i))$ and $((q, i), (q', i))$ according to w' is the same as the weight of (q, q') according to w. The states (q, i) such that $p(q) = i$ have priority 0 according to p' (they are the Büchi states), and all the other states in Q' (including sink) have priority 1.

Lemma 3. *Given an MDP M with energy parity objective, we can construct in quadratic time an MDP M' with energy Büchi objective such that for all states q_0 in M, there exists an almost-sure winning strategy from q_0 in M if and only if there exists an almost-sure winning strategy from q_0 in M' (with the same initial credit).*

From the proof of Lemma 3, it follows that the memory requirement is the same as for energy Büchi MDPs. And if the weights are in $\{-1, 0, 1\}$, it follows that the energy parity problem can be solved in polynomial time.

Theorem 1. *For energy parity MDPs, (1) the decision problem of whether a given state is almost-sure winning is in NP $\cap$ coNP, and there is a pseudo-polynomial time algorithm in $O(|E| \cdot d \cdot |Q|^5 \cdot W)$ to solve it; and (2) memory of size $2 \cdot |Q| \cdot W$ is sufficient for almost-sure winning strategies.*

4 MDPs with Mean-Payoff Parity Objectives

In this section we present a polynomial-time algorithm for solving MDPs with mean-payoff parity objective. We first recall some useful properties of MDPs.

For an end-component $U \in \mathcal{E}(M)$, consider the memoryless strategy σ_U that plays in every state $s \in U \cap Q_1$ all edges in $E(s) \cap U$ uniformly at random. Given the strategy σ_U, the end-component U is a closed connected recurrent set in the Markov chain obtained by fixing σ_U.

Lemma 4. *Given an MDP M and an end-component $U \in \mathcal{E}(M)$, the strategy σ_U ensures that for all states $s \in U$, we have $\mathbb{P}_s^{\sigma_U}(\{\omega \mid \mathsf{Inf}(\omega) = U\}) = 1$.*

Expected mean-payoff value. Given an MDP M with a weight function w, the *expected mean-payoff value*, denoted $\mathsf{ValMP}(w)$, is the function that assigns to every state the maximal expectation of the mean-payoff objective that can be guaranteed by any strategy. Formally, for $q \in Q$ we have $\mathsf{ValMP}(w)(q) = \sup_{\sigma \in \Sigma} \mathbb{E}_q^{\sigma}(\mathsf{MP}(w))$, where $\mathsf{MP}(w)$ is the measurable function that assigns to a play ρ the long-run average $\mathsf{MP}(w, \rho)$ of the weights. By the classical results of MDPs with mean-payoff objectives, it follows that there exists pure memoryless optimal strategies [15], i.e., there exists a pure memoryless optimal strategy σ^* such that for all $q \in Q$ we have $\mathsf{ValMP}(w)(q) = \mathbb{E}_q^{\sigma^*}(\mathsf{MP}(w))$.

It follows from Lemma 4 that the strategy σ_U ensures that from any starting state s, any other state t is reached in finite time with probability 1. Therefore, the value for mean-payoff parity objectives in MDPs can be obtained by computing values for end-components and then playing a strategy to maximize the expectation to reach the values of the end-components.

We now present the key lemma where we show that for an MDP that is an end-component such that the minimum priority is even, the mean-payoff parity objective $\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{\geq \nu}$ is satisfied with probability 1 if the expected mean-payoff value is at least ν at some state (the result also holds for strict inequality). In other words, from the expected mean-payoff value of at least ν we ensure that both the mean-payoff and parity objective is satisfied with probability 1 from all states. The proof of the lemma considers two pure memoryless strategies: one for stochastic shortest path and the other for optimal expected mean-payoff value, and combines them to obtain an almost-sure winning strategy for the mean-payoff parity objective (details in [8]).

Lemma 5. *Consider an MDP M with state space Q, a priority function p, and weight function w such that (a) M is an end-component (i.e., Q is an end-component) and (b) the smallest priority in Q is even. If there is a state $q \in Q$ such that $\mathsf{ValMP}(w) \geq \nu$ (resp. $\mathsf{ValMP}(w) > \nu$), then there exists a strategy σ^* such that for all states $q \in Q$ we have $\mathbb{P}_q^{\sigma^*}(\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{\geq \nu}) = 1$ (resp. $\mathbb{P}_q^{\sigma^*}(\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{> \nu}) = 1$).*

Memory required by strategies. Lemma 5 shows that if the smallest priority in an end-component is even, then considering the sub-game restricted to the end-component, the mean-payoff parity objective is satisfied if and only if the mean-payoff objective is satisfied. The strategy constructed in Lemma 5 requires infinite memory, and in the case of loose inequality (i.e., $\mathsf{MeanPayoff}^{\geq \nu}$) infinite memory is required in general (see [11] for an example on graphs), and if the inequality is strict (i.e., $\mathsf{MeanPayoff}^{> \nu}$), then finite memory strategies exist [17]. For the purpose of computation we show that both strict and non-strict inequality can be solved in polynomial time. Since Lemma 5 holds for both strict and non-strict inequality, in sequel of this section we consider non-strict inequality and all the results hold for strict inequality as well.

Winning end-component. Given an MDP M with a parity objective $\mathsf{Parity}(p)$ and a mean-payoff objective $\mathsf{MeanPayoff}^{\geq \nu}$ for a weight function w, we call an end-component U *winning* if (a) $\min(p(U))$ is even; and (b) there exists a state with expected mean-payoff value at least ν in the sub-MDP induced by U, i.e., $\max_{q \in U} \mathsf{ValMP}(w)(q) \geq \nu$ in the sub-MDP induced by U. We denote by $\mathcal{W}$ the set of winning end-components, and let $\mathsf{Win} = \bigcup_{U \in \mathcal{W}} U$ be the union of the winning end-components.

Reduction to reachability of winning end-component. By Lemma 5 it follows that in every winning end-component the mean-payoff parity objective is satisfied with probability 1. Conversely, consider an end-component U that is not winning, then either the smallest priority is odd, or the maximal expected mean-payoff value that can be ensured for any state in U by staying in U is less than ν. Hence if only states in U are visited infinitely often, then with probability 1 (i) either the parity objective is not satisfied, or (ii) the mean-payoff objective is not satisfied. In other words, if an end-component that is not winning is visited infinitely often, then the mean-payoff parity objective is satisfied with probability 0. It follows that the value function for MDPs with mean-payoff parity objective can be computed by computing the value function for reachability to the set Win, i.e., formally, $\sup_{\sigma\in\Sigma} \mathbb{P}_q^\sigma(\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{\geq\nu}) = \sup_{\sigma\in\Sigma} \mathbb{P}_q^\sigma(\mathsf{Reach}(\mathsf{Win}))$, where $\mathsf{Reach}(\mathsf{Win})$ is the set of paths that reaches a state in Win at least once. Since the value function in MDPs with reachability objectives can be computed in polynomial time using linear programming [15], it suffices to present a polynomial-time algorithm to compute Win in order to obtain a polynomial-time algorithm for MDPs with mean-payoff parity objectives.

Computing winning end-components. The computation of the winning end-components is done iteratively by computing winning end-components with smallest priority 0, then winning end-components with smallest priority 2, and so on. The computation of Win is as follows:

- For $i \geq 0$, let $\mathcal{W}_{2i}$ be the set of maximal end-components U with states with priority at least $2i$ and that contain at least one state with priority $2i$, i.e., U contains only states with priority at least $2i$, and contains at least one state with priority $2i$. Let $\mathcal{W}'_{2i} \subseteq \mathcal{W}_{2i}$ be the set of maximal end-components $U \in \mathcal{W}_{2i}$ such that there is a state $q \in U$ such that the expected mean-payoff value in the sub-MDP restricted to U is at least ν. Let $\mathsf{Win}_{2i} = \bigcup_{U\in\mathcal{W}'_{2i}} U$.

The set $\mathsf{Win} = \bigcup_{i=0}^{\lfloor d/2\rfloor} \mathsf{Win}_{2i}$ is the union of the states of the winning end-components (formal pseudo-code in [8]).

Complexity of computing winning end-components. The winning end-component algorithm runs for $O(d)$ iterations and in each iteration requires to compute a maximal end-component decomposition and compute mean-payoff values of at most n end-components, where n is the number of states of the MDP. The maximal end-component decomposition can be achieved in polynomial time [12,13,9]. The mean-payoff value function of an MDP can also be computed in polynomial time using linear programming [15]. It follows that the value function of an MDP with mean-payoff parity objectives can be computed in polynomial time. The almost-sure winning set is obtained by computing almost-sure reachability to Win in polynomial time [12,13,9]. This polynomial-time complexity provides a tight upper bound for the problem.

Theorem 2. *The following assertions hold:*

1. *The set of almost-sure winning states for mean-payoff parity objectives can be computed in polynomial time for MDPs.*

2. *For mean-payoff parity objectives, almost-sure winning strategies require infinite memory in general for non-strict inequality (i.e, for mean-payoff parity objectives* $\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{\geq \nu}$*) and finite-memory almost-sure winning strategies exist for strict inequality (i.e., for* $\mathsf{Parity}(p) \cap \mathsf{MeanPayoff}^{> \nu}$*).*

References

1. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
2. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)
3. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite runs in weighted timed automata with energy constraints. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008)
4. Brázdil, T., Brozek, V., Etessami, K., Kucera, A., Wojtczak, D.: One-counter Markov decision processes. In: Proc. of SODA, pp. 863–874. SIAM, Philadelphia (2010)
5. Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., Raskin, J.-F.: Faster algorithms for mean-payoff games. Formal Methods in System Design (2010)
6. 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)
7. 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)
8. Chatterjee, K., Doyen, L.: Energy and mean-payoff parity Markov decision processes. Technical report, IST Austria (February 2011), http://pub.ist.ac.at/Pubs/TechRpts/2011/IST-2011-0001.pdf
9. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: Proc. of SODA. ACM SIAM (2011)
10. 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)
11. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Mean-payoff parity games. In: Proc. of LICS, pp. 178–187. IEEE Computer Society, Los Alamitos (2005)
12. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM 42(4), 857–907 (1995)
13. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
14. Emerson, E.A., Jutla, C.: Tree automata, mu-calculus and determinacy. In: Proc. of FOCS, pp. 368–377. IEEE, Los Alamitos (1991)
15. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
16. Gimbert, H., Horn, F.: Solving simple stochastic tail games. In: Proc. of SODA, pp. 847–862 (2010)
17. Gimbert, H., Oualhadj, Y., Paul, S.: Computing optimal strategies for Markov decision processes with parity and positive-average conditions. Technical report, LaBRI, Université de Bordeaux II (2011)

18. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Inf. Process. Lett. 68(3), 119–124 (1998)
19. Kucera, A., Stražovský, O.: On the controller synthesis for finite-state markov decision processes. In: Proc. of FSTTCS, pp. 541–552 (2005)
20. Pacuk, A.: Hybrid of mean payoff and total payoff. In: Talk at the Workshop Games for Design and Verification, St Anne's College Oxford (September 2010)
21. W. Thomas. Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, ch.7, pp. 389–455. Springer, Heidelberg (1997)
22. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state systems. In: FOCS 1985. IEEE Computer Society Press, Los Alamitos (1985)
23. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. 158(1&2), 343–359 (1996)

The Role of Polymorphism in the Characterisation of Complexity by Soft Types*

Jacek Chrząszcz and Aleksy Schubert

The University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
{chrzaszcz,alx}@mimuw.edu.pl

Abstract. Soft type assignment systems STA, STA_+, and STA_B characterise by means of reduction of terms the computation in complexity classes PTIME, NP, and PSPACE, respectively. All these systems are inspired by linear logic and include polymorphism similar to the one of System F. We show that the presence of polymorphism gives undecidability of typechecking and type inference. We also show that reductions in decidable monomorphic versions of these systems also capture the same complexity classes in a way sufficient for the traditional complexity theory. The reductions we propose show in addition that the monomorphic systems to serve as a programming language require some metalanguage support since the program which operates on data has form and type which depend on the size of the input.

1 Introduction

One of the goals of Implicit Computational Complexity studies is the search of a programming language which captures a particular complexity class, e.g. PTIME. The research may lead not only to a different theoretical view on the class, but also to a programming language with strong guarantees on time or memory consumption.

The relation of the computational complexity to type systems was studied already in the Statman's characterisation of nonelementary computation by reductions in the simply typed lambda calculus [16] which was later simplified by [13] and, in the context of the calculus with additional δ reduction rules, refined by Goerdt [7]. This was also studied in the pure simply typed calculus for types with low functional orders [15,18].

The current paper has its roots in the study of complexity classes by means of linear logic started by Girard in Light Linear Logic (LLL) and Elementary Linear Logic (ELL) [6] where polynomial and elementary time complexities respectively are considered and related to the computation via cut-elimination in proof nets. Polynomial time is also characterised by Soft Linear Logic (SLL) of Lafont [10]. The line started by Lafont brought the starting point of the current paper, i.e. systems STA, STA_+, and STA_B of Gaboardi et al [2,3,4] which, when appropriately tailored, capture the complexity classes of PTIME, NP, and PSPACE, respectively. As we explain it below, all the systems capture the classes in a specific way which is not compatible with the traditional reduction notion of the complexity theory. The papers provide for each Turing Machine in PTIME (NP, PSPACE respectively) a term which when applied to an encoding of an input gives the answer which encodes an answer of the original machine.

* This work was partly supported by the Polish government grant no N N206 355836.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 219–230, 2011.

We focus in the current paper on the question of what can be said on the systems STA, STA_+, STA_B when we apply the notion of problem reduction as used in the traditional complexity theory. We define a particular problem of computation by terms and show that the PTIME- and NP-completeness can be obtained even in monomorphic systems STA-mono, STA_+-mono, respectively, which satisfy additional restrictions on the functional order of redexes. In case of PSPACE, the additional constraint can be lifted and we can obtain PSPACE-completeness in case of the monomorphic system STA_B-mono with the same restriction as in the case of the polymorphic one. This is due to the presence of algebraic type of booleans. This observation leads us to a conclusion that corresponding addition to STA-mono, STA_+-mono leads to systems where the additional constraint on the order of redexes can be lifted. It is known since the papers [11,12] that well chosen algebraic constructs make possible a characterisation of the polynomial complexity classes even in the standard simply typed lambda calculus. In our paper we obtain a significantly more restricted set of constants that can give a sensible characterisation of the complexity classes. Still, there is a cost of this austere choice of means, namely, our characterisation does not give a single term to simulate a Turing Machine. This elucidates from a different perspective the point already mentioned in [14] that the polymorphism is needed in the systems that characterise complexity classes to combine structurally similar definitions into a single term.

A big drawback of the polymorphic systems is that the typechecking and type inference problems are, as shown here, undecidable. This can be mitigated by a system with a limited polymorphism. In particular the monomorphic version of STA, where there is no polymorphism, enjoys decidable typechecking and type inference problems [5]. It is now interesting to look for systems with amount of polymorphism sufficient to obtain decidable type inference, but strong enough to combine necessary definitions into programs which work for all inputs.

The paper is organised as follows. Section 2 presents the soft type assignment systems. In Section 3 we show that the polymorphic systems have undecidable typechecking and type inference problems. The characterisation of complexity classes by the monomorphic systems is given in Section 4.

2 Presentation of Systems

The main focus of the paper is on a presentation of the distinction between the polymorphic systems STA, STA_+, STA_B and their monomorphic counterparts STA-mono, STA_+-mono, STA_B-mono. Therefore we need to present all the systems. The types of the systems are built using the same grammar:

Polymorphic types	Monomorphic types	
$A ::= \alpha \mid \sigma \multimap A \mid \forall\alpha.A$	$A ::= \alpha \mid \sigma \multimap A$	(Linear Types)
$\sigma ::= A \mid !\sigma$	$\sigma ::= A \mid !\sigma$	(Bang Types)

The systems STA_B and STA_B-mono, in addition, use a fixed type variable $\mathbb{B}$ which is a domain for boolean values $\mathbf{0}$ and $\mathbf{1}$. The quantifier $\forall$ is a binding operator and the variable α in $\forall\alpha.A$ is bound. We identify types that differ only in the names of bound type variables. The set of free type variables in A is written $\mathrm{FTV}(A)$. We can substitute a free type variable α in a type A by a type B so that no free variable is captured

during the operation. The operation is written $A[B/\alpha]$. We write $\forall.A$ to express a type where all free type variables in A are bound by a series of quantifiers, e.g. $\forall.\alpha \multimap \beta = \forall\alpha\beta.\alpha \multimap \beta$. We also use the shorthand $A \stackrel{i}{\multimap} B$ to mean the type $A \multimap \cdots \multimap A \multimap B$ with i occurrences of A, so that $A \stackrel{1}{\multimap} B$ is $A \multimap B$ and $A \stackrel{0}{\multimap} B$ is B.

The terms of the systems are defined using the following grammars:

$$\begin{array}{ll} M ::= x \mid \lambda x.M \mid M_1 M_2 & \text{terms of STA,} \\ M ::= x \mid \lambda x.M \mid M_1 M_2 \mid M_1 + M_2 & \text{terms of STA}_+, \\ M ::= x \mid \lambda x.M \mid M_1 M_2 \mid \mathbf{0} \mid \mathbf{1} \mid \texttt{if } M_1 \texttt{ then } M_2 \texttt{ else } M_3 & \text{terms of STA}_B. \end{array}$$

As usual, the λ operator is a binding operator and the variable x is bound in the term $\lambda x.M$. We consider terms up to α-conversion i.e. two terms which differ only in names of the bound variables are considered to be the same. The set of free variables in a term M is written $\mathrm{FV}(M)$. A capture avoiding substitution of a variable x by a term N in a term M is denoted $M[N/x]$.

The systems assign types to terms. They use contexts which are sets of pairs $x : A$ to assign types to free variables. We sometimes use $\bigotimes_{j=k}^{l} x_j : \tau_j$ for $k \leq l$ to denote the context $x_k : \tau_k, \ldots, x_l : \tau_l$. We write $\mathrm{dom}(\Gamma)$ to denote the set of variables for which the context Γ assigns the types, e.g. $\mathrm{dom}(x : A, y : B) = \{x, y\}$. The type assignment rules for all the systems are presented in Figure 1. The figure is divided into sections containing different groups of rules marked with labels (F) for fundamental rules, (P) for polymorphic rules, (N) for nondeterministic rules, and (B) for booleans rules with additive `if`. The groups are combined together to create the following 6 systems:

$$\begin{array}{rr} \text{STA} : (F) + (P) & \text{STA-mono} : (F) \\ \text{STA}_+ : (F) + (P) + (N) & \text{STA}_+\text{-mono} : (F) + (N) \\ \text{STA}_B : (F) + (P) + (B) & \text{STA}_B\text{-mono} : (F) + (B) \end{array}$$

We sometimes write $\Gamma \vdash_Z M : A$, to denote a derivation in a suitable system Z. One of the crucial characteristics of derivations is degree. A degree of a derivation is the maximal number of the rules (sp) one can meet when traversing the derivation from the root to the leaves. Inductively $\mathsf{d}(\mathcal{D})$ is for rules other than (sp) the maximum of $\mathsf{d}(\mathcal{D}')$ over all $\mathcal{D}'$ that derive the premises of the final judgement of $\mathcal{D}$ and $\mathsf{d}(\mathcal{D}') + 1$ in case the final rule is (sp) and $\mathcal{D}'$ derives its premise.

These systems are also equipped with their computational semantics of β-reduction. The reductions are defined as the syntactical closures of the base rules, which in all systems include $(\lambda x.M)N \to_\beta M[N/x]$ and moreover contain:

$$\begin{array}{ll} \text{for STA}_+, \text{STA}_+\text{-mono:} & \text{for STA}_B, \text{STA}_B\text{-mono:} \\ M_1 + M_2 \to_\beta M_1 & \texttt{if } \mathbf{0} \texttt{ then } M_1 \texttt{ else } M_2 \to_\beta M_1 \\ M_1 + M_2 \to_\beta M_2 & \texttt{if } \mathbf{1} \texttt{ then } M_1 \texttt{ else } M_2 \to_\beta M_2 \end{array} \quad (1)$$

2.1 Presentation of the Problem

The main theorems in papers [2,3,4] consider the problem of simulation of the respective complexity classes in the type systems in such a way that a computation of a Turing Machine is simulated by a reduction of terms. From the point of view of the traditional complexity theory the papers consider the following problem:

The fundamental rules (F)

$$\frac{}{x : A \vdash x : A}\ (Ax) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\ (\multimap I) \qquad \frac{\Gamma \vdash M : \sigma \multimap A \qquad \Delta \vdash N : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash MN : A}\ (\multimap E)$$

$$\frac{\Gamma, x_1 : \sigma, \ldots, x_n : \sigma \vdash M : \tau}{\Gamma, x : !\sigma \vdash M[x/x_1, \ldots, x/x_n] : \tau}\ (m) \qquad \frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M : !\sigma}\ (sp)$$

The polymorphic rules (P)

$$\frac{\Gamma \vdash M : A \quad \alpha \notin \mathrm{FTV}(\Gamma)}{\Gamma \vdash M : \forall \alpha.A}\ (\forall I) \qquad \frac{\Gamma \vdash M : \forall \alpha.A}{\Gamma \vdash M : A[B/\alpha]}\ (\forall E)$$

The nondeterministic rule (N)

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash M + N : A}\ (sum)$$

The additive boolean rules (B)

$$\frac{}{\vdash \mathbf{0} : \mathbb{B}}\ (\mathbb{B}_0 I) \qquad \frac{}{\vdash \mathbf{1} : \mathbb{B}}\ (\mathbb{B}_1 I) \qquad \frac{\Gamma \vdash M : \mathbb{B} \quad \Gamma \vdash N_0 : A \quad \Gamma \vdash N_1 : A}{\Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ N_0\ \mathtt{else}\ N_1 : A}\ (\mathbb{B}E)$$

Fig. 1. The systems STA (F+P), STA$_+$ (F+P+N), STA$_B$ (F+P+B), and STA-mono (F), STA$_+$-mono (F+N), STA$_B$-mono (F+B)

Definition 1 (reduction problem — naïve version). *The* naïve reduction problem *for the system STA (STA$_+$, STA$_B$) is, given a pair of terms M_1, M_2 typable in the system STA (STA$_+$, STA$_B$ respectively), can M_1 be β-reduced to M_2?*

There is one subtlety about the problem. Namely, the problem is a correct decision problem only when there is a witness of the adjective *typable*. In many cases, the typability is a non-trivial property and may be undecidable. Therefore, the input data to the problem must also include a witness of the typability of the subject terms. We assume that it is a derivation of a type in the system for which the problem is considered. Even if we add the derivations $\mathcal{D}_1, \mathcal{D}_2$ we still have one more issue. The problem itself has the time complexity $|M_1|^{O(\mathsf{d}(\mathcal{D}_1))}$ in case of STA, STA$_+$ (see [4, Theorem 15], [3, Lemma 5.7]) and space complexity in the same range (see [2, Theorem 2]). Therefore, the measures are exponential provided that the grade is not fixed. In this view the papers rather prove the completeness for the following problem.

Definition 2 (reduction problem with degree). *The* reduction problem for the degree *k in the system STA (STA$_+$, STA$_B$) is, given a pair of terms M_1, M_2 with their derivations $\mathcal{D}_1, \mathcal{D}_2$ respectively where $\mathsf{d}(\mathcal{D}_1), \mathsf{d}(\mathcal{D}_2) \leq k$ in the system STA (STA$_+$, STA$_B$ respectively), can M_1 be β-reduced to M_2?*

The following result can be inferred from the papers on soft type assignment systems.

Theorem 1. *The reduction problem for the degree k, where k is sufficiently large, in STA (STA$_+$, STA$_B$) is complete for PTIME (NP, PSPACE, respectively).*

Proof. The problems are, as discussed, in the classes. They are complete for the classes since one can evaluate the polynomials which bound the length of the tape and the length of the computation as part of the reduction procedure. The other terms in the proofs stay in a fixed degree. □

3 Undecidability of Type Related Problems with Polymorphism

One of major requirements for a well designed functional programming language is the ability to automatically check and infer types of the programs. This led to consideration of the *typechecking* and *type inference* problems which are defined as follows.

Definition 3 (typechecking (TCP) and type inference problems (TIP)). *The* typechecking problem *is given a term M, a context Γ, and a type A, is there a derivation of the judgement $\Gamma \vdash M : A$?*

The type inference problem *is given a term M and a context Γ is there a type A and a derivation of the judgement $\Gamma \vdash M : A$?*

The first problem is used when the whole typing information is supplied by a programmer while the second one when the final type should be computed by the programmer's environment. In this paper we present a proof that the problems of typechecking and type inference are undecidable for all the type assignment systems STA, STA$_+$, and STA$_B$. For this we adapt the argument presented by Wells [17]. The proof is based on a reduction of the undecidable semiunification problem (SUP) [9]. The problem manipulates expressions built of variables and a binary symbol. For the purposes of our paper we assume that the variables are type variables and the binary symbol is $\multimap$.

Definition 4 (semiunification problem (SUP)). *The* semiunification problem *is, given a pair of inequalities $A_1 \leq B_1, A_2 \leq B_2$, decide if there are substitutions S, S_1, S_2 such that the equalities $S_1(S(A_1)) = S(B_1)$, $S_2(S(A_2)) = S(B_2)$ hold.*

The reduction of the semiunification problem can be made in a comprehensive way only when we consider derivations of a special form. Here is a definition of this form.

Definition 5 (regular form). *We say that a derivation for a judgement $\Gamma \vdash M : A$ is in* regular form *when it obeys all the conditions*

1. *The rule (w) which introduces $x : A$ to the context is used only right before $(\multimap I)$ which abstracts the variable x. We say that such a derivation is* (w)-sparing.
2. *The rule (m) which introduces $x : A$ to the context is used only right before $(\multimap I)$ which abstracts the variable x or right before a series of (m) rules where one of them substitutes x by certain y. We say that such a derivation is* (m)-sparing.
3. *For each use of the $(\forall E)$ rule the premise of the rule is not obtained using the $(\forall I)$ rule. We say that such a derivation obeys* $(\forall E)$ before $(\forall I)$ property.

The $(\forall E)$ before $(\forall I)$ property is called *INST before GEN property* in [17].

Lemma 1 (regular form). *Let X be any of STA, STA$_+$, STA$_B$. For each derivation $\mathcal{D}$ of a judgement $\Gamma \vdash_X M : A$ there is a derivation $\mathcal{D}'$ of the same judgement which is in regular form.*

Proof. The proof is done by showing that a derivation can be stepwise modified to obey the first condition, then the second one without loosing the first one and at last the third one without loosing all the previous ones. In each case the proof is by induction over the initial derivation. □

Now, we can obtain our undecidability result.

Theorem 2 (undecidability of TCP for polymorphic types). *The TCP for any of the systems STA, STA$_+$, STA$_B$ is undecidable.*

Proof. We reduce the SUP to the TCP. Consider an instance $\mathcal{E} = \{A_1 \leq B_1, A_2 \leq B_2\}$ of the SUP. We encode it in the following instance of the TCP

$$\begin{array}{l} b : \forall\alpha.(!\alpha \multimap \alpha) \multimap \beta, \\ c : \forall.(B_1 \multimap \gamma_1) \multimap (\gamma_2 \multimap B_2) \multimap (A_1 \multimap A_2) \vdash b(\lambda x.cxx) : \beta. \end{array} \tag{2}$$

We show now that this judgement is derivable in any of the systems STA, STA$_+$, STA$_B$ if and only if the instance $\mathcal{E}$ is solvable. We denote by A_c and A_b the types of b and c in the context of the judgement. We can present the judgement as $\Gamma_{SUP} \vdash M_{SUP} : \beta$.

($\Leftarrow$) Suppose first that $\mathcal{E}$ is solvable. This means there are substitutions S, S_1, S_2 such that the equalities $S_1(S(A_1)) = S(B_1)$, $S_2(S(A_2)) = S(B_2)$ hold (*). It is straightforward to derive judgements

$x_1 : \forall.S(A_1) \multimap S(A_2) \vdash x_1 : S_1(S(A_1)) \multimap S_1(S(A_2))$
$x_2 : \forall.S(A_1) \multimap S(A_2) \vdash x_2 : S_2(S(A_1)) \multimap S_2(S(B_2))$
$c : \forall.(B_1 \multimap \gamma_1) \multimap (\gamma_2 \multimap B_2) \multimap (A_1 \multimap A_2) \vdash$
$\quad c : (S(B_1) \multimap S_1(S(A_2))) \multimap (S_2(S(A_1)) \multimap S(B_2)) \multimap (S(A_1) \multimap S(A_2))$
$b : \forall\alpha.(!\alpha \multimap \alpha) \multimap \beta \vdash b : (!(\forall.S(A_1) \multimap S(A_2)) \multimap \forall.S(A_1) \multimap S(A_2)) \multimap \beta$

Note that we instantiated in the derivation for c the variable γ_1 with $S_1(S(A_2))$ and the variable γ_2 with $S_2(S(A_1))$ as well as all other variables in the type of c according to the substitution S. We can now use (*) to match the types of x_1 and x_2 with the subtypes of the type obtained for c. We can use the (m) rule, then abstract with $(\multimap I)$ and apply b to obtain (2).

($\Rightarrow$) The use of c defines the substitution S. The type of b forces the type of cxx to be the same as the type of x, namely $\forall\delta_1 \ldots \delta_k.S(A_1) \multimap S(A_2)$ for some $\delta_1 \ldots \delta_k$. Now, the two uses of x define the substitutions S_1, S_2 with domains $\delta_1 \ldots \delta_k$ which must make $S_1(S(A_1))$ equal to $S(B_1)$ in the first use of x and $S_2(S(A_2))$ equal to $S(B_2)$ in the second use of x. Due to Lemma 1 we can restrict the derivations so that the substitutions are gathered directly from consecutive $(\forall E)$ rules rhather than by a complicated procedure which combines introductions and eliminations of $\forall$. Note that the expressions A_1, A_2, B_1, B_2 do not contain $\forall$ and ! so we may assume S, S_1, S_2 do not use these connectives either. □

Observe that $b(\lambda x.cxx)$ can be typed only to β under the context presented above. This immediately makes the construction above prove also the following theorem.

Theorem 3 (undecidability of TIP for polymorphic types). *The TIP for any of the systems STA, STA$_+$, STA$_B$ is undecidable.*

4 Characterisation of Complexity in Monomorphic Systems

Unlike their polymorphic counterparts, the monomorphic systems have decidable type-checking and type inference [5]. Yet they are expressive enough to characterise the polynomial complexity classes. This characterisation, however, is based on a different restriction than the one on the degree of the derivation. We define the order of a type as $\mathrm{ord}(\alpha) = 1$ for a base type α, $\mathrm{ord}(!\sigma) = \mathrm{ord}(\sigma), \mathrm{ord}(\sigma \multimap \tau) = \max(\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau))$. The order of a redex $(\lambda x.M)N$ in a derivation $\mathcal{D}$ when the type of $(\lambda x.M)$ in the derivation is τ equals to $\mathrm{ord}(\tau)$. The order of the redex $M_1 + M_2$ of the type τ is $\mathrm{ord}(\tau) + 1$. The order $\mathrm{ord}(M, \mathcal{D})$ of a term M with the derivation $\mathcal{D}$ is the maximal order of redexes in the derivation. We can now present a version of the reduction problem we deal with in the case of the systems STA, STA_+.

Definition 6 (reduction problem with order). *The* reduction problem for the order k *in the system STA is, given a pair of terms M_1, M_2 with their derivations $\mathcal{D}_1, \mathcal{D}_2$ respectively where $\mathrm{ord}(M_1, \mathcal{D}_1), \mathrm{ord}(M_2, \mathcal{D}_2) \leq k$ in the system STA, can M_1 be β-reduced to M_2?*

The reduction problem for the existential order k *in the system STA_+ is as above but $M_1 = C[M_1']$ where $C[x]$ is a linear context of any order, i.e. $C[x]$ is typable by a derivation without (m) and (sp) rules for some Γ and M_1' is a term of STA of order k.*

The paper [15] proves the following result concerning the problem of reduction for the order 3 in the simply typed lambda calculus $\lambda_{\to}$.

Theorem 4 (the reduction problem of order 3 in $\lambda_{\to}$). *The problem of reduction for terms of order 3 in the simply typed lambda calculus is in PTIME.*

We can make use of the theorem since the typability in STA-mono implies the typability in $\lambda_{\to}$. We define the !-erasing as $\|\alpha\| = \alpha$ where α is atomic, $\|\sigma \multimap A\| = \|\sigma\| \to \|A\|$, and $\|!\sigma\| = \|\sigma\|$. As usual, we extend $\|\cdot\|$ to contexts in such a way that $\|\Gamma\| = \{x : \|\sigma\| \mid x : \sigma \in \Gamma\}$.

Proposition 5 (typability in STA and $\lambda_{\to}$) *If $\Gamma \vdash_{\text{STA-mono}} M : \sigma$ then $\|\Gamma\| \vdash_{\lambda_{\to}} M : \|\sigma\|$. Moreover the order of the term M in the derivation in $\lambda_{\to}$ is the same as the order of the term M in the derivation in STA.*

The reductions for STA, STA_+, and STA_B involve encoding of some logical problems. The first two are done in calculi with no boolean values, so we define them here.

Definition 7. *Let $Bool_i$ be a type $!^i\alpha \multimap !^i\alpha \multimap \alpha$, where α is a fixed type variable. One can define boolean constants and connectives with the following STA-mono types:*

$$tt = \lambda x.\lambda y.x : Bool_i \qquad ff = \lambda x.\lambda y.y : Bool_i$$

$$and = \lambda b_1.\lambda b_2.\lambda x.\lambda y.b_1\ (b_2\ x\ y)\ y : Bool_i \multimap\ !^i Bool_j \multimap Bool_{i+j+1}$$

$$or = \lambda b_1.\lambda b_2.\lambda x.\lambda y.b_1\ x\ (b_2\ x\ y) : Bool_i \multimap\ !^i Bool_j \multimap Bool_{i+j+1}$$

$$not = \lambda b.\lambda x.\lambda y.b\ y\ x : Bool_i \multimap Bool_i$$

We obviously have $\Gamma \vdash \lambda x.\lambda y.t : Bool_j$ whenever $\Gamma \vdash \lambda x.\lambda y.t : Bool_i$ and $j \geq i$.

Theorem 6 (PTIME Completeness of STA-mono). *The problem of reduction with order 3 in case of terms typable in STA-mono is complete for PTIME.*

Proof. The problem is in PTIME since the derivability in STA-mono implies the derivability in $\lambda_{\to}$ in the same functional order by Proposition 5. The reduction in the functional order 3 can be done in polynomial time by Theorem 4.

The problem is hard for PTIME since we can reduce in polynomial time the Circuit Value Problem to the problem of reduction of terms typable in STA-mono.

We use a variant of CVP, the Topologically Ordered CVP (TopCVP, Problem A.1.2 in [8]). The instance of the problem may be given as a sequence of assignments $x_1 := e_i, \ldots, x_n := e_n$ where each e_i is one of $0, 1, x_j, \neg x_j, x_j \vee x_k, x_j \wedge x_k$ where $j, k < i$ and $j \neq k$ (where applicable). The additional restriction $j \neq k$ in the last two clauses is not fundamental and is compensated by assignments of the form $x_i := x_j$. The result of the circuit is the value of the variable x_n.

Given terms or, and and not we encode the problem instance as follows:

$$(\lambda x_1.(\lambda x_2.\ldots(\lambda x_n.x_n)[\![e_n]\!]\ldots)[\![e_2]\!])[\![e_1]\!]$$

where $[\![e_i]\!]$ are the straightforward translations of e_i, defined as the right one of ff, tt, $\lambda x.\lambda y.x_j\ x\ y, not\ x_j, or\ x_j\ x_k, and\ x_j\ x_k$. It is easy to see that this term reduces to tt or ff if and only if the input circuit evaluates to true or false respectively.

We now give a step by step definition of the translation together with a sketch of the proof that the above term is typable with the type $Bool_{2^n-1}$.

Let $\hat{e}_i$ be the translation of the assignments $i+1, \ldots, n$, defined inductively from n down to 0 as $\hat{e}_n = x_n$, $\hat{e}_{i-1} = (\lambda x_i.\hat{e}_i)[\![e_i]\!]$. It is easy to see that $\hat{e}_0$ is the translation of the whole problem instance. Now, given $\Gamma_i^k = \bigotimes_{j=1}^{k} x_j : !^{2^{2n-i}-1} Bool_{2^j-1}$, for each $i = 1 \ldots n$ (for the left judgement below) and $i = 0 \ldots n$ (for the right one) we have:

$$\Gamma_i^{i-1} \vdash [\![e_i]\!] : Bool_{2^i-1}, \qquad \Gamma_i^i \vdash \hat{e}_i : Bool_{2^n-1}. \tag{3}$$

All the left judgments can be shown directly. Note that variables used in e_i have index smaller than i and $i \leq n$, so $2^{2n-i} - 1 \geq 2^i - 1$ and the number of ! is sufficient.

To derive the right judgments, we proceed by simple induction from n down to 0.

In the end we get $\vdash \hat{e}_0 : Bool_{2^n-1}$ as expected.

It is possible to repeat the above proof in such a way that all judgments are preceded with m input variables (numbered with non-positive numbers $-(m-1) \ldots 0$ for simplicity), arriving at the encoding of a circuit with input values:

$$\vdash \lambda x_{-(m-1)}.\ldots\lambda x_0.\hat{e}_0 : !^{2^{2n}-1} Bool_0 \xrightarrow{m}\!\circ\ Bool_{2^n-1}. \tag{4}$$

Our construction is therefore independent from the input data, but not from the size of input data. Note that in the above proof, the number of needed ! is estimated very roughly. Nevertheless, if precise account of ! is taken, their number in certain cases remains exponential.

Even though the number of ! is high, the term can be reduced to a normal form in linear number of β reductions for any boolean input. Indeed, each $[\![e_i]\!]$ can be normalized in constant time to either tt or ff and there are n redexes which translate the assignments of the initial circuit. □

We must remark that since the monomorphic binary boolean connectives are not linear, the construction in the theorem above employs an exponential number of !. The bang

type constructors are, however, introduced in blocks, i.e. many ! in a single sequence of (sp) applications or (m) applications. Therefore, we can turn to a compressed representation of derivations where n uses of (sp) one after another is replaced with a single one which introduces $!^n$ and analogously for (m) which introduces ! to a single variable in the context. This kind of compressed representation of derivations makes possible to realise the constructions above in polynomial time.

Theorem 7 (NP Completeness of STA$_+$-mono). *The problem of reduction with existential order* 3 *in case of terms typable in STA$_+$-mono is complete for NP.*

Proof. The problem is in NP since we first evaluate the linear context with $+$ which takes the time proportional to the size of the context by [3, Theorem 5.12]. Then we obtain a third order term which can be evaluated in polynomial time by the construction in [15]. Note that since the context is linear it does not matter at which point $+$ redexes are executed.

The problem is hard for NP since we can reduce in polynomial time the CIRCUIT-SAT problem to the problem of reduction of terms typable in STA$_+$-mono. Indeed, it is enough to use the translation of circuit from Theorem 6 and nondeterministically choose the boolean value of some input variables. However, in order to do it in a consistent way (i.e. each occurrence of a variable gets the same chosen value), one has to use the following choice functions $\lambda f.ftt + fff$ of order 4 which take the λ-abstracted CVP translation and nondeterministically choose between its instantiation with tt and its instantiation with ff. □

Theorem 8 (PSPACE Completeness of STA$_B$-mono). *The problem of reduction for the degree* 1 *in case of the terms typable in STA$_B$-mono is complete for PSPACE.*

Proof. The problem is in PSPACE since STA$_B$-mono is a subsystem of STA$_B$ where the reduction problem above is in APTIME by [2, Theorem 2]. The problem is hard for PSPACE since we can reduce in polynomial time the QBF problem to the problem of reduction of terms typable in STA$_B$-mono: using built-in booleans, the boolean connectives can be easily encoded as linear terms:

$$\mathbf{and}(b_1, b_2) = \texttt{if } b_1 \texttt{ then } b_2 \texttt{ else } \mathbf{1} \qquad \mathbf{or}(b_1, b_2) = \texttt{if } b_1 \texttt{ then } \mathbf{0} \texttt{ else } b_2$$
$$\mathbf{not}(b) = \texttt{if } b \texttt{ then } \mathbf{1} \texttt{ else } \mathbf{0}$$

and to encode quantifiers we use terms

$$\mathbf{forall} = \lambda p.\texttt{if } p\ \mathbf{0} \texttt{ then } p\ \mathbf{1} \texttt{ else } \mathbf{1} \qquad \mathbf{exists} = \lambda p.\texttt{if } p\ \mathbf{0} \texttt{ then } \mathbf{0} \texttt{ else } p\ \mathbf{1}$$

applied to the λ abstracted formula. The resulting translated term can be typed without ! so it has degree 1. □

There is a contrast between the result for PSPACE and PTIME (NP) as the construction for PSPACE uses basically the same machinery, but the degree of the terms used in the proof for PSPACE is constant while in the other ones changes with the size of the input data. This is because the `if-then-else` construct is polymorphic. In fact we can add this construct to the languages STA-mono and STA$_+$-mono by similar means to the

STA$_B$-mono system: the type constant $\mathbb{B}$, constants $\mathbf{0}$ and $\mathbf{1}$ together with typing rules $(\mathbb{B}_{\mathbf{0}}I)$ and $(\mathbb{B}_{\mathbf{1}}I)$, the following multiplicative rule for `if-then-else`

$$\frac{\Gamma_1 \vdash M : \mathbb{B} \quad \Gamma_2 \vdash N_0 : A \quad \Gamma_3 \vdash N_1 : A \qquad \Gamma_1 \# \Gamma_2 \quad \Gamma_1 \# \Gamma_3 \quad \Gamma_2 \# \Gamma_3}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \texttt{if } M \texttt{ then } N_0 \texttt{ else } N_1 : A} \ (\mathbb{B}_* E) \tag{5}$$

and δ-reduction rules

$$\texttt{if } \mathbf{0} \texttt{ then } M_1 \texttt{ else } M_2 \to_\delta M_1 \qquad \texttt{if } \mathbf{1} \texttt{ then } M_1 \texttt{ else } M_2 \to_\delta M_2 \tag{6}$$

These systems use a significantly weaker set of additional algebraic types than the ones used in [11]. Namely, we only use one algebraic type with finite number of inhabitants. Such systems, called STA*-mono and STA$^*_+$-mono below, have decidable typechecking and type-inference problems [5] and we obtain the following theorem similar in flavour to the results obtained in [14].

Theorem 9 (Completeness of star systems). *The reduction problem for terms typable in* STA**-mono by a derivation of degree* 1 *is complete for PTIME. The reduction problem for terms typable in* STA$^*_+$*-mono by a derivation of degree* 1 *is complete for NP.*

Proof. The problems are in PTIME and NP respectively, because one can translate the terms to the polymorphic STA and STA$_+$ (resp.): the type $\mathbb{B}$ can be translated into the usual polymorphic encoding $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$ with constants $\lambda xy.x$ and $\lambda xy.y$ and $\texttt{if } M \texttt{ then } N_0 \texttt{ else } N_1$ translated to $M\ N_0\ N_1$. This translation does not affect the degree of derivations.

The problems are hard for PTIME and NP respectively, but the proofs of Theorems 6 and 7 have to be adapted to get rid of explicit duplication of boolean variables: using the polymorphic `if-then-else` one can implicitly duplicate boolean values using tuples. Let us define some abbreviations:

$$\mathbb{B}^n \equiv (\mathbb{B} \overset{n}{\multimap} \mathbb{B}) \multimap \mathbb{B}, \qquad t^n \equiv \lambda z.z \underbrace{t \ldots t}_{n \text{ times}}.$$

We clearly have $t^n : \mathbb{B}^n$ whenever $t = \mathbf{0}$ or $\mathbf{1}$.

Let $x_1 := e_i, \ldots, x_n := e_n$ be an instance of the CVP problem with output value x_n. Let k_i be the number of occurrences of the variable x_i in $e_{i+1}, \ldots, e_n$. We replace each occurrence of x_i in the translation of $e_{i+1}, \ldots, e_n$ with "linear copies" of x_i, named $x_i^1, \ldots, x_i^{k_i}$. We set $k_n = 1$. We translate expressions e_i in the following way:

$$\tilde{e_i} = \texttt{if } [\![e_i]\!] \texttt{ then } \mathbf{0}^{k_i} \texttt{ else } \mathbf{1}^{k_i}$$

where $[\![e_i]\!]$ is the right one of $\mathbf{0}, \mathbf{1}, x_j^l, \mathbf{not}(x_j^l), \mathbf{or}(x_j^l, x_k^m), \mathbf{and}(x_j^l, x_k^m)$, and x_j^l (and optionally x_k^m) denotes a suitable linear copy of x_j (and x_k respectively). The translation of the CVP instance is defined as before, by backward induction for $i = n$ down to 0, where $\hat{e}_0$ denotes the translation of the whole instance:

- $\hat{e}_n = x_n^1$,
- $\hat{e}_{i-1} = (\lambda \tilde{x}_i.\tilde{x}_i\ (\lambda x_i^1, \ldots, x_i^{k_i}.\hat{e}_i))\ \tilde{e}_i$.

Now, for $i = 1 \ldots n$, let $\Gamma_i = \emptyset$ or $x_j^l : \mathbb{B}$ or $x_j^l : \mathbb{B}, x_k^m : \mathbb{B}$ according to the form of e_i, we have

$$\Gamma_i \vdash [\![e_i]\!] : \mathbb{B} \qquad \text{and} \qquad \Gamma_i \vdash \tilde{e}_i : \mathbb{B}^{k_i}. \tag{7}$$

We also have

$$\Delta_i \vdash \hat{e}_i : \mathbb{B} \tag{8}$$

for $i = 0 \ldots n$, where Δ_i is defined by backward induction as follows

- $\Delta_n = x_n^1 : \mathbb{B}$,
- $\Delta_{i-1} = \Delta_i - \{x_i^1, \ldots, x_i^{k_i}\} \cup \Gamma_i$.

Note that $\Delta_i = \Gamma_{i+1} \cup \cdots \cup \Gamma_n \cup \{x_n^1 : \mathbb{B}\} - \{x_j^1, \ldots, x_j^{k_j} \mid j > i\}$, the sets Δ_i and Γ_i are disjoint, $\Delta_i \supseteq \{x_i^1, \ldots, x_i^{k_i}\}$ and $\Delta_0 = \emptyset$. The proofs of (7) are trivial. The proof of (8) can be done by a simple induction for $i = n$ down to 0.

From the above, we conclude that $\vdash \hat{e}_0 : \mathbb{B}$ and it is obvious that $\hat{e}_0$ reduces to $\mathbf{0}$ if and only if the value of the initial CVP instance is true. In the whole proof we did not use a single ! so the degree of the derivation is 0.

Using the tricks similar to proofs of theorems 6 and 7 one can separate data (initial values of several first variables) from the computation (the remaining assignments performing boolean operations). Since the term is linear, in order to simulate nondeterminism, one does not have to use higher order choice functions, but one can simply partially apply the function to a suitable number of nondeterministic terms $\mathbf{0}+\mathbf{1}$. □

5 Final Remarks

Conclusions. The study we presented here shows that a meaningful account of polynomial complexity classes PTIME, NP, and PSPACE, can be obtained already in the monomorphic versions of the systems STA, STA_+, and STA_B. The systems enjoy decidable type inference problem which makes them to be a good theoretical basis for very simple programming languages with respective complexity bounds. However, the restriction to monomorphic systems makes part of the program construction to happen on the metalanguage level, i.e. our simulations require different term for each length of a machine tape. This raises an interesting open question which polymorphic systems have decidable type inference and simulate PTIME, NP and PSPACE with no need of metalanguage artifacts. Note that the paper [1] suggests that the addition of the complexity annotations (! in our case) to terms typed in System F can be done automatically in polynomial time. It is interesting to see if the hypothetical decidable polymorphic systems which characterise the polynomial classes also have the polynomial overhead compared to the systems with no ! connective.

Acknowledgements. We would like to thank Paweł Urzyczyn for the inspiration to take up the problem combined with many fruitful discussions on the content of the paper and to anonymous referees who helped greatly in shaping up the paper.

References

1. Atassi, V., Baillot, P., Terui, K.: Verification of PTIME reducibility for System F terms: type inference in dual light affine logic. Logical Methods in Computer Science 3(4:10), 1–32 (2007)
2. Gaboardi, M., Marion, J.-Y., Rocca, S.R.D.: A logical account of PSPACE. In: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, pp. 121–131. ACM, New York (2008)

3. Gaboardi, M., Marion, J.-Y., Rocca, S.R.D.: Soft linear logic and polynomial complexity classes. ENTCS 205, 67–87 (2008)
4. Gaboardi, M., Ronchi Della Rocca, S.: A Soft Type Assignment System for λ-Calculus. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 253–267. Springer, Heidelberg (2007)
5. Gaboardi, M., Rocca, S.R.D.: Type inference for a polynomial lambda calculus. In: Berardi, S., Damiani, F., De'Liguoro, U. (eds.) Types for Proofs and Programs, pp. 136–152. Springer, Heidelberg (2009)
6. Girard, J.-Y.: Light linear logic. Information and Computation 143, 175–204 (1998)
7. Goerdt, A.: Characterizing complexity classes by general recursive definitions in higher types. In: Proceedings of the 2nd Workshop on Computer Science Logic, London, UK, pp. 99–117. Springer, Heidelberg (1989)
8. Greenlaw, R., Hoover, J.H., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press, USA (1995)
9. Kfoury, A.J., Tiuryn, J., Urzyczyn, P.: The undecidability of the semi-unification problem. Information and Computation 102(1), 83–101 (1993)
10. Lafont, Y.: Soft linear logic and polynomial time. Theoretical Computer Science 318, 163–180 (2004)
11. Leivant, D., Marion, J.-Y.: Lambda calculus characterizations of poly-time. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 274–288. Springer, Heidelberg (1993)
12. Leivant, D., Marion, J.-Y.: Ramified recurrence and computational complexity ii: Substitution and poly-space. In: Kleine Büning, H. (ed.) CSL 1995. LNCS, vol. 1092, pp. 486–500. Springer, Heidelberg (1996)
13. Mairson, H.: A simple proof of a theorem of Statman. Theoretical Computer Science (103), 213–226 (1992)
14. Mairson, H.G., Terui, K.: On the Computational Complexity of Cut-Elimination in Linear Logic. In: Blundo, C., Laneve, C. (eds.) ICTCS 2003. LNCS, vol. 2841, pp. 23–36. Springer, Heidelberg (2003)
15. Schubert, A.: The complexity of β-reduction in low orders. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 400–414. Springer, Heidelberg (2001)
16. Statman, R.: The typed λ-calculus is not elementary recursive. Theoretical Computer Science (9), 73–81 (1979)
17. Wells, J.B.: Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In: Proceedings of Ninth Annual IEEE Symposium on Logic in Computer Science, pp. 176–185 (July 1994)
18. Wierzbicki, T.: Complexity of the higher order matching. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 82–96. Springer, Heidelberg (1999)

An Algebraic Theory of Complexity for Valued Constraints: Establishing a Galois Connection*

David A. Cohen[1], Páidí Creed[1], Peter G. Jeavons[2], and Stanislav Živný[3]

[1] Department of Computer Science, Royal Holloway, University of London, UK
[2] Department of Computer Science, University of Oxford, UK
[3] University College, University of Oxford, UK
{d.cohen,paidi}@rhul.ac.uk, {peter.jeavons,standa.zivny}@cs.ox.ac.uk

Abstract. The complexity of any optimisation problem depends critically on the form of the objective function. Valued constraint satisfaction problems are discrete optimisation problems where the function to be minimised is given as a sum of cost functions defined on specified subsets of variables. These cost functions are chosen from some fixed set of available cost functions, known as a valued constraint language. We show in this paper that when the costs are non-negative rational numbers or infinite, then the complexity of a valued constraint problem is determined by certain algebraic properties of this valued constraint language, which we call *weighted polymorphisms*. We define a Galois connection between valued constraint languages and sets of weighted polymorphisms and show how the closed sets of this Galois connection can be characterised. These results provide a new approach in the search for tractable valued constraint languages.

1 Introduction

Classical constraint satisfaction is concerned with the feasibility of satisfying a collection of constraints. The extension of this framework to include optimisation is now also being investigated and a theory of so-called *soft constraints* is being developed. Several alternative mathematical frameworks for soft constraints have been proposed in the literature, including the very general frameworks of 'semi-ring based constraints' and 'valued constraints' [6]. For simplicity, we shall adopt the valued constraint framework here as it is sufficiently powerful to model a wide range of optimisation problems [17]. In this framework, every tuple of values allowed by a constraint has an associated *cost*, and the goal is to find an assignment with minimal total cost. The general constraint satisfaction problem (CSP) is NP-hard, and so is unlikely to have a polynomial-time algorithm. However, there has been much success in finding tractable fragments of the CSP by restricting the types of relation allowed in the constraints. A set of allowed relations has been called a *constraint language* [26]. For some constraint languages the associated constraint satisfaction problems with constraints chosen

* This research was supported by EPSRC grant EP/F01161X/1. Stanislav Živný is supported by a Junior Research Fellowship at University College, Oxford.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 231–242, 2011.

from that language are solvable in polynomial-time, whilst for other constraint languages this class of problems is NP-hard [27,26,23]; these are referred to as *tractable languages* and *NP-hard languages*, respectively. Dichotomy theorems, which classify each possible constraint language as either tractable or NP-hard, have been established for languages over 2-element domains [32], 3-element domains [10], for conservative languages [13,4], and maximal languages [11,9].

The general *valued* constraint satisfaction problem (VCSP) is also NP-hard, but again we can try to identify tractable fragments by restricting the types of allowed *cost functions* that can be used to define the valued constraints. A set of allowed cost functions has been called a *valued constraint language* [17]. Much less is known about the complexity of the optimisation problems associated with different valued constraint languages, although some results have been obtained for certain special cases. In particular, a complete characterisation of complexity has been obtained for valued constraint languages over a 2-element domain with real-valued or infinite costs [17]. This result generalises a number of earlier results for particular optimisation problems such as Max-Sat [20] and Min-Ones [21]. One class of tractable cost functions that has been extensively studied is the class of *submodular* cost functions [21,17,28,22,29,34].

In the classical CSP framework it has been shown that the complexity of any constraint language over any finite domain is determined by certain algebraic properties known as *polymorphisms* [27,26]. This result has reduced the problem of the identification of tractable constraint languages to that of the identification of suitable sets of polymorphisms. In other words, it has been shown to be enough to study just those constraint languages which are characterised by having a given set of polymorphisms. Using the algebraic approach, considerable progress has now been made towards a complete characterisation of the complexity of constraint languages over finite domains of arbitrary size [23,12,3,1,2,5].

In the VCSP framework it has been shown that a more general algebraic property known as a *multimorphism* can be used to analyse the complexity of certain valued constraint languages [14,17]. Multimorphisms have been used to show that there are precisely eight maximal tractable valued constraint languages over a 2-element domain with real-valued or infinite costs, and each of these is characterised by having a particular form of multimorphism [17]. Furthermore, it was shown that many known maximal tractable valued constraint languages over larger finite domains are precisely characterised by a single multimorphism and that key NP-hard examples have (essentially) no multimorphisms [17,16].

Cohen et al. [15] later generalised the notion of a multimorphism slightly, to that of a *fractional polymorphism*. They showed that fractional polymorphisms, together with the polymorphisms of the underlying feasibility relations, characterise the complexity of any valued constraint language with non-negative rational or infinite costs over any finite domain [15].

Contributions. In this paper, we extend the results of [15] by introducing a new algebraic construct which we call a *weighted polymorphism*. We are able to show, using the ideas of [15], that the weighted polymorphisms of a valued constraint language are sufficient on their own to determine the complexity of that

language. In addition, we are now able to define a Galois connection between valued constraint languages and sets of weighted polymorphisms, and characterise the closed sets on both sides.

The Galois connection we establish here can be applied to the search for tractable valued constraint languages in a very similar way to the application of polymorphisms to the search for tractable constraint languages in the classical CSP. First, we need only consider valued constraint languages characterised by weighted polymorphisms. This greatly simplifies the search for a characterisation of all tractable valued constraint languages. Second, any tractable valued constraint language with finite rational or infinite costs must have a non-trivial weighted polymorphism. Hence the results of this paper provide a powerful new set of tools in the search for a polynomial-time/NP-hard dichotomy for finite-domain optimisation problems defined by valued constraints. In the conclusion section we will mention recent results obtained using the Galois connection established in this paper.

Despite the fact that the proof of the main result uses similar techniques to [15], namely linear programming and Farkas Lemma, the main contribution of this paper is significantly different: [15] has shown that fractional polymorphisms capture the complexity of valued constraint languages. Here, we prove the same for weighted polymorphisms, but also establish a 1-to-1 correspondence between valued constraint languages and particular sets of weighted polymorphisms, which we call weighted clones. This is crucial for using weighted polymorphisms in searching for new tractable valued constraint languages. Our results show that a linear program can be set up not only to answer the question of whether a given cost function is expressible over a valued constraint language, but also for the question of whether a given weighted operation belongs to a weighted clone. (We do not elaborate on this application in much detail, but it follows straightforwardly from the proofs of the main results.)

The structure of the paper is as follows. In Section 2 we describe the Valued Constraint Satisfaction Problem and define the notion of expressibility. In Sections 3 and 4 we introduce weighted relational clones (valued constraint languages closed under a certain notion of expressibility) and weighted clones respectively, and state the main result: weighted relational clones are in 1-to-1 correspondence with weighted clones. In Section 5 we give a proof of the main new theorem establishing the Galois connection. Finally, in Section 6, we mention some recent results based on the results of this paper.

2 Valued Constraint Satisfaction Problems

We shall denote by $\mathbb{Q}_+$ the set of all non-negative rational numbers.[1] We define $\overline{\mathbb{Q}}_+ = \mathbb{Q}_+ \cup \{\infty\}$ with the standard addition operation extended so that for all $a \in \mathbb{Q}_+$, $a + \infty = \infty$ and $a\infty = \infty$. Members of $\overline{\mathbb{Q}}_+$ are called *costs*.

[1] To avoid computational problems, we work with rational numbers rather than real numbers. We could work with the algebraic reals, but the rationals are sufficiently general to encode many standard optimisation problems; see, for example [17].

A function ϕ from D^r to $\overline{\mathbb{Q}}_+$ will be called a *cost function* on D of *arity* r.

Definition 1. *An instance of the* **valued constraint satisfaction problem***, (***VCSP***), is a triple* $\mathcal{P} = \langle V, D, C\rangle$ *where:* V *is a finite set of* **variables***;* D *is a set of possible* **values***;* C *is a multi-set of* **constraints***. Each element of* C *is a pair* $c = \langle \sigma, \phi\rangle$ *where* σ *is a tuple of variables called the* **scope** *of* c*, and* ϕ *is a* $|\sigma|$*-ary cost function on* D *taking values in* $\overline{\mathbb{Q}}_+$*. An* **assignment** *for* $\mathcal{P}$ *is a mapping* $s : V \to D$*. The* **cost** *of an assignment* s*, denoted* $Cost_{\mathcal{P}}(s)$*, is given by the sum of the costs for the restrictions of* s *onto each constraint scope, that is,*

$$Cost_{\mathcal{P}}(s) \stackrel{\text{def}}{=} \sum_{\langle\langle v_1, v_2, \ldots, v_m\rangle, \phi\rangle \in C} \phi(s(v_1), s(v_2), \ldots, s(v_m)).$$

A **solution** *to* $\mathcal{P}$ *is an assignment with minimal cost, and the question is to find a solution.*

A **valued constraint language** is any set Γ of cost functions from some fixed set D. We define VCSP(Γ) to be the set of all VCSP instances in which all cost functions belong to Γ. A valued constraint language Γ is called **tractable** if, for every finite subset $\Gamma_f \subseteq \Gamma$, there exists an algorithm solving any instance $\mathcal{P} \in \text{VCSP}(\Gamma_f)$ in polynomial time. Conversely, Γ is called **NP-hard** if there is some finite subset $\Gamma_f \subseteq \Gamma$ for which VCSP(Γ_f) is NP-hard.

We now define a closure operator on cost functions, which adds to a set of cost functions all other cost functions which can be *expressed* using that set, in the sense defined below.

Definition 2. *For any VCSP instance* $\mathcal{P} = \langle V, D, C\rangle$*, and any list* $L = \langle v_1, \ldots, v_r\rangle$ *of variables of* $\mathcal{P}$*, the* **projection** *of* $\mathcal{P}$ *onto* L*, denoted* $\pi_L(\mathcal{P})$*, is the* r*-ary cost function defined as follows:*

$$\pi_L(\mathcal{P})(x_1, \ldots, x_r) \stackrel{\text{def}}{=} \min_{\{s:V\to D \;\mid\; \langle s(v_1),\ldots,s(v_r)\rangle = \langle x_1,\ldots,x_r\rangle\}} Cost_{\mathcal{P}}(s)\,.$$

We say that a cost function ϕ *is* **expressible** *over a constraint language* Γ *if there exists a VCSP instance* $\mathcal{P} \in \text{VCSP}(\Gamma)$ *and a list* L *of variables of* $\mathcal{P}$ *such that* $\pi_L(\mathcal{P}) = \phi$*. We define* Express($\Gamma$) *to be the* **expressive power** *of* Γ*; that is, the set of all cost functions expressible over* Γ*.*

Note that the list of variables L may contain repeated entries, and we define the minimum over an empty set of costs to be ∞.

Example 1. Let $\mathcal{P}$ be the VCSP instance with a single variable v and no constraints, and let $L = \langle v, v\rangle$. Then, by Definition 2,

$$\pi_L(\mathcal{P})(x, y) = \begin{cases} 0 & \text{if } x = y \\ \infty & \text{otherwise} \end{cases}.$$

Hence for any valued constraint language Γ, over any set D, Express(Γ) contains this binary cost function, which will be called the **equality** cost function.

The next result shows that expressibility preserves tractability.

Theorem 1 ([15]). *A valued constraint language Γ is tractable if and only if* Express(Γ) *is tractable; similarly, Γ is NP-hard if and only if* Express(Γ) *is NP-hard.*

This result shows that, when trying to identify tractable valued constraint languages, it is sufficient to consider only languages of the form Express(Γ). In the following sections, we will show that such languages can be characterised using certain algebraic properties.

3 Weighted Relational Clones

Definition 3. *We denote by $\mathbf{\Phi}_D$ the set of cost functions on D taking values in $\overline{\mathbb{Q}}_+$ and by $\mathbf{\Phi}_D^{(r)}$ the r-ary cost functions in $\mathbf{\Phi}_D$.*

Definition 4. *Any cost function $\phi : D^r \to \overline{\mathbb{Q}}_+$ has an associated cost function which takes only the values 0 and ∞, known as its* **feasibility relation***, denoted* Feas(ϕ)*, which is defined as follows:*

$$\mathrm{Feas}(\phi)(x_1, \ldots, x_r) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \phi(x_1, \ldots, x_r) < \infty \\ \infty & \text{otherwise} \end{cases} .$$

We now define a closure operator on cost functions with rational costs, which adds to a set of cost functions all other cost functions which can be obtained from that set by a certain affine transformation.

Definition 5. *We say $\phi, \phi' \in \mathbf{\Phi}_D$ are* **cost-equivalent***, denoted by $\phi \sim \phi'$, if there exist $\alpha, \beta \in \mathbb{Q}$ with $\alpha > 0$ such that $\phi = \alpha\phi' + \beta$. We denote by $\Gamma_\sim$ the smallest set of cost functions containing Γ which is closed under cost-equivalence.*

The next result shows that adding feasibility relations or cost-equivalent cost functions does not increase the complexity of Γ.

Theorem 2 ([15]). *For any valued constraint language Γ, we have:*

1. *$\Gamma \cup$ Feas(Γ) is tractable if and only if Γ is tractable, and $\Gamma \cup$ Feas(Γ) is NP-hard if and only if Γ is NP-hard.*
2. *$\Gamma_\sim$ is tractable if and only if Γ is tractable, and $\Gamma_\sim$ is NP-hard if and only if Γ is NP-hard.*

The algebraic approach to complexity for the classical CSP uses standard algebraic notions of polymorphisms, clones and relational clones [12,7,24].

Here we introduce an algebraic theory for valued constraints based on the notions of *weighted polymorphisms*, *weighted clones* and *weighted relational clones*, defined below.

Definition 6. *We say a set $\Gamma \subseteq \mathbf{\Phi}_D$ is a* **weighted relational clone** *if it contains the equality cost function and is closed under cost-equivalence and feasibility; rearrangement of arguments; addition of cost functions; and minimisation over arbitrary arguments. For each $\Gamma \subseteq \mathbf{\Phi}_D$ we define* wRelClone(Γ) *to be the smallest weighted relational clone containing Γ.*

It is a straightforward consequence of Definitions 2 and 6 that, for any valued constraint language $\Gamma \subseteq \mathbf{\Phi}$, the set of cost functions that are cost equivalent to the expressive power of Γ, together with all associated feasibility relations, is given by the smallest weighted relational clone containing Γ, as the next result indicates.

Proposition 1. *For any* $\Gamma \subseteq \mathbf{\Phi}_D$, $\mathrm{Express}(\Gamma \cup \mathrm{Feas}(\Gamma))_\sim = \mathrm{wRelClone}(\Gamma)$.

Hence, by Theorem 1 and Theorem 2, the search for tractable valued constraint languages taking values in $\overline{\mathbb{Q}}_+$ corresponds to a search for suitable weighted relational clones. As has been done in the crisp case [12], we will now proceed to establish an alternative characterisation for weighted relational clones which facilitates this search.

4 Weighted Clones

For any finite set D, a function $f : D^k \to D$ is called a k-ary *operation* on D.

Definition 7. *We denote by* $\mathbf{O}_D$ *the set of all finitary operations on* D *and by* $\mathbf{O}_D^{(k)}$ *the* k*-ary operations in* $\mathbf{O}_D$.

Definition 8. *The* k*-ary* **projections** *on* D *are the operations*

$$e_i^{(k)} : D^k \to D\,, \quad (a_1, \ldots, a_k) \mapsto a_i\,.$$

Definition 9. *We define a* k*-ary* **weighted operation** *on a set* D *to be a* partial function $\omega : \mathbf{O}_D^{(k)} \to \mathbb{Q}$ *such that* $\omega(f) < 0$ *only if* f *is a projection and*

$$\sum_{f \in \mathbf{dom}(\omega)} \omega(f) = 0\,.$$

The **domain** *of* ω, *denoted* $\mathbf{dom}(\omega)$, *is the subset of* $\mathbf{O}_D^{(k)}$ *on which* ω *is defined. We denote by* $ar(\omega) = k$ *the arity of* ω.

We denote by $\mathbf{W}_D$ *the finitary weighted operations on* D *and by* $\mathbf{W}_D^{(k)}$ *the* k*-ary weighted operations on* D.

Definition 10. *We say that two* k*-ary weighted operations* $\omega, \mu \in \mathbf{W}_D^{(k)}$ *are* **weight-equivalent** *if* $\mathbf{dom}(\omega) = \mathbf{dom}(\mu)$ *and there exists some fixed positive rational* c, *such that* $\omega(f) = c\mu(f)$, *for all* $f \in \mathbf{dom}(\omega)$.

Definition 11. *For any* $\omega_1, \omega_2 \in \mathbf{W}_D^{(k)}$, *we define the sum of* ω_1 *and* ω_2, *denoted* $\omega_1 + \omega_2$, *to be the* k*-ary weighted operation* ω *with* $\mathbf{dom}(\omega) = \mathbf{dom}(\omega_1) \cup \mathbf{dom}(\omega_2)$ *and*

$$\omega(f) = \begin{cases} \omega_1(f) + \omega_2(f) & f \in \mathbf{dom}(\omega_1) \cap \mathbf{dom}(\omega_2) \\ \omega_1(f) & f \in \mathbf{dom}(\omega_1) \setminus \mathbf{dom}(\omega_2) \\ \omega_2(f) & f \in \mathbf{dom}(\omega_2) \setminus \mathbf{dom}(\omega_1) \end{cases}\,. \tag{1}$$

Definition 12. *Let* $f \in \mathbf{O}_D^{(k)}$ *and* $g_1, \ldots, g_k \in \mathbf{O}_D^{(l)}$. *The* superposition *of* f *and* $g_1, \ldots, g_k$ *is the* l*-ary operation* $f[g_1, \ldots, g_k] : D^l \to D$, $(x_1, \ldots, x_l) \mapsto f(g_1(x_1, \ldots, x_l), \ldots, g_k(x_1 \ldots, x_l))$.

Definition 13. *For any* $\omega \in \mathbf{W}_D^{(k)}$ *and any* $g_1, g_2, \ldots, g_k \in \mathbf{O}_D^{(l)}$, *we define the* **translation** *of* ω *by* $g_1, \ldots, g_k$, *denoted* $\omega[g_1, \ldots, g_k]$, *to be the* partial function $\omega[g_1, \ldots, g_k]$ *from* $\mathbf{O}_D^{(l)}$ *to* $\mathbb{Q}$ *defined by*

$$\omega[g_1, \ldots, g_k](f) \stackrel{\text{def}}{=} \sum_{\substack{f' \in \mathbf{dom}(\omega) \\ f = f'[g_1, \ldots, g_k]}} \omega(f') \,. \tag{2}$$

The domain of $\omega[g_1, \ldots, g_k]$ *is the set of* l*-ary operations* $\{f'[g_1, g_2, \ldots, g_k] \mid f' \in \mathbf{dom}(\omega)\}$.

Example 2. Let ω be the 4-ary weighted operation on D given by

$$\omega(f) = \begin{cases} -1 & \text{if } f \text{ is a projection} \\ +1 & \text{if } f \in \{\max(x_1, x_2), \min(x_1, x_2), \max(x_3, x_4), \min(x_3, x_4)\} \end{cases},$$

and let

$$\langle g_1, g_2, g_3, g_4 \rangle = \left\langle e_1^{(3)}, e_2^{(3)}, e_3^{(3)}, \max(x_1, x_2) \right\rangle .$$

Then, by Definition 13 we have

$$\omega[g_1, g_2, g_3, g_4](f) = \begin{cases} -1 & \text{if } f \text{ is a projection} \\ +1 & \text{if } f \in \{\max(x_1, x_2, x_3), \min(x_1, x_2), \min(x_3, \max(x_1, x_2))\} \\ 0 & \text{if } f = \max(x_1, x_2) \end{cases} .$$

Note that $\omega[g_1, g_2, g_3, g_4]$ satisfies the conditions of Definition 9 and hence is a weighted operation.

Example 3. Let ω be the same as in Example 2 but now consider

$$\langle g_1', g_2', g_3', g_4' \rangle = \left\langle e_1^{(4)}, \max(x_2, x_3), \min(x_2, x_3), e_4^{(4)} \right\rangle .$$

By Definition 13 we have

$$\omega[g_1', g_2', g_3', g_4'](f) = \begin{cases} -1 & \text{if } f \in \{e_1^{(4)}, \max(x_2, x_3), \min(x_2, x_3), e_4^{(4)}\} \\ +1 & \text{if } f \in \left\{ \begin{array}{l} \max(x_1, x_2, x_3), \min(x_1, \max(x_2, x_3)), \\ \max(\min(x_2, x_3), x_4), \min(x_2, x_3, x_4) \end{array} \right\} \end{cases} .$$

Note that $\omega[g_1', g_2', g_3', g_4']$ does not satisfy the conditions of Definition 9 because, for example, $\omega[g_1', g_2', g_3', g_4'](f) < 0$ when $f = \max(x_2, x_3)$, which is not a projection. Hence $\omega[g_1', g_2', g_3', g_4']$ is not a weighted operation.

Definition 14. *If the result of a translation is a weighted operation, then that translation will be called a* **proper** *translation.*

Remark 1. For any $\omega \in \mathbf{W}_D^{(k)}$, if $g_1, \ldots, g_k$ are *projections*, then it can be shown that the function $\omega[g_1, \ldots, g_k]$ satisfies the conditions of Definition 9, and hence is a weighted operation. This means that a translation by any list of projections is always a proper translation.

We are now ready to define *weighted clones*.

Definition 15. *Let C be a clone of operations on D. We say a set $W \subseteq \mathbf{W}_D$ is a* **weighted clone** *with* **support** *C if it contains all zero-valued weighted operations whose domains are subsets of C and is closed under weight-equivalence, addition, and proper translation by operations from C.*

For each $W \subseteq \mathbf{W}_D$ we define $\mathrm{wClone}(W)$ *to be the smallest weighted clone containing W.*

Remark 2. The support of $\mathrm{wClone}(W)$ is the clone generated by the domains of the elements of W. That is, the support of $\mathrm{wClone}(W)$ is $\mathrm{Clone}(\cup_{\omega \in W} \mathbf{dom}(\omega))$.

Example 4. For any clone of operations, C, there exists a unique weighted clone which consists of all weighted operations assigning weight 0 to each subset of C.

Definition 16. *Let $\phi \in \mathbf{\Phi}_D^{(r)}$ and let $\omega \in \mathbf{W}_D^{(k)}$. We say that ω is a* **weighted polymorphism** *of ϕ if, for any $x_1, x_2, \ldots, x_k \in D^r$ such that $\phi(x_i) < \infty$ for $i = 1, \ldots, k$, we have*

$$\sum_{f \in \mathbf{dom}(\omega)} \omega(f)\phi(f(x_1, x_2, \ldots, x_k)) \leq 0\,. \tag{3}$$

If ω is a weighted polymorphism of ϕ we say ϕ is **improved** *by ω.*

Note that, because $a\infty = \infty$ for any value a (including $a = 0$), if inequality (3) holds we must have $\phi(f(x_1, \ldots, x_k)) < \infty$, for all $f \in \mathbf{dom}(\omega)$, i.e., each $f \in \mathbf{dom}(\omega)$ is a polymorphism of ϕ.

Example 5. Consider the class of submodular set functions [31]. These are precisely the cost functions on $\{0, 1\}$ satisfying

$$\phi(\min(x_1, x_2)) + \phi(\max(x_1, x_2)) - \phi(x) - \phi(y) \leq 0\,.$$

In other words, the set of submodular functions are defined as the set of cost functions on $\{0, 1\}$ with the 2-ary weighted polymorphism

$$\omega(f) = \begin{cases} -1 & \text{if } f \in \{e_1^{(2)}, e_2^{(2)}\} \\ +1 & \text{if } f \in \{\min(x_1, x_2), \max(x_1, x_2)\} \end{cases}.$$

Definition 17. *For any $\Gamma \subseteq \mathbf{\Phi}_D$, we denote by* $\mathrm{wPol}(\Gamma)$ *the set of all finitary weighted operations on D which are weighted polymorphisms of all cost function $\phi \in \Gamma$ and by* $\mathrm{wPol}^{(k)}(\Gamma)$ *the k-ary weighted operations in* $\mathrm{wPol}(\Gamma)$.

Definition 18. *For any $W \subseteq \mathbf{W}_D$, we denote by* $\mathrm{Imp}(W)$ *the set of all cost functions in $\mathbf{\Phi}_D$ that are improved by all weighted operations $\omega \in W$ and by* $\mathrm{Imp}^{(r)}(W)$ *the r-ary cost functions in* $\mathrm{Imp}(W)$.

It follows immediately from the definition of a Galois connection [8] that, for any set D, the mappings wPol and Imp form a Galois connection between $\mathbf{W}_D$ and $\mathbf{\Phi}_D$. A characterisation of this Galois connection for finite sets D is given by the following two theorems:

Theorem 3. *For any finite set D, and any finite $\Gamma \subseteq \mathbf{\Phi}_D$,*

$$\mathrm{Imp}(\mathrm{wPol}(\Gamma)) = \mathrm{wRelClone}(\Gamma).$$

Theorem 4. *For any finite set D, and any finite $W \subseteq \mathbf{W}_D$,*

$$\mathrm{wPol}(\mathrm{Imp}(W)) = \mathrm{wClone}(W).$$

As with any Galois connection [8], this means that there is a one-to-one correspondence between weighted clones and weighted relational clones. Hence, by Proposition 1, Theorem 1, and Theorem 2, the search for tractable valued constraint languages taking values in $\overline{\mathbb{Q}}_+$ corresponds to a search for suitable weighted clones of operations.

5 Proof of Theorems 3 and 4

A similar result to Theorem 3 was obtained in [15, Theorem 4] using the related algebraic notion of fractional polymorphism. The proof given in [15] can be adapted in a straightforward way, and we omit the details due to space constraints. We will sketch the proof of Theorem 4. First, we show in Proposition 2 that the weighted polymorphisms of a set of cost functions form a weighted clone. The rest of the theorem then follows from Theorem 5, which states that any weighted operation that improves all cost functions in $\mathrm{Imp}(W)$ is an element of the weighted clone $\mathrm{wClone}(W)$. Due to space constraints we will not include the proof of Theorem 5.

Proposition 2. *Let D be a finite set.*

1. *For all $\Gamma \subset \mathbf{\Phi}_D$, $\mathrm{wPol}(\Gamma)$ is a weighted clone with support $\mathrm{Pol}(\Gamma)$.*
2. *For all $W \subset \mathbf{W}_D$, $\mathrm{wClone}(W) \subseteq \mathrm{wPol}(\mathrm{Imp}(W))$.*

Proof. Certainly $\mathrm{wPol}(\Gamma)$ contains all zero-valued weighted operations with domains contained in $\mathrm{Pol}(\Gamma)$, since all of these satisfy the conditions set out in Definition 16. Similarly, $\mathrm{wPol}(\Gamma)$ is closed under addition and weight-equivalence, since both of these operations preserve inequality (3). Hence, to show $\mathrm{wPol}(\Gamma)$ is a weighted clone we only need to show $\mathrm{wPol}(\Gamma)$ is closed under proper translations by members of $\mathrm{Pol}(\Gamma)$.

Let $\omega \in \mathrm{wPol}^{(k)}(\Gamma)$ and suppose $\omega' = \omega[g_1, \ldots, g_k]$ is a proper translation of ω, where $g_1, g_2, \ldots, g_k \in \mathrm{Pol}^{(l)}(\Gamma)$. We will now show that $\omega' \in \mathrm{wPol}^{(l)}(\Gamma)$. Suppose ϕ is an r-ary cost function satisfying $\omega \in \mathrm{wPol}(\{\phi\})$, i.e., ϕ and ω satisfy (3) for any $x_1, x_2, \ldots, x_k \in \mathrm{Feas}(\phi)$. Given any $x'_1, x'_2, \ldots, x'_l \in \mathrm{Feas}(\phi)$, set $x_i = g_i(x'_1, x'_2, \ldots, x'_l)$ for $i = 1, 2, \ldots, k$. Then, if we set $f' = f[g_1, \ldots, g_k]$, we have

$f'(x'_1, x'_2, \ldots, x'_l) = f(x_1, x_2, \ldots, x_k)$, for any $f \in \mathbf{O}_D^{(k)}$. Hence, by Definition 13, we have

$$\sum_{f' \in \mathbf{dom}(\omega')} \omega'(f')\phi(f'(x'_1, x'_2, \ldots, x'_k) = \sum_{f \in \mathbf{dom}(\omega)} \omega(f)\phi(f(x_1, x_2, \ldots, x_k) \leq 0\,.$$

For the second part, we observe that $W \subseteq \mathrm{wPol}(\mathrm{Imp}(W))$. Hence, $\mathrm{wClone}(W) \subseteq \mathrm{wClone}(\mathrm{wPol}(\mathrm{Imp}(W))) = \mathrm{wPol}(\mathrm{Imp}(W))$. □

We will make use of the following lemma, which shows that a weighted sum of arbitrary translations of any weighted operations ω_1 and ω_2 can be obtained by translating ω_1 and ω_2 by projection operations, forming a weighted sum, and then translating the result.

Lemma 1. *For any weighted operations $\omega_1 \in \mathbf{W}_D^{(k)}$ and $\omega_2 \in \mathbf{W}_D^{(l)}$ and any $g_1, \ldots, g_k \in \mathbf{O}_D^{(m)}$ and $g'_1, \ldots, g'_l \in \mathbf{O}_D^{(m)}$,*

$$c_1\,\omega_1[g_1, \ldots, g_k] + \; c_2\,\omega_2[g'_1, \ldots, g'_l] = \omega[g_1, \ldots, g_k, g'_1, \ldots, g'_l]\,, \tag{4}$$

where $\omega = c_1\,\omega_1[e_1^{(k+l)}, \ldots, e_k^{(k+l)}] + \; c_2\,\omega_2[e_{k+1}^{(k+l)}, \ldots, e_{k+l}^{(k+l)}]$

Proof. For any $f \in \mathbf{dom}(\omega)$, the result of applying the right-hand side expression in equation (4) to f is:

$$\sum_{\substack{f' \in \mathbf{dom}(\omega) \\ f = f'[g_1, \ldots, g_k, g'_1, \ldots, g'_l]}} \left(\sum_{\substack{h' \in \mathbf{dom}(\omega_1) \\ f' = h'[e_1^{(k+l)}, \ldots, e_k^{(k+l)}]}} c_1\,\omega_1(h') \;\; + \sum_{\substack{h' \in \mathbf{dom}(\omega_2) \\ f' = h'[e_{k+1}^{(k+l)}, \ldots, e_{k+l}^{(k+l)}]}} c_2\,\omega_2(h') \right).$$

Replacing each f' by the equivalent superposition of h' with projections, we obtain:

$$\sum_{\substack{h' \in \mathbf{dom}(\omega_1) \\ f = h'[g_1, \ldots, g_k]}} c_1\,\omega_1(h') + \sum_{\substack{h' \in \mathbf{dom}(\omega_2) \\ f = h'[g'_1, \ldots, g'_l]}} c_2\,\omega_2(h')\,,$$

which is the result of applying the left-hand-side of Equation 4 to f. □

Theorem 5. *For all finite $W \subset \mathbf{W}_D$, $\mathrm{wPol}(\mathrm{Imp}(W)) \subseteq \mathrm{wClone}(W)$.*

6 Conclusions

We have presented an algebraic theory of valued constraint languages analogous to the theory of clones used to study the complexity of the classical constraint satisfaction problem. We showed that the complexity of any valued constraint language with rational costs is determined by certain algebraic properties of the cost functions allowed in the language: the weighted polymorphisms. Compared to the results in [15], not only have we captured the complexity of valued

constraint languages, but we have also established a 1-to-1 connection between valued constraint languages and sets of weighted polymorphisms.

In previous work [27,26] it has been shown that every tractable crisp constraint language can be characterised by an associated clone of operations. This work initiated the use of algebraic properties in the search for tractable constraint languages, an area that has seen considerable activity in recent years; see, for instance, [11,13,12,10,28,22,3,1,2,5]. The results in this paper show that a similar result holds for the valued constraint satisfaction problem: every tractable valued constraint language is characterised by an associated weighted clone.

We believe that our results here will provide a similar impetus for the investigation of tractable valued constraint satisfaction problems. In fact, we have already commenced investigating minimal weighted clones in order to understand maximal valued constraint languages [19]. Building on Rosenberg's famous classification of minimal clones, we have obtained a similar classification of minimal weighted clones [19]. Furthermore, using the results from this paper, we have proved maximality of several known tractable valued constraint languages, including an alternative proof of the characterisation of all maximal Boolean valued constraint languages from [17]. Details on minimal weighted clones and other applications of our results will be included in the full version of this paper.

References

1. Barto, L., Kozik, M.: Constraint Satisfaction Problems of Bounded Width. In: FOCS 2009, pp. 461–471 (2009)
2. Barto, L., Kozik, M., Maróti, M., Niven, T.: CSP dichotomy for special triads. Proceedings of the American Mathematical Society 137(9), 2921–2934 (2009)
3. Barto, L., Kozik, M., Niven, T.: The CSP dichotomy holds for digraphs with no sources and no sinks. SIAM Journal on Computing 38(5), 1782–1802 (2009)
4. Barto, L.: The dichotomy for conservative constraint satisfaction problems revisited. In: LICS 2011 (2011)
5. Berman, J., Idziak, P., Marković, P., McKenzie, R., Valeriote, M., Willard, R.: Varieties with few subalgebras of powers. Trans. of AMS 362(3), 1445–1473 (2010)
6. Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-based CSPs and Valued CSPs Constraints 4(3), 199–240 (1999)
7. Bodnarčuk, V., Kalužnin, L., Kotov, V., Romov, B.: Galois theory for Post algebras. I. Cybernetics and Systems Analysis 5(3), 243–252 (1969)
8. Börner, F.: Basics of Galois Connections. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 38–67. Springer, Heidelberg (2008)
9. Bulatov, A.: A Graph of a Relational Structure and Constraint Satisfaction Problems. In: LICS 2004, pp. 448–457 (2004)
10. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. Journal of the ACM 53(1), 66–120 (2006)
11. Bulatov, A., Krokhin, A., Jeavons, P.: The complexity of maximal constraint languages. In: STOC 2001, pp. 667–674 (2001)
12. Bulatov, A., Krokhin, A., Jeavons, P.: Classifying the Complexity of Constraints using Finite Algebras. SIAM Journal on Computing 34(3), 720–742 (2005)

13. Bulatov, A.A.: Tractable Conservative Constraint Satisfaction Problems. In: LICS 2003, pp. 321–330 (2003)
14. Cohen, D.A., Cooper, M., Jeavons, P.G., Krokhin, A.A.: Soft constraints: Complexity and multimorphisms. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 244–258. Springer, Heidelberg (2003)
15. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: An algebraic characterisation of complexity for valued constraint. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 107–121. Springer, Heidelberg (2006)
16. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. Theoretical Computer Science 401(1-3), 36–51 (2008)
17. Cohen, D.A., Cooper, M.C., Jeavons, P.G., Krokhin, A.A.: The Complexity of Soft Constraint Satisfaction. Artificial Intelligence 170(11), 983–1016 (2006)
18. Cohen, D.A., Jeavons, P.G., Živný, S.: The expressive power of valued constraints: Hierarchies and collapses. Theoretical Computer Science 409(1), 137–153 (2008)
19. Creed, P., Živný, S.: On minimal weighted clones. In: CP 2011 (2011)
20. Creignou, N.: A dichotomy theorem for maximum generalized satisfiability problems. Journal of Computer and System Sciences 51(3), 511–522 (1995)
21. Creignou, N., Khanna, S., Sudan, M.: Complexity Classification of Boolean Constraint Satisfaction Problems. SIAM, Philadelphia (2001)
22. Deineko, V., Jonsson, P., Klasson, M., Krokhin, A.: The approximability of Max CSP with fixed-value constraints. Journal of the ACM 55(4) (2008)
23. Feder, T., Vardi, M.: The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. SIAM Journal on Computing 28(1), 57–104 (1998)
24. Geiger, D.: Closed systems of functions and predicates. Pacific Journal of Mathematics 27(1), 95–100 (1968)
25. Gyssens, M., Jeavons, P., Cohen, D.: Decomposing Constraint Satisfaction Problems Using Database Techniques. Artificial Intelligence 66(1), 57–89 (1994)
26. Jeavons, P.: On the Algebraic Structure of Combinatorial Problems. Theoretical Computer Science 200(1-2), 185–204 (1998)
27. Jeavons, P., Cohen, D., Gyssens, M.: Closure Properties of Constraints. Journal of the ACM 44(4), 527–548 (1997)
28. Jonsson, P., Klasson, M., Krokhin, A.: The Approximability of Three-valued MAX CSP. SIAM Journal on Computing 35(6), 1329–1349 (2006)
29. Kolmogorov, V., Živný, S.: The complexity of conservative finite-valued CSPs, Tech. rep., arXiv:1008.1555 (August 2010)
30. Kolmogorov, V., Živný, S.: Generalising tractable VCSPs defined by symmetric tournament pair multimorphisms, Tech. rep., arXiv:1008.3104 (August 2010)
31. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley & Sons, Chichester (1988)
32. Schaefer, T.: The Complexity of Satisfiability Problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226 (1978)
33. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., Chichester (1986)
34. Živný, D.A., Cohen, P.G.: Jeavons, The Expressive Power of Binary Submodular Functions. Discrete Applied Mathematics 157(15), 3347–3358 (2009)

On the Use of Guards for Logics with Data*

Thomas Colcombet[1], Clemens Ley[2], and Gabriele Puppis[2]

[1] CNRS/LIAFA
[2] Department of Computer Science, Oxford University

Abstract. The notion of orbit finite data monoid was recently introduced by Bojańczyk as an algebraic object for defining recognizable languages of data words. Following Büchi's approach, we introduce the new logic 'rigidly guarded MSO' and show that the data languages definable in this logic are exactly those recognizable by orbit finite data monoids. We also establish, following this time the approach of Schützenberger, McNaughton and Papert, that the first-order variant of this logic defines exactly the languages recognizable by aperiodic orbit finite data monoids. Finally, we give a variant of the logic that captures the larger class of languages recognized by non-deterministic finite memory automata.

1 Introduction

Data languages are languages over an infinite alphabet – the letters of which are called data values – which are closed under permutation of the data values. This invariance under permutation makes any property concerning the data values, other than equality, irrelevant. Some examples of data languages are:

- *The sets of words containing exactly k distinct data values.*
- *The sets of words where the first and last data values are equal.*
- *The sets of words with no consecutive occurrences of the same data value.*
- *The sets of words where each data value occurs at most once.* ($\star$)

The intention behind data values in data words (or data trees, ...) is to model, e.g. the id's in a database, or the process or users numbers in the log of a system. Those numbers are used as identifiers, and we are interested only in comparing them by equality. The invariance under permutation of data languages captures this intention. Data words can also be defined to have both a data value and a letter from a finite alphabet at each position. This is more natural in practice, and does not make any difference in the results to follow.

The paper aims at understanding better how the classical theory of regular languages can be extended to data languages. The classical theory associates regular languages to finite state automata or, equivalently, to finite monoids. For instance, important properties of regular languages can be detected by exploiting equivalences with properties of the monoid – see, e.g. Straubing's book [14] or Pin's survey [11] for an overview of the approach.

* Supported by the French ANR 2007 JCJC 0051 JADE.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 243–255, 2011.

Recently, Bojańczyk introduced the notion of *data monoids* [3] as a framework for algebraic characterizations of data languages. Bojańczyk focused on the languages of data words recognizable by *orbit finite data monoids*, an analog of finite monoids for data languages. All the above examples but $\star$ are recognizable with this definition. Our main objective is to understand better the expressive power of the orbit finite data monoid model by comparing it with automaton-based models and logical formalisms for data words.

In terms of logic, there is a natural way to define logics for data words. It is sufficient for this to use a predicate $x \sim y$ meaning that the data values at positions x and y are equal. In particular, one may think that the monadic (second-order) logic with this new predicate is a good candidate to equivalently specify recognizable languages, i.e., would play the role of monadic logic in the standard theory of regular languages. However, this is not the case, as monadic logic happens to be much too expressive. One inclusion indeed holds: every language of data words recognized by an orbit finite monoid is definable in monadic logic. However, the converse does not hold, as witnessed by the formula

$$\forall\, x, y.\ x \neq y \rightarrow x \nsim y, \tag{$\star\star$}$$

which defines the above (non-recognizable) language $\star$. More generally, it has been shown that monadic logic (indeed, even first-order logic with data equality) has an undecidable satisfiability problem and it can express properties not implementable by automaton models, such as finite memory automata (FMA, described below) [10]. We naturally aim at answering the following question:

> *Is there a natural variant of monadic logic which defines precisely the data languages recognizable by orbit finite data monoids?*

We answer this question positively, introducing *rigidly guarded MSO* (abbreviating *monadic second-order logic with rigidly guarded data equality tests*). This logic allows testing equality of two data values only when the two positions are related in an injective way (we say rigid). That is, data equality tests are allowed only in formulas of the form $\varphi(x, y) \ \wedge \ x \sim y$, where φ is rigid, i.e., defines a partial injection. For instance, it is easy to check whether there are two consecutive positions sharing the same data value, e.g., by the formula $\exists\, x, y.\ (x = y + 1) \ \wedge \ x \sim y$. The guard $(x = y + 1)$ is rigid since x uniquely determines y, and vice versa. However, it is impossible to describe the language $\star$ in this logic. In particular, the above formula $\star\star$ can be rewritten as $\neg\exists\, x, y.\ (x \neq y) \ \wedge \ x \sim y$, but this time the guard $x \neq y$ is not rigid: for a given x, there can be several y such that $x \neq y$. It may seem a priori that the fact that rigidity is a semantic property is a severe drawback. This is not the case since (i) rigidity can be enforced syntactically, and (ii) rigidity is decidable for formulas in our logic.

To validate the robustness of our approach, we also answer to the following question inspired from the seminal Schtzenberger-McNaughton-Papert result:

> *Does the rigidly guarded FO logic (i.e., the first-order fragment of rigidly guarded MSO) correspond to aperiodic orbit finite data monoids?*

We answer this question positively as well. We finally consider data languages recognizable by finite memory automata and we prove that a natural variant of rigidly guarded MSO, called ∃*backward-rigidly guarded MSO* captures the class of data languages recognized by non-deterministic finite memory automata.

Overall, we don't claim that data languages recognizable by orbit finite data monoids are the counterpart to the notion of regular languages in the standard theory, since this model is rather expressively weak (see related work below). However, in this restricted framework, we are able to recover several of the major results which hold for usual regular languages.

Related work. This work is highly related to the well known theory of regular languages. We refer by this to the key equivalence between monadic logic and regular languages due to Büchi [5], and the Schützenberger-McNaughton-Papert result that characterizes the subclass of first-order definable languages [13, 9].

The other branch of related work is the one concerned with languages of data words. The first contribution in this direction is due to Kaminski and Francez [7], who introduced finite memory automata (FMA for short). These automata possess a fixed finite set of registers that can store data values. At each step such an automaton can compare the current data value with the values stored in the registers, and can decide to store this value in some register (forgetting the previous content of the register). This model of automaton, in its non-deterministic form, has a decidable emptiness problem and an undecidable universality problem (decidability of the latter problem is recovered in the deterministic variant).

Recently, the deterministic model of FMA has been modified by requiring a stricter policy in the use of registers [2]. This modification does not affect the expressive power of the model, but, as opposed to the original model, the new model can be efficiently minimized. In [1] partial results on relating automata to logics are also given. The question of characterizing the first-order logic definable language among the languages recognized by deterministic FMA is still open.

Many other automaton models for data languages have been studied in the literature (see [12] for a survey). These include pebble automata [10] and data automata [4], the latter introduced as a mean of capturing decidable logics over data words. The algebraic theory for these models has not been developed, nor is there an exact characterization of definability in logics for any of these models.

Contribution and structure of the paper. These are our contributions:

1. We show how infinite orbit finite data monoids can be finitely represented.
2. We introduce a new logic called "rigidly guarded MSO" – a natural weakening of MSO logic with data equality tests. Although the syntax of our logic is based on a semantic property, one can decide whether a formula belongs to the logic or not.
3. We show that satisfiability of rigidly guarded MSO formulas is decidable.
4. We show that rigidly guarded MSO is as expressive as orbit finite data monoids, and that its first-order fragment corresponds precisely to aperiodic orbit finite data monoids.

5. We give a decidable variant of rigidly guarded MSO that captures the data languages recognized by non-deterministic finite memory automata and has a decidable satisfiability problem. We also provide a decidable logic for data trees along the same lines.

Section 2 gives preliminaries on data languages and data monoids, and explains how to define representations of data monoids with finitely many orbits. Section 3 introduces rigidly guarded MSO and its first-order fragment. Section 4 describes the translation from rigidly guarded MSO (resp., FO) formulas to orbit finite data monoids (resp., aperiodic orbit finite data monoids) recognizing the same languages. Section 5 describes the converse translation, namely, from (aperiodic) orbit finite data monoids to rigidly guarded MSO (resp., FO) formulas. Section 6 introduces a variant of rigidly guarded MSO that captures precisely the class of languages recognized by non-deterministic finite memory automata. Finally, Section 7 provides an assessment of the results and related open problems.

2 Data Monoids

In this paper, D will usually denote an infinite set of *data values* (e.g., $d, e, f, \ldots$) and A will denote a finite set of *symbols* (e.g., $a, b, c, \ldots$). A *data word* over the alphabet $D \times A$ is a finite sequence $u = (d_1, a_1) \ldots (d_n, a_n)$ in $(D \times A)^*$. The domain of u, denoted $\mathcal{D}om(u)$, is $\{1, \ldots, n\}$.

Given a set $C \subseteq D$ of data values, a *(data) renaming* on C is a permutation on C that is the identity for all but finitely many values of C. We denote by G_C the set of all renamings on C. Renamings are naturally extended to tuples of data values, data words, sets of data words, and so on. A *data language* over $D \times A$ is a set of data words in $(D \times A)^*$ that is closed under renamings in G_D.

Recall that a monoid is an algebraic structure $\mathcal{M} = (M, \cdot)$ where $\cdot$ is an associative product on M and M contains an identity $1_{\mathcal{M}}$. A monoid is *aperiodic* if for all elements s, there is n such that $s^n = s^{n+1}$. We say that the set G_C of renamings *acts* on a monoid $\mathcal{M} = (M, \cdot)$ if there is a function $\hat{}$ that maps every renaming $\tau \in G_C$ to an automorphism $\hat{\tau}$ on $\mathcal{M}$. That is, for all renamings $\tau, \pi \in G_C$ and all elements $s, t \in M$, we have (i) $\widehat{\tau \circ \pi} = \hat{\tau} \circ \hat{\pi}$, (ii) $\hat{\tau}_{\mathrm{id}}(s) = s$, where τ_{id} is the identity function on C, (iii) $\hat{\tau}(s) \cdot \hat{\tau}(t) = \hat{\tau}(s \cdot t)$, and (iv) $\hat{\tau}(1_{\mathcal{M}}) = 1_{\mathcal{M}}$. For example, consider the free monoid $((D \times A)^*, \cdot)$ consisting of all finite words over $D \times A$ equipped with the operation of juxtaposition (the empty word ε playing the role of the identity). The group G_D of data renamings acts on the free monoid when the action is defined by $\hat{\tau}((d_1, a_1) \ldots (d_n, a_n)) = (\tau(d_1), a_1) \ldots (\tau(d_n), a_n)$.

We say that a renaming τ is a *stabilizer* of an element s of a monoid $\mathcal{M}$ acted upon by G_C, if $\hat{\tau}(s) = s$. A set $C' \subseteq D$ of data values *supports* an element s if all renamings that are the identity on C' are stabilizers of s. It is known that the intersection of two sets that support s is a set that supports s as well [3, 6]. Hence we can define *the memory* of s, denoted $\mathrm{mem}(s)$, as the intersection of all sets that support s. Note that there are finite monoids whose elements have infinite memory (see [3] for an example). On the other hand, monoids that are homomorphic images of the free monoid contains only elements with finite

memory. As we are interested in homomorphic images of the free monoid, we will consider monoids whose elements have finite memory (this property is called *finite support axiom* and the resulting algebraic objects *data monoids*).

Definition 1. *A* data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$ *over* C *is a monoid* $(M, \cdot)$ *that is acted upon by* G_C*, in which every element has finite memory.*

Unless otherwise stated, data monoids are defined over the set D of data values.

The *orbit* of an element s of $\mathcal{M} = (M, \cdot, \hat{\ })$ is the set of all elements $\hat{\tau}(s)$ with $\tau \in G_D$. Note that two orbits are either disjoint or equal. We say that $\mathcal{M}$ is *orbit finite* if it contains finitely many orbits. It is easy to see that if two elements are on the same orbit, then their memories have the same size. Hence an orbit finite data monoid has a uniform bound on the size of the memories (this is not true for arbitrary data monoids).

A *morphism* between two data monoids $\mathcal{M} = (M, \cdot, \hat{\ })$ and $\mathcal{N} = (N, \odot, \check{\ })$ is a monoid morphism that commutes with the action of renamings.A data language $L \subseteq (D \times A)^*$ is *recognized* by a morphism $h : (D \times A)^* \to \mathcal{M}$ if the membership of a word $u \in (D \times A)^*$ in L is determined by the element $h(u)$ of $\mathcal{M}$, namely, if $L = h^{-1}(h(L))$. As L is closed under renamings, $h(L)$ is a union of orbits.

Finite presentations of data monoids. Since orbit finite data monoids are infinite objects, we need suitable representations that ease algorithmic manipulation. The basic idea is to consider the restriction of an orbit finite data monoid to a finite set of data values:

Definition 2. *Given a data monoid* $\mathcal{M} = (M, \cdot, \hat{\ })$ *and* $C \subseteq D$*, we define the* restriction *of* $\mathcal{M}$ *to* C *to be* $\mathcal{M}|_C = \big(M|_C, \cdot|_C, \hat{\ }|_C\big)$*, where* $M|_C$ *consists of all elements* $s \in M$ *such that* $\mathrm{mem}(s) \subseteq C$*,* $\cdot|_C$ *is the restriction of* $\cdot$ *to* $M|_C$*, and* $\hat{\ }|_C$ *is the restriction of* $\hat{\ }$ *to* G_C *and* $M|_C$*.*

Note that $s \cdot t \in M|_C$ and $\hat{\tau}(s) \in M|_C$ for all $s, t \in M|_C$ and $\tau \in G_C$. Hence, if C is finite, $\mathcal{M}|_C$ is a finite data monoid.[1] Hereafter, we denote by $\|\mathcal{M}\|$ the maximum cardinality of the memories of the elements of an orbit finite data monoid $\mathcal{M}$.

Proposition 1. *Let* $\mathcal{M}$*,* $\mathcal{M}'$ *be orbit finite data monoids such that* $\|\mathcal{M}\| = \|\mathcal{M}'\|$ *and let* $C \subseteq D$ *be of cardinality at least* $2\|\mathcal{M}\|$*. If* $\mathcal{M}|_C$ *and* $\mathcal{M}'|_C$ *are isomorphic, then so are* $\mathcal{M}$ *and* $\mathcal{M}'$*.*

The above proposition shows that the restriction of an orbit finite data monoid $\mathcal{M}$ over a *sufficiently large* finite set C uniquely determines $\mathcal{M}$. A more careful analysis shows that many operations on orbit finite data monoids (e.g., the product of two such monoids, the quotient with respect to a congruence) can be performed at the level of the finite restriction. This allows us to effectively compute the results of algebraic operations on orbit finite data monoids.

[1] One has to keep in mind that data monoids over finite sets do not satisfy the same properties as those over infinite sets. For instance, the Memory Theorem, as stated in [3], does not hold for data monoids over finite sets.

Term-based presentations of data monoids. We have just shown how we can represent an infinite data monoid by a finite one. It is also possible to give a more explicit presentation of orbit finite data monoids using what we call a *term-based presentation system*. Each element is a term of the form $o(d_1, \ldots, d_k)$ in which o is an *orbit name* (belonging to some fixed set) of a fixed arity k, and $d_1, \ldots, d_k$ are distinct values. Those terms are furthermore considered modulo an equivalence relation, and equipped with a binary operation. Such a presentation is valid if the binary operation is associative over the equivalence classes, and if the data values respect the renaming policy required for data monoids. Under those suitable assumptions, the equivalence classes of terms equipped with the associative operation as product and the natural renaming operations form a data monoid. Furthemore, if there are finitely many orbit names, then the represented data monoid is orbit finite. We also show that conversely, every orbit finite data monoid can be represented by such a term-based representation, using finitely many orbit names.

This kind of presentation ease algorithmic manipulations of the elements of the data monoid, and are heavily used in the proofs. Some open questions are directly related to this presentation such as: is it possible to get rid of the equivalence relation for recognizing a language of data words?

3 Rigidly Guarded Logics

From now on, we abbreviate monadic second-order logic with data equality tests by *MSO*. MSO formulas are built up from atoms of the form $x < y$, $a(x)$, $x \in X$, or $x \sim y$, using boolean connectives and existential quantifications over first-order variables (e.g., $x, y, \ldots$) and monadic second-order variables (e.g., $X, Y, \ldots$). The meaning of an atom $x \sim y$ is that the data values at the two positions that correspond to the interpretation of the variables x and y must be the same. The meaning of the other predicates is as usual.

Here we introduce a new logic called "rigidly guarded MSO". We say that a formula $\varphi(x, y)$ with two free first-order variables x, y is *rigid* if for all data words $u \in (D \times A)^*$ and all positions x (resp., y) in u, there is *at most one* position y (resp., x) in u such that $u \vDash \varphi(x, y)$. *Rigidly guarded MSO* is obtained from MSO by enforcing the following restriction: every data equality test of the form $x \sim y$ must be guarded by a rigid formula $\varphi(x, y)$. Precisely, the formulas of rigidly guarded MSO are build up using the following grammar:

$$\varphi \;:=\; \exists\, x.\, \varphi \mid \exists\, Y.\, \varphi \mid x{<}y \mid a(x) \mid x \in Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi_{\text{rigid}}(x, y) \wedge x{\sim}y$$

where $a \in A$ and $\varphi_{\text{rigid}}(x, y)$ is a *rigid* formula that is generated by the same grammar. *Rigidly guarded FO* is the first-order fragment of rigidly guarded MSO.

The notion of rigidity is a semantic property, and this may seem problematic. However, we can enforce rigidity syntactically as follows. Instead of a guard $\varphi_{\text{rigid}}(x, y)$ in a formula, one uses the new guard

$$\widetilde{\varphi}_{\text{rigid}}(x, y) \;=^{\text{def}}\; \varphi_{\text{rigid}}(x, y) \;\wedge\; \forall\, x', y'.\, \varphi_{\text{rigid}}(x', y') \to (x = x' \leftrightarrow y = y')\ .$$

It is easy to check that $\widetilde{\varphi}_{\texttt{rigid}}$ is always rigid, and that furthermore, if φ_{rigid} is rigid then it is equivalent to $\widetilde{\varphi}_{\texttt{rigid}}$. This trick allows us to enforce rigidity syntactically. We will also see in Corollary 2 below that one can decide if a formula respects the rigidity assumption in all its guards (the problem being undecidable when data tests are not guarded).

We remark that in this logic, the similar constructions $\varphi_{\text{rigid}}(x,y) \wedge x \nsim y$, $\varphi_{\text{rigid}}(x,y) \to x \sim y$, and $\varphi_{\text{rigid}}(x,y) \to x \nsim y$ can be derived. This is thanks to the Boolean equivalences $\varphi \to \varphi'$ iff $\varphi \to (\varphi \wedge \varphi')$, $\varphi \wedge \neg\varphi'$ iff $\neg(\varphi \to \varphi')$, and $\varphi \to \neg\varphi'$ iff $\neg(\varphi \wedge \varphi')$.

Example 1. Let us consider the language $L_{\geq k}$ of all data words that contain at least k different data values. If $k = 1$ we just need to check the non-emptiness of the word by the sentence $\exists\, x.\ \texttt{true}$. For $k = 2$ it is sufficient to test for the existence of two distinct *consecutive* data values, using for instance the formula $\exists\, x, y.\ (x+1 = y) \wedge x \nsim y$. For $k > 2$, one can proceed by induction as follows. One first observes that if a word has at least k distinct data values, then there is a minimal factor witnessing this property, say $[x, y]$. A closer inspection reveals that, in this case, $[x+1, y-1]$ is a maximal factor that uses exactly $k-2$ data values and thus belongs to the language $L_{\geq k-2} \setminus L_{\geq k-1}$. By induction, the fact that $[x+1, y-1]$ is a maximal factor that belongs to $L_{\geq k-2} \setminus L_{\geq k-1}$ is expressible in rigidly guarded FO by a formula $\varphi(x, y)$. Furthermore, this formula $\varphi(x, y)$ is *rigid* according to its semantic definition. We conclude that the language $L_{\geq k}$ is defined by the formula $\exists\, x, y.\ \varphi(x, y) \wedge x \nsim y$.

To simplify the notation, it is sometimes convenient to think of a first-order variable x as a second-order variable X interpreted as a singleton set. Therefore, by a slight abuse of notation, we shall often write variables in uppercase letters, without explicitly saying whether these are first-order or second-order variables (their correct types can be inferred from the atoms they appear in). As usual, we write $\varphi(X_1, \ldots, X_m)$ whenever we want to make explicit that the free variables of φ are among $X_1, \ldots, X_m$. Moreover, given a formula $\varphi(X_1, \ldots, X_m)$, a data word $u \in (D \times A)^*$, and some unary predicates $U_1, \ldots, U_m \subseteq \mathcal{D}om(u)$, we write $u \vDash \varphi(U_1, \ldots, U_m)$ whenever φ holds on u by interpreting the free variables $X_1, \ldots, X_m$ with the predicates $U_1, \ldots, U_m$.

As usual, given a formula $\varphi(\bar{X})$ with some free (first-order or monadic second-order) variables $X_1, \ldots, X_m$, one can see it as defining the language

$$[\![\varphi]\!] \;=\; \{\langle u, U_1, \ldots, U_m\rangle \;:\; u \vDash \varphi(U_1, \ldots, U_m)\} \;\subseteq\; (D \times A \times B^m)^*$$

where B denotes the binary alphabet $\{0, 1\}$ and $\langle u, U_1, \ldots, U_m\rangle$ is the word over the alphabet $D \times A \times B^m$ that has letter $(d, a, b_1, \ldots, b_m)$ at position i iff (d, a) is the i-th letter of u, and for all $j = 1 \ldots m$, b_j is 1 if $i \in U_j$, and 0 otherwise.

4 From the Logic to Data Monoids

In this section, we show that every data language defined by a rigidly guarded MSO sentence is recognized by an orbit finite data monoid. Our proof follows

the classical technique for showing that MSO definable languages over standard words can be recognized by monoids. Namely, we show that each construction in the logic corresponds to a closure under some operation on recognizable languages: disjunction corresponds to union, negation corresponds to complement, existential quantification corresponds to projection, etc.

The principle of the proof is to establish that, given a rigidly guarded MSO formula $\varphi(\bar{X})$, the language $[\![\varphi]\!]$ is recognized by an orbit finite data monoid. Though this statement is true, it cannot be used – as it is the case in the standard theory – as an induction hypothesis. The problem is that the operation that corresponds to existential quantification (i.e. projection) transforms an orbit finite data monoid into a data monoid which is not orbit finite, in general. That is why our induction hypothesis is stronger, and states that $[\![\varphi]\!]$ is recognized by an orbit finite data monoid via a *projectable* morphism, to be defined below (we write $s \doteq t$ whenever the elements s and t are in the same orbit):

Definition 3. *Let h be a morphism from the free data monoid $(D \times A \times B^m)^*$ to a data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$. We say that h is* projectable *if for all data words $u \in (D \times A)^*$ and all tuples of predicates $\bar{U} = (U_1, \dots, U_m)$ and $\bar{V} = (V_1, \dots, V_m)$,*

$$h(\langle u, \bar{U}\rangle) \doteq h(\langle u, \bar{V}\rangle) \quad \textit{implies} \quad h(\langle u, \bar{U}\rangle) = h(\langle u, \bar{V}\rangle) \ .$$

We now state the theorem, which is at the same time our induction hypothesis:

Theorem 1. *For all rigidly guarded MSO formulas $\varphi(\bar{X})$, the language $[\![\varphi]\!]$ is effectively recognized by an orbit finite data monoid with a projectable morphism.*

From the above theorem we obtain, in particular, the following key corollaries:

Corollary 1. *Every data language definable in rigidly guarded MSO (resp., rigidly guarded FO) is recognizable by an orbit finite data monoid (resp., aperiodic orbit finite data monoid).*

Note that the claim for the first-order case is deduced using the result that every language recognized by an orbit finite data monoid and definable in FO (without any rigidity assumption) is recognized by an aperiodic orbit finite data monoid [3]. That is why Theorem 1 needs not to consider the first-order case.

Corollary 2. *The satisfiability problem for rigidly guarded MSO logic is decidable. Moreover, one can decide whether a formula belongs to the rigidly guarded MSO logic, and in this case whether the formula is rigid.*

The proof of Theorem 1 is by structural induction on the rigidly guarded MSO formulas: the translation of the atomic formulas $x < y$, $a(x)$, $x \in Y$ are easy (at least towards non-aperiodic monoids) and the translations of the Boolean connectives are as in the classical case.

The translation of the existential closures uses a powerset construction on orbit finite data monoids. Since data monoids are in general infinite objects, the standard powerset construction would yield infinitely many orbits even if the original data monoid has finitely many of them. In our case, the construction remembers all possible elements of the original monoid, but since the morphism is projectable, one never has to store more than one element per orbit. Indeed, whenever another element in the same orbit is encountered, it has to be equal to the one already present: this limitation allows us to preserve orbit finiteness.

The most technical part concerns the translation of the rigidly guarded data tests $\varphi(x,y) \ \wedge \ x \sim y$. The rigidity assumption on the guard $\varphi(x,y)$ is crucial for this result: if $\varphi(x,y)$ were not rigid, then the data monoid recognizing $[\![\varphi(x,y) \ \wedge \ x \sim y]\!]$ would still be orbit finite, but the morphism would in general not be projectable. The proof that $[\![\varphi(x,y) \ \wedge \ x \sim y]\!]$ is recognized via a projectable morphism requires a bit of analysis since rigidity is a semantic assumption and hence one cannot directly deduce from it a property for the data monoid. However, one can use the rigidity property for "normalizing" the data monoid, allowing the construction to go through.

5 From Data Monoids to the Logic

Having shown that every language defined by a rigidly guarded MSO (resp., FO) formula is recognized by an orbit finite data monoid (resp., by an aperiodic orbit finite data monoid), we now show the converse.

Theorem 2. *Given an orbit finite data monoid $\mathcal{M}$, a morphism h from the free data monoid to $\mathcal{M}$, and an orbit o, one can compute a rigidly guarded MSO sentence φ that defines the data language $L = h^{-1}(o)$. Moreover, if $\mathcal{M}$ is aperiodic, then φ is a rigidly guarded FO sentence.*

This is the most technical result of the paper. Note that in the classical theory of regular languages, the analogous of Theorem 2 (at least the part involving only MSO) is straightforward: indeed, a monoid can be used as an automaton, and it is sufficient to write an MSO formula that guesses a run of such an automaton and checks that it is valid and accepting. We cannot use such a proof for data monoids: not only there is no equivalent automaton model, but furthermore, the above approach is intrinsically not compatible with the notion of rigidity.

Our proof follows a structure similar to Schützenberger's proof that languages recognized by aperiodic monoids are definable by star-free expressions (i.e., in FO logic). The proof relies on an induction on the structure of ideals of the data monoid, the so called *Green's relations* [11]. This requires specific study of this theory for orbit finite data monoids. Such a study was initiated by Bojańczyk [3], but we had to develop several new tools for our proof to go through (these tools concern the size of $\mathcal{H}$-classes and the analysis of the memory inside the $\mathcal{J}$-classes). As opposed to the classical case, the proof is significantly more involved for MSO compared to FO.

6 Logics for Finite Memory Automata

In this section, we try to see how guards as introduced above can help constructing decidable logics. We consider languages recognized by *finite memory automata* (FMA) [7]. These extend finite state automata by a finite set of registers, storing values from an infinite alphabet D. Data words are processed from left to right. At each step the automaton compares (up to equality) the current input value with the values that are stored in its registers. Based on this information, the automaton decides whether or not to store the input value in one of its registers, updates its state, and moves one symbol to the right.

The deterministic variant of FMA can be viewed as the natural automaton counterpart of orbit finite data monoids. However, deterministic FMA are more expressive than orbit finite data monoids, as witnessed by the language

$$L \ =^{\text{def}} \ \{d_1 \dots d_n \ : \ n \in \mathbb{N},\ d_1 = d_i \text{ for some } 1 < i \leq n\}$$

which is recognizable by deterministic FMA, but not by orbit finite data monoids. Moreover, unlike classical finite automata, non-deterministic FMA are even more expressive than deterministic FMA, as witnessed by the language

$$L' \ =^{\text{def}} \ \{d_1 \dots d_n \ : \ n \in \mathbb{N},\ d_i = d_j \text{ for some } 1 \leq i < j \leq n\}.$$

It thus comes natural to look for logical characterizations of data languages recognizable by deterministic (resp., non-deterministic) FMA.

A natural attempt at finding a logic for FMA consists in relaxing the notion of rigidity. One could imagine using backward-rigid guards for data tests. These are formulas $\varphi(x, y)$ that determine the leftmost position $\min(x, y)$ from the rightmost position $\max(x, y)$ (but possibly not the other way around). Formulas of *backward-rigidly guarded MSO* are built up using the grammar:

$$\varphi \ := \ \exists\, x.\, \varphi \mid \exists\, Y.\, \varphi \mid x < y \mid a(x) \mid x \in Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi_{\text{backward}}(x, y) \wedge x \sim y$$

where $\varphi_{\text{backward}}$ is a backward-rigid formula generated from the same grammar (as usual, we can enforce backward-rigidity syntactically). For example the above language L can be easily defined by the backward-rigidly guarded MSO sentence $\exists\, x, y.\, (x < y \wedge \forall\, z.\, x \leq z) \wedge x \sim y$. One can translate backward-rigidly guarded MSO formulas to equivalent deterministic FMA, but not the other way around:

Proposition 2. *Every language definable in backward-rigidly guarded MSO is recognizable by deterministic FMA. There is a language recognized by a deterministic FMA which cannot be defined in backward-rigidly guarded MSO.*

We do not have a candidate logic that corresponds precisely to deterministic FMA. However, we are able to characterize the larger class of languages recognized by non-deterministic FMA. The logic for this class is obtained from backward-rigidly guarded MSO by allowing the guards to use additional second-order variables (which however needs to be quantified existentially in the outermost part of the formula). The logic, abbreviated $\exists$*backward-rigidly guarded MSO*, consists of the formulas $\exists\, \bar{Z}.\, \varphi$, with φ is generated by the grammar

$$\varphi := \exists x.\, \varphi \mid \exists Y.\, \varphi \mid x < y \mid a(x) \mid x \in Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi_{\exists\text{backward}}(x, y, \bar{Z}) \wedge x \sim y$$

where $\varphi_{\exists\text{backward}}$ is a formula from the same grammar that determines $\min(x, y)$ from $\max(x, y)$ and $\bar{Z}$, and where the quantifications are over variables different from $\bar{Z}$ (the variables $\bar{Z}$ are quantified only in the outermost part of $\exists\ \bar{Z}.\ \varphi$).

Theorem 3. *A language is definable in ∃backward-rigidly guarded MSO iff it is recognizable by non-deterministic FMA.*

As it happens for rigidly guarded MSO logic, one can derive from Theorem 3 the following decidability results:

Corollary 3. *The satisfiability problem for ∃backward-rigidly guarded MSO is decidable. Moreover, one can decide whether a formula belongs to the ∃backward-rigidly guarded MSO, and in this case whether the formula is ∃backward-rigid.*

It is also easy to generalize both Theorem 3 and Corollary 3 to data *tree* languages recognized by non-deterministic finite memory tree automata [8]. For this we use a natural variant of ∃backward-rigidly guarded MSO on data trees. The guarded tests in this case are of the form

$$\varphi_{\exists\,\text{downward}}(x, y, \bar{Z})\ \wedge\ \varphi'_{\exists\,\text{downward}}(x, z, \bar{Z})\ \wedge\ y \sim z$$

where $\varphi_{\exists\,\text{downward}}(x, y, \bar{Z})$ (resp. $\varphi'_{\exists\,\text{downward}}(x, z, \bar{Z})$) is a formula in the logic that determines the position y (resp., z) from an ancestor x in the data tree and the second-order variables $\bar{Z}$. This logic happens to be equivalent with the natural extension of non-deterministic FMA to trees.

Finally, it is natural to look for effective characterizations of data languages that are both recognizable by non-deterministic FMA and definable in (unrestricted) FO. However, it is known that such characterization cannot be achieved: in [1] it has been shown that the problem of determining whether a language recognized by a non-deterministic FMA is definable in FO is undecidable. The problem of characterizing FO within the class of languages recognizable by deterministic FMA is still open.

7 Conclusion and Future Work

We have shown that the algebraic notion of orbit finite data monoid corresponds to a variant of the logic MSO which is – and this is of course subjective – natural. It is natural in the sense that it only relies on a single and understandable principle: guarding data equality tests by rigidly definable relations.

We believe that this notion of guard is interesting by itself. Of course, it is not the first time that guards are used to recover some decidability properties from a too expressive logic. What is more original in the present context is the equivalence with the algebraic object, which shows that this approach is in some sense maximal: it is not just a particular technique among others for having decidability, but it is sufficient for completely capturing the expressiveness of the very natural algebraic model.

Another contribution of the present work is the development of the structural understanding of orbit finite data monoids. By structural understanding, we refer to Green's relations. These relations form a major tool in most involved proofs concerning finite monoids. The corresponding study of Green's relations for orbit finite data monoids was already a major argument in the proof of [3], and it had to be developed even further in the present work.

Finally, we proved that a variant of the same logic captures the larger class of data languages recognized by non-deterministic NFA.

We are only at the beginning of understanding the various notions of recognizability for data languages. However, several interesting questions were raised during our study. Some of them concern the fine structure of the logic:

> *The nesting level of guards seems to be a robust and relevant parameter in our logic. Can we understand it algebraically? Can we decide it?*

Other questions concern the more general model of FMA:

> *Can we characterize among the languages recognized by deterministic FMA the ones recognizable by orbit finite data monoids? Can we give a logic equivalent to deterministic FMA and characterize its FO fragment?*

Acknowledgments. We would like to thank Michael Benedikt and Anca Muscholl for the many helpful remarks on the paper.

References

[1] Benedikt, M., Ley, C., Puppis, G.: Automata vs. Logics on data words. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 110–124. Springer, Heidelberg (2010)

[2] Benedikt, M., Ley, C., Puppis, G.: What you must remember when processing data words. In: Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW). CEUR Workshop Proceedings. CEUR-WS.org, vol. 619 (2010)

[3] Bojańczyk, M.: Data monoids. In: Proceedings of the 28th Annual Symposium on Theoretical Aspects of Computer Science (STACS). LIPIcs, vol. 9, pp. 105–116. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)

[4] Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: Proceedings of the 21th IEEE Symposium on Logic in Computer Science LICS, pp. 7–16. IEEE Computer Society, Los Alamitos (2006)

[5] Büchi, R.J.: Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6(1-6), 66–92 (1960)

[6] Gabbay, M., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2002)

[7] Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134(2), 329–363 (1994)

[8] Kaminski, M., Tan, T.: Tree automata over infinite alphabets. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 386–423. Springer, Heidelberg (2008)

[9] McNaughton, R., Papert, S.: Counter-free Automata. M.I.T. Research Monograph. Elsevier MIT Press (1971)

[10] Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic 5(3), 403–435 (2004)
[11] Pin, J.E.: Mathematical foundations of automata theory (2010), http://www.liafa.jussieu.fr/~jep/MPRI/MPRI.html
[12] Schwentick, T.: Automata for XML - a survey. Journal of Computer and System Sciences 73(3), 289–315 (2007)
[13] Schützenberger, M.P.: On finite monoids having only trivial subgroups. Information and Control 8(2), 190–194 (1965)
[14] Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Basel (1994)

An Elementary Proof of a $3n - o(n)$ Lower Bound on the Circuit Complexity of Affine Dispersers*

Evgeny Demenkov[1] and Alexander S. Kulikov[2]

[1] St. Petersburg State University
[2] St. Petersburg Department of Steklov Institute of Mathematics

Abstract. A Boolean function $f: \mathbb{F}_2^n \to \mathbb{F}_2$ is called an affine disperser of dimension d, if f is not constant on any affine subspace of $\mathbb{F}_2^n$ of dimension at least d. Recently Ben-Sasson and Kopparty gave an explicit construction of an affine disperser for sublinear d. The main motivation for studying such functions comes from extracting randomness from structured sources of imperfect randomness. In this paper, we show another application: we give a very simple proof of a $3n-o(n)$ lower bound on the circuit complexity (over the full binary basis) of affine dispersers for sublinear dimension. The same lower bound $3n-o(n)$ (but for a completely different function) was given by Blum in 1984 and is still the best known.

The main technique is to substitute variables by linear functions. This way the function is restricted to an affine subspace of $\mathbb{F}_2^n$. An affine disperser for sublinear dimension then guarantees that one can make $n - o(n)$ such substitutions before the function degenerates. It remains to show that each such substitution eliminates at least 3 gates from a circuit.

1 Introduction

Proving lower bounds on the circuit complexity of explicitly defined Boolean functions is one of the most famous and difficult problems in theoretical computer science. Already in 1949 Shannon [1] showed by a counting argument that almost all Boolean functions have circuits of size $\Omega(2^n/n)$ only. Still, we have no example of an explicit function requiring super-linear circuit size. Moreover, only a few proofs of linear lower bounds are known. We review some of them is Sect. 3. The best lower bound $3n-o(n)$ for the basis B_2 was proved by Blum in 1984 [2], the current record lower bound $5n - o(n)$ for the basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$ was given in 2002 by Iwama, Lachish, Morizumi, and Raz [3].

* Research is partially supported by Federal Target Program "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009–2013, Russian Foundation for Basic Research, RAS Program for Fundamental Research, Grant of the President of Russian Federation (NSh-5282.2010.1), and PDMI Computer Science Club scholarship.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 256–265, 2011.

All bounds mentioned above are proved by the gate elimination method. The main idea of this method is the following. One considers a Boolean function on n variables from a certain class of functions and shows that for any circuit computing this function setting some variables to constants yields a sub-function of the same type and eliminates several gates. Usually, a gate is eliminated just because one of its inputs becomes a constant. By induction, one concludes that the original circuit must have many gates. Though this method is essentially the only known method for proving non-trivial lower bounds for general circuit complexity, as many authors note it is unlikely that it will allow to prove non-linear bounds.

In this paper, we prove a $3n - O(d)$ lower bound on the circuit complexity of a Boolean function $f: \mathbb{F}_2^n \to \mathbb{F}_2$ that is not constant on any affine subspace of $\mathbb{F}_2^n$ of dimension at least d. Such functions are called affine dispersers for dimension d. The proof of a lower bound is much simpler than the proof of the currently strongest lower bound $3n - o(n)$ given by Blum in 1984 [2]. However, it is not easy to construct explicitly an affine disperser for small d. Only recently Ben-Sasson and Kopparty [4] presented a construction for sublinear $d = o(n)$ (namely, $d = \Theta(n^{4/5})$).

The main idea of the proof is as follows. Consider an affine disperser f for dimension d. We know that f is not constant on any affine subspace of $\mathbb{F}_2^n$ of dimension at least d. Hence for any $I_1, \ldots, I_{n-d} \subseteq \{1, \ldots, n\}$ and $c_1, \ldots, c_{n-d} \in \mathbb{F}_2$, f is not constant on affine subspace

$$\{x \in \mathbb{F}_2^n \mid \bigoplus_{i \in I_k} x_i = c_k, \text{ for all } 1 \le k \le n - d\}$$

of dimension at least d. We consequently find substitutions of the form $x_{i_k} = \bigoplus_{i \in I_k} x_i \oplus c_k$ so that at least 3 gates are eliminated under each of them from the current circuit. This way we eliminate at least $3n - O(d)$ gates.

To find a substitution under which at least 3 gates are eliminated we just take the topologically first non-linear gate R (i.e., a gate that does not compute a function of the form $\bigoplus_{i \in I} x_i \oplus c$, for $I \subseteq \{1, 2, \ldots, n\}$, $c \in \mathbb{F}_2$) of a circuit. Since it is the first such gate, both its inputs P and Q are linear functions. By an appropriate substitution, we make P constant which also makes R constant. This kills P, R and all the successors of R, i.e., at least 3 gates in total. In the example given in Fig. 1, one can make a substitution $x_1 = x_2 \oplus 1$. Then P evaluates to 0, R also evaluates to 0 and T is also eliminated. The formal proof is given in Section 4.

Similar ideas (substituting variables by linear functions) were used by Boyar and Peralta [5] for proving lower bounds on the multiplicative complexity of Boolean functions. The multiplicative complexity of a Boolean function is defined as the smallest number of $\wedge$ gates in a circuit over $\{\wedge, \oplus, 1\}$ computing this function. As with the circuit complexity, it is known [6] that the multiplicative complexity of almost all Boolean functions is about $2^{n/2}$, while the best known lower bound for an explicit function is $n - 1$. As an easy consequence we obtain a lower bound $n - d - 1$ on the multiplicative complexity of an affine disperser for dimension d.

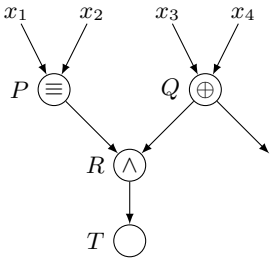


Fig. 1. Substitution $x_1 = x_2 \oplus 1$ eliminates at least 3 gates from this circuit

2 General Setting

By B_n we denote the set of all Boolean functions $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$. A circuit over a basis $\Omega \subseteq B_2$ is a directed acyclic graph with nodes of in-degree 0 or 2. Nodes of in-degree 0 are marked by variables from $\{x_1, \ldots, x_n\}$ and are called inputs. Nodes of in-degree 2 are marked by functions from Ω and are called gates. There is also a special output gate where the result is computed. The size of a circuit is its number of gates. By $C_\Omega(f)$ we denote the minimum size of a circuit over Ω computing f. The two commonly studied bases are B_2 and $U_2 = B_2 \setminus \{\oplus, \equiv\}$.

We call a function $f \in B_2$ degenerate if it does not depend essentially on some of its variables, i.e., there is a variable x_i such that the sub-functions $f|_{x_i=0}$ and $f|_{x_i=1}$ are equal. It is easy to see that a gate computing a degenerate function from B_2 can be easily eliminated from a circuit without increasing its size (when eliminating this gate one may need to change the functions computed at its successors). The set B_2 contains the following sixteen functions $f(x, y)$:

- six degenerate functions: 0, 1, x, $x \oplus 1$, y, $y \oplus 1$;
- eight functions of the form $((x \oplus a)(y \oplus b)) \oplus c$, where $a, b, c \in \mathbb{F}_2$ (throughout all the paper we write xy instead of $x \wedge y$); we call them $\wedge$-type functions;
- two functions of the form $x \oplus y \oplus a$, where $a \in \mathbb{F}_2$; these are called $\oplus$-type functions.

An example on simplifying a circuit is given in Fig. 2. We assign x_2 the value 1. Then Q computes the constant 1, so P and hence also R compute $x_1 \oplus 1$. These 3 gates can be eliminated from the circuit. After that S computes $(x_1 \oplus 1) \oplus x_4$, i.e., $x_1 \equiv x_4$, while T computes $(x_1 \oplus 1)S$. The negation sign on the wire from x_1 to T is intended to reflect the fact that the binary function computed at T is not just xy as in the picture, but $(x \oplus 1)y$.

Below we state several simple but important facts illustrated in this example.

- The substitution $x_2 = 1$ trivializes the gate Q (i.e., makes it constant), so not only Q is eliminated, but also all its successors. At the same time, P is not trivialized, but becomes a degenerate function. This illustrates the difference between $\oplus$- and $\wedge$-type gates and explains why currently best lower bounds for circuits over U_2 are stronger than those for circuits over B_2.

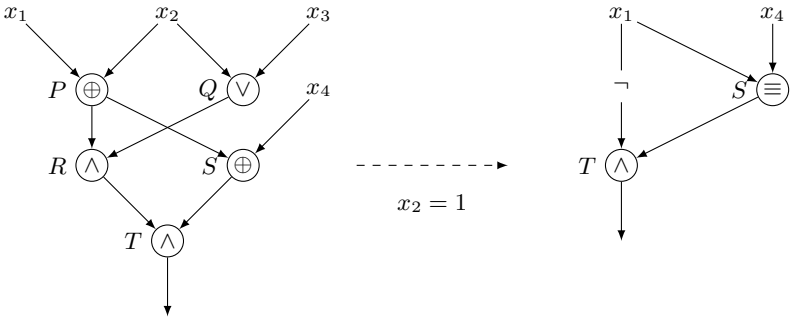


Fig. 2. Example of simplifying a circuit under a substitution $x_2 = 1$

– While simplifying a circuit under a substitution one may need to change the functions computed at gates.
– The resulting circuit depends on neither x_2 nor x_3, though only x_2 was substituted.

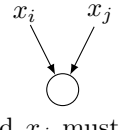
3 Known Lower Bounds

Below we review some of the known lower bounds on circuit complexity and in each case indicate a property of a Boolean function that is important for the proof. We concentrate on the circuit size, while there are many other models such as formulas, branching programs, monotone circuits, constant-depth circuits, where functions with other interesting properties are needed. Note that apart from the properties described below, each function for which one would like to prove a lower bound by the gate elimination method must also satisfy the following natural property: it must remain essentially the same after replacing a variable by a constant.

– Bounds on C_{B_2}
  • Schnorr [7] proved a $2n - c$ lower bound on C_{B_2} for a function satisfying the following property: for any two different input variables x_i and x_j, there are at least three different sub-functions among

$$f|_{x_i=0,x_j=0}, f|_{x_i=1,x_j=0}, f|_{x_i=0,x_j=1}, f|_{x_i=1,x_j=1}\,.$$

  This property is needed to argue that a top of a circuit cannot look like this:



  That is, at least one of x_i and x_j must feed also some other gate (as otherwise one would get at most two different subfunctions w.r.t. x_i and x_j). One then assigns a constant to this variable and kills two gates. A $2n - c$ lower bound follows by induction.

There are many natural functions satisfying this property. E.g., $\mathrm{MOD}^n_{m,r}$, for $m \geq 3$, defined as follows:

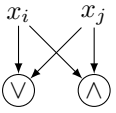
$$\mathrm{MOD}^n_{m,r}(x_1, \ldots, x_n) = 1 \text{ iff } \sum_{i=1}^{n} x_i \equiv r \pmod{m}.$$

- Paul [8] proves a $2n - o(n)$ lower bound on C_{B_2} for the storage access function: for $a \in \mathbb{F}_2^{\log n}$ and $x \in \mathbb{F}_2^n$, $f(a, x) = x_{\bar{a}}$, where $\bar{a}$ is the number from $\{0, \ldots, n-1\}$ whose binary representation is a, and $x_{\bar{a}}$ is the corresponding bit of a. An important property of this function is that for any input variable x_i, when all the bits of a are already assigned, the output of f either equals x_i or does not depend on x_i at all. This allows to substitute x_i not only by a constant, but by an arbitrary function.
- Stockmeyer [9] proved a $2.5n - c$ lower bound on C_{B_2} for many symmetric functions (in particular, for all $\mathrm{MOD}^n_{m,r}$ functions). He essentially uses the fact that for symmetric functions substituting $x_i = h$, $x_j = h \oplus 1$ for a function h is the same as just saying that $x_i \oplus x_j = 1$.
- A function for which Blum [2] proved a $3n - o(n)$ lower bound on C_{B_2} (a similar function was also used by Paul [8]) is defined as follows. Let $a, b, c \in \mathbb{F}_2^{\log n}$, $x \in \mathbb{F}_2^n$, $p, q, r \in \mathbb{F}_2$. Then

$$f(a, b, c, p, q, r, x) = q((x_{\bar{a}} x_{\bar{b}}) \vee (p x_{\bar{b}}(x_{\bar{c}} \oplus r))) \vee (1 \oplus q)(x_{\bar{a}} \oplus x_{\bar{b}}).$$

  For any x_i and x_j, one can get $x_i \oplus x_j$ as well as $x_i x_j$ from f by assigning some of the remaining variables.
- Kojevnikov and Kulikov [10] proved a $7n/3 - c$ lower bound for functions with high multiplicative complexity. Any circuit computing such a function must have several $\wedge$-type gates. This allows to assign different weights to $\oplus$- and $\wedge$-type gates when counting the number of gates that are eliminated.

– Bounds on C_{U_2}
  - Schnorr [7] proved a $3n - c$ lower bound on C_{U_2} for the parity function. A property that helps here is that an optimal circuit cannot contain a variable of out-degree exactly 1. Indeed, if such a variable x_i existed, one could substitute all the other variables to trivialize the unique gate fed by x_i. This would make the function independent of x_i, a contradiction.
  - Zwick [11] proved a $4n - c$ lower bound for all $\mathrm{MOD}^n_{m,r}$ functions, $m \geq 3$. He noticed that any optimal circuit for such a function can contain only a constant number of out-degree 1 variables. This allows to remove such variables from the consideration by using a circuit complexity measure equal to the number of gates minus the number of out-degree 1 variables.
  - Iwama, Lachish, Morizumi, and Raz [3] used Zwick's circuit complexity measure to prove a lower bound $5n - o(n)$ on C_{U_2} for strongly two-dependent functions, i.e., functions satisfying the following property: for any two variables all the four sub-functions resulting by fixing the values of these variables are different. This property guarantees that a top of a circuit cannot look like this:



This case is the main bottleneck in Zwick's proof. An explicit construction of a strongly two-dependent function was previously given by Savicky and Zak [12]. In fact, this function is even k-mixed, for $k = n - o(n)$: for any subset of k variables, all the 2^k sub-functions w.r.t. these k variables are different. Recently, Amano and Tarui [13] showed that this property is not enough for proving stronger than $5n$ lower bounds on C_{U_2} by constructing a function of circuit complexity $5n + o(n)$ that is k-mixed, for $k = n - o(n)$.

4 A $3n - o(n)$ Lower Bound

In this section we consider only circuits over B_2. Let $\mu(\mathcal{C}) = s(\mathcal{C}) + N(\mathcal{C})$, where $s(\mathcal{C})$ is the size (number of gates) of $\mathcal{C}$ and $N(\mathcal{C})$ is the number of input variables of $\mathcal{C}$ with out-degree at least 1.

Lemma 1. *Let P be a gate of a circuit $\mathcal{C}$ that is a $\oplus$-type gate that depends only on $\oplus$-type gates of out-degree 1 and variables. Then there is a variable x_j and a (possibly empty) subset of variables $I \subseteq \{1, \ldots, n\} \setminus \{j\}$ such that for any constant $c \in \mathbb{F}_2$, the substitution $x_j = \bigoplus_{i \in I} x_i \oplus c$ makes the gate P constant and reduces $N(\mathcal{C})$ at least by 1.*

Proof. Clearly P computes a function $\bigoplus_{i \in I} x_i \oplus x_j \oplus c_0$ for some $1 \leq j \leq n$, $I \subseteq \{1, \ldots n\} \setminus \{j\}$, $c_0 \in \mathbb{F}_2$. We analyse the effect of reducing $\mathcal{C}$ under the substitution $x_j = \bigoplus_{i \in I} x_i \oplus c$, for $c \in \mathbb{F}_2$. Let S be the set of gates that P depends on. Clearly S contains at least $|I| - 1$ gates (as $\bigoplus_{i \in I} x_i \oplus c_0$ cannot be computed by less than $|I| - 1$ gates). To simplify $\mathcal{C}$ under the substitution $x_j = \bigoplus_{i \in I} x_i \oplus c$, we eliminate the gate P (as it now computes the constant $c \oplus c_0$) and all its successors (as they now compute degenerate functions).

To reduce $N(\mathcal{C})$ by 1, we need to replace x_j by $\bigoplus_{i \in I} x_i \oplus c$. For this, we eliminate all the gates from S (they were needed only for computing P) and add $|I| - 1$ gates computing $\bigoplus_{i \in I} x_i$. We then use them instead of x_j. Clearly, the resulting circuit outputs the same as the initial circuit for all $x \in \mathbb{F}_2^n$ such that $x_j = \bigoplus_{i \in I} x_i \oplus c$. □

An example of such simplification is given in Fig. 3 ($I = \{2, 3, 4, 5\}$, $j = 1$).

Theorem 1. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be an affine disperser for dimension d, A be an affine subspace of $\mathbb{F}_2^n$ of dimension D, and $\mathcal{C}$ be a circuit with n inputs such that $\forall x \in A, \mathcal{C}(x) = f(x)$. Then*

$$\mu(\mathcal{C}) \geq 4(D - d)\,.$$

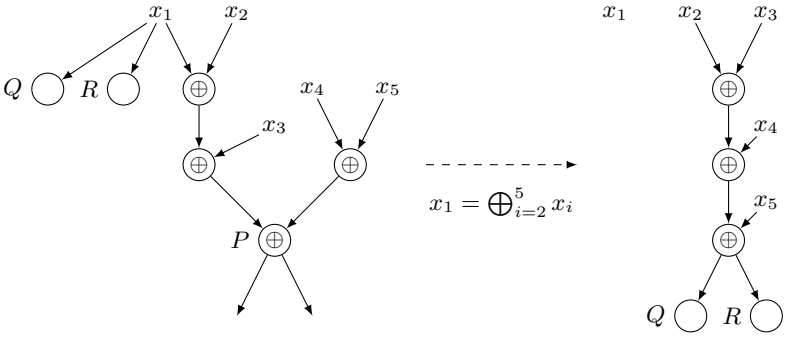


Fig. 3. Example of a linear substitution

Proof. We prove the inequality by induction on D. The base case $D \leq d$ is trivial. Consider now the case $D \geq d+1$. Take a circuit $\mathcal{C}$ computing f on A with the minimal possible $\mu(\mathcal{C})$. Assume wlog that $\mathcal{C}$ does not contain degenerate gates (all such gates can be eliminated without increasing $\mu(\mathcal{C})$). Note also that $\mathcal{C}$ cannot compute a linear function. Indeed, if $\mathcal{C}$ computed a function of the form $\bigoplus_{i\in I} x_i \oplus c$, then f would be constant on an affine space $A' = \{x \in A : \bigoplus_{i\in I} x_i = c\}$ of dimension at least $D - 1 \geq d$. Thus, $\mathcal{C}$ contains at least one $\wedge$-type gate.

In the following we find a substitution of the form $\bigoplus_{i\in I} x_i \oplus c$ under which $\mathcal{C}$ is reduced to $\mathcal{C}'$ such that $\mu(\mathcal{C}) \geq \mu(\mathcal{C}') + 4$ and $\mathcal{C}(x) = \mathcal{C}'(x)$ for all $x \in A' = \{x \in A : \bigoplus_{i\in I} x_i = c\}$. Since A' has dimension at least $D-1$, we conclude by induction that $\mu(\mathcal{C}) \geq 4(D-1-d)+4 = 4(D-d)$. Note that any gate that becomes constant under such substitution cannot be an output gate, as otherwise $\mathcal{C}$ would compute a linear function.

Consider a topological order on all the gates of $\mathcal{C}$ and let P be the first gate in this order that is not a $\oplus$-type gate of out-degree 1. Since it depends only on $\oplus$-type gates and input variables, functions computed at both inputs of P are of the form $\bigoplus_{i\in I_1} x_i \oplus c_1$ and $\bigoplus_{i\in I_2} x_i \oplus c_2$. Below we consider five cases, Fig. 4 shows all of them.

- **Case 1.** P is a $\oplus$-type gate of out-degree at least 2. Then it clearly computes a function of the form $\bigoplus_{i\in I} x_i \oplus c$. By the lemma above, by making P constant we reduce μ at least by 4.
- **Case 2.** P is an $\wedge$-type gate.
  - **Case 2.1.** One of the inputs of P is a gate Q. Then Q is a $\oplus$-type gate. By making Q the constant (as in the lemma) which trivializes P we kill P, Q, and all the successors of P. Also, $N(\mathcal{C})$ is reduced at least by 1, hence μ is reduced at least by 4.
  - **Case 2.2.** Both inputs of P are variables x_i and x_j and at least one of them (say, x_i) have out-degree at least 2. By assigning x_i the constant which trivializes P we kill all the successors of x_i and all the successors of P. Clearly $N(\mathcal{C})$ is reduced at least by 1. By considering two sub-cases we show that $s(\mathcal{C})$ is reduced at least by 3.

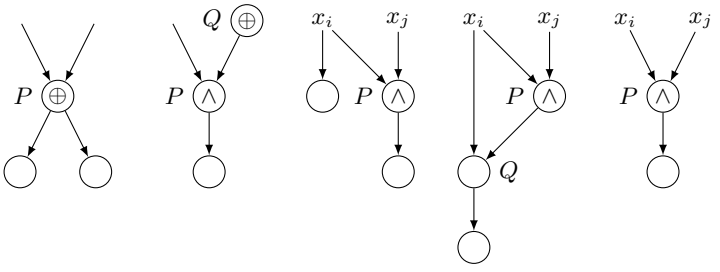


Fig. 4. All cases of the proof

  - **Case 2.2.1.** x_i has a successor that is not fed by P. Then this successor is eliminated as well as P and all the successors of P.
  - **Case 2.2.2.** The only successor of P is the gate Q and it is also fed by x_i. Then clearly it becomes constant (as both its inputs are constants) and so all its successors are also eliminated.
- **Case 2.3.** Both inputs of P are out-degree 1 variables x_i and x_j. By assigning x_i the constant which trivializes P, we eliminate P and all its successors and reduce $N(\mathcal{C})$ at least by 2. Hence μ is reduced at least by 4. □

Corollary 1. *Any circuit over B_2 computing an affine disperser for dimension d has at least $3n - 4d$ gates.*

Proof. Indeed, by Theorem 1, for any circuit $\mathcal{C}$ computing an affine disperser for dimension d,

$$s(\mathcal{C}) = \mu(\mathcal{C}) - N(\mathcal{C}) \geq 4(n-d) - N(\mathcal{C}) \geq 3n - 4d\,. \qquad \square$$

Thus, an affine disperser for sublinear dimension requires circuits of size at least $3n - o(n)$. It is also easy to see that by the same method one can prove a lower bound $n - o(n)$ on the multiplicative complexity of affine dispersers for sublinear dimension. For this, we just make $n - o(n)$ linear substitutions each time killing the first $\wedge$-type gate.

5 Further Directions

1. It would be interesting to improve the presented lower bound by a more involved case analysis or to find another property of Boolean functions implying a stronger than $3n$ lower bound.
2. Another interesting direction is to prove a non-trivial upper bound for an affine disperser.

3. An (apparently) easier problem is to close one of the following gaps (see [9], [14], [11]):

$$2.5n - c \leq C_{B_2}(\mathrm{MOD}_3^n) \leq 3n + c,$$
$$4n - c \leq C_{U_2}(\mathrm{MOD}_4^n) \leq 5n + c\,.$$

   Also it is still not known whether $C(\mathrm{MOD}_p^n)$ is strictly greater than $C(\mathrm{MOD}_q^n)$ for primes $p > q$. Note however that any symmetric function can be computed by a circuit (over B_2) of size $4.5n + o(n)$ [14].
4. It would also be interesting to find a Boolean function of multiplicative complexity at least cn, for a constant $c > 1$.

Acknowledgements. We would like to thank Edward A. Hirsch and Arist Kojevnikov as well as the anonymous referees for helpful comments and Arnab Bhattacharyya for pointing us out the paper [4].

References

1. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell System Technical Journal 28, 59–98 (1949)
2. Blum, N.: A Boolean function requiring $3n$ network size. Theoretical Computer Science 28, 337–345 (1984)
3. Iwama, K., Lachish, O., Morizumi, H., Raz, R.: An explicit lower bound of $5n-o(n)$ for boolean circuits (2005) (unpublished manuscript), http://www.wisdom.weizmann.ac.il/~ranraz/publications/Podedl.ps
4. Ben-Sasson, E., Kopparty, S.: Affine dispersers from subspace polynomials. In: Proceedings of the Annual Symposium on Theory of Computing (STOC), vol. 679, pp. 65–74. ACM Press, New York (2009)
5. Boyar, J., Peralta, R.: Tight bounds for the multiplicative complexity of symmetric functions. Theoretical Computer Science 396, 223–246 (2008)
6. Boyar, J., Peralta, R., Pochuev, D.: On The Multiplicative Complexity of Boolean Functions over the Basis $(\wedge, \oplus, 1)$. Theoretical Computer Science 235(1), 1–16 (2000)
7. Schnorr, C.P.: Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. Computing 13, 155–171 (1974)
8. Paul, W.J.: A $2.5n$-lower bound on the combinational complexity of Boolean functions. SIAM Journal of Computing 6(3), 427–433 (1977)
9. Stockmeyer, L.J.: On the combinational complexity of certain symmetric Boolean functions. Mathematical Systems Theory 10, 323–336 (1977)
10. Kojevnikov, A., Kulikov, A.S.: Circuit Complexity and Multiplicative Complexity of Boolean Functions. In: Ferreira, F., Löwe, B., Mayordomo, E., Mendes Gomes, L. (eds.) CiE 2010. LNCS, vol. 6158, pp. 239–245. Springer, Heidelberg (2010)
11. Zwick, U.: A $4n$ lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic Boolean functions. SIAM Journal on Computing 20, 499–505 (1991)

12. Savicky, P., Zak, S.: A large lower bound for 1-branching programs. Technical Report TR96-036, ECCC (1996)
13. Amano, K., Tarui, J.: A well-mixed function with circuit complexity $5n \pm o(n)$: Tightness of the lachish-raz-type bounds. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 342–350. Springer, Heidelberg (2008)
14. Demenkov, E., Kojevnikov, A., Kulikov, A.S., Yaroslavtsev, G.: New upper bounds on the Boolean circuit complexity of symmetric functions. Information Processing Letters 110(7), 264–267 (2010)

On the Complexity of the *l*-diversity Problem

Riccardo Dondi[1], Giancarlo Mauri[2], and Italo Zoppis[2]

[1] Dipartimento di Scienze dei Linguaggi, della Comunicazione e degli Studi Culturali, Università degli Studi di Bergamo, Bergamo, Italy
[2] DISCo, Università degli Studi di Milano-Bicocca, Milano, Italy
riccardo.dondi@unibg.it, {mauri,zoppis}@disco.unimib.it

Abstract. The problem of publishing personal data without giving up privacy is becoming increasingly important. Different interesting formalizations have been recently proposed in this context, i.e. *k-anonymity* [17,18] and *l-diversity* [12]. These approaches require that the rows in a table are clustered in sets satisfying some constraint, in order to prevent the identification of the individuals the rows belong to. In this paper we focus on the *l-diversity* problem, where the possible attributes are distinguished in *sensible* attributes and *quasi-identifier* attributes. The goal is to partition the set of rows, where for each set C of the partition it is required that the number of rows having a specific value in the sensible attribute is at most $\frac{1}{l}\ |C|$.

We investigate the approximation and parameterized complexity of *l-diversity*. Concerning the approximation complexity, we prove the following results: (1) the problem is not approximable within factor $c \ln l$, for some constant $c > 0$, even if the input table consists of two columns; (ii) the problem is APX-hard, even if $l = 4$ and the input table contains exactly 3 columns; (iii) the problem admits an approximation algorithm of factor m (where $m + 1$ is the number of columns in the input table), when the sensitive attribute ranges over an alphabet of constant size. Concerning the parameterized complexity, we prove the following results: (i) the problem is W[1]-hard even if parameterized by the size of the solution, l, and the size of the alphabet; (ii) the problem admits a fixed-parameter algorithm when both the maximum number of different values in a column and the number of columns are parameters.

1 Introduction

In recent years the topic of releasing personal data to the public without giving up privacy has been widely investigated. In different contexts such as epidemic analysis, it is fundamental to publish data avoiding the identification of the individuals the data belong to. One of the approaches that have been proposed in literature to deal with the problem, is the abstraction model [17]. In the abstraction model, some of the data are suppressed or generalized in order to preserve data integrity.

Two of the most relevant approaches based on the abstraction model are *k-anonymity* [18] and *l-diversity* [12]. Given a table consisting of n rows and m

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 266–277, 2011.

columns, *k-anonymity* asks for a partition of the rows in sets, each of size at least k. Then, some of the entries of the rows in the same set are suppressed, i.e. substituted with the value $*$, so that all the rows in a set become identical. Hence it is not possible to distinguish the rows inside a set, and this should prevent to identify which row is related to an individual.

The complexity of the *k-anonymity* problem has been deeply investigated. The problem is known to be NP-hard [13] and APX-hard [1], even when $k = 3$ and the rows are over a binary alphabet [4,6], while it admits a polynomial time algorithm when $k = 2$ [4]. Several approximation algorithms have been proposed for *k-anonymity*, where the approximation factors are functions of k. Namely, an $O(k \log k)$-approximation algorithm has been given in [13], an $O(k)$-approximation algorithm has been given in [1] and an $O(\log k)$-approximation algorithm has been given in [15,9]. The parameterized complexity of *k-anonymity* have been investigated, under several natural parameterizations [5,8].

A different model proposed to deal with some deficiencies of *k-anonymity*, is *l-diversity* [12]. Given a table consisting of n input rows and $m + 1$ columns, *l-diversity* distinguishes between the values contained in one column (referred in the following as sensitive column) and the values contained in the remaining columns (referred in the following as quasi-identifier columns) [1]. A sensitive attribute regards an individual's privacy, as is the case of a disease, and must be kept secret. Quasi-identifier attributes, like age or gender, can be combined with other data to infer an individual's identity, but they are not directly related to an individual's privacy.

The *l-diversity* problem requires that the rows in the input table are clustered in sets satisfying the l-diverse property, that is for each set C of the partition, it is required that the number of rows having a specific value σ in the sensible column is at most a fraction $\frac{1}{l}|C|$. As in the case of *k-anonymity*, some of the entries of quasi-identifier columns are suppressed, so that all the rows in C become identical in the quasi-identifier columns. Hence, it is hard to identify which row is related to an individual (although an adversary may be able to identify rows using background knowledge). In *k-anonymity*, it can happen that all the rows in a set have the same value in a sensitive column, hence in some cases this value may be related to an individual [12]. To avoid this drawback, *l-diversity* requires that each set of the solution must be l-diverse.

A related problem that has been recently introduced, is clustering with diversity [11]. This problem asks for a partions of the input rows in l-diverse clusters, with the goal of minimizing the maximum radius of any cluster. Clustering with diversity admits a 2-approximation algorithm [11], and it is known to be not approximable within a factor less than 2, unless $P = NP$ [11].

The *l-diversity* problem is known to be NP-hard [20], when $l = 3$, the sensitive column is over a ternary alphabet and the quasi-identifier columns are over a binary alphabet [4], while it is in P when $l = 2$ [20]. The only non-trivial

[1] *l-diversity* can be generalized to contain multiple sensitive columns; however, in this paper, following the approach of [20], we assume that there is exactly one sensitive column in the input table.

approximation algorithm for *l-diversity* is due to Xiao et al. [20] and achieves an lm approximation factor.

In this paper, we investigate different aspects of the complexity of *l-diversity* problem. First, we investigate the approximation complexity of the problem. We show that *l-diversity* is not approximable within factor $c \ln l$, for some constant $c > 0$, even if the input matrix consists of two columns (a sensible column and a quasi-identifier column). Furthermore, we show that the problem is APX-hard even if $l = 4$ and the input table consists of exactly 3 columns (a sensible column and two quasi-identifier columns). Finally, we give an approximation algorithm of factor m, when the sensitive column ranges over an alphabet of constant size.

Then, we investigate the parameterized complexity of *l-diversity*. We show that the problem is W[1]-hard, when parameterized by the size of the solution, l, and the size of the alphabet, while it is fixed-parameter tractable when both the maximum number of different values in a column and the number of columns are parameters. Some of the proofs are omitted due to space limitation.

2 Preliminary Definitions

In this section we introduce some preliminary definitions that will be used in the rest of the paper. Given a graph $G = (V, E)$ and a vertex $v \in V$, we denote by $N(v)$ the set of vertices adjacent to v in G. A graph $G = (V, E)$ is cubic $N(v) = 3$ for each $v \in V$.

Given an alphabet Σ, a row r over Σ of length m is a vector of m elements taken from the set Σ. The j-th element of r is denoted by $r[j]$. The *l-diversity* problem distinguishes between two kinds of attributes: *sensitive* and *quasi-identifier*. In what follows we represent an instance R of *l-diversity* as a table (or equivalently a set of rows) consisting of n rows and $m + 1$ columns. A column can be either a q-column or a s-column, if it is referred to a quasi-identifier attribute or a sensitive attribute respectively. An instance of *l-diversity* contains exactly one s-column (in the following we assume that it is associated with index 0) and m q-columns. We denote by Σ_s the alphabet associated with the s-column (that is the rows in the s-column range over Σ_s), and by Σ_q the alphabet of the q-columns. We define $\Sigma = \Sigma_s \cup \Sigma_s$, and we denote by $|\Sigma_{max}|$ the maximum number of different values in a column of R.

Given an instance R of *l-diversity*, and a row $r \in R$ over Σ, a *suppression* of the i-th entry in r, $1 \leq i \leq m$, consists of replacing the value $r[i]$ with the value $*$, with $* \notin \Sigma$. Notice that a suppression cannot occur in the s-column. Given a partition $\Pi = (P_1, \ldots, P_t)$ of R, we define the cost of a set P_i, denoted by $c(P_i)$, as $|P_i|\{j : 1 \leq j \leq m, \exists r_1, r_2 \in P_i,\ r_1[j] \neq r_2[j]\}|$, that is the number of entries in q-columns of the rows in P_i that must be suppressed in order to make all such rows identical. The cost of Π, denoted by $c(\Pi)$, is defined as $\sum_{P_i \in \Pi} c(P_i)$.

A *resolution vector* is a vector of length m over alphabet $\Sigma_q \cup \{*\}$. Given a resolution vector v, $Del(v)$ denotes the number of entries of v having value $*$. A resolution vector v is called a *full resolution vector* if $Del(v) = 0$. Given a set of rows P, we define the resolution vector associated with P, denoted by $r(P)$, as

a resolution vector such that: (1) if $r_a[j] = \alpha$, with $1 \leq j \leq m$ and $\alpha \in \Sigma_q$, for each $r_a \in P$, then $r(P)[j] = \alpha$; (2) else $r(P)[j] = *$, with $1 \leq j \leq m$. Given a row $r \in R$ and a resolution vector v, we say that r and v are *compatible*, if $r[i] \neq v[i]$ implies $v[i] = *$. Given a set of rows P and an integer l, P is l-diverse if, for each value $\sigma \in \Sigma_s$, there are at most $\frac{|P|}{l}$ rows in P having value σ in the s-column. A partition $\Pi = (P_1, \ldots, P_t)$ of R is l-diverse if each set P_i, $1 \leq i \leq t$, is l-diverse. We are now able to formally define the *l-diversity* problem:

Problem 1. l-diversity.
Input: *a set R of rows over an alphabet Σ.*
Output: *an l-diverse partition $\Pi = (P_1, \ldots, P_t)$ of R.*
Goal: *to minimize $c(\Pi)$.*

Given a solution $\Pi = (P_1, \ldots, P_t)$ of *l-diversity* over instance R, let $r \in P_j$; then we say that Π induces $Del(r(P_j))$ suppressions in r. Let R be the set of rows input of *l-diversity*, and let $r_1, r_2 \in R$. We define the Hamming distance between r_1 and r_2, denoted as $H(r_1, r_2)$, as the Hamming distance between r_1 and r_2 restricted to the q-columns, i.e. $H(r_1, r_2) = |\{i : 1 \leq i \leq m \wedge r_1[i] \neq r_2[i]\}|$. Notice that, given a partition $\Pi = (P_1, \ldots, P_t)$ of R, $|P_i| \max_{r_1, r_2 \in P_i}\{H(r_1, r_2)\}$ is a lower bound for $c(P_i)$. An important property of *l-diversity* that can be easily derived from Lemma 1 of [20], is the following.

Lemma 1. *[20]. Given disjoint sets S_1, S_2, ..., S_h of rows, if each S_i, $1 \leq i \leq h$ is l-diverse, then $\bigcup_{i=1}^{h} S_i$ is l-diverse.*

In [20] it is introduced a problem related to *l-diversity*, called Tuple Minimization problem. The Tuple Minimization problem stems from a preprocessing phase on the instance R of *l-diversity*. The preprocessing phase computes a partition $\Pi = (P_1, P_2, \ldots, P_k, Z)$, called a *sound partition*, of R. A partition of R is *sound* if each set P_j, $1 \leq j \leq k$, is an l-diverse set of identical (in the q-columns) rows, while $Z = R \setminus (\bigcup_{j=1}^{k} P_j)$. Notice that Z is the only set of Π that may contain not identical (in the q-columns) rows and that may be not l-diverse. The Tuple Minimization problem, given a set R of rows over an alphabet Σ, and a sound partition $\Pi = (P_1, P_2, \ldots, P_k, Z)$ of R, asks for a sound partition $\Pi' = (P'_1, \ldots, P'_t, Z')$ of R such that: (i) $Z' \supseteq Z$; (ii) for each j, with $1 \leq j \leq t$, $P'_j \subseteq P_i$, for some i with $1 \leq i \leq k$; (iii) Π' is l-diverse; (iv) $|Z'|$ is minimized.

Starting from a set of rows R input of *l-diversity*, it is easy to compute in polynomial time a sound partition $\Pi = (P_1, P_2, \ldots, P_k, Z)$ of R (for details see [20]). It is easy to see that given an approximation algorithm A of factor α for Tuple Minimization, then A approximates within factor $m \cdot \alpha$ the *l-diversity* problem [20].

3 Approximation Complexity

In this section, we discuss the approximation complexity of the *l-diversity* problem. First we focus on the inapproximability of the *l-diversity* problem and of

the Tuple Minimization problem. Then, we prove the APX-hardness for a restricted case of *l-diversity* (when $l = 4$), and we give a factor m approximation algorithm, when the size of Σ_s is bounded by a fixed constant.

3.1 Inapproximability of *l-diversity* and Tuple Minimization

In this section we investigate the approximation complexity of the *l-diversity* problem and of Tuple Minimization problem. We show that both problems cannot be approximated within factor $c \ln l$ unless $P = NP$, for some constant $c > 0$, even if the input table consists of two columns.

In order to prove the result, we give a gap-preserving reduction from the Minimum Set Cover problem (MIN SET COVER). We refer the reader to [19] for details on gap-preserving reduction. Given a universe $U = \{u_1, \ldots, u_h\}$ and a collection $\mathcal{C}$ of subsets $S_1, \ldots S_p$ of the universe U, MIN SET COVER, asks for a collection $\mathcal{C}' \subseteq \mathcal{C}$ of minimum size such that every element in U belongs to at least one member of $\mathcal{C}'$. MIN SET COVER is known to be not approximable in polynomial time to within a factor of $d \ln h$, for some constant $d > 0$ [2,16].

Let $(U, \mathcal{C})$ be an instance of MIN SET COVER. Let us define the corresponding instance R of *l-diversity*, where all the rows in R are over two columns, an s-column (of index 0) and a q-column (of index 1). Define $\Sigma_s = \{\lambda_1, \ldots, \lambda_{2h}\}$ as the alphabet associated with the s-column. Define $\Sigma_q = \{x_i : 1 \leq i \leq h\} \cup \{s_i : 1 \leq i \leq p\}$ as the alphabet associated with the s-column.

Let us now define the rows in R:

- for each $u_i \in U$, $1 \leq i \leq h$, there is a row $r_{u,i} \in R$, defined as follows:
  - $r_{u,i}[0] = \lambda_i$; - $r_{u,i}[1] = x_i$;
- for each $S_i \in U$, there is a set $R_{s,i} \subseteq R$ consisting of $h + 1$ rows, where $R_{s,i} = \{r_{s,i,j} : 1 \leq j \leq h + 1\}$; each row $r_{s,i,j} \in R_{s,i}$ is defined as follows:
  - $r_{s,i,j}[0] = \lambda_k$, with $1 \leq j \leq |U| - |S_i|$, where u_k is the j-th element of $U \setminus S_i$;
  - $r_{s,i,j}[0] = \lambda_{h+q}$, with $|U| - |S_i| + 1 \leq j \leq h + 1$ and $q = j - (|U| - |S_i|)$;
  - $r_{s,i,j}[1] = s_i$, with $1 \leq j \leq h + 1$.

Set $l = h + 1$ and define the set $R_U = \bigcup_{u_i \in U} r_{u,i}$. Notice that the set $R = \left(\bigcup_{S_i \in \mathcal{C}} R_{s,i}\right) \cup R_U$. By construction, each set $R_{s,i}$, $1 \leq i \leq p$, is l-diverse, and it is a (maximal) set of identical rows with respect to column 1, while R_U is not l-diverse.

Lemma 2. *Let $(\mathcal{C}, U)$ be an instance of MIN SET COVER and let R be the corresponding instance of* l-diversity*. Given a solution $\mathcal{C}^*$ of MIN SET COVER over instance $(\mathcal{C}, U)$, such that $\mathcal{C}^*$ consists of d sets, there exists a solution of* l-diversity *over instance R of cost at most $d(h + 1) + h$.*

Lemma 3. *Let $(\mathcal{C}, U)$ be an instance of MIN SET COVER and let R be the corresponding instance of* l-diversity*. Given a solution Π of* l-diversity *over instance R such that $c(\Pi) \leq h + d(h + 1)$, there exists a solution of MIN SET COVER over instance $(\mathcal{C}, U)$ consisting of at most d sets.*

Proof. Assume that there is a solution Π of *l-diversity* over instance R such that $c(\Pi) \leq h + d(h+1)$. Notice that we can assume that there exists exactly one set $R' \subseteq R$, such that Π induces some suppressions in the rows of R'.

Let R' be the only set of Π, where for each row of R' the entry in the q-column (of index 1) is suppressed. Notice that $R_U \subseteq R'$, as each row $r_{u,i} \in R_U$, $1 \leq i \leq h$, is the only row of R with $r_{u,i}[1] = x_i$. Since R_U is not l-diverse, we can assume that $R_U \subset R'$, and that there exists at least one row $r_{s,i,j} \in R_{s,i} \cap R'$, $1 \leq j \leq h+1$, for some $R_{s,i}$, $1 \leq i \leq p$.

Consider a set $R_{s,i}$ such that there exists a row $r_{s,i,j} \in R_{s,i}$, for some $1 \leq j \leq h+1$, with the property that $r_{s,i,j} \in R_{s,i} \cap R'$. We claim that Π induces a suppression in each row $r_{s,i,k} \in R_{s,i}$, $1 \leq k \leq h+1$. Indeed, notice that Π induces a suppression in $r_{s,i,j}$, since $r_{s,i,j} \in R'$. Now, consider the rows in $R_{s,i} \setminus \{r_{s,i,j}\}$. Since $|R_{s,i} \setminus \{r_{s,i,j}\}| = l-1$, it follows that $R_{s,i} \setminus \{r_{s,i,j}\}$ is not l-diverse. Furthermore, for each row r in $R \setminus R_{s,i}$, $r[1] \neq s_i$, while for each row $r_{s,i,k} \in R_{s,i}$, $1 \leq k \leq h+1$, it holds $r_{s,i,k}[i] = s_i$. Hence each row in $R_{s,i}$ must be clustered in Π with at least one row at Hamming distance 1 (for the q-column). As a consequence we can assume that if $R' \cap R_{s,i} \neq \emptyset$, then $R_{s,i} \subset R'$.

Now, we show that for each row $r_{u,j} \in R_U$, there exists at least one set $R_{s,i} \subset R'$, such that for each row $r_{s,i,k} \in R_{s,i}$, $1 \leq k \leq h+1$, it holds $r_{s,i,k}[0] \neq r_{u,j}[0]$. Assume that this is not the case. Then, for each $R_{s,i} \subset R'$, there exists at least one row $r_{s,i,k} \in R_{s,i}$, $1 \leq k \leq h+1$, such that $r_{s,i,k}[0] = r_{u,j}[0]$. It follows that R' is not l-diverse. Indeed, let $r_{u,j}[0] = \lambda_j$. Assume that R' contains z sets $R_{s,i}$, with $1 \leq i \leq h+1$. Then there are $z+1$ rows in R' having value λ_j in the s-column, while $|R'| = lz + l - 1$.

As a consequence, for each row in $r_{u,j} \in R_U$, $1 \leq j \leq h$, there exists at least one set $R_{s,i}$, $1 \leq i \leq p$, in R', such that for each row $r_{s,i,k} \in R_{s,i}$, $1 \leq k \leq h+1$, it holds $r_{s,i,k}[0] \neq r_{u,j}[0]$. Then, define $\mathcal{C}' = \{S_i : R_{s,i} \subset R'\}$. It follows that $\mathcal{C}'$ is a solution of MIN SET COVER, that is for each element $u_j \in U$, there is a set $S_i \in \mathcal{C}'$ that covers u_j. Indeed, consider a row $r_{u,j}$ (associated with the element $u_j \in U$) and let $R_{s,i,j}$ (associated with the set $S_i \in \mathcal{C}'$) be one of the set in R' such that for each row $r_{s,i,k} \in R_{s,i}$, it holds $r_{s,i,k}[0] \neq r_{u,j}[0]$. By construction $u_j \in S_i$, hence $\bigcup_{S_j \in \mathcal{C}'} S_j = U$. □

Theorem 1. *The* l-diversity *problem cannot be approximated with factor* $c \ln l$ *unless* $P = NP$*, even if the input table consists of two columns.*

Proof. The MIN SET COVER problem is known to be inapproximable within factor $d \ln h$, for some constant $d > 0$, where $h = |U|$. Given an instance $I = (\mathcal{C}, U)$ of MIN SET COVER, let R be the corresponding instance of *l-diversity*. We denote by $OPT_{MINSETCOVER}(I)$ ($OPT_{l\text{-}diversity}(R)$ respectively) the value of an optimal solution of MIN SET COVER (*l-diversity* respectively), over instance I (R respectively). Let $f : I \to \mathbb{N}$ be a function, we have proved in Lemma 2 that it holds $OPT_{MINSETCOVER}(I) \leq f(I) \Rightarrow OPT_{l\text{-}diversity}(R) \leq f(I)(h+1) + h$ and, by Lemma 3, $OPT_{MINSETCOVER}(I) > d \ln h f(I) \Rightarrow OPT_{l\text{-}diversity}(R) > d \ln h f(I)(h+1) + h$, for some constant $d > 0$. It follows we cannot approximate *l-diversity* within factor $\frac{d \ln h}{2}$, for some constant d, which

implies, since in the reduction $l = h+1$, that *l-diversity* cannot be approximated within factor $c \ln l$, for some constant $c > 0$. □

As a consequence of the previous theorem, the MIN TUPLE problem cannot be approximated within factor $c \ln l$, for some constant $c > 0$, unless P=NP.

3.2 APX-Hardness of *l-diversity* with $l = 4$ and $m = 3$

In this section we investigate the inapproximability of *l-diversity* when l is a fixed constant, namely $l = 4$, and the set R of rows is over one s-column and two q-columns. We denote this restriction by $4 - diversity(3)$. We prove that $4-diversity(3)$ is APX-hard, giving an L-reduction from Minimum Vertex Cover on Cubic graphs (MVCC). We recall that, given a cubic graph $G = (V, E)$, with $V = \{v_1, \ldots, v_h\}$ and $|E| = k$, MVCC asks for a minimum cardinality set $V' \subseteq V$, such that for each $(v_i, v_j) \in E$, at least one of $v_i, v_j \in V'$. MVCC is known to be APX-hard [3].

Given a cubic graph $G = (V, E)$, we define an instance R_G of $4-diversity(3)$, consisting of a set R_G of rows over three columns, one s-column (the column of index 0) and two q-columns (columns of indices $1, 2$). Now, let us define the instance R_G of $4 - diversity(3)$.

For each vertex $v_i \in V$, the instance R_G has a set $R_G(v_i) = \{r_x(v_i), 1 \leq x \leq 5\}$. The rows $r_x(v_i)$, $1 \leq x \leq 5$, are defined as follows:

- $r_x(v_i)[0] = s_{i,x}$; - $r_x(v_i)[1] = r_x(v_i)[2] = p_i$.

For each edge $\{v_i, v_i\} \in E$ (assume $i < j$), R_G contains a set $R_G(e_{i,j}) = \{r_x(e_{i,j}), 1 \leq x \leq 5\}$. The rows $r_x(e_{i,j})$, $1 \leq x \leq 5$, are defined as follows:

- $r_x(e_{i,j})[0] = s_{i,j,x}$, for $1 \leq x \leq 3$;
- $r_x(e_{i,j})[0] = s_{i,j}$, for $4 \leq x \leq 5$;
- $r_x(e_{i,j})[1] = p_i$, for $1 \leq x \leq 3$;
- $r_x(e_{i,j})[1] = p_i$, for $x = 4$;
- $r_x(e_{i,j})[1] = p_j$, for $x = 5$;
- $r_x(e_{i,j})[2] = p_{i,j}$, for $1 \leq x \leq 5$.

Finally R_G contains a set $Z = \{z_1, z_2, z_3, z_4\}$ of four rows, where $z_i[0] = x_i$, with $1 \leq i \leq 4$, and $z_i[1] = z_i[2] = q_i$, with $1 \leq i \leq 4$. Now, in the next lemmata we show the relation between a vertex cover of G and a solution of $4 - diversity(3)$ on the corresponding instance R_G.

Lemma 4. *Let $G = (V, E)$ be a cubic graph input of MVCC and let R_G be the corresponding instance of $4 - diversity(3)$. Then, starting from a cover V' of G, we can compute in polynomial time a solution Π of $4 - diversity(3)$ over instance R_G such that $c(\Pi) \leq 6k + 2|V'| - 2|V|$.*

Lemma 5. *Let R_G be an instance of $4 - diversity(3)$ corresponding to a cubic graph $G = (V, E)$, input of MVCC. Then, starting from a solution Π of $4 - diversity(3)$ such that $c(\Pi) \leq 6k + 2p - 2|V|$, we can compute in polynomial time a cover V' of G, such that $|V'| \leq p$.*

Proof. (*Sketch*) Let us consider a solution Π of $4 - diversity(3)$ over input R_G. By construction, we can assume that Π contains exactly one set (denoted as S_z), such that Π induces two suppressions in each row of S_z. Notice that $Z \subseteq S_z$.

We say that Π contains an *I-set for* $R_G(v_i)$ if there exists a set X of Π such that X contains one row of each of the sets $R_G(e_{i,j})$ (in this case $r_4(e_{i,j})$), $R_G(e_{i,h})$, $R_G(e_{i,k})$ and $R(v_i)$, and Π induces exactly one suppression for each row in X (a suppression in the q-column of index 2). Now, starting from solution Π, we can compute in polynomial time a solution Π', such that: (i) $c(\Pi') \leq c(\Pi)$; (ii) Π' induces 5 suppressions for the rows in $R_G(e_{i,j})$ if and only if Π' contains an I-set for for $R_G(v_i)$ or $R_G(v_j)$; (iii) the set of vertices $v_i \in V$, such that Π' induces an I-set for $R_G(v_i)$, is an independent set of G.

For each set $R_G(v_i)$ such that Π' does not include an I-set for $R_G(v_i)$, we can assume that $R_G(v_i)$ is a set of Π', as in this case no suppression is induced by Π' in rows of $R_G(v_i)$. Furthermore, for each set $R_G(e_{i,j})$ such that Π' has neither an I-set for $R_G(v_i)$ nor $R_G(v_j)$, then Π' induces at least 6 suppressions for the rows in $R_G(e_{i,j})$. Hence, we can assume that $R_G(e_{i,j}) \setminus \{r_5(e_{i,j})\}$ is a set of Π', while $r_5(e_{i,j}) \in S_Z$, as in this case Π' induces exactly 6 suppressions in the rows of $R_G(e_{i,j})$.

Now, consider a solution Π' of $4 - diversity(3)$ over instance R_G. Let us define a vertex cover V' of G as follows: for each set $R_G(v_i)$, such that the overall number of suppressions for the rows in $R_G(v_i)$ is 1, let v_i be in $V \setminus V'$, else v_i is in the cover V' of G. It is easy to see that, if $c(\Pi') = 6k + 2q - 2|V|$, there are exactly $|V| - q$ sets $R_G(v_i)$ such that Π' has an I-set for $R_G(v_i)$. Since the set of vertices v_i corresponding to I-sets of Π' is an independent set of G, it follows that $|V'|$ is a cover of G, and $|V'| = q \leq p$. □

As a direct consequence of Lemmata 4 and 5, it follows that $4 - diversity(3)$ is APX-hard.

3.3 An Approximation Algorithm for Bounded $|\Sigma_s|$

In this section we present an approximation algorithm of factor m for the *l-diversity* problem, when $|\Sigma_s|$ is a fixed constant, of time complexity $O(n^{2|\Sigma_s|+1})$. Notice that the *l-diversity* is NP-hard when $|\Sigma_s|$ is a fixed constant greater or equal to 3 [4]. The approximation algorithm is obtained by showing that Tuple Minimization can be solved optimally in polynomial time when $|\Sigma_s|$ is a fixed constant. An exact algorithm for the Tuple Minimization problem directly implies a factor m approximation algorithm for the *l-diversity* problem. Let us show that the Tuple Minimization problem can be solved in polynomial time when $\Sigma_s = \{\sigma_1, \ldots, \sigma_q\}$, where q is a constant, by dynamic programming. Recall that the input of Tuple Minimization consists of a set R of rows and a sound partition $P = (S_1, S_2, \ldots, S_h, Z)$ of R, where each set S_i, $1 \leq i \leq h$, is l-diverse and consists of identical rows (in the q-columns), while Z may not be l-diverse. In order to compute a feasible solution for Tuple Minimization we have to transfer some rows from the sets $S_1, S_2, \ldots, S_h$ to the set Z. Each transfer of rows from a set S_x to the set Z results in a set $S_x^t \subseteq S_x$ and a set $Z' = Z \cup (S_x \setminus S_x^t)$ (where one of S_x^t, $(S_x \setminus S_x^t)$ can possibly be empty). A transfer from the sets $S_1, \ldots, S_j$, $1 \leq j \leq h$, to Z is *feasible*, if each S_x^t, $1 \leq x \leq h$, is l-diverse.

Define $S[t_1, t_2, \ldots, t_q; j]$ be equal to 1 if there exists a feasible transfer from the sets $S_1, \ldots, S_j$, $1 \leq j \leq h$, to Z, such that t_i rows, $1 \leq i \leq q$, having value

σ_i in the s-column have been transferred, else $S[t_1, t_2, \ldots, t_q; j]$ is equal to 0. Let us present the recurrence to compute $S[t_1, t_2, \ldots, t_q; j]$:

$$S[t_1, t_2, \ldots, t_q; j] = \bigvee_{t'_1, t'_2, \ldots, t'_q} S[t''_1, t''_2, \ldots, t''_q; j-1]$$

such that S^t_j is obtained by moving $t'_i = t_i - t''_i$ rows, $1 \leq i \leq q$, having value σ_i in the s-column from S_j to Z, and S^t_j is l-diverse. For the basic case, it holds $S[t_1, t_2, \ldots, t_z; 1] = 1$ if S^t_1 is obtained by moving t_i rows, $1 \leq i \leq q$, having value σ_i in the s-column from S_1 to Z, and S^t_1 is l-diverse. Now, let us show that the algorithm computes correctly an optimal solution for Tuple Minimization.

Lemma 6. *There is a feasible transfer from $S_1, \ldots, S_j$ to Z, such that for each $i \in \{1, \ldots, q\}$, t_i rows of value σ_i in the s-column are transferred, if and only if $S[t_1, t_2, \ldots, t_q; j] = 1$.*

As a consequence of Lemma 6, it follows that Tuple Minimization can be solved in time $O(n^{2|\Sigma_s|+1})$.

4 Parameterized Complexity of *l-diversity*

In this section we consider the parameterized complexity of the *l-diversity* problem, under two natural parameterizations. First, we show that the *l-diversity* problem is W[1]-hard when parameterized by the size w of the solution (i.e. the number of suppressions in an optimal solution), by l, and by $|\Sigma|$. Then we investigate the parameterized complexity of *l-diversity*, when the problem is parameterized by m and by the maximum number $|\Sigma_{max}|$ of different values in a column. We refer the reader to [14] for an introduction to parameterized complexity.

4.1 W[1]-Hardness of *l-diversity*

In this section we consider the *l-diversity* problem parameterized by the size of the solution, denoted by w, by l and by $|\Sigma|$, the size of the alphabet Σ. We denote such a restriction by *l-diversity*$(l, w, |\Sigma|)$. Given a set R of rows, the *l-diversity*$(l, w, |\Sigma|)$ problem asks for a solution of *l-diversity* that induces at most w suppressions.

We show that *l-diversity*$(l, w, |\Sigma|)$ is W[1]-hard, by giving a parameter preserving reduction from the h-Clique problem. Given a graph $G = (V, E)$, a h-clique is a set $V' \subseteq V$, with $|V'| = h$, such that each pair of vertices in V' is connected by an edge of E. The h-Clique problem, given a graph $G = (V, E)$ and a parameter h, asks if there exists a h-clique in G. Notice that the vertices of a h-clique are connected by exactly $\binom{h}{2}$ edges.

Let us describe the reduction from h-Clique to *l-diversity*$(l, w, |\Sigma|)$. Given a graph $G = (V, E)$, denote by m_G and n_G respectively $|E|$ and $|V|$. Now, let us describe the instance R of *l-diversity*$(l, w, |\Sigma|)$ associated with G. Set $l = 2h^2$

and $l_z = 2h^2 - \binom{h}{2}$. The rows in R are over the columns of indices $0, \ldots, n_G$ (recall that the column of index 0 is the s-column). The rows in R have value over the alphabet $\Sigma = \{0,1\} \cup \{f_k : 1 \leq k \leq l+1\}$. Hence $|\Sigma| = l+3$. Each q-column i, $1 \leq i \leq n_G$, is associated with the vertex v_i of G. For each edge $\{v_i, v_j\} \in E$, R contains a set $S(i,j) = \{r_{i,j,k} : 1 \leq k \leq l+1\}$ of rows defined as follows: $r_{i,j,k}[0] = f_k$, for $1 \leq k \leq l+1$; $r_{i,j,k}[i] = r_{i,j,k}[j] = 1$, for $1 \leq k \leq l+1$; $r_{i,j,k}[p] = 0$, for $p \neq i, j$, $1 \leq p \leq n_G$, and $1 \leq k \leq l+1$.

Furthermore, R contains a set $Z = \{z_1, \ldots, z_{l_z}\}$ of rows defined as follows: $z_i[0] = f_i$, for $1 \leq i \leq l_z$; $z_i[k] = 0$, for $1 \leq i \leq l_z$ and $1 \leq k \leq n_G$. Notice that $|Z| = l_z = l - \binom{h}{2}$ and that Z is the only set of R that is not l-diverse. Now, we are able to prove the two main properties of the reduction.

Lemma 1. *Let $G = (V, E)$ be an instance of h-Clique, and let R be the corresponding instance of* l-diversity$(l, w, |\Sigma|)$. *Given $V' \subseteq$ a h-clique of G, we can compute in polynomial time a solution of* l-diversity$(l, w, |\Sigma|)$ *over instance R with at most $hl = 2h^3$ suppressions.*

Lemma 2. *Let $G = (V, E)$ be an instance of h-Clique, and let R be the corresponding instance of* l-diversity$(l, w, |\Sigma|)$. *Let Π be a solution of* l-diversity$(l, w, |\Sigma|)$ *over instance R with $c(\Pi) \leq hl = 2h^3$, then we can compute in polynomial time a h-clique of G.*

Proof. (*Sketch.*) Given a solution Π of *l-diversity*$(l, w, |\Sigma|)$ over instance R, with $c(\Pi) \leq hl = 2h^3$, it is possible to prove the following claim: there are at least $\binom{h}{2}$ sets $S(i,j)$, such that Π induces a suppression in some row of $S(i,j)$, otherwise Π is not l-diverse.

Assume now that Π is a solution of *l-diversity* such that Π induces some suppressions in the rows of at least $k = \binom{h}{2}$ sets $S(i,j)$. Starting from a solution Π with $c(\Pi) \leq lh$, it is possible to compute in polynomial time a solution Π^* such that $c(\Pi^*) \leq c(\Pi)$ and such that: (i) for each set $S(i,j)$, Π^* contains a set $S'(i,j)$ of at least l identical rows (in the q-columns); (ii) there is exactly one row $r_{i,j}$ from each of $\binom{h}{2}$ sets $S(i,j)$, $1 \leq i \leq n_G$ and $1 \leq j \leq m_G$, for which Π^* induces some suppressions. Each row $r_{i,j}$ corresponds to the edge of G incident into vertices v_i and v_j. It follows that the $\binom{h}{2}$ rows corresponds to a set of $\binom{h}{2}$ edges of G incident in a set V' of vertices, with $|V'| = h$ vertices. Hence V' is a h-clique of G. □

Theorem 3 is a consequence of Lemmata 1 and 2.

Theorem 3 l-diversity$(l, w, |\Sigma|)$ *is W[1]-hard.*

4.2 Bounding $|\Sigma_{max}|$ and m

Since *l-diversity* is NP-hard when either the number $|\Sigma_{max}|$ of different values in a column (see [4]) or m (see Section 3) is bounded by a fixed constant, a natural question which may arise is whether the problem remains hard if it is

parameterized by both these values. We denote by *l-diversity*$(|\Sigma_{max}|, m)$ the *l-diversity* problem parameterized by $|\Sigma_{max}|$ and m. In this section, we show that *l-diversity*$(|\Sigma_{max}|, m)$ admits a fixed-parameter algorithm.

Let R be a set of rows over Σ. A solution for *l-diversity*$(|\Sigma_{max}|, m)$ is a partition $\Pi = (P_1, \ldots, P_z)$ of R, where each set P_i must be l-diverse. Each resolution vector $r(P_i)$ is a vector over the alphabet $\Sigma^* = \Sigma \cup \{*\}$. Notice that the number of resolution vectors is at most $(|\Sigma_{max}| + 1)^m$. Let Res be the set of possible resolution vectors. Recall that the cost associated with a resolution vector v is equal to $Del(v)$, that is the number of $*$ in r. For each resolution vector in $v \in Res$, let $r(v)$ be the number of rows in R compatible to v. Now, define S as the set of full resolution vector (i.e. those resolution vectors that do not contain suppressions) such that, for each $s \in S$, there is a row r compatible with s. Notice that $|S| \leq |\Sigma_{max}|^m$. Given $s \in S$, $r(s)$ is the number of rows in R compatible to s. Given the set of resolution vector Res, and the set R, *l-diversity*$(|\Sigma_{max}|, m)$ asks for a solution with minimum number of suppressions. A solution of *l-diversity* is obtained by computing an assignment of the rows in R to the resolution vectors in Res, which is is represented as an assignment of the rows identical to a vector $s \in S$ to the vectors of Res. Each set of the partition consists of the rows assigned to a specific resolution vector. Since in a feasible solution each set must be l-diverse, the set of rows assigned to each resolution vector must be an l-diverse set.

In order to compute the solution of minimum cost of *l-diversity*$(|\Sigma_{max}|, m)$, we introduce an integer linear program (ILP). Let $s \in S$ be a full resolution vector, and v be a resolution vector of Res compatible with s. We define the variable $x_{s,v,\lambda}$ as the number of rows equal to s having sensitive attribute λ, which are assigned to the resolution vector v.

$$\begin{array}{ll} \min \sum_{s,v,\lambda} Del(s) x_{s,v,\lambda} & \\ \text{s.t.} \sum_{s,\lambda} x_{s,v,\lambda} \geq l(\sum_{s} x_{s,v,\lambda'}) & \text{for each } v \in Res \text{ and for each } \lambda' \in \Sigma_s \\ \sum_{v,\lambda} x_{s,v,\lambda} = r(s) & \text{for each } s \in S \\ x_{s,v,\lambda} \in \mathbb{Z}^+ & \end{array} \quad (1)$$

The ILP has $O((|\Sigma_{max}| + 1)^{m+1})$ number of variables. By the result in [10], a integer linear program with z variables can be solved with $O(z^{9z/2}L)$ arithmetic operations in integers of $O(z^{2z}L)$ bits in size, where z represents the number of variable in the ILP, while L is the number of bits in the input. Hence *l-diversity*$(|\Sigma_{max}|, m)$ is fixed-parameter tractable.

References

1. 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)
2. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic Construction of Sets for k-restrictions. ACM Trans. Algorithms 2(2), 153–177 (2006)
3. Alimonti, P., Kann, V.: Some APX-Completeness Results for Cubic Graphs. Theoretical Computer Science 237(1-2), 123–134 (2000)

4. 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)
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. Bonizzoni, P., Della Vedova, G., Dondi, R.: Anonymizing Binary and Small Tables is Hard to Approximate. Journal of Comb. Opt. 22(1), 97–119 (2011)
7. Downey, R.G., Fellows, M.R.: Fixed-Parameter Tractability and Completeness ii: On Completeness for W[1]. Theoretical Computer Science 141, 109–131 (1995)
8. Evans, P.A., Wareham, T., Chaytor, R.: Fixed-parameter Tractability of Anonymizing Data by Suppressing Entries. J. Comb. Optim. 18(4), 362–375 (2009)
9. Gionis, A., Tassa, T.: k-Anonymization with Minimal Loss of Information. IEEE Trans. Knowl. Data Eng. 21(2), 206–219 (2009)
10. Lenstra, H.: Integer Programming with a Fixed Number of Variables. Mathematics of Operations Research 4(8) (1983)
11. Li, J., Yi, K., Zhang, Q.: Clustering with diversity. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 188–200. Springer, Heidelberg (2010)
12. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M.: l-Diversity: Privacy Beyond k-Anonymity. In: Liu, L., Reuter, A., Whang, K., Zhang, J. (eds.) 22nd International Conference on Data Engineering, p. 24. IEEE Computer Society, New York (2006)
13. Meyerson, A., Williams, R.: On the Complexity of Optimal K-Anonymity. In: Deutsch, A. (ed.) 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 223–228. ACM, New York (2004)
14. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
15. Park, H., Shim, K.: Approximate Algorithms for k-Anonymity. In: Chan, C.Y., Ooi, B.C., Zhou, A. (eds.) ACM SIGMOD International Conference on Management of Data, pp. 67–78. ACM Press, New York (2007)
16. Raz, R., Safra, S.: A Sub-Constant Error-probability Low-degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP. In: Twenty-ninth Annual ACM Symposium on Theory of Computing (STOC 1997), pp. 475–484 (1997)
17. Samarati, P.: Protecting Respondents' Identities in Microdata Release. IEEE Trans. Knowl. Data Eng. 13(6), 1010–1027 (2001)
18. Sweeney, L.: k-Anonymity: a Model for Protecting Privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10(5), 557–570 (2002)
19. Vazirani, V.: Approximation Algorithms. Springer, Heidelberg (2001)
20. Xiao, X., Yi, K., Tao, Y.: The Hardness and Approximation Algorithms for l-diversity. In: Manolescu, I., Spaccapietra, S., Teubner, J., Kitsuregawa, M., Lger, A., Naumann, F., Ailamaki, A., Ozcan, F. (eds.) 13th International Conference on Extending Database Technology (EDBT 2010), pp. 135–146. ACM Press, New York (2010)

Infinite Synchronizing Words for Probabilistic Automata*

Laurent Doyen[1], Thierry Massart[2], and Mahsa Shirmohammadi[2]

[1] LSV, ENS Cachan & CNRS, France
doyen@lsv.ens-cachan.fr
[2] Université Libre de Bruxelles, Brussels, Belgium
{thierry.massart,mahsa.shirmohammadi}@ulb.ac.be

Abstract. Probabilistic automata are finite-state automata where the transitions are chosen according to fixed probability distributions. We consider a semantics where on an input word the automaton produces a sequence of probability distributions over states. An infinite word is accepted if the produced sequence is synchronizing, i.e. the sequence of the highest probability in the distributions tends to 1. We show that this semantics generalizes the classical notion of synchronizing words for deterministic automata. We consider the emptiness problem, which asks whether some word is accepted by a given probabilistic automaton, and the universality problem, which asks whether all words are accepted. We provide reductions to establish the PSPACE-completeness of the two problems.

1 Introduction

Probabilistic automata (PA) are finite-state automata where the transitions are chosen according to fixed probability distributions. In the traditional semantics, a run of a probabilistic automaton over an input word is a path (i.e., a sequence of states and transitions), and the *classical acceptance conditions* over runs (such as in finite automata, Büchi automata, etc.) are used to define the probability to accept a word as the measure of its accepting runs [11,2]. Over finite and infinite words, several undecidability results are known about probabilistic automata in the traditional semantics [10,1].

Recently, an alternative semantics for probabilistic automata has been proposed, with applications in sensor networks, queuing theory, and dynamical systems [9,8,5]. In this new semantics, a run over an input word is the sequence of probability distributions produced by the automaton. For an example, consider the probabilistic automaton with alphabet $\Sigma = \{a, b\}$ on Fig. 1 and the sequence of probability distributions produced by the input word $a(aba)^\omega$.

Previous works have considered *qualitative* conditions on this semantics. The space of probability distributions (which is a subset of $[0, 1]^n$) is partitioned into regions defined by linear predicates, and classical acceptance conditions are used to define

* This work has been partly supported by the MoVES project (P6/39) which is part of the IAP-Phase VI Interuniversity Attraction Poles Programme funded by the Belgian State, Belgian Science Policy.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 278–289, 2011.

accepting sequences of regions. It is known that reachability of a region is undecidable for linear predicates, and that it becomes decidable for a class of qualitative predicates which essentially constrain only the support of the probability distributions [8].

In this paper, we consider a *quantitative* semantics which has decidable properties, defined as follows [5]. A sequence $\bar{X} = X_0 X_1 \dots$ of probability distributions over a set of states Q is *synchronizing* if in the long run, the probability mass tends to accumulate in a single state. More precisely, we consider two definitions: the sequence $\bar{X}$ is *strongly synchronizing* if $\liminf_{i\to\infty} \|X_i\| = 1$ where $\|X_i\| = \max_{q\in Q} X_i(q)$ is the highest probability in X_i; it is *weakly synchronizing* if $\limsup_{i\to\infty} \|X_i\| = 1$. Intuitively, strongly synchronizing means that the probabilistic automaton behaves in the long run like a deterministic system: eventually, at every step i (or at infinitely many steps for weakly synchronizing) there is a state $\hat{q}_i$ which accumulates almost all the probability, and therefore the sequence $\hat{q}_i \hat{q}_{i+1} \dots$ is almost deterministic. Note that the state $\hat{q}_i$ needs not be the same at every step i. For instance, in the sequence in Fig. 1, the maximal probability in a state tends to 1, but it alternates between the three states q_2, q_3, and q_4. We define the synchronizing language $L(\mathcal{A})$ of a probabilistic automaton $\mathcal{A}$ as the set of words[1] which induce a synchronizing sequence of probability distributions. In this paper, we consider the decision problems of emptiness and universality for synchronizing language, i.e. deciding whether $L(\mathcal{A}) = \emptyset$, and $L(\mathcal{A}) = \mathcal{D}(\Sigma)^\omega$ respectively.

Synchronizing words have applications in planning, control of discrete event systems, biocomputing, and robotics [3,15]. For deterministic finite automata (DFA), a (finite) word w is synchronizing if reading w from any state of the automaton always leads to the same state. Note that DFA are a special case of probabilistic automata. A previous generalization of synchronizing words to probabilistic automata was proposed by Kfoury, but the associated decision problem is undecidable [7]. By contrast, the results of this paper show that the definition of strongly and weakly synchronizing words is a decidable generalization of synchronized words for DFA. More precisely, we show that there exists a (finite) synchronizing word for a DFA $\mathcal{A}$ if and only if there exists an (infinite) synchronizing word for $\mathcal{A}$ viewed as a probabilistic automaton with uniform initial distribution over all states.

We show that the emptiness and universality problems for synchronizing languages is PSPACE-complete, for both strongly and weakly synchronizing semantics. For emptiness, the PSPACE upper bound follows from a reduction to the emptiness problem of an exponential-size Büchi automaton. The construction relies on an extension of the classical subset construction. The PSPACE lower bound is obtained by a reduction from the universality problem for nondeterministic finite automata.

For universality, the upper bound follows from a reduction to the emptiness problem of an exponential-size coBüchi automaton, and the lower bound is obtained by a reduction from the emptiness problem of traditional probabilistic coBüchi automata in positive semantics [4,14].

[1] Words can be randomized, i.e. their letters can be probability distributions over the alphabet Σ. We denote by $\mathcal{D}(\Sigma)$ the set of all probability distributions over Σ.

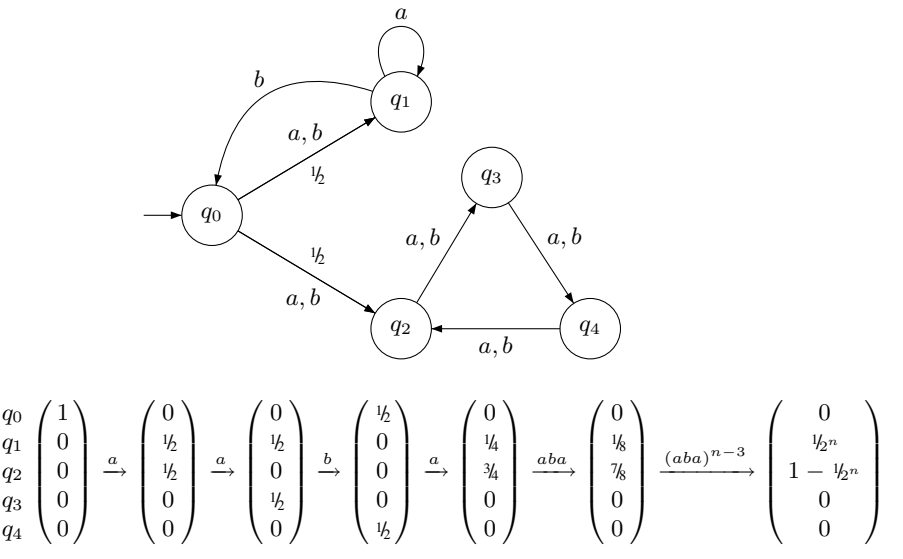


Fig. 1. The word $a(aba)^\omega$ is strongly synchronizing

The PSPACE-completeness bounds improve the results of [5] where it is shown that the emptiness and universality problems for synchronizing languages are decidable[2] using a characterization which yields doubly exponential algorithms.

Due to lack of space, the details of some proofs are omitted and can be found in [6].

2 Automata and Synchronizing Words

A *probability distribution* over a finite set S is a function $d : S \to [0,1]$ such that $\sum_{s\in S} d(s) = 1$. The *support* of d is the set $\mathsf{Supp}(d) = \{s \in S \mid d(s) > 0\}$. We denote by $\mathcal{D}(S)$ the set of all probability distributions over S.

Given a finite alphabet Σ, we denote by Σ^* the set of all finite words over Σ, and by Σ^ω the set of all infinite words over Σ. The length of a word w is denoted by $|w|$ (where $|w| = \infty$ for infinite words). An infinite *randomized word* over Σ is a sequence $w = d_0 d_1 \dots$ of probability distributions over Σ. We denote by $\mathcal{D}(\Sigma)^\omega$ the set of all infinite randomized words over Σ. A word $w \in \Sigma^\omega$ can be viewed as a randomized word $d_0 d_1 \dots$ in which the support of all probability distributions d_i is a singleton. We sometimes call $w \in \Sigma^\omega$ a *pure word* to emphasize this.

Finite Automata. A nondeterministic finite automaton (NFA) $\mathcal{A} = \langle L, \ell_0, \Sigma, \delta, \mathcal{F}\rangle$ consists of a finite set L of states, an initial state $\ell_0 \in L$, a finite alphabet Σ, a transition relation $\delta : L \times \Sigma \to 2^L$, and an acceptance condition $\mathcal{F}$ which can be either finite, Büchi, or coBüchi (and then $\mathcal{F} \subseteq L$), or generalized Büchi (and then $\mathcal{F} \subseteq 2^L$).

[2] Probabilistic automata are equivalent to Markov decision processes with blind strategies.

Finite acceptance conditions define languages of finite words, other acceptance conditions define languages of infinite words. Automata with Büchi, coBüchi, and generalized Büchi condition are called ω-automata. A *run* over a (finite or infinite) word $w = \sigma_0\sigma_1\dots$ is a sequence $\rho = r_0r_1\dots$ such that $r_0 = \ell_0$ and $r_{i+1} \in \delta(r_i, \sigma_i)$ for all $0 \leq i < |w|$. A finite run $r_0 \dots r_k$ is *accepting* if $r_k \in \mathcal{F}$, and an infinite run $r_0r_1\dots$ is *accepting* for a Büchi condition if $r_j \in \mathcal{F}$ for infinitely many j, for a coBüchi condition if $r_j \notin \mathcal{F}$ for finitely many j, for a generalized Büchi condition if for all $s \in \mathcal{F}$, we have $r_j \in s$ for infinitely many j.

The *language* of a (finite- or ω-) automaton is the set $L_{\mathsf{f}}(\mathcal{A})$ (resp., $L_\omega(\mathcal{A})$) of finite (resp., infinite) words over which there exists an accepting run. The *emptiness problem* for (finite- or ω-) automata is to decide, given an automaton $\mathcal{A}$, whether $L_{\mathsf{f}}(\mathcal{A}) = \emptyset$ (resp., $L_\omega(\mathcal{A}) = \emptyset$), and the *universality problem* is to decide whether $L_{\mathsf{f}}(\mathcal{A}) = \Sigma^*$ (resp., $L_\omega(\mathcal{A}) = \Sigma^\omega$). For both finite and Büchi automata, the emptiness problem is NLOGSPACE-complete, and the universality problem is PSPACE-complete [13,12].

A *deterministic* finite automaton (DFA) is a special case of NFA where the transition relation is such that $\delta(\ell, \sigma)$ is a singleton for all $\ell \in L$ and $\sigma \in \Sigma$, which can be viewed as a function $\delta : L \times \Sigma \to L$, and can be extended to a function $\delta : L \times \Sigma^* \to L$ defined inductively as follows: $\delta(\ell, \epsilon) = \ell$ with ϵ the empty word and $\delta(\ell, \sigma \cdot w) = \delta(\delta(\ell, \sigma), w)$ for all $w \in \Sigma^*$. A *synchronizing* word for a DFA is a word $w \in \Sigma^*$ such that $\delta(\ell, w) = \delta(\ell', w)$ for all $\ell, \ell' \in L$, i.e. such that from all states, a unique state is reached after reading w. Synchronizing words have applications in several areas from planning to robotics and system biology, and they gave rise to the famous Černý's conjecture [3,15].

Probabilistic Automata. A *probabilistic automaton* (PA) $\mathcal{A} = \langle Q, \mu_0, \Sigma, \delta\rangle$ consists of a finite set Q of states, an initial probability distribution $\mu_0 \in \mathcal{D}(Q)$, a finite alphabet Σ, and a probabilistic transition function $\delta : Q \times \Sigma \to \mathcal{D}(Q)$. In a state $q \in Q$, the probability to go to a state $q' \in Q$ after reading a letter $\sigma \in \Sigma$ is $\delta(q, \sigma)(q')$. Define $\mathsf{Post}(q, \sigma) = \mathsf{Supp}(\delta(q, \sigma))$, and for a set $s \subseteq Q$ and $\Sigma' \subseteq \Sigma$, let $\mathsf{Post}(s, \Sigma') = \bigcup_{q\in s} \bigcup_{\sigma\in\Sigma'} \mathsf{Post}(q, \sigma)$.

The *outcome* of an infinite randomized word $w = d_0d_1\dots$ is the infinite sequence $X_0X_1\dots$ of probability distributions $X_i \in \mathcal{D}(Q)$ such that $X_0 = \mu_0$ is the initial distribution, and for all $n > 0$ and $q \in Q$,

$$X_n(q) = \sum_{\sigma\in\Sigma} \sum_{q'\in Q} X_{n-1}(q') \cdot d_{n-1}(\sigma) \cdot \delta(q', \sigma)(q)$$

The *norm* of a probability distribution X over Q is $\|X\| = \max_{q\in Q} X(q)$. We say that w is a *strongly synchronizing* word if

$$\liminf_{n\to\infty} \|X_n\| = 1, \tag{1}$$

and that it is a *weakly synchronizing* word if

$$\limsup_{n\to\infty} \|X_n\| = 1. \tag{2}$$

Intuitively, a word is synchronizing if in the outcome the probability mass tends to concentrate in a single state, either at every step from some point on (for strongly

synchronizing), or at infinitely many steps (for weakly synchronizing). Note that equivalently, the randomized word w is strongly synchronizing if the limit $\lim_{n\to\infty}\|X_n\|$ exists and equals 1. We denote by $\mathcal{L}_S(\mathcal{A})$ (resp., $\mathcal{L}_W(\mathcal{A})$) the set of strongly (resp., weakly) synchronizing words of $\mathcal{A}$.

In this paper, we are interested in the *emptiness problem* for strongly (resp., weakly) synchronizing languages which is to decide, given a probabilistic automaton $\mathcal{A}$, whether $\mathcal{L}_S(\mathcal{A}) = \emptyset$ (resp., $\mathcal{L}_W(\mathcal{A}) = \emptyset$), and in the *universality problem* which is to decide, whether $\mathcal{L}_S(\mathcal{A}) = \mathcal{D}(\Sigma)^\omega$ (resp., $\mathcal{L}_W(\mathcal{A}) = \mathcal{D}(\Sigma)^\omega$).

Synchronizing sequences of probability distributions have been first introduced for Markov decision processes (MDP) [5]. A probabilistic automaton can be viewed as an MDP where a word corresponds to a blind strategy (in the terminology of [5]) which chooses letters (or actions) independently of the sequence of states visited by the automaton and it only depends on the number of rounds that have been played so far. It is known that the problem of deciding the existence of a blind synchronizing strategy for MDPs is decidable[3] [5, Theorem 5]. In Section 3 we provide a solution in PSPACE to this problem, as well as a matching PSPACE lower bound.

Remark 1. From the results of [5], it follows that if there exists a (strongly or weakly) synchronizing word, then there exists a pure one.

A deterministic finite automaton is also a special case of probabilistic automaton where the probabilistic transition function is such that $\mathsf{Post}(q,\sigma)$ is a singleton for all $q \in Q$ and $\sigma \in \Sigma$ (and disregarding the initial distribution μ_0). We show that the definition of strongly (and weakly) synchronizing word generalizes to probabilistic automata the notion of synchronizing words for DFA, in the following sense.

Theorem 1. *Given a deterministic finite automaton $\mathcal{A}$, the following statements are equivalent:*

1. *There exists a (finite) synchronizing word for $\mathcal{A}$.*
2. *There exists an (infinite) strongly (or weakly) synchronizing word for $\mathcal{A}$ (viewed as a probabilistic automaton) with uniform initial distribution.*

Proof. First, if $w \in \Sigma^*$ is a synchronizing word for the DFA $\mathcal{A}$, there is a state q which is reached from all states of A by reading w. This implies that $X_{|w|}(q) = 1$ in the PA $\mathcal{A}$ (no matter the initial distribution) and since the transition function of $\mathcal{A}$ is deterministic, any infinite word with prefix w is both strongly (and thus also weakly) synchronizing for $\mathcal{A}$.

Second, assume that w is a strongly (or weakly) synchronizing word for the PA $\mathcal{A}$ with initial distribution μ_0 such that $\mu_0(q) = \frac{1}{m}$ where $m = |Q|$ is the number of states of $\mathcal{A}$. By Remark 1, we assume that $w = \sigma_0\sigma_1\cdots \in \Sigma^\omega$ is pure. Let $X_0X_1\ldots$ be the outcome of w in $\mathcal{A}$. Since the transitions in $\mathcal{A}$ are deterministic, all probabilities $X_i(q)$ for $i \geq 0$ and $q \in Q$ are multiples of $\frac{1}{m}$, i.e. $X_i(q) = \frac{c}{m}$ for some $0 \leq c \leq m$. Therefore, the fact that $\liminf_{n\to\infty} \|X_n\| = 1$ (or $\limsup_{n\to\infty} \|X_n\| = 1$) implies that $X_i(q) = 1$ for some $i \geq 0$ and $q \in Q$. Then, the finite word $\sigma_0\sigma_1\ldots\sigma_{i-1}$ is synchronizing for $\mathcal{A}$. □

[3] The results in [5] suggest a doubly exponential algorithm for solving this problem.

Note that the problem of deciding whether there exists a synchronizing word for a given DFA can be solved in polynomial time, while the emptiness problem for synchronizing languages (for probabilistic automata) is PSPACE-complete (see Theorem 2).

End-Components. A set $C \subseteq Q$ is *closed* if for every state $q \in C$, there exists $\sigma \in \Sigma$ such that $\mathsf{Post}(q, \sigma) \subseteq C$. For each $q \in C$, let $D_C(q) = \{\sigma \in \Sigma \mid \mathsf{Post}(q, \sigma) \subseteq C\}$. The graph induced by C is $\mathcal{A} \restriction C = (C, E)$ where E is the set of edges $(q, q') \in C \times C$ such that $\delta(q, \sigma)(q') > 0$ for some $\sigma \in D_C(q)$. An *end-component* is a closed set U such that the graph $\mathcal{A} \restriction C$ is strongly connected.

3 The Emptiness Problem is PSPACE-Complete

In this section, we present constructions to reduce the emptiness problem for synchronizing languages of probabilistic automata to the emptiness problem for ω-automata, with Büchi condition for strongly synchronizing language, and with generalized Büchi condition for weakly synchronizing language. The constructions are exponential and therefore provide a PSPACE upper bound for the problems. We also prove a matching lower bound.

Lemma 1. *The emptiness problem for strongly synchronizing language of probabilistic automata is decidable in PSPACE.*

We give the main idea of the proof of Lemma 1. The details can be found in [6].

Given a PA $\mathcal{A} = \langle Q, \mu_0, \Sigma, \delta \rangle$, we construct a Büchi automaton $\mathcal{B} = \langle L, \ell_0, \Sigma, \delta_{\mathcal{B}}, \mathcal{F}_{\mathcal{B}} \rangle$ such that $\mathcal{L}_S(\mathcal{A}) = \emptyset$ iff $L(\mathcal{B}) = \emptyset$. The automaton $\mathcal{B}$ is exponential in the size of $\mathcal{A}$, and thus the PSPACE bound follows from the NLOGSPACE-completeness of the emptiness problem for Büchi automata.

The construction of $\mathcal{B}$ relies on the following intuition. A strongly synchronizing word induces a sequence of probability distributions X_i in which the probability mass tends to accumulate in a single state $\hat{q}_i$ at step i. It can be shown that for all sufficiently large i, there exists a deterministic transition from $\hat{q}_i$ to $\hat{q}_{i+1}$, i.e. there exists $\sigma_i \in \Sigma$ such that $\mathsf{Post}(\hat{q}_i, \sigma_i) = \{\hat{q}_{i+1}\}$. The Büchi automaton $\mathcal{B}$ will guess the *witness sequence* $\hat{q}_i \hat{q}_{i+1} \ldots$ and check that the probability mass is 'injected' into this sequence. The state of $\mathcal{B}$ keeps track of the support $s_i = \mathsf{Supp}(X_i)$ of the outcome sequence on the input word, and at some point guesses that the witness sequence $\hat{q}_i \hat{q}_{i+1} \ldots$ starts. Then, using an *obligation* set $o_i \subseteq s_i$, it checks that every state in s_i eventually 'injects' some probability mass in the witness sequence.

The construction of $\mathcal{B} = \langle L, \ell_0, \Sigma, \delta_{\mathcal{B}}, \mathcal{F}_{\mathcal{B}} \rangle$ is as follows:

- $L = 2^Q \cup (2^Q \times 2^Q \times Q)$ is the set of states. A state $s \subseteq Q$ is the support of the current probability distribution. A state $(s, o, \hat{q}) \in 2^Q \times 2^Q \times Q$ consists of the support s, the obligation set $o \subseteq s$, and a state $\hat{q} \in s$ of the witness sequence.
- $\ell_0 = \mathsf{Supp}(\mu_0)$ is the initial state.
- $\delta_{\mathcal{B}} : L \times \Sigma \to 2^L$ is defined as follows. For all $s \in 2^Q$ and $\sigma \in \Sigma$, let $s' = \mathsf{Post}(s, \sigma)$, and define $\delta_{\mathcal{B}}(s, \sigma) = \{s'\} \cup \{(s', s', \hat{q}) \mid \hat{q} \in s'\}$. For all $(s, o, \hat{q}) \in 2^Q \times 2^Q \times Q$ and $\sigma \in \Sigma$, let $s' = \mathsf{Post}(s, \sigma)$. If $\mathsf{Post}(\hat{q}, \sigma)$ is not a singleton, then $\delta_{\mathcal{B}}((s, o, \hat{q}), \sigma) = \emptyset$, otherwise let $\{\hat{q}'\} = \mathsf{Post}(\hat{q}, \sigma)$, and

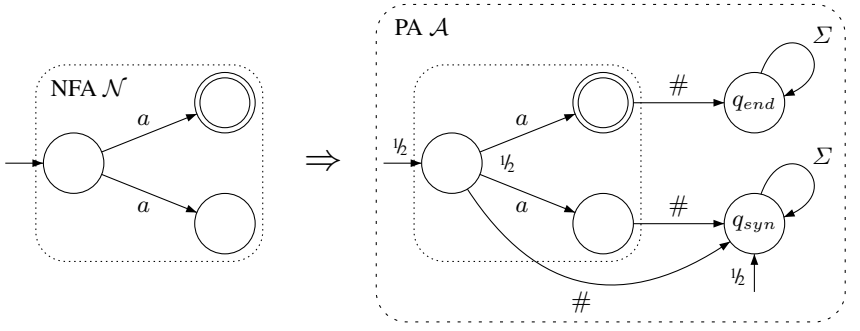


Fig. 2. Sketch of the reduction for PSPACE-hardness of the emptiness problem

- if $o \neq \emptyset$, then $\delta_{\mathcal{B}}((s,o,\hat{q}),\sigma) = \{(s', o' \setminus \{\hat{q}'\}, \hat{q}') \mid \forall q \in o : o' \cap \mathsf{Post}(q,\sigma) \neq \emptyset\}$,
- if $o = \emptyset$, then $\delta_{\mathcal{B}}((s,o,\hat{q}),\sigma) = \{(s', s', \hat{q}')\}$.

– $\mathcal{F}_{\mathcal{B}} = \{(s,o,\hat{q}) \in 2^Q \times 2^Q \times Q \mid o = \emptyset\}$ is the set of accepting states.

Lemma 2. *The emptiness problem for weakly synchronizing language of probabilistic automata is decidable in PSPACE.*

The proof of Lemma 2 is by a reduction to the emptiness problem of an exponential-size ω-automaton with generalized Büchi condition. It can be found in [6].

Lemma 3. *The emptiness problem for strongly synchronizing language and for weakly synchronizing language of probabilistic automata is PSPACE-hard.*

Proof. We present a proof for strongly synchronizing words using a reduction from the universality problem for nondeterministic finite automata. The proof and the reduction for weakly synchronizing words is analogous.

Given a NFA $\mathcal{N}$, we construct a PA $\mathcal{A}$, such that $L(\mathcal{N}) = \Sigma^*$ iff $\mathcal{L}_S(\mathcal{A}) = \emptyset$. The reduction is illustrated in Fig. 2. The nondeterministic transitions of $\mathcal{N}$ become probabilistic in $\mathcal{A}$ with uniform probability. The initial probability distribution assigns probability $\frac{1}{2}$ to the absorbing state q_{sync}. Therefore, a synchronizing word needs to inject all that probability into q_{sync}. This can be done with the special symbol $\#$ from the non-accepting states of the NFA. From the accepting states, the $\#$ symbol leads to a sink state q_{end} from which there is no way to synchronize the automaton.

Let $\mathcal{N} = \langle L, \ell_0, \Sigma, \delta_{\mathcal{N}}, \mathcal{F}_{\mathcal{N}} \rangle$ be a NFA, we construct the PA $\mathcal{A} = \langle Q, \mu_0, \Sigma', \delta, \mathcal{F} \rangle$ as follows:

– $Q = L \cup \{q_{sync}, q_{end}\}$.
– $\mu_0(\ell_0) = \mu_0(q_{sync}) = \frac{1}{2}$, and $\mu_0(q) = 0$ for all $q \in Q \setminus \{\ell_0, q_{sync}\}$.
– $\Sigma' = \Sigma \cup \{\#\}$.
– $\delta : Q \times \Sigma \to \mathcal{D}(Q)$ is the probabilistic transition function defined as follows. For all $\sigma \in \Sigma'$, $\delta(q_{sync},\sigma)(q_{sync}) = 1$ and $\delta(q_{end},\sigma)(q_{end}) = 1$. For all $q \in \mathcal{F}_N$, $\delta(q,\#)(q_{end}) = 1$, and for all $q \notin \mathcal{F}_N$, $\delta(q,\#)(q_{sync}) = 1$. Finally, for all $q, q' \in L$ and $\sigma \in \Sigma$, $\delta(q,\sigma)(q') = \frac{1}{|\delta_N(q,\sigma)|}$ if $q' \in \delta_N(q,\sigma)$, and $\delta(q,\sigma)(q') = 0$ otherwise.

We show that $L(\mathcal{N}) \neq \Sigma^*$ iff $\mathcal{L}_S(\mathcal{A}) \neq \emptyset$. First, assume that $L(\mathcal{N}) \neq \Sigma^*$. Let $w \in \Sigma^*$ such that $w \notin L(\mathcal{N})$. Then all runs of $\mathcal{N}$ over w end in a non-accepting state, and in $\mathcal{A}$ the state q_{sync} is reached with probability 1 on the word $w \cdot \#$. Therefore, $w \cdot (\#)^\omega$ is a strongly synchronizing word for $\mathcal{A}$ and $\mathcal{L}_S(\mathcal{A}) \neq \emptyset$.

Second, assume that $\mathcal{L}_S(\mathcal{A}) \neq \emptyset$. Let $w' \in \mathcal{L}_S(\mathcal{A})$ be a strongly synchronizing word for $\mathcal{A}$, and let $X_0 X_1 \ldots$ be the outcome of w' in $\mathcal{A}$. Since $\mu_0(q_{sync}) = \frac{1}{2}$ and q_{sync} is a sink state, we have $X_k(q_{sync}) \geq \frac{1}{2}$ for all $k \geq 0$ and since w' is strongly synchronizing, it implies that $\lim_{k\to\infty} X_k(q_{sync}) = 1$. Then w' has to contain $\#$, as this is the only letter on a transition from a state in L to q_{sync}. Let $w \in \Sigma^*$ be the prefix of w' before the first occurrence of $\#$. We claim that w is not accepted by $\mathcal{N}$. By contradiction, if there is an accepting run r of $\mathcal{N}$ over w, then positive probability is injected in q_{end} by the finite word $w \cdot \#$ and stays there forever, in contradiction with the fact that $\lim_{k\to\infty} X_k(q_{sync}) = 1$. Therefore $w \notin L(\mathcal{N})$ and $L(\mathcal{N}) \neq \Sigma^*$. □

The following result follows from Lemma 1, Lemma 2, and Lemma 3.

Theorem 2. *The emptiness problem for strongly synchronizing language and for weakly synchronizing language of probabilistic automata is PSPACE-complete.*

4 The Universality Problem is PSPACE-Complete

In this section, we present necessary and sufficient conditions for probabilistic automata to have a universal strongly (resp., weakly) synchronizing language. We show that the construction can be checked in PSPACE. Unlike for the emptiness problem, it is not sufficient to consider only pure words for universality of strongly (resp., weakly) synchronizing languages. For instance, all infinite pure words for the probabilistic automaton in Fig. 3 are strongly (and weakly) synchronizing, but the uniformly randomized word over $\{a, b\}$ is not strongly (nor weakly) synchronizing. Formally, we say an infinite randomized word is a uniformly randomized word over Σ denoted by w_u, if $d_i(\sigma) = \frac{1}{|\Sigma|}$ for all $\sigma \in \Sigma$ and $i \in \mathbb{N}$.

Lemma 4. *There is a probabilistic automaton for which all pure words are strongly synchronizing, but not all randomized words .*

The reason is that there are two sets ($\{q_1\}$ and $\{q_2\}$) for which the probability can not go out. For a given PA $\mathcal{A} = \langle Q, \mu_0, \Sigma, \delta, \mathcal{F}\rangle$, a maximal end-component $U \subseteq Q$ is *terminal*, if $\mathsf{Post}(U, \Sigma) \subseteq U$. It is easy to see that a terminal end-component keeps probability inside. To have a universal strongly/weakly synchronizing language, the PA $\mathcal{A}$ needs to have only a unique terminal end-component. Otherwise, the uniformly randomized word w_u would reach all terminal end-components and would not be strongly synchronizing. Though having only a terminal end-component is necessary, it is not sufficient. For example, the infinite word $(ab)^\omega \notin \mathcal{L}_S(A)$ for the PA $\mathcal{A}$ in Fig. 5 which contains only one terminal end-component. The probabilistic automaton needs to ensure that for all randomized words, all of the probability mass tends to accumulate in the unique terminal end-component. We express this property for a terminal end-component as being absorbing. We say that a terminal end-component U is *absorbing*,

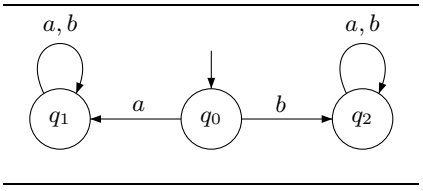


Fig. 3. Randomization is necessary

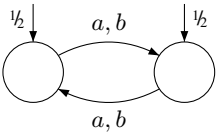


Fig. 4. Randomization is not sufficient

if $\lim_{n\to\infty} \sum_{q\in U} X_n(q) = 1$ for the outcome $X_0X_1\ldots$ of all infinite randomized words $w \in D(\Sigma)^\omega$. Fig. 6 shows an automaton where the unique end competent is absorbing and the strongly synchronizing language is universal.

Lemma 5. *For a given PA $\mathcal{A}$, deciding whether a given terminal end-component U is absorbing is decidable in PSPACE.*

We give the main idea of the proof of Lemma 5. The details are provided in [6].

Given a terminal end-component $U \subseteq Q$ of the PA $\mathcal{A} = \langle Q, \mu_0, \Sigma, \delta, \mathcal{F}\rangle$, we construct a coBüchi automaton $\mathcal{C} = \langle L, \ell_0, 2^\Sigma, \delta_\mathcal{C}, \mathcal{F}_\mathcal{C}\rangle$ such that U is absorbing iff $L(\mathcal{C}) = \emptyset$. The coBüchi automaton $\mathcal{C}$ is exponential in the size of $\mathcal{A}$, and as a consequence of NLOGSPACE-completeness of the emptiness problem for coBüchi automata, the PSPACE bound follows.

The automaton $\mathcal{C}$ is constructed to guess an infinite word w as a witness, to prove that the terminal end-component U is not absorbing. This word induces an infinite sequence of supports $s_0s_1s_2\ldots$ produced by its outcome $X_0X_1X_2\ldots$ (i.e., $s_i = \mathsf{Supp}(X_i)$ for all $i \in \mathbb{N}$). At some point n, there is a subset $s \subseteq s_n$ form which U cannot be reached. Therefore, the states of $\mathcal{C}$ keeps track of the support $s_i = \mathsf{Supp}(X_i)$ of the outcome, and at some point guesses the set s and checks that U is never reached from states in s. Then the acceptance condition requires that eventually all the reached states are outside of the end-component U. Since, by Lemma 4, the pure words are not sufficient, the alphabet of $\mathcal{C}$ is 2^Σ. A word over this alphabet is a sequence of subsets of letters which can be viewed as the sequence of supports of the distributions of a randomized word.

The construction of $\mathcal{C} = \langle L, \ell_0, 2^\Sigma, \delta_\mathcal{C}, \mathcal{F}_\mathcal{C}\rangle$ is as follows:

- $L = 2^Q \times \{0, 1\}$.
- $\ell_0 = (\mathsf{Supp}(\mu_0), 0)$ is the initial state.
- $2^\Sigma \setminus \{\emptyset\}$ is the alphabet.
- $\delta_\mathcal{C} : L \times 2^\Sigma \to 2^L$ is the transition function defined as follows. For all $s \subseteq Q$ and $\Sigma' \subseteq \Sigma$, let $s' = \mathsf{Post}(s, \Sigma')$ and define $\delta_\mathcal{C}((s, 0)) = \{(s', 0)\} \cup \{(s'', 1) \mid s'' \neq \emptyset \wedge s'' \subseteq s' \setminus U\}$ and define $\delta_\mathcal{C}((s, 1)) = \{(s', 1)\}$ if $s' \cap U = \emptyset$, and $\delta_\mathcal{C}((s, 1)) = \emptyset$ otherwise.
- and $\mathcal{F}_\mathcal{C} = 2^Q \times \{1\}$ is the coBüchi acceptance condition.

Another necessary condition to have a universal strongly (resp., weakly) synchronizing language for a probabilistic automaton is that the uniformly randomized word is synchronizing as well. For instance, the automaton presented in Fig. 4 has an absorbing end-component, but since the uniformly randomized word is not strongly synchronizing, the strongly synchronizing language is not universal.

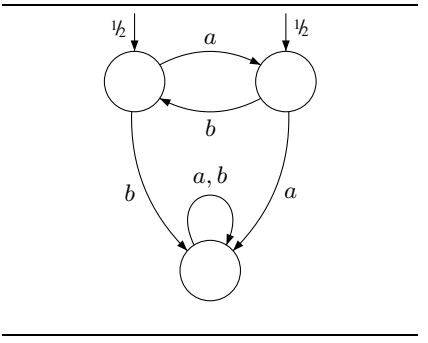


Fig. 5. Non-absorbing end-component

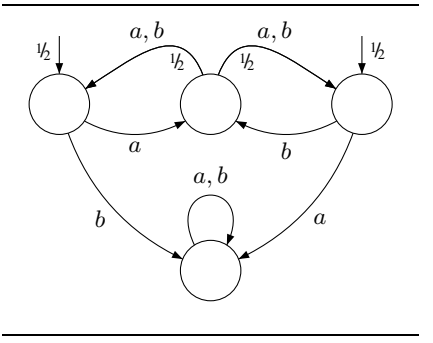


Fig. 6. Absorbing end-component

Lemma 6. *The universality problem for strongly synchronizing language and for weakly synchronizing language of probabilistic automata is decidable in PSPACE.*

We state the main idea of the proof of Lemma 6 for strongly synchronizing languages. The detailed proof can be found in [6]. The proof for weakly synchronizing languages follows an analogous discussion which is left to the reader.

We establish the following characterization. The synchronizing language of a given PA $\mathcal{A}$ is universal iff (I) there is a (then necessarily unique) absorbing end-component in $\mathcal{A}$, and (II) the uniformly randomized word w_u is strongly (resp., weakly) synchronizing. The above arguments show that these conditions are necessary and we now briefly explain why they are also sufficient. Since the uniformly randomized word w_u is strongly synchronizing, it can be shown that the unique terminal end-component U of $\mathcal{A}$ consists of a simple cycle, in the sense that $|\mathsf{Post}(q, \Sigma)| = 1$ for all states $q \in U$. It follows that if word w is not strongly synchronizing, then two different states of U would be reached after the same number of steps. But since all states reachable by w are also reachable by w_u, it would mean that w_u is not strongly synchronizing, a contradiction.

Condition (I) can be checked in PSPACE by Lemma 5, and Condition (II) reduces to check that a Markov chain is synchronizing, which can be done in polynomial time by steady state analysis. The PSPACE bound follows.

Lemma 7. *The universality problem for strongly synchronizing language and for weakly synchronizing language of probabilistic automata is PSPACE-hard.*

Proof. We present a proof using a reduction from a PSPACE-complete problem so called *initial state problem*. Given a nondeterministic finite automaton $\mathcal{N} = \langle Q, q_0, \Sigma, \delta, \mathcal{F}\rangle$ and a state $q \in Q$, we denote by $\mathcal{N}_q$ the automaton $\mathcal{N}$ in which the initial state is q, i.e. $\mathcal{N}_q = \langle Q, q, \Sigma, \delta, \mathcal{F}\rangle$. The *initial state problem* is to decide, given $\mathcal{N}$, whether the exists a state $q \in Q$ and a word $w \in \Sigma^\omega$ such that all runs r of $\mathcal{N}_q$ over w avoid $\mathcal{F}$, i.e. $r_i \notin \mathcal{F}$ for all $i \geq 0$. From the results of [4,14], it follows that the initial state problem is PSPACE-complete. We present a polynomial-time reduction from the the initial state problem to the universality problem, establishing the PSPACE hardness of the universality problem.

Given an NFA $\mathcal{N} = \langle L, \ell_0, \Sigma, \delta_{\mathcal{N}}, \mathcal{F}_{\mathcal{N}} \rangle$ with $\mathcal{F}_{\mathcal{N}} \neq \emptyset$, we construct a PA $\mathcal{A} = \langle Q, \mu_0, \Sigma, \delta \rangle$ as follows:

- $Q = L \cup \{q_{end}\}$.
- $\mu_0(\ell_0) = 1$, and $\mu_0(q) = 0$ for all $q \in Q \setminus \{\ell_0\}$.
- $\delta : Q \times \Sigma \to \mathcal{D}(Q)$ is the probabilistic transition function defined as follows. For all $q \in Q$ and $\sigma \in \Sigma$, if $q \notin \mathcal{F}$, then $\delta(q, \sigma)$ is the uniform distribution over $\delta_{\mathcal{N}}(q, \sigma)$, and if $q \in \mathcal{F}_{\mathcal{N}}$, $\delta(q, \sigma)(q') = \frac{1}{2|\delta_{\mathcal{N}}(q,\sigma)|}$ for all $q' \in \delta_{\mathcal{N}}(q, \sigma)$ and $\delta(q, \sigma)(q_{end}) = \frac{1}{2}$.

We show that the answer to the initial state problem for $\mathcal{N}$ is YES if and only if $\mathcal{A}$ is not universal. We assume w.l.o.g that all states in $\mathcal{N}$ are reachable. First, if the answer to the initial state problem for $\mathcal{N}$ is YES, then let $\hat{q}$ be an initial state and $w \in \Sigma^\omega$ be a word satisfying the problem. We construct a word that is not (strongly neither weakly) synchronizing for $\mathcal{A}$. First, consider the $|Q|$-times repetition of the uniform distribution d_u over Σ. Then, with positive probability the state q_{end} is reached, and also with positive probability the state $\hat{q}$ is reached, say after k steps. Let $w' \in \Sigma^\omega$ such that $w = v \cdot w'$ and $|v| = |Q| - k$. Note that from state $\hat{q}$ the finite word v is played with positive probability by the repetition of uniform distribution d_u. Therefore, on the word $(d_u)^{|Q|} \cdot w'$, with some positive probability the set q_{end} is never reached, and thus it is not synchronizing, and $\mathcal{A}$ is not universal. Second, if $\mathcal{A}$ is not universal, then the terminal end-component $\{q_{end}\}$ is not absorbing and by the construction in Lemma 5, there exists a state $\hat{q}$ and a pure word $w \in \Sigma^\omega$ such that all runs from $\hat{q}$ on w avoid q_{end}, and therefore also avoid $\mathcal{F}_{\mathcal{N}}$. Hence, the answer to the initial state problem for $\mathcal{N}$ is YES. □

The following result follows from Lemma 6, and Lemma 7.

Theorem 3. *The universality problem for strongly synchronizing language and for weakly synchronizing language of probabilistic automata is PSPACE-complete.*

5 Discussion

The complexity results of this paper show that both the emptiness and the universality problems for synchronizing languages are PSPACE-complete. The results in this paper apply also to a more general definition of synchronizing sequence of probability distribution, where groups of equivalent states are clustered together. A labeling function assigns a color to each group of equivalent states. The definition of synchronizing sequences then corresponds to the requirement that the automaton essentially behaves deterministically according to the sequence of colors produced in the long run. A labeled probabilistic automaton is a PA $\mathcal{A}\langle Q, \mu_0, \Sigma, \delta \rangle$ with a labeling function $\mathsf{L} : Q \to \Gamma$ where Γ is a finite set of colors. The L-norm of a probability distribution $X \in \mathcal{D}(Q)$ is $\|X\|_{\mathsf{L}} = \max_{\gamma \in \Gamma} \sum_{q:\mathsf{L}(q)=\gamma} X(q)$, and a sequence $X_0 X_1 \ldots$ is strongly synchronizing (resp., weakly synchronizing) if $\liminf_{n\to\infty} \|X_n\|_{\mathsf{L}} = 1$, (resp., $\limsup_{n\to\infty} \|X_n\|_{\mathsf{L}} = 1$). The constructions of ω-automata in Lemma 1 and Lemma 2 can be adapted to show that the emptiness problem remains in PSPACE for

labeled probabilistic automata. Roughly, the ω-automaton will guess the witness sequence $\hat{\gamma}_i\hat{\gamma}_{i+1}\ldots$ of colors rather than a witness sequence of states. The solution of the universality problem is adapted analogously.

References

1. Baier, C., Bertrand, N., Größer, M.: On decision problems for probabilistic büchi automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 287–301. Springer, Heidelberg (2008)
2. Baier, C., Größer, M.: Recognizing omega-regular languages with probabilistic automata. In: Proc. of LICS: Logic in Comp. Science, pp. 137–146. IEEE, Los Alamitos (2005)
3. Benenson, Y., Adar, R., Paz-Elizur, T., Livneh, Z., Shapiro, E.: DNA molecule provides a computing machine with both data and fuel. Proc. National Acad. Sci. 100, 2191–2196 (2003)
4. Chadha, R., Sistla, A.P., Viswanathan, M.: On the expressiveness and complexity of randomization in finite state monitors. In: Proc. of LICS: Logic in Computer Science, pp. 18–29. IEEE Comp. Soc., Los Alamitos (2008)
5. Doyen, L., Massart, T., Shirmohammadi, M.: Synchronizing objectives for Markov decision processes. In: Proc. of iWIGP, pp. 61–75 (2011)
6. Doyen, L., Massart, T., Shirmohammadi, M.: Infinite synchronizing words for probabilistic automata. Tech. Rep, 138, Université Libre de Bruxelles (ULB), Belgium (June 2011)
7. Kfoury, D.J.: Synchronizing sequences for probabilistic automata. Studies in Applied Mathematics 29, 101–103 (1970)
8. Korthikanti, V.A., Viswanathan, M., Kwon, Y., Agha, G.: Reasoning about mdps as transformers of probability distributions. In: Proc. of QEST: Quantitative Evaluation of Systems, pp. 199–208. IEEE Computer Society, Los Alamitos (2009)
9. Kwon, Y., Agha, G.: Linear inequality ltl (iltl): A model checker for discrete time Markov chains. In: ICFEM, pp. 194–208 (2004)
10. Paz, A.: Introduction to Probabilistic Automata. Academic Press, New York (1971)
11. Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230–245 (1963)
12. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theor. Comput. Sci. 49, 217–237 (1987)
13. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proceedings of the 5th Annual Symposium on Theory of Computing, pp. 1–9. ACM Press, New York (1973)
14. Tracol, M., Baier, C., Größer, M.: Recurrence and transience for probabilistic automata. In: Proc. of FSTTCS: Foundations of Software Technology and Theoretical Computer Science. LIPIcs, vol. 4, pp. 395–406. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
15. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

Characterizing EF over Infinite Trees and Modal Logic on Transitive Graphs

Balder ten Cate[1,⋆] and Alessandro Facchini[2,⋆⋆]

[1] University of California, Santa Cruz
[2] Warsaw University

Abstract. We provide several effective equivalent characterizations of EF (the modal logic of the descendant relation) on arbitrary trees. More specifically, we prove that, for EF-bisimulation invariant properties of trees, being definable by an EF formula, being a Borel set, and being definable in weak monadic second order logic, all coincide. The proof builds upon a known algebraic characterization of EF for the case of *finitely branching* trees due to Bojańczyk and Idziaszek. We furthermore obtain characterizations of modal logic on transitive Kripke structures as a fragment of weak monadic second order logic and of the μ-calculus.

1 Introduction

Determining effective characterizations of logics on trees is an important problem in theoretical computer science, motivated by the desire to better understand the expressive power of logics such as first-order logic (FO) or the temporal logic CTL^* on trees.

While for finite words this kind of problem is well-studied, the list of results for trees is much more frugal. The situation has improved a little in the last years, thanks to the successful use of *forest algebras* (see [9]). Notably, this formalism has been used for obtaining decidable characterizations for the classes of tree languages definable in $\mathsf{EF} + \mathsf{EX}$ [8], $\mathsf{EF} + \mathsf{F}^{-1}$ [4,15], BC-$\Sigma_1(<)$ [6,15], $\Delta_2(\leq)$ [7,15], in FO with the child relation and in FO with counting quantifiers (modulo an integer) and the child relation [1]. This approach has then been extended in the case of the temporal logic EF on infinite but finitely branching trees by Bojańczyk and Idziaszek [5].

These results all demonstrate the importance of the algebraic approach to obtaining decidable characterizations of logics. In the case of infinite trees, it is natural to ask whether such logics also admit *topological* characterizations. Take for instance the logic EF, the modal logic of the descendant relation, which is a fragment of the modal μ-calculus. The formula $\mu x.\Box x$ of the modal μ-calculus, which defines the class of well-founded trees, is not equivalent to any EF-formula. This follows from results of Bojańczyk and Idziaszek [5] since the syntactic ω-forest algebra of the class of finitely branching trees satisfying $\mu x.\Box x$ fails to satisfy one of the algebraic equations that characterize the expressive power of EF on finitely branching trees. However, there is another explanation for the fact that $\mu x.\Box x$ is not equivalent to an EF-formula, which involves topology: the class of (arbitrarily branching) well-founded trees is not Borel,

⋆ The first author was supported by the NSF grant IIS-0905276.
⋆⋆ The second author is supported by a grant from the SNFS, n. PBLAP2-132006.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 290–302, 2011.

while EF-formulae can only define tree languages that are Borel. This raises the question whether EF, as a fragment of MSO, can be characterized by topological means.

In this work we give a positive answer to this question. We prove that a (not necessarily finitely branching) tree language is EF-definable if and only if it is invariant for EF-bisimilarity and Borel. Since EF is a fragment of *weak monadic-second order logic* (WMSO) and all WMSO-definable tree languages are Borel, we obtain as a corollary that EF is the EF-bisimulation invariant fragment of WMSO. All the proofs make crucial use of the results in [5]. Secondly, these characterizations are effective: given an MSO formula, one can effectively test whether it defines an EF-definable tree language. Thirdly, these characterizations show that for EF-bisimilar invariant languages WMSO is the Borel fragment of MSO. Since for all tree languages WMSO and MSO are incomparable (cf. Remark 1 in Subsection 2.2), this cannot be true in general. It is however an open question whether such a relative characterization of WMSO holds when restricted to *complete* binary trees (cf. [13]).

Theorem 1. *Let L be any* MSO*-definable tree language. The following are equivalent and decidable:*

(1) L is EF*-definable,*
(2) L is closed under EF*-bisimulation and* WMSO*-definable,*
(3) L is closed under EF*-bisimulation and Borel,*
(4) L is closed under EF*-bisimulation, and for every L-idempotent context c, and for every forest f, $c(f)$ and $(c + cf)^\infty$ are L-equivalent.*

Note that here, we consider the monadic second-order language with the child relation and without a horizontal sibling-order relation.

The fourth condition, which is essentially the condition that was used in [5] to characterize EF on finitely branching trees, involves some algebraic notions that we will introduce in Section 2. Thm. 1 does not hold for *finitely* branching trees. This is because on finitely branching trees, well-foundedness is equivalent to finiteness, which is Borel and closed under EF-bisimulation but not EF-definable.

From Thm. 1, we finally obtain the following effective characterizations of modal logic on transitive Kripke models (with a suitable definition of what it means for a class of pointed Kripke models to be Borel):

Theorem 2. *For every μ-formula ϕ, the following are equivalent and decidable:*

(1) ϕ is equivalent on transitive Kripke models to a modal formula,
(2) ϕ is equivalent on transitive Kripke models to a WMSO*-formula,*
(3) the class of pointed transitive Kripke models satisfying ϕ is Borel.

Theorem 3. *For every* WMSO*-formula $\phi(x)$, the following are equivalent:*

(1) $\phi(x)$ is equivalent on transitive Kripke models to a modal formula,
(2) $\phi(x)$ is bisimulation invariant on transitive Kripke models.

Since EF is a fragment of first-order logic (with the descendant relation), Thm. 1, 2 and 3 imply that EF is the EF-bisimulation invariant fragment of FO on trees and that modal logic is the bisimulation invariant fragment of FO on transitive Kripke models. Both were known (cf. [11, Thm. 4.12]).

2 Preliminaries and Groundwork

2.1 The Beauty of Trees

Trees, forests, contexts. A colored directed graph g over a finite alphabet Σ, called a Σ-colored directed graph, is given by a triple $(V_g, \prec_g, \lambda_g)$ such that the domain V_g is a (possibly empty) set of nodes, $\prec_g \subseteq V_g \times V_g$ is a directed edge (or arrow) relation and $\lambda_g : V_g \to \Sigma$ is a coloring function. A node $x \in V_g$ is called a root if there is no node $y \in V_g$ such that $y \prec_g x$. A path from a node x to a node y is a sequence $(x_1, \ldots, x_n)$ of nodes such that $x_1 = x$, $x_n = y$ and for every $0 < i < n$, $x_i \prec_g x_{i+1}$. We say that a node x is reachable from a node y if there is a path from y to x. A pointed Σ-colored directed graph is a pair (g, s) where g is a Σ-colored directed graph and $s \in V_g$.

Unordered trees (*trees*, for short) will be colored directed graphs with a unique root (if the graph is not empty) and where every node is reachable along a unique path from the root. Given two nodes $x, y \in V_g$, we say that y is a son of x and x is a parent of y if $x \prec_g y$. In general, if there is a path of length at least 2 from x to y, then x is called an *ancestor* of y and y is called a *descendant* of x. A node that is not an ancestor of any node of a tree is called a *leaf*. A tree t is said to be *finitely branching* if for every node x, the set of sons of x is finite. The depth of a node x in a tree t is the length of the path from the root of t to x. Given a tree $t = (V_t, \prec_t, \lambda_t)$ and node $x \in V_t$, the subtree rooted in x is the tree $t.x = (V_{t.x}, \prec_t |_{V_{t.x}}, \lambda_t|_{V_{t.x}})$ where $V_{t.x}$ is the set of all reachable nodes of V_t from x, $\prec_t |_{V_{t.x}} = \prec_t \cap (V_{t.x} \times V_{t.x})$ and $\lambda_t|_{V_{t.x}} = \lambda_t \cap (V_{t.x} \times \Sigma)$. A tree is called *regular* if, up to isomorphism, it has only finitely many subtrees. The set of all trees over a finite alphabet Σ is denoted by T_Σ. A subset $L \subseteq T_\Sigma$ is called a *Σ-tree language*, or simply a tree language.

A forest over a finite set Σ is a colored directed graph over a finite alphabet Σ with possibly more than one root (and at least one if the domain is not empty) but such that any node is reached by an unique path from a single root. Given a forest f, and a node x of its domain, by $f.x$ we denote the subtree of f rooted in x. A context is a forest with a hole. Formally, a context over Σ is a forest over $\Sigma \cup \{\blacksquare\}$ where exactly one node is labeled by "$\blacksquare$", and it is a leaf but not a root. As usual, we do not distinguish between two isomorphic trees or forests.

Operations on forests and contexts. We define two types of operations on forests and contexts: a (horizontal) *concatenation* operation, denoted by $+$, and a (vertical) *composition* operation, denoted by $\cdot$. Contrary to the concatenation operation, the composition operation is not commutative.

Given a sequence of forests $(f_i : i \in \alpha)$, with $\alpha \in \omega \cup \{\omega\}$, we want to concatenate these forests. Note that each forest can contain infinitely many rooted subtrees, and the length of the sequence itself, i.e., α, can be infinite. The concatenation of the forests f_i is defined as the forest $\sum_{i \in \alpha} f_i$ obtained by taking the disjoint union of all forests f_i. Since we identify isomorphic forests, this operation is commutative and associative.

We allow also to concatenate a sequence of forests where one of them is a context. Clearly the result of such a concatenation is a context.

Concerning vertical composition, we can compose a context c with a forest, resp. a context, f, and obtain as a result a forest, resp. a context, $c(f)$ simply by replacing the

hole node of c by f. More formally, let c be the context $(V_c, \prec_c, \lambda_c)$ and f be the forest $(V_f, \prec_f, \lambda_f)$. Without loss of generality, we can assume that $V_c \cap V_f = \emptyset$. Let $h \in V_c$ be the unique node labelled by "■" and $h_0 \in V_c$ be its unique parent, and let ρ_f be the set of roots of f. Then $c(f)$ is the forest defined by $((V_c \setminus \{h\}) \cup V_f, \prec', \lambda')$ where

- $\prec' |_{V_c \setminus \{h\}} = \prec_c |_{V_c \setminus \{h\}}$ and $\prec' |_{V_f} = \prec_f$, and for all $x \in \rho_f$, $h_0 \prec' x$,
- $\lambda' |_{V_c \setminus \{h\}} = \lambda_c |_{V_c \setminus \{h\}}$ and $\lambda' |_{V_f} = \lambda_f$.

For every pair of contexts c, c' and every forest f, we have that $c(c'(f)) = (c(c'))(f)$.

Note that, restricted to finitely (unordered) branching contexts and forests, the operations of concatenation and composition correspond to the ones in [5].

Myhill-Nerode equivalence. Given a tree language L, we want to define a notion of equivalence with respect to L, analogous to the well known Myhill-Nerode equivalence relation for finite words. Intuitively, two forests, or two contexts, are L-equivalent if they "*behave the same*" with respect to L. The definition we use, which we will now present, is essentially the one in [5] (cf. Remark 2).

The crucial notion here is that of a *template*. There are two kinds of templates, *forest-templates* and *context-templates*. A forest-template for an alphabet Σ is a forest over the alphabet $\Sigma \cup \{\star\}$ in which one or more leafs (possibly infinitely many) are labeled $\star$, and no non-leaf node is labeled $\star$. Similarly, a context-template for Σ is a forest over the alphabet $\Sigma \cup \{\star\}$ in which one or more nodes (possibly infinitely many) are labeled "$\star$". Intuitively, the occurrences of "$\star$" in a forest-template are placeholders for forests, and the occurrences of "$\star$" in a context-template are placeholders for contexts.

In what follows, we will make use of the operation of replacing a subtree with another forest. We will not give a formal definition of this operation. Just note that it can be defined in a straightforward manner by using horizontal concatenation and a generalization of the substitution operation used in defining vertical composition.

Thus, given a forest-template f over $\Sigma \cup \{\star\}$ and a forest g over Σ, we denote by $f[\star \mapsto g]$ the forest obtained by replacing every node labeled "$\star$" in f with the forest g. Similarly, given a context-template f over $\Sigma \cup \{\star\}$ and a context c over Σ, we denote by $f[\star \mapsto c]$ the forest obtained by replacing every subtree starting at a node labeled "$\star$" in f with the forest obtained by composing the context c with the (possibly empty) forest given by all the subtrees rooted at a child of the considered node labeled "$\star$". In the special case where f is the infinite unary tree labeled by "$\star$", then for every context c, $f[\star \mapsto c]$ will be denoted also by c^∞.

We call a forest-template *guarded* if no root is labelled by "$\star$". Analogously for context-templates. Let $L \subseteq T_\Sigma$ be a tree language over an alphabet Σ. We say that two contexts, c_1, c_2, over Σ are L-equivalent (denoted by $c_1 \equiv_L c_2$) if for every guarded context-template t over alphabet $\Sigma \cup \{\star\}$, $t[\star \mapsto c_1] \in L$ iff $t[\star \mapsto c_2] \in L$. Similarly, two forests f_1, f_2 are L-equivalent (denoted by $f_1 \equiv_L f_2$) if for every guarded forest-template t over $\Sigma \cup \{\star\}$, $t[\star \mapsto f_1] \in L$ iff $t[\star \mapsto f_2] \in L$.

Note that since we are working with *tree* languages, we can safely ignore forests $t[\star \mapsto f]$ or $t[\star \mapsto c]$ that are not trees. Moreover, note that two trees (seen as forests) can be L-equivalent while they do not agree on membership in L.

Proposition 1. *L-equivalence is an equivalence relation (both on contexts and on forests).*

There is a natural generalization of forest-templates and context-templates, where the holes are marked. More precisely, multi-forest-templates and multi-context-templates for an alphabet Σ are forests over an extended alphabet $\Sigma \cup \{\star_1, \ldots, \star_n\}$, satisfying the same conditions as forest-templates and context-templates. Given a multi-forest-template t over $\Sigma \cup \{\star_1, \ldots, \star_n\}$ and forests $f_1, \ldots, f_n$, the forest obtained by replacing every node labeled "$\star_i$" in t with the forest f_i, with $i = 1, \ldots, n$ is denoted by $t[\star_1 \mapsto f_1, \ldots, \star_n \mapsto f_n]$. Similarly for multi-context-templates.

The next lemma for L-equivalence and multi-forest-templates will be very useful:

Lemma 1. *Let L be any Σ-tree language and f any multi-forest-template over the extended alphabet $\Sigma \cup \{\star_1, \ldots, \star_n\}$. Consider two finite sequences of Σ-forests $(g_1, \ldots, g_n)$ and $(h_1, \ldots, h_n)$ such that g_i and h_i are L-equivalent, for each $i = 1, \ldots, n$. Then $f[\star_1 \mapsto g_1, \ldots, \star_n \mapsto g_n]$ and $f[\star_1 \mapsto h_1, \ldots, \star_n \mapsto h_n]$ are also L-equivalent.*

The previous lemma can analogously be shown to hold for multi-context templates.

From now on, when speaking about forest, templates we always mean *guarded* forest, analogously with context.

2.2 Monadic Second Order Logics

A colored directed graph g over an alphabet Σ can be viewed as a relational structure

$$M_g := \langle V_g, \prec_g, (P_a^g : a \in \Sigma) \rangle$$

where P_a^g are unary predicates with the interpretation $P_a^g = \{v \in V_g : \lambda_g(v) = a\}$. Colored directed graphs, and in particular forests, as relational structures can be described in first- or second-order logics with the child relation ($\prec$) plus unary predicates. In this paper we are interested in monadic second-order logic (MSO) and in weak monadic second-order logic (WMSO).

Let $\mathsf{Var}_1 = \{x, y, \ldots\}$ be a countable infinite set of first-order variables, and $\mathsf{Var}_2 = \{X, Y, \ldots\}$ a countable infinite set of monadic second-order variables. Given a finite alphabet Σ, the set of MSO-formulae over Σ is defined by the following grammar:

$$\phi ::= x \prec y \mid x = y \mid P_a(x) \mid X(x) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x\phi \mid \exists X\phi$$

with $a \in \Sigma$, $x \in \mathsf{Var}_1$, $X \in \mathsf{Var}_2$. A formula with no free variables is called a *sentence*.

The semantics of MSO-formulae is given in terms of valuations. Let g be a colored directed graph. We can identify a valuation val over g with a pair of functions $(\mathsf{val}_1, \mathsf{val}_2)$ where $\mathsf{val}_1 : \mathsf{Var}_1 \to V_g$ and $\mathsf{val}_2 : \mathsf{Var}_2 \to \wp(V_g)$ (we use $\wp$ to denote powerset). Thus, a valuation for a (nonempty) colored directed graph g assigns to every first-order variable an element from the domain and to each second order variable a set of elements from the domain. A *weak* valuation is a valuation that assigns to each monadic second-order variable a finite set, i.e., such that $\mathsf{val}_2 : \mathsf{Var}_2 \to \wp_{\mathsf{fin}}(V_g)$.

The meaning of a formula ϕ in a nonempty colored directed graph g relative to a valuation val for g is then defined in a standard way, using arbitrary valuations or using weak valuations. When we say that a formula is an MSO-formula, resp. a WMSO-formula, we mean that the formula is interpreted using arbitrary, resp. weak valuations.

Remark 1. The adjective "weak" is a bit misleading, since WMSO is in general not a fragment of MSO. Indeed, the class of finitely branching trees is not definable in MSO (because, as we will see, every MSO-formula that is satisfiable on trees is true of some finitely branching tree) but is defined by the WMSO-formula $\forall x \exists X \forall y (x \prec y \rightarrow Xy)$. The class of well-founded trees is definable in MSO but not in WMSO, as will follow from results we discuss below (in particular, from the fact that the class of well-founded trees is not Borel). On finitely branching trees, WMSO *is* strictly less expressive than MSO. This follows from the fact that on finitely branching trees "X is a finite set" is expressed by the MSO-formula $\forall Y (\forall x (Yx \rightarrow Xx) \wedge \forall y (Yy \rightarrow \exists z (Yz \wedge y \prec z)) \rightarrow \neg \exists y (Yy))$ ("X does not contain an infinite path").

Given an MSO sentence ϕ over the alphabet Σ, the tree language defined by ϕ is the set of non empty trees:

$$L(\phi) = \{t \in T_\Sigma \setminus \{\emptyset\} : \phi \text{ is true in } M_t\}$$

Analogously for a WMSO-formula. We say that a tree language L is (W)MSO-definable if there is a (W)MSO-formula ϕ such that $L = L(\phi)$.

For $n \geq 1$, we denote by $\equiv_n$ the equivalence relation that holds between two structures if they cannot be distinguished by an MSO-sentence of quantifier depth n. It is well known that, for each $n \geq 1$, over a finite signature, there are only finitely many $\equiv_n$-classes of structures. It follows that every structure can be completely described up to $\equiv_n$-equivalence by a single MSO-sentence of quantifier depth n. Furthermore, the equivalence relation $\equiv_n$ can be characterized using a variant of Ehrenfeucht-Fraïssé games. These games are similar to the ones for first-order logic, except that there is a second type of move, where Spoiler selects a set of elements in one of the two structures and Duplicator responds with a corresponding set of elements in the other structure (in order for Duplicator to win the game, the resulting bijection should not only be a partial isomorphism with respect to the relations in the structures, but should also preserve membership in the chosen sets). As in the first-order case, Duplicator has a winning strategy in the n-round game ($n \geq 1$) iff the two structures satisfy the same MSO-formulae of quantifier depth at most n.

Proposition 2. *If L is an* MSO*-definable tree language, then there are only finitely many L-equivalence classes of forests and contexts. Moreover, every L-equivalence class (of forests and of contexts) has a finitely branching and regular member.*

Remark 2. It can be naturally asked what is the relation between the introduced notion of L-equivalence and the one originally introduced by Bojańczyk and Idziaszek. It follows by the previous Prop. 2 that if L-equivalence were defined using regular forest-templates and regular context-templates only, the result would be the same as L-equivalence the way we defined, assuming that L is a MSO definable tree language. This means that our definition essentially coincides with the one used in [5].

We still need a slightly more fine-grained version of Prop. 2. Let c be a context and f a forest. By a *forest built from c and f*, we will mean a forest g for which there exists a forest s such that replacing each non-leaf node in s by a copy of c and replacing each leaf-node of s by a copy of f yields g. We then say that s is the *skeleton* of g.

Proposition 3. *Let L be any* MSO*-definable tree language. Let c be a context, f a forest and g a forest built from c and f whose skeleton satisfies some* MSO*-sentence ψ. Then there is an L-equivalent forest g' built from c and f, whose skeleton is regular, finitely branching and satisfies ψ as well.*

2.3 The Logic EF and EF-Bisimulation

Fix an alphabet Σ. The set of formulae of EF over Σ is defined by the grammar

$$\phi ::= a \mid \phi \wedge \phi \mid \neg\phi \mid \mathsf{EF}\phi \qquad (a \in \Sigma)$$

The semantics of EF over non empty Σ-trees (where the distinguished node is the root of the tree) is defined inductively as usual by saying that every EF-formula $a \in \Sigma$ is true in trees with root label a and that an EF formula $\mathsf{EF}\phi$ is true in trees that have a proper subtree where ϕ is true. For any EF formula ϕ, the class of trees where ϕ is true is denoted by $L(\phi)$. Given a tree language L, we say that L is EF-definable if there is an EF-formula ϕ such that $L = L(\phi)$. The following is well-known (see, e.g., [3]):

Proposition 4. *Every* EF*-definable tree language is also* WMSO*-definable (and, in fact, definable in first-order logic with the descendant relation).*

Following [5], we introduce a special bisimilarity game on forests, called the EF bisimulation game. We first define the game in the case of trees. Let t_0 and t_1 be two trees. The EF bisimulation game over t_0 and t_1 is played by two players: Bob and Anne. The game proceeds in rounds. At the beginning of each round, the state in the game is a pair of trees (t'_0, t'_1). A round is played as follows. First if the root labels a_0, a_1 of t'_0, t'_1 are different, then Bob wins the whole game. Otherwise Bob selects one of the trees t'_i, for $i = 0, 1$, and its subtree s_i. Then Anne selects a subtree s_{1-i} in the other tree t'_{1-i}. The round is finished, and a new round is played with the state updated to (s_0, s_1). If Anne can survive for infinitely many rounds in the EF bisimulation game on t_0 and t_1, then we say that the trees t_0 and t_1 are EF-bisimilar.

Note that clearly if two trees are bisimilar in the standard way, they also are EF-bisimilar. The converse need not to be true. Consider for example the binary tree t on the alphabet $\{a, b\}$ where the only nodes labelled by a are the nodes 0^{2k+1}, with $k > 0$, and the tree t' on the alphabet $\{a, b\}$ where the only nodes labelled by a are the nodes 0^{2k}, with $k > 0$. The two trees are EF-bisimilar but not bisimilar.

A tree language L is called *invariant under* EF*-bisimulation* if it is impossible to find two trees, $t_0 \in L$ and $t_1 \notin L$ that are EF-bisimilar. From the previous remark on the interrelation between standard bisimilarity and EF bisimilarity, if two tree languages are EF-bisimilar they are also bisimilar, but the converse is in general not true.

This game is so designed that all tree languages defined by an EF formula are invariant under EF-bisimulation. Formally, we have that:

Proposition 5 ([5]). *Every* EF*-definable tree language is invariant under* EF*-bisimulation.*

The converse is false. A counter-example is the language of all well-founded trees over a fixed finite alphabet. This language is invariant under EF-bisimulation but it cannot be defined by an EF-formula, as follow from Thm. 1.

We say that a context c is L-*idempotent* if the composition $c(c)$ is L-equivalent to c.

Theorem 4. *An* MSO *definable tree language L can be defined by an* EF *formula iff*
(1) L is invariant under EF*-bisimulation;*
(2) for every L-idempotent context c and for every forest f, $c(f) \equiv_L (c + c(f))^\infty$
Moreover the previous two conditions are decidable.

Remark 3. This characterization of the logic EF was proved by Bojańczyk and Idziaszek in [5] for the case of finitely branching trees. It follows from Prop. 2 that the result holds for arbitrarily branching trees as well. Strictly speaking, the result in [5] is different as it characterizes definability by EF-formulae that are Boolean combination of formulae of the form $\mathsf{EF}\phi$. However, as explained in [5], this is not an essential restriction. More precisely, call an EF-formula ψ a EF-*forest formula* if ψ is a Boolean combinations of formulae of the form $\mathsf{EF}\phi$. Then, one can prove that every EF-formula ϕ is logically equivalent to $\bigwedge_{a\in A}(a \to \phi_a)$, where each ϕ_a is an EF-forest formula. This means that a tree language L is EF-definable iff, for every $a \in A$, the language L_a is definable by an EF-forest formula, where a tree t is said to be in L_a iff the tree obtained from t by relabeling its root with a is in L. Thus, the equivalence relation $\equiv_L$ in the previous theorem is, strictly speaking, the finite intersection of all $\equiv_{L_a}$.

We extend the notion of EF-bisimilarity to forests by saying that two forests f_1, f_2 are EF-bisimilar if the trees obtained from f_1 and f_2 by adding a "fresh" root, are EF-bisimilar. More precisely, let f_1 and f_2 be two forests over Σ. Let t_1 and t_2 be any pair of trees over $\Sigma \cup \{a\}$, $a \notin \Sigma$, such that: (i) each forest f_i is obtained from the corresponding tree t_i by removing the root node and (ii) $t_1(\varepsilon) = t_2(\varepsilon) = a$. Then we say that f_1 and f_2 are EF-bisimilar if the trees t_1 and t_2 are EF-bisimilar. Note that for forests f_1 and f_2 consisting of a single tree t_1 and t_2 respectively, saying that f_1 and f_2 are EF-bisimilar is not the same as saying that t_1 and t_2 are EF-bisimilar.

The following lemma, relating EF-bisimilarity to L-equivalence, will come in handy later on. Call two contexts EF-bisimilar, if they are bisimilar when viewed as forests over an alphabet containing an additional label "■".

Lemma 2. *Let L be any* EF*-bisimulation-invariant tree language. Then every two* EF*-bisimilar forests are L-equivalent and every two* EF*-bisimilar contexts are L-equivalent.*

2.4 The Topological Complexity of Tree Languages

A topological space is a set X together with a collection τ of subsets of X (called the open sets) containing the empty set and X itself, such that τ is closed under arbitrary unions and finite intersections. Subsets of a topological space X can be classified according to their topological complexity, where the open sets and their complements (the closed sets) are considered to have the lowest complexity. In this context, the notion of a Borel set naturally arises. The set $\mathbf{Borel}(X)$ of Borel sets of a topological space (X, τ) is the smallest set that contains all open sets of X, and is closed under countable unions and complementation. The Borel sets of a topological space can be further classified into an infinite hierarchy, but this will be irrelevant for present purposes.

The topology we will consider here on trees is the *prefix topology*, where the open sets are, intuitively, those sets of trees for which membership of a tree is witnessed by a finite-depth prefix of the tree. Thus, for example the set of trees containing a node labeled a is an open set, but the complement is not.

Formally, for a Σ-tree t and a natural number $n \geq 1$, the *depth-n prefix* of t, denoted by $t^{(n)}$, is the Σ-tree obtained by restricting the domain of t to all the elements of V_t of depth at most n. We say that two trees t, t' are *equivalent up to depth* n if $t^{(n)} = t'^{(n)}$. We call a set X of trees open if for each $t \in X$ there is a natural number $n \geq 1$ such that for all trees t', if t and t' are equivalent up to depth n then $t' \in X$. The reader may verify that this indeed yields a topological space. When we say that a set of trees is Borel, we will mean that it is Borel with respect to this topology.

By induction on the structure of a formula, it is easy to verify that:

Proposition 6. *Every* WMSO*-definable set of trees is Borel.*

A tree is *well-founded* if it has no infinite branch. It is well-known that the tree language of all well-founded trees is an example of a tree language that is *not* Borel.

Given two topological spaces X and Y, a function $F : X \to Y$ is continuous if for every open set $B \subseteq Y$, $F^{-1}(B) \subseteq X$ is also open. If $F : X \to Y$ is a continuous function, and $A \subseteq X$ and $B \subseteq Y$ are sets such that $F^{-1}(B) = A$, then we write $A \leq_W B$, and we say that A continuously reduces to B. The relation $\leq_W$ is clearly a pre-order. Intuitively, $A \leq_W B$ means that A is topologically no more complex than B. In particular, we have the following well known fact:

Proposition 7. *If X and Y are topological spaces $A \subseteq X$, $B \subseteq Y$ are sets, such that $A \leq_W B$, then $B \in$ **Borel**(Y) implies $A \in$ **Borel**(X).*

Given a depth-n prefix $t^{(n)}$ over Σ, by $t^{(n)} \cdot T_\Sigma$ we denote the set of all trees over Σ extending $t^{(n)}$. The next proposition, determining a sufficient condition for a function in order to be continuous, will be very useful.

Proposition 8. *Let Σ and Σ' be two finite sets, and F be a function from T_Σ into $T_{\Sigma'}$. If for every depth-n prefix $t^{(n)}$ over Σ there exists a depth-n prefix $t'^{(n)}$ over Σ' such that $F(t^{(n)} \cdot T_\Sigma) \subseteq t'^{(n)} \cdot T_{\Sigma'}$, then F is continuous.*

3 Main Result

We now prove the equivalence of the given characterizations of the logic EF on trees.

Theorem 1. Let L be any MSO-definable tree language. The following are equivalent and decidable:

(1) L is EF-definable,
(2) L is closed under EF-bisimulation and WMSO-definable,
(3) L is closed under EF-bisimulation and Borel,
(4) L is closed under EF-bisimulation, and for every L-idempotent context c, and for every forest f, $c(f)$ and $(c + cf)^\infty$ are L-equivalent.

Proof. The proof proceeds in a round-robin fashion. Decidability follows from Thm. 4. (1)⇒(2) follows by applying Prop. 5 and Prop. 4, (2)⇒(3) by Prop. 6, while (4)⇒(1) is given by Thm. 4. The proof of (3)⇒(4) is by contraposition. Suppose there is an idempotent context c and a forest t such that $c(t)$ and $(c + c(t))^\infty$ are not L-equivalent. There are two cases:

(a) for some forest-template e, $e[\star \mapsto c(t)] \in L$ and $e[\star \mapsto (c + c(t))^\infty] \notin L$,
(b) for some forest-template e, $e[\star \mapsto c(t)] \notin L$ and $e[\star \mapsto (c + c(t))^\infty] \in L$.

We will show that both these two possibilities imply that L is not Borel, by giving a continuous reduction of the non-Borel set WF or its complement to L. Recall that WF is the set of all well-founded trees over an alphabet consisting of a single letter a.

For any tree t' over $\{a\}$, let $\widehat{t'}$ be the forest obtained as follows: if t' is the empty tree then $\widehat{t'}$ is ct, otherwise $\widehat{t'}$ is obtained by replacing every leaf node of t' by the forest ct, and replacing every non-leaf node by the context $(c + ct)$. Finally, let $F(t')$ be the tree $e(\widehat{t'})$, i.e., the result of replacing every $\star$ in e by $\widehat{t'}$. Because for every k, the first k levels of t determines the first k levels of $F(t)$, by Prop. 8 F is continuous.

Claim (i) if $t' \in \mathsf{WF}$, then $\widehat{t'}$ is L-equivalent to ct
Claim (ii) if $t' \notin \mathsf{WF}$, then $\widehat{t'}$ is L-equivalent to $(c + ct)^\infty$

This implies that L is not Borel. Indeed, suppose case (a) holds. From the above claim, we obtain that $t' \in \mathsf{WF}$ iff $F(t') \in L$. This means that $F^{-1}(L) = \mathsf{WF}$, proving that $\mathsf{WF} \leq_W L$. Analogously, if case (b) holds, from the previous claim we obtain that $t' \in \mathsf{WF}^{\mathsf{C}}$ iff $F(t') \in L$, where WF^{C} is the complement of WF. Thus $F^{-1}(L) = \mathsf{WF}^{\mathsf{C}}$ and therefore $\mathsf{WF}^{\mathsf{C}} \leq_W L$. It remains only to prove the above claim.

We first introduce some terminology. Clearly, $\widehat{t'}$ is a forest that is built from the context c and the tree t. Let us denote the skeleton of $\widehat{t'}$ by s. Suppose that s is not well-founded. If x is a node in the domain of s such that the subtree $s.x$ is well-founded and there is no ancestor y of x such that $s.y$ is also well-founded, then we call the subforest of $\widehat{t'}$ built up from c and t whose skeleton is $s.x$ a *maximal well-founded subforest of* $\widehat{t'}$.

We first prove claim (i). Suppose that $t' \in \mathsf{WF}$. Then, clearly, s is well-founded. By Prop. 3, $\widehat{t'}$ is L-equivalent to a finitely branching, regular forest t'' built up from c and t whose skeleton s'' is well-founded (recall that well-foundedness is MSO-definable). Since every infinite finitely branching tree has an infinite branch, we then know that the skeleton s'' is in fact finite. Let k be the length of the longest branch of s''. It is easy to see that s'' is EF-bisimilar to a tree that has a single path whose labels (read from the root to the leaf) form the word $\mathsf{c}^{k+1}\mathsf{t}$. For convenience, we denote this tree itself by $\mathsf{c}^{k+1}\mathsf{t}$. Analogously, it is not hard to see that t'' is EF-bisimilar to the forest built up from c and t whose skeleton is $\mathsf{c}^{k+1}\mathsf{t}$, i.e., the forest $c^{k+1}t$. Since EF-bisimilarity implies L-equivalence (Lemma 2), $\widehat{t'}$ is L-equivalent to $c^{k+1}t$. And since c is an L-idempotent context, we have that $\widehat{t'}$ is in fact L-equivalent to ct.

Next, we prove claim (ii). Suppose that $t' \notin \mathsf{WF}$. Clearly, s is non well-founded. By Prop. 3, $\widehat{t'}$ is L-equivalent to a regular (finitely-branching) forest t'' built up from c and t whose skeleton s'' is non well-founded (recall that non well-foundedness is MSO-definable). Because $\widehat{t''}$ is regular, there are only finitely many maximal well-founded subforests of $\widehat{t''}$ built up from c and t. Thus since we know that every well-founded

forest built from c and t is L-equivalent to ct, we can take all maximal well-founded subforests of $\widehat{t'}$ built up from c and t and replace each of these subforests by ct. Here, we use the substitution lemma (Lemma 1). Furthermore, because EF-bisimilarity implies L-equivalence (Lemma 2), if the resulting forest contains several copies of ct next to each other as siblings, they can be collapsed into one. This yields a new forest that can be viewed as a forest built up from the context $c+ct$ alone, whose skeleton has no leafs. But any such forest is EF-bisimilar, hence L-equivalent, to the forest $(c + ct)^\infty$. □

4 Eliminating Recursion from μ-Formulae on Transitive Structures

Recall that a Kripke model is a non-empty $\wp(\mathrm{Prop})$-colored directed graph, where Prop is some set of proposition letters, and that a *pointed* Kripke model is a Kripke model with a distinguished state. We may assume that Prop is finite. Kripke models are more general than trees: they can contain cycles, there is not necessarily a unique root, and a node may satisfy any number of proposition letters.

We refer to [10] for the syntax and semantics of the modal μ-calculus. The class of all pointed models of a μ-formula ϕ is denoted by $\|\phi\|$, whereas the class of all *transitive* pointed models of ϕ is denoted by $\|\phi\|^{\mathsf{tr}}$. Notice that the logic EF can be seen as modal logic interpreted on the transitive closure of trees.

On arbitrary Kripke models, modal logic corresponds to the bisimulation invariant fragment of first-order logic [2], whereas the modal μ-calculus is the bisimulation invariant fragment of MSO [12]. This means that on arbitrary Kripke models, a μ-formula is equivalent to a modal formula iff it is equivalent to a FO formula. Moreover, by a result of Martin Otto [14], this is decidable. We prove an analogous effective characterization for transitive models:

Theorem 2. For every μ-formula ϕ, the following are equivalent and decidable:
(1) ϕ is equivalent on transitive Kripke models to a modal formula,
(2) ϕ is equivalent on transitive Kripke models to a WMSO-formula,
(3) the class of pointed transitive Kripke models satisfying ϕ is Borel.

For a μ-formula ϕ, let $(\phi)^*_\mu$ be the μ-formula obtained from ϕ by replacing every subformula of the form $\Diamond\phi$ by $\mu x.\Diamond(x \vee \phi)$ where x is a fresh variable. For a WMSO formula ϕ, let $(\phi)^*_{\mathsf{WMSO}}$ be the WMSO formula obtained from ϕ by replacing each subformula of the form $x \prec y$ by the WMSO formula that says that the node y is reachable from x along $\prec$. It is easy to see that the following holds :

Lemma 3. *For every pointed Kripke model (g, s)*
(1) for every μ-formula ϕ, we have that $(g, s) \in \|(\phi)^*_\mu\|$ iff $(g^{\mathsf{tr}}, s) \in \|\phi\|^{\mathsf{tr}}$
(2) for every WMSO-formula ϕ, we have that $(\phi)^*_{\mathsf{WMSO}}$ is true in g iff ϕ is true in g^{tr}
where by g^{tr} we denote the transitive closure of g.

We now define what it means for a class of models to be "Borel". Given a pointed model (g, s), its *set of tree companions* is the set $T(g, s)$ of pointed models (t, r) such that (g, s) is bisimilar (in the usual sense) with (t, r), and the underlying directed graph

of t is a tree with root r. For a set P of pointed models, $T(P) = \bigcup_{(g,s)\in P} T(g,s)$. We say that the set of pointed transitive models of a μ-formula ϕ is Borel iff $T(\|(\phi)^*_\mu\|)$, viewed as a tree language over $\wp(\mathrm{Prop})$, is Borel. We can finally proceed to show how Thm. 2 follows from Thm. 1.

Proof of Theorem 2. The implication (1) $\Rightarrow$ (2) is easily verified. Thus, if we verify that (2) $\Rightarrow$ (3) and (3) $\Rightarrow$ (1) we are done. For the first implication we reason as follows. Suppose that ϕ is equivalent on transitive Kripke models to a WMSO-formula γ. Then by Lemma 3, $(\phi)^*_\mu$ is equivalent on $\wp(\mathrm{Prop})$-trees to the WMSO-formula $(\gamma)^*_{\mathsf{WMSO}}$. By Prop. 6 we obtain that $T(\|(\phi)^*_\mu\|)$ is a Borel set. For the second implication, first recall that for every EF-formula φ, there is a modal formula ψ such that for every tree t over Prop, φ is true over t iff $(t,r) \in \|(\psi)^*_\mu\|$, where r is the root of t. Suppose that $\|\phi\|^{\mathsf{tr}}$ is Borel. By definition, this means that $T(\|(\phi)^*_\mu\|)$ is a Borel set. But on the one hand we have that the set $T(\|(\phi)^*_\mu\|)$ is the set of tree models of the μ-formula $(\phi)^*_\mu$, meaning that it is MSO-definable. This set is clearly closed under EF-bisimulation. By Theorem 1, $T(\|(\phi)^*_\mu\|)$ is EF-definable, meaning that, by applying Lemma 3, there is modal formula ψ such that $\|\psi\|^{\mathsf{tr}} = \|\phi\|^{\mathsf{tr}}$. □

Theorem 3. For every WMSO-formula $\phi(x)$, the following are equivalent:

(1) $\phi(x)$ is equivalent on transitive Kripke models to a modal formula,
(2) $\phi(x)$ is bisimulation invariant on transitive Kripke models.

Proof. (1) $\Rightarrow$ (2) is clear. In order to prove (2) $\Rightarrow$ (1), we reason as follows. Let $\phi(x) \in$ WMSO be bisimulation invariant on transitive models. Then $\|(\phi(x))^*_{\mathsf{WMSO}}\|$ is closed under bisimulation. By [12], there is a μ-formula ψ equivalent to $(\phi(x))^*_{\mathsf{WMSO}}$ on all models, meaning that $\phi(x)$ and ψ are equivalent on transitive models. By applying Thm. 2 we conclude the proof. □

References

1. Benedikt, M., Segoufin, L.: Regular Tree Languages Definable in FO and in FO_{mod}. ACM Trans. on Computational Logic 11(1) (2009)
2. van Benthem, J.: Modal Correspondence Theory. PhD thesis, Mathematisch Instituut & Instituut voor Grondslagenonderzoek, University of Amsterdam (1976)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Univ. Press, Cambridge (2001)
4. Bojańczyk, M.: Two-Way Unary Temporal Logic over Trees. In: LICS 2007, pp. 121–130 (2007)
5. Bojańczyk, M., Idziaszek, T.: Algebra for Infinite Forests with an Application to the Temporal Logic EF. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 131–145. Springer, Heidelberg (2009)
6. Bojańczyk, M., Segoufin, L., Straubing, H.: Piecewise Testable Tree Languages. In: LICS 2008, pp. 442–451 (2008)
7. Bojańczyk, M., Segoufin, L.: Tree languages defined in first-order logic with one quantifier alternation. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 233–245. Springer, Heidelberg (2008)

8. Bojańczyk, M., Walukiewicz, I.: Characterizing EF and EX Tree Logics. Theoretical Computer Science 358(2-3), 255–272 (2006)
9. Bojańczyk, M., Walukiewicz, I.: Forest Algebras. In: Automata and Logic: History and Perspectives, pp. 107–132. Amsterdam University Press, Amsterdam (2007)
10. Bradfield, J., Stirling, C.: Modal Logic and Mu-Calculi. In: Bergstra, J., et al. (eds.) Handbook of Process Algebra, pp. 293–332. Elsevier, North-Holland (2001)
11. Dawar, A., Otto, M.: Modal Characterisation Theorems over Special Classes of Frames. Ann. Pure Appl. Logic 161(1), 1–42 (2009)
12. Janin, D., Walukiewicz, I.: On the Expressive Completeness of the Propositional μ-Calculus with Respect to Monadic Second Order Logic. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 263–277. Springer, Heidelberg (1996)
13. Murlak, F.: Weak Index vs Borel Rank. In: STACS 2008, pp. 573–584 (2008)
14. Otto, M.: Eliminating recursion in the μ-calculus. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 531–540. Springer, Heidelberg (1999)
15. Place, T.: Characterization of logics over ranked tree languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 401–415. Springer, Heidelberg (2008)
16. Walukiewicz, I.: Monadic Second-Order Logic on Tree-Like Structures. Theoretical Computer Science 275(1-2), 311–346 (2002)

Parity Games on Graphs with Medium Tree-Width

John Fearnley[1] and Oded Lachish[2]

[1] Department of Computer Science, University of Liverpool, UK
[2] Department of Computer Science and Information Systems, Birkbeck, University of London, UK

Abstract. This paper studies the problem of solving parity games on graphs with bounded tree-width. Previous work by Obdržálek has produced an algorithm that uses $n^{O(k^2)}$ time and $n^{O(k^2)}$ space, where k is the tree-width of the graph that the game is played on. This paper presents an algorithm that uses $n^{O(k \log n)}$ time and $O(n + k \log n)$ space. This is the fastest known algorithm for parity games whose tree-width k satisfies (in standard asymptotic notation) $k \in \omega(\log n)$ and $k \in o(\sqrt{n}/\log n)$.

1 Introduction

In this paper we study the problem of solving parity games on graphs with bounded tree-width. A parity game is a two player game that is played on a finite directed graph. Parity games have received much attention due to the fact that they share a polynomial time equivalence with the μ-calculus model checking problem [7,16], and with the non-emptiness problem for non-deterministic parity tree automata [8]. The complexity of solving parity games is also interesting, because it is one of the few non-cryptographic problems that lies in NP $\cap$ co-NP (and even in UP $\cap$ co-UP [9]). This implies that they are highly unlikely to be either NP-complete or co-NP-complete. However, despite much effort from the community, no polynomial time algorithm has been found.

Problems that are not known to admit polynomial time algorithms for general inputs are often studied from the perspective of *parametrised complexity* [5]. Here, the objective is to find a class of restricted inputs for which a polynomial time algorithm can be devised, and tree-width has turned out to be one of the most successful restrictions for this purpose. The tree-width of a graph is a parameter that measures how close the graph is to being a tree, which originated from the work of Robertson and Seymour on graph minors [14].

When we are given a guarantee that the input graph has tree-width bounded by some constant k, we want to find algorithm whose running time can be expressed as $O(f(k) \cdot n^{g(k)})$. Since k is constant, this is a polynomial time upper bound. A more restrictive notion is *fixed parameter tractability*, where we are looking for an algorithm whose running time can be expressed as $O(f(k) \cdot n^c)$, where c is a constant that is independent of k. Many problems that are NP-hard in general have been found to admit fixed parameter tractable algorithms when the input is restricted to be a graph with bounded tree-width [4].

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 303–314, 2011.

Since parity games have no known polynomial time algorithm, it is natural to ask whether the problem becomes easier when the game graph has bounded tree-width. Although no fixed parameter tractable algorithm is known for parity games, Obdržálek has found a polynomial time algorithm [13]. His algorithm uses a dynamic programming technique that exploits the structure of a graph with bounded tree-width. This yields an algorithm that solves a parity game on a graph with tree-width k in $n^{O(k^2)}$ time and $n^{O(k^2)}$ space.

Our contribution. While Obdržálek's algorithm uses dynamic programming, which is a technique that is traditionally applied to graphs of bounded tree-width, we approach the problem using the techniques that have been developed for solving parity games. Our work builds upon the algorithm of Jurdziński, Paterson, and Zwick, that solves parity games in sub-exponential time for arbitrary graphs [10]. Their algorithm is a modification of McNaughton's exponential-time recursive algorithm for parity games [11]. They introduced the concept of a dominion, which is a structure that appears in parity games. They showed that if McNaughton's algorithm is equipped with a preprocessing procedure that removes all small dominions, then the resulting algorithm runs in $n^{O(\sqrt{n})}$ time. This is currently the fastest algorithm for solving parity games on general graphs. These techniques have since been adapted by Schewe to produce the best known algorithm for parity games with a small number of distinct priorities [15].

Our algorithm applies these techniques to parity games on graphs of bounded tree-width. Instead of using preprocessing to remove small dominions, our algorithm uses the fact that the graph has bounded tree-width to break it into smaller pieces, and then uses preprocessing to remove every dominion that is entirely contained within each piece. We show that equipping McNaughton's algorithm with this preprocessing procedure produces an algorithm that runs in $n^{O(k \log n)}$ time for graphs with tree-width bounded by k. Therefore, our algorithm is asymptotically faster than Obdržálek's algorithm whenever $k \in \omega(\log n)$, where ω is standard asymptotic notation. It is also asymptotically faster than the algorithm of Jurdziński, Paterson, and Zwick whenever $k \in o(\sqrt{n}/\log n)$.

The other advantage that our algorithm has over Obdržálek's algorithm is the amount of space that is uses. Each step in Obdržálek's dynamic programming creates a vast amount of information about the game that must be stored, which leads to the $n^{O(k^2)}$ space complexity of the algorithm. By contrast, our algorithm always uses $O(n^2)$ space. This may make our algorithm more attractive even in the cases where $k \notin \omega(\log n)$.

2 Parity Games

A parity game is defined by a tuple $(V, V_0, V_1, E, \text{pri})$, where V is a set of vertices and E is a set of edges, which together form a finite directed graph. We assume that every vertex has at least one outgoing edge. The sets V_0 and V_1 partition V into vertices belonging to player Even and vertices belonging to player Odd, respectively. The function $\text{pri} : V \to \mathbb{N}$ assigns a *priority* to each vertex. We will use $\text{MaxPri}(G) = \max\{\text{pri}(v) \ : \ v \in V\}$ to denote the largest priority in G.

At the beginning of the game a token is placed on the starting vertex v_0. In each step, the owner of the vertex that holds the token must choose one outgoing edge from that vertex and move the token along it. In this fashion, the two players form an infinite path $\pi = \langle v_0, v_1, v_2, \dots \rangle$, where $(v_i, v_{i+1}) \in E$ for every $i \in \mathbb{N}$. To determine the winner of the game, we consider the set of priorities that occur *infinitely often* along the path. This is defined to be:

$$\mathrm{Inf}(\pi) = \{d \in \mathbb{N} \ : \ \text{For all } j \in \mathbb{N} \text{ there is an } i > j \text{ such that } \mathrm{pri}(v_i) = d\}.$$

Player Even wins the game if the highest priority occurring infinitely often is even, and player Odd wins the game otherwise. In other words, player Even wins the game if and only if $\max(\mathrm{Inf}(\pi))$ is even.

A positional strategy for Even is a function that chooses one outgoing edge for every vertex in V_0. A strategy is denoted by $\sigma : V_0 \to V$, with the condition that $(v, \sigma(v)) \in E$, for every Even vertex v. Positional strategies for player Odd are defined analogously. The sets of positional strategies for Even and Odd are denoted by Π_0 and Π_1, respectively. Given two positional strategies σ and τ, for Even and Odd respectively, and a starting vertex v_0, there is a unique path $\langle v_0, v_1, v_2 \dots \rangle$, where $v_{i+1} = \sigma(v_i)$ if v_i is owned by Even and $v_{i+1} = \tau(v_i)$ if v_i is owned by Odd. This path is known as the *play* induced by the two strategies σ and τ, and will be denoted by $\mathrm{Play}(v_0, \sigma, \tau)$.

An infinite path $\langle v_0, v_1, \dots \rangle$ is said to be *consistent* with an Even strategy $\sigma \in \Pi_0$ if $v_{i+1} = \sigma(v_i)$ for every i such that $v_i \in V_0$. If $\sigma \in \Pi_0$ is strategy for Even, and v_0 is a starting vertex then we define $\mathrm{Paths}(v_0, \sigma)$ to be the function that gives every path starting at v_0 that is consistent with σ:

$$\mathrm{Paths}_0(v_0, \sigma) = \{\langle v_0, v_1, \dots \rangle \in V^\omega \ : \ \text{for all } i \in \mathbb{N}, \text{ if } v_i \in V_0 \text{ then } v_{i+1} = \sigma(v_i), \text{ and if } v_i \in V_1 \text{ then } (v_i, v_{i+1}) \in E)\}.$$

An Even strategy is called a *winning strategy* from a given starting vertex if player Even can use the strategy to ensure a win when the game is started at that vertex, no matter how player Odd plays in response. In other words, a strategy $\sigma \in \Pi_0$ is a winning strategy for player Even from the starting vertex v_0 if $\max(\mathrm{Inf}(\pi))$ is even for every path $\pi \in \mathrm{Paths}_0(v_0, \sigma)$. The strategy σ is said to be winning for a set of vertices $W \subseteq V$ if it is winning for every vertex $v \in W$. Winning strategies for player Odd are defined analogously.

A game is said to be *positionally determined* if one of the two players always has a positional winning strategy. We now give a fundamental theorem, which states that parity games are positionally determined.

Theorem 1 ([6,12]). *In every parity game, the set of vertices V can be partitioned into two sets (W_0, W_1), where Even has a positional winning strategy for W_0, and Odd has a positional winning strategy for W_1.*

The sets W_0 and W_1, whose existence is implied by Theorem 1, are called *winning sets*, and our objective is to find an algorithm that computes this partition. Occasionally, we will want to refer to winning sets from different games, and we will use W_i^G to refer to the winning set for player i in the game G.

3 McNaughton's Algorithm for Parity Games

In this section we describe the McNaughton's algorithm for parity games. This algorithm was first formulated by McNaughton [11], and was presented for parity games by Zielonka [17]. We will also describe the improvements that have been made to this algorithm by Jurdziński, Paterson, and Zwick. Our algorithm will build upon the techniques that have been developed by these authors.

To begin, we must define the *attractor* of a set of vertices W. This is the set of vertices from which one of the players can force the token to arrive at W, no matter how the opponent plays in response. We will give the definitions for attractors of player Even. To define the attractor we use the Pre^0 operator:

$$\text{Pre}^0(W) = \{v \in V_0 \ : \ \text{There exists } (v,u) \in E \text{ such that } u \in W\} \cup \{v \in V_1 \ : \ \text{For all } (v,u) \in E \text{ we have } u \in W\}$$

This operator gives the set of vertices from which Even can force play into the set W in exactly one step. To compute the attractor, we iteratively apply this operator until a fixed point is reached:

$$W_0 = W$$
$$W_{i+1} = \text{Pre}^0(W_i) \cup W_i$$

Since each set $W_i \subseteq W_{i+1}$, we know that a fixed point must be reached after at most $|V|$ iterations. Therefore, we define the attractor for player Even of a set of vertices W to be $\text{Attr}^0(W) = W_{|V|}$. The corresponding function $\text{Attr}^1(W)$, which gives attractors for player Odd, can be defined analogously.

Algorithm 1. `McNaughton`(G)

Input: A parity game $G = (V, V_0, V_1, E, \text{pri})$
Output: The partition of winning sets (W_0, W_1)

```
p := MaxPri(G); d := p mod 2
A := Attr^d({v ∈ V : pri(v) = p})
if A = V then
    (W_d, W_{1-d}) := (V, ∅)
    return (W_0, W_1)
end if
(W'_0, W'_1) := McNaughton(G ↾ (V \ A))
if W'_{1-d} = ∅ then
    (W_d, W_{1-d}) := (V, ∅)
else
    D := Attr^{1-d}(W'_{1-d})
    (W''_0, W''_1) := McNaughton(G ↾ (V \ D))
    (W_d, W_{1-d}) := (W''_d, W''_{1-d} ∪ D)
end if
return (W_0, W_1)
```

McNaughton's algorithm is shown as Algorithm 1. Given a set of vertices $W \subseteq V$ such that for each $v \in W$ there is an edge (v, u) with $u \in W$, we use $G \restriction W$ to denote the *sub-game* induced by the set W. In other words, $G \restriction W$ is the game obtained by removing every vertex in $V \setminus W$ from the game G, and all edges that are incident to these vertices. The algorithm makes two recursive calls on sub-games of size at most $|V| - 1$, which implies the following theorem.

Theorem 2 ([11,17]). *If McNaughton's algorithm is applied to a parity game with n vertices, then it will compute the partition into winning sets in $O(2^n)$ time.*

Jurdziński, Paterson, and Zwick introduced the concept of a *dominion* [10], which is useful for analysing McNaughton's algorithm. A set of vertices W is a *trap* for player d if there is no edge (v, u) such that $v \in V_d \cap W$ and $u \notin W$, and if for every vertex $v \in V_{1-d} \cap W$ there is an edge (v, u) such that $u \in W$. In other words, the set of vertices W is a trap for a player d if player $1 - d$ has a strategy to ensure that player d cannot leave W. A trap W for player d is a dominion if player $1 - d$ wins everywhere in the game $G \restriction W$.

Definition 3 (Dominion ([10])). *A set $W \subseteq V$ is a dominion for player d if W is a trap for player $1 - d$, and player d has a winning strategy for every vertex in the game $G \restriction W$.*

To see why dominions are useful for analysing McNaughton's algorithm, we will give two trivial facts about traps in parity games. Firstly, the complement (with respect to the set V) of an Even attractor is a trap for Even, and the complement of an Odd attractor is a trap for Odd. Secondly, the set W_0 is a trap for Odd and the set W_1 is a trap for Even. These two facts imply that the set W'_{1-d} computed by the first recursive call of the algorithm is a dominion.

Proposition 4. *The set W'_{1-d} is a dominion for player $1 - d$.*

Jurdziński, Paterson, and Zwick used this idea to speed up McNaughton's algorithm. Before making the first recursive call, their algorithm performs a pre-processing step that removes all player $1 - d$ dominions that are smaller than some threshold. This guarantees that the size of the set W'_{1-d}, which is returned by the first recursive call, must be larger than this threshold. Since the set W'_{1-d} is removed from the game, this means that the second recursive call will be performed on a significantly smaller game. By carefully balancing the amount of time spent on pre-processing and the size of the dominions that are removed, they obtained a bound of $n^{O(\sqrt{n})}$ on the running time of their algorithm.

4 Tree Width

To define the tree-width of a parity game, we will use the tree-width of the undirected graph that is obtained when the orientation of the edges in the game is ignored. Therefore, we will use the following definition of a tree decomposition.

Definition 5 (Tree Decomposition). *For each game G, the pair (T, X), where $T = (I, F)$ is an undirected tree and $X = \{X_i : i \in I\}$ is a family of subsets of V, is a tree decomposition of G if all of the following hold:*

1. $\bigcup_{i \in I} X_i = V$,
2. *for every $(v, u) \in E$ there is an $i \in I$ such that $v \in X_i$ and $u \in X_i$, and*
3. *for every $i, j \in I$, if $k \in I$ is on the unique path from i to j in T, then $X_i \cap X_j \subseteq X_k$.*

The *width* of a tree decomposition (T, X) is $\max\{|X_i| - 1 : i \in I\}$, which is the cardinality of the largest set contained in X minus 1. The *tree-width* of a game G is the smallest width that is obtained by a tree decomposition of G. We will discuss the computation of a tree decomposition for a parity game in Section 6, and for the time being we will assume that a tree decomposition is given along with the input to our algorithm.

We can now explain the properties of a tree decomposition that will be used in our algorithm. Suppose that $i \in I$ is some vertex in the tree decomposition. If we remove i from the graph T, then we will produce a forest, and each of the trees in this forest will have exactly one vertex that is adjacent to i in T. This allows us to use the edges of i to identify each tree in the forest. For each edge $(i, j) \in F$, we use $T_{(i,j)}$ to denote the tree rooted at j that appears when i is removed from T. For each tree $T_{(i,j)}$, we define $V_{(i,j)} = \{v \in V : v \in X_l \text{ for some } l \in T_{(i,j)}\}$ to be the set of vertices in G that is contained in $T_{(i,j)}$.

For our algorithm, the most important property of a tree decomposition is that each set X_i is a *separator* in the graph G. This means that in order to move from some vertex $v \in V_{(i,j)}$ to some vertex $u \in V_{(i,l)}$, where $j \neq l$, we must pass through some vertex in X_i. In other words, if every vertex in X_i is removed from the graph, then the sets $V_{(i,j)}$ will be disconnected from each other.

Proposition 6. *Suppose that $i \in I$ and let (i, j) and (i, l) be two edges in F such that $j \neq l$. If $v \in V_{(i,j)}$ and $u \in V_{(i,l)}$, then every path from v to u must pass through at least one vertex in X_i.*

Our algorithm will use separators to break the graph into smaller pieces. In particular, it will find a separator that splits the graph into at least two pieces, where each piece contains at most two-thirds of the vertices in the original graph. The following proposition is a standard result for graphs of bounded tree width [14], which shows that such a separator must always exist.

Proposition 7. *Let G be a game with at least $3k+3$ vertices, and let (T, X) be a tree decomposition of G with width k. There is some $i \in I$ such that $|V_{(i,j)}| \leq 2|V|/3$ for all $(i, j) \in F$.*

We define $\text{Split}(G, (T, X))$ to be a function that, given a game G with tree decomposition (T, X), selects some vertex $i \in I$ that satisfies the condition given by Proposition 7. Obviously, this function can be computed in polynomial time.

5 Our Algorithm

In this section, we will describe the approach that our algorithm takes in order to solve a game $G = (V, V_0, V_1, E, \text{pri})$ with tree decomposition $(T = (I, F), X)$ of width k, where $i = \text{Split}(G, (T, X))$ and $d = \text{MaxPri}(G) \bmod 2$. The key idea behind our algorithm is to break the game into sub-games using the separator X_i. Each sub-game will be preprocessed separately in order to remove every player $1 - d$ dominion that is entirely contained within the sub-game.

We begin by describing the preprocessing procedure that is used to remove every dominion for player $1 - d$ that does not contain a vertex in X_i. For each edge $(i, j) \in F$, we define a preprocessing game G_j on the vertices in $V_{(i,j)} \cup X_i$. The only difference between $G \restriction (V_{(i,j)} \cup X_i)$ and G_j is that all of the vertices in X_i are given to player d. Moreover, every vertex in X_i is given a self loop edge, and its priority is changed to 0 if d is even, and 1 if d is odd. These changes allow player d to win the game if the token arrives at a vertex in X_i.

Definition 8 (Preprocessing Game G_j). *Let $G = (V, V_0, V_1, E, \text{pri})$ be a game where $d = \text{MaxPri}(G) \bmod 2$, let (T, X) be a tree decomposition of G, and let $i = \text{Split}(G, (T, X))$. For each edge $(i, j) \in F$ we define the game $G_j = (V', V'_0, V'_1, E', \text{pri}')$ as follows:*

$$\begin{aligned} V' &= V_{(i,j)} \cup X_i \\ V'_d &= (V_0 \cap V') \cup X_i, \\ V'_{1-d} &= (V_1 \cap V') \setminus X_i, \\ E' &= (E \cap (V' \times V')) \cup \{(v, v) \ : \ v \in X_i\} \end{aligned}$$

$$\text{pri}'(v) = \begin{cases} 0 & \text{if } d \text{ is even and } v \in X_i, \\ 1 & \text{if } d \text{ is odd and } v \in X_i, \\ \text{pri}(v) & \text{otherwise.} \end{cases}$$

Note that the definition of a preprocessing game ensures that every vertex must have at least one outgoing edge. This is because the fact that X_i is a separator implies that the preprocessing game must include all outgoing edges from the vertices in $V_{(i,j)}$, and every vertex in $v \in X_i$ is given a self loop (v, v). Moreover, since no new vertices are added, and the only new edges are self loops, if (T, X) is a tree decomposition of width k for the original game, then (T, X) is also a tree decomposition of width k for the preprocessing game.

The algorithm will call itself recursively in order to solve each preprocessing game G_j. It will therefore compute a partition $(W_0^{G_j}, W_1^{G_j})$, which are the winning sets for the two players in the game G_j. The first thing that we can prove is that the winning set for player $1 - d$ in the preprocessing game G_j must be contained in the winning set for player $1 - d$ in the game G. Therefore, we can remove the vertices in the set $W_{1-d}^{G_j}$ from the game G and add them to the set W_{1-d}, which is the winning set for player $1 - d$ in the game G.

Proposition 9. *If $d = \text{MaxPri}(G) \bmod 2$, then we have $W_{1-d}^{G_j} \subseteq W_{1-d}^{G}$ for every $(i, j) \in F$.*

Proof. We begin by arguing that $W_{1-d}^{G_j}$ is a trap for player d in the game G. We will prove this claim by contradiction. Suppose that there is an edge (v, u) with $v \in V_d \cap W_{1-d}^{G_j}$ and $u \in V \setminus W_{1-d}^{G_j}$. We must have $v \in X_i$, because otherwise the edge (v, u) would be contained in E', and this would imply that $W_{1-d}^{G_j}$ is not a trap for player d in G_j. Since $v \in X_i$, we know that $(v, v) \in E'$ and that $\text{pri}'(v) = d \bmod 2$. Therefore, player d has a winning strategy for the vertex v in the game G_j, which contradicts the fact that $v \in W_{1-d}^{G_j}$.

Since $W_{1-d}^{G_j}$ is a trap for player d in the game G, player $1 - d$ can use the winning strategy for $W_{1-d}^{G_j}$ to force a win from that set in the game G. This proves the claim that $W_{1-d}^{G_j} \subseteq W_{1-d}^{G}$. □

The next proposition gives the reason why this preprocessing procedure is useful. It states that, after each of the sets $W_{1-d}^{G_j}$ has been removed, every dominion for player $1 - d$ must use at least one vertex from the separator X_i.

Proposition 10. *If* $d = \text{MaxPri}(G) \bmod 2$ *then for every player* $1 - d$ *dominion* D *that is contained in* $V \setminus \bigcup_{(i,j)\in F} W_{1-d}^{G_j}$ *we have* $D \cap X_i \neq \emptyset$.

Proof. We will prove this claim by contradiction. Suppose that there is a player $1 - d$ dominion D that is contained in $V \setminus \bigcup_{(i,j)\in F} W_{1-d}^{G_j}$ with the property $D \cap X_i = \emptyset$. If D is not entirely contained in some set $V_{(i,j)}$, then we will consider the dominion $D' = D \cap V_{(i,j)}$ for some edge $(i, j) \in F$ such that $D \cap V_{(i,j)} \neq \emptyset$. The fact that X_i is a separator in G implies that D' is also a dominion for player $1 - d$. Therefore, from now on, we can assume that $D \subseteq V_{(i,j)}$.

Since D is a player $1 - d$ dominion, Definition 3 implies that player $1 - d$ has a winning strategy for every vertex in the sub-game $G \restriction D$. Since D is a trap for player d and $D \cap X_i = \emptyset$, this strategy allows player $1 - d$ to win from every vertex in D in the preprocessing game G_j. This implies that $D \subseteq W_{1-d}^{G_j}$, which contradicts the fact that $D \subseteq V \setminus \bigcup_{(i,j)\in F} W_{1-d}^{G_j}$. □

Recall that Proposition 4 implies that the set D returned by the first recursive call of McNaughton's algorithm must be a dominion for player $1-d$. The property given by Proposition 10 allows us to conclude that D must contain at least one vertex in the separator X_i. Since the algorithm removes D from the game, we know that at least one vertex will be removed from the separator X_i. If the graph has tree width k, then repeating this procedure $k + 1$ times must remove every vertex from the set X_i. Since X_i is a separator, this implies that the game will have been split into at least two disjoint parts, which can be solved separately.

Algorithm 2 shows the algorithm `Preprocess`(G, i), which performs preprocessing for the game G around the separator X_i. It returns the set W_{1-d}, which is the attractor for player $1 - d$ to the union of the winning sets $W_{1-d}^{G_j}$. Algorithm 3 is identical to Algorithm 1, except that calls to `McNaughton` have been replaced by calls to `Solve`.

Algorithm 4 shows the main algorithm. It has two special cases: when the number of vertices is small the algorithm applies McNaughton's algorithm, and

Algorithm 2. `Preprocess`$(G, (T, X), i)$

Input: A parity game $G = (V, V_0, V_1, E, \mathrm{pri})$, a tree decomposition $(T = (I, F), X)$, and the index of a separator X_i
Output: Every player-$(1-d)$ dominion in G that is disjoint from X_i

$p := \mathrm{MaxPri}(G)$; $d := p \bmod 2$
for all $(i, j) \in F$ **do**
  $(W_0^{G_j}, W_1^{G_j}) := \mathtt{Solve}(G_j, (T, X), \mathrm{Split}(G_j, (T, X)))$
end for
$W_{1-d} := \mathrm{Attr}^{1-d}(\bigcup_{(i,j)\in F} W_{1-d}^{G_j})$
return W_{1-d}

Algorithm 3. `New-McNaughton`$(G, (T, X), i)$

Input: A parity game $G = (V, V_0, V_1, E, \mathrm{pri})$, a tree decomposition $(T = (I, F), X)$, and the index of a separator X_i
Output: The partition of winning sets (W_0, W_1)

$p := \mathrm{MaxPri}(G)$; $d := p \bmod 2$
$A := \mathrm{Attr}^d(\{v \in V \ : \ \mathrm{pri}(v) = p\})$
if $A = V$ **then**
  $(W_d, W_{1-d}) := (V, \emptyset)$
  return (W_0, W_1)
end if
$(W_0', W_1') := \mathtt{Solve}(G \restriction (V \setminus A), (T, X), i)$
if $W_{1-d}' = \emptyset$ **then**
  $(W_d, W_{1-d}) := (V, \emptyset)$
else
  $D := \mathrm{Attr}^{1-d}(W_{1-d}')$
  $(W_0'', W_1'') := \mathtt{Solve}(G \restriction (V \setminus D), (T, X), i)$
  $(W_d, W_{1-d}) := (W_d'', W_{1-d}'' \cup D)$
end if
return (W_0, W_1)

when the separator X_i is empty the algorithm calls itself recursively to solve each piece independently. If neither of the two special cases are applicable, then the algorithm runs the preprocessing procedure on the game. Note that the preprocessing procedure may remove every vertex v that has $\mathrm{pri}(v) = \mathrm{MaxPri}(G)$, which could cause the player $d := \mathrm{MaxPri}(G) \bmod 2$ to change. If this were to occur, then we would have to run preprocessing for the other player to ensure that Proposition 10 can be applied. Therefore, the algorithm repeats the preprocessing procedure until the player d does not change. Once preprocessing is complete, the algorithm runs `New-McNaughton` on the remaining game.

We will use the notation $T(n, l)$ to denote the running time of Algorithm 4 when the input graph G has n vertices, and when $|X_i| = l$. Since the best bound for $|X_i|$ that we have is $|X_i| \leq k$, the running time of our algorithm is given by $T(n, k)$, where k is the width of the tree decomposition. However, we must still

Algorithm 4. `Solve`$(G, (T, X), i)$

Input: A parity game $G = (V, V_0, V_1, E, \text{pri})$, a tree decomposition $(T = (I, F), X)$ and the index of a separator X_i
Output: The partition of winning sets (W_0, W_1)

```
if |V| < 12k then
    (W_0, W_1) := McNaughton(G)
else if X_i = ∅ then
    for all (i, j) ∈ F do
        (W_0^{G_j}, W_1^{G_j}) := Solve(G ↾ V_(i,j), (T, X), Split(G ↾ V_(i,j), (T, X)))
    end for
    W_0 := ⋃_{(i,j)∈F} W_0^{G_j}; W_1 := ⋃_{(i,j)∈F} W_1^{G_j}
else
    (W_0, W_1) := (∅, ∅)
    repeat
        p := MaxPri(G); d := p mod 2
        W_{1−d} := W_{1−d} ∪ Preprocess(G, (T, X), i)
        V := V \ W_{1−d}
    until MaxPri(G) mod 2 = d
    (W'_0, W'_1) := New-McNaughton(G, (T, X), i)
    return (W_0 ∪ W'_0, W_1 ∪ W'_1)
end if
return (W_0, W_1)
```

consider $T(n, l)$ for $l < k$, because our algorithm reduces the size of $|X_i|$ as it progresses. We have the following recurrence for $T(n, l)$:

$$T(n,l) \leq \begin{cases} n \cdot T(2n/3, k) & \text{if } l = 0, \\ 2^{12k} & \text{if } n \leq 12k, \\ n^2 \cdot T(2n/3 + k, k) + T(n-1, l) + T(n-1, l-1) + n^2 & \text{otherwise.} \end{cases}$$

The first case follows from the fact that the algorithm solves each piece of the game independently when the separator X_i is empty. Proposition 7 implies that each of these pieces can have size at most $2n/3$, and no separator in the piece can contain more than k vertices. The second case follows from the use of McNaughton's algorithm to solve games that have fewer than $12k$ vertices. The running time of McNaughton's algorithm is given by Theorem 2.

The final case of the recurrence deals with the case where X_i is non-empty, and where there are a large number of vertices. The term $n^2 \cdot T(2n/3+k, k)$ represents the cost of preprocessing. The algorithm `Solve`, can invoke `Preprocess` at most $|V|$ times before calling `New-McNaughton`. This is because if $\text{MaxPri}(G) \bmod 2 \neq d$ then at least one vertex must have been removed by the previous call to `Preprocess`. The algorithm `Preprocess` itself solves at most n preprocessing games, each of which has at most $2n/3 + k$ vertices and tree-width at most k. The term $T(n-1, l)$ comes from the first recursive call of `New-McNaughton`, where the removal of an attractor is guaranteed to reduce the number of vertices by 1. The term $T(n-1, l-1)$ is for the second recursive call of `New-McNaughton`,

where Proposition 10 guarantees that at least one vertex has been removed from X_i. Finally, in each application of `New-McNaughton` must compute at least one attractor, and an attractor can be computed in $O(n^2)$ time. Solving the recurrance yields the following theorem.

Theorem 11. *When given a parity game with a tree decomposition of width k, Algorithm 4 will compute the partition into winning sets in time $n^{O(k \log n)}$ and $O(n^2)$ space.*

6 Computing the Tree Decomposition

It has been shown that the problem of deciding whether a graph has tree-width smaller than some bound k is NP-hard [2], which poses a significant problem for our algorithm, since it requires a tree decomposition as an input. Bodlaender has given an exact algorithm for computing tree decompositions [3]: If the tree-width of the graph is bounded by some constant k, then his algorithm will run in time $O(n \cdot f(k))$. However, the constant hidden by the $O(\cdot)$ notation is in the order 2^{k^3}. Thus, the cost of computing a tree decomposition using this algorithm dwarfs any potential advantage that our algorithm can offer over Obdržálek's algorithm, which must also compute a tree decomposition for the input graph.

As a solution to this problem, we propose that an *approximate* tree decomposition should be computed instead. A paper by Amir gives a plethora of approximation algorithms for this purpose [1]. One such algorithm takes an arbitrary graph G and an integer k, and in $O(2^{3k} n^2 k^{3/2})$ time provides either a tree decomposition of width at most $4.5k$ for G, or reports that the tree-width of G is larger than k.

Therefore, we can take the following approach to computing our tree decomposition. Apply Amir's algorithm with the input k set to 1. If a tree decomposition is found then halt, otherwise increase k by 1 and repeat. Once $k > \sqrt{n}/(4.5 \log n \cdot c)$, where c is the constant hidden by the $O(\cdot)$ notation in Theorem 11, we stop attempting to find a tree decomposition and apply the algorithm of Jurdziński, Paterson, and Zwick. If the procedure produces a tree decomposition for some k, then we apply either Obdržálek's algorithm or our algorithm, depending on how large k is.[1]

If the procedure halts at the value k, either because a tree decomposition has been found or because k is to large, then the total amount of time spent computing the tree decomposition is $\sum_{i=1}^{k} O(2^{3k} n^2 k^{3/2}) \in O(2^{3(k+1)} n^2 k^3)$. Thus, if the input graph has treewidth k and Obdržálek's algorithm is applied, then the running time will be $O(2^{3(k+1)} n^2 k^3) + n^{O((4.5k)^2)} \in n^{O(k^2)}$. If our algorithm is applied then the total running time will be $O(2^{3(k+1)} n^2 k^3) + n^{O(4.5k \log n)} \in n^{O(k \log n)}$.

[1] A more efficient technique would be to double k in each iteration and, once a tree decomposition is found, to use binary search to find the smallest value of k for which Amir's algorithm produces a tree decomposition. However, using this approach does not lead to better asymptotic running times for the algorithms.

Since the final value of $k < \sqrt{n}$, if the algorithm of Jurdziński, Paterson, and Zwick is applied then the total running time will be $O(2^{3(k+1)}n^2k^3) + n^{O(\sqrt{n})} \in n^{O(\sqrt{n})}$.

References

1. Amir, E.: Efficient approximation for triangulation of minimum treewidth. In: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, pp. 7–15. Morgan Kaufmann Publishers Inc., San Francisco (2001)
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic and Discrete Methods 8(2), 277–284 (1987)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)
4. Bodlaender, H.L.: Treewidth: Algorithmic techniques and results. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 19–36. Springer, Heidelberg (1997)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, pp. 368–377. IEEE Computer Society Press, Washington, DC (1991)
7. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 385–396. Springer, Heidelberg (1993)
8. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
9. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Information Processing Letters 68(3), 119–124 (1998)
10. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA, pp. 117–123. ACM/SIAM (2006)
11. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65(2), 149–184 (1993)
12. Mostowski, A.W.: Games with forbidden positions. Technical Report 78, University of Gdańsk (1991)
13. Obdržálek, J.: Fast mu-calculus model checking when tree-width is bounded. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 80–92. Springer, Heidelberg (2003)
14. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7(3), 309–322 (1986)
15. Schewe, S.: Solving parity games in big steps. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 449–460. Springer, Heidelberg (2007)
16. Stirling, C.: Local model checking games (Extended abstract). In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 1–11. Springer, Heidelberg (1995)
17. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200, 135–183 (1998)

Satisfiability of Systems of Equations of Real Analytic Functions Is Quasi-decidable

Peter Franek[1], Stefan Ratschan[1], and Piotr Zgliczynski[2]

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic
[2] Jagellonian University in Krakow

Abstract. In this paper we consider the problem of checking whether a system of equations of real analytic functions is satisfiable, that is, whether it has a solution. We prove that there is an algorithm (possibly non-terminating) for this problem such that (1) whenever it terminates, it computes a correct answer, and (2) it always terminates when the input is robust. A system of equations of robust, if its satisfiability does not change under small perturbations. As a basic tool for our algorithm we use the notion of degree from the field of (differential) topology.

1 Introduction

It is well known that, while the theory of real numbers with addition and multiplication is decidable [23], any periodic function makes the problem undecidable, since it allows encoding of the integers. Recently, several papers [7,17,18,5] have argued, that in continuous domains (where we have notions of neighborhood, perturbation etc.) such decidability results do not always have much practical relevance. The reason is, that real-world manifestations of abstract mathematical objects in such domains will always be exposed to perturbations (imprecision of production, engineering approximations, unpredictable influences of the environment etc.). Engineers take these perturbations into account by coming up with *robust* designs, that is, designs that do not change essentially under such perturbations. Hence, in this context, it is sufficient to come up with algorithms that are able to decide such robust problem instances. They are allowed to run forever in non-robust cases, but—since robustness may not be checkable—*must not* return incorrect results, in whatever case. In a recent paper we called problems possessing such an algorithm *quasi-decidable* [19].

In this paper we show quasi-decidability of a certain fragment of the first-order theory of the reals. We allow n equalities over n variables ranging over closed intervals $I_1, \ldots, I_n$, and verify the existence of a solution of the equalities in those intervals. The allowed function symbols include addition, multiplication, exponentiation, and sine. More specifically, they have to be real analytic, and for compact intervals $I_1, \ldots, I_n$, we need to be able to compute an interval $J \supseteq f(I_1, \ldots, I_n)$ such that the over-approximation of J over $f(I_1, \ldots, I_n)$ can be made arbitrarily small.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 315–326, 2011.

Verification of zeros of systems of equations is a major topic in the interval computation community [15,20,11,8]. However, here people are usually not interested in some form of completeness of their methods, but in usability within numerical solvers for systems of equations or global optimization.

The main tool we use is the notion of the *degree of a continuous function* that comes from differential topology [9,13,16]. For continuous functions $f : [a, b] \to \mathbb{R}$, the degree $\deg(f, [a, b], 0)$ is 0 iff $f(a)$ and $f(b)$ have the same sign, otherwise the degree is either -1 or 1, depending on whether the sign changes from negative to positive or the other way round. Hence, in this case, the degree gives the information given by the intermediate value theorem plus some directional information. For higher dimensional functions, the degree is an integer whose value may be greater than 1, and that generalizes this information to higher dimensions. However, the degree is defined only when the dimensions of the domain and target space of f are equal.

If we can over-approximate the function f arbitrarily precisely on intervals, then the degree is algorithmically computable. Our algorithm for proving satisfiability of a function f consists of over-approximating the connected components of the zero set of f by small neighborhoods U_i and checking, whether $\deg(f, U_i, 0)$ are zero. If any of them is nonzero, then $f(x) = 0$ has a solution. Otherwise we show that there exists an arbitrarily small perturbation $\tilde{f}$ of f such that $\tilde{f}(x) = 0$ does not have a solution. However, such neighborhoods U_i may not exist for a general continuous or even differentiable function. Therefore, we restrict ourselves to analytic functions. The zero set of an analytic function consists of a finite number of closed connected components and we may take disjoint small neighborhoods of them.

Collins [4] presents a similar result to ours, formulated in the language of computable analysis [24]. However, the paper unfortunately contains only very rough sketch proofs, for which—from our point of view—it is not at all clear how they can be completed into complete proofs.

Since this work applies results from a quite distant field—topology—to automated reasoning, it is not possible to keep the paper self-contained within a reasonable number of pages. Still, we tried to keep the basic material self-contained and to refer to topological results only later in the paper. The necessary topological pre-requisites can be found in standard textbooks [14, e.g.].

The work of Stefan Ratschan and Peter Franek was supported by MŠMT project number OC10048 and by the institutional research plan AV0Z100300504.

2 Main Theorem

In the following, we define a *box* B in $\mathbb{R}^n$ to be the Cartesian product of n closed intervals of finite positive length (a hyper-rectangle). More general, we define a k-dimensional box (k-box) in $\mathbb{R}^n$ to be a product of k closed intervals of positive finite length and $(n - k)$ constants.

On $\mathbb{R}^n$, we denote the norm of a vector $(x_1, \ldots, x_n)$ by $|x|$ and the norm of a continuous function $f : \Omega \to \mathbb{R}^n$ by $||f|| := \sup\{|f(x)|;\ x \in \Omega\}$. Moreover, we will denote the solution set $\{f(x) = 0 \mid x \in \Omega\}$ by $\{f = 0\}$.

For a set $\Omega \subset \mathbb{R}^n$, $\bar{\Omega}$ will be its closure, Ω° its interior and $\partial\Omega = \bar{\Omega} \backslash \Omega^\circ$ its boundary with respect to the Euclidean topology. If Ω is a k-box in $\mathbb{R}^n$, we will usually denote $\partial\Omega$ the boundary in the topology of Ω (i.e., union of the $2k$ faces).

Definition 1. *Let $\Omega \subseteq \mathbb{R}^n$. We call a function $f : \Omega \to \mathbb{R}$* interval-computable *iff there exists an algorithm $I(f)$ that, for every box with rational endpoints such that this box is a subset of Ω, computes a closed interval such that*

- *for every box $B \subseteq \Omega$, $I(f)(B) \supseteq \{f(x) \mid x \in B\}$, and*
- *for every box $S \subseteq \Omega$, for every $\varepsilon > 0$ there is a $\delta > 0$ such that for every box B with $B \subseteq S$, $diam(B) < \delta$, for all $y \in I(f)(B)$, there is an $x \in B$ such that $|f(x) - y| \leq \varepsilon$.*

We call a function $f = (f_1, \ldots, f_n) : \Omega \to \mathbb{R}^n$ interval-computable iff each f_i is interval-computable. In this case, the algorithm $I(f)$ returns a tuple of intervals, one for each f_i.

Usually such functions are written in terms of symbolic expressions containing symbols denoting certain basic functions such as rational constants, addition, multiplication, exponentiation, and sine. In that case, the first property above follows from the fundamental theorem of interval arithmetic, and the second property from Lipschitz continuity of interval arithmetic (e.g., Theorem 2.1.1 in Neumaier's book [15]). However, in this paper we do not fix a certain notation and will allow an arbitrary language for denoting interval computable functions. We will use interval computable functions and expressions denoting them interchangeably.

Checking satisfiability of a system of equations $f_1 = 0, \ldots, f_n = 0$ in n variables amounts to checking whether the function given by $f_1, \ldots, f_n$ has a zero. Since we aim at showing that this problem is quasi-decidable we have to define some notion of robustness of zeros of such functions.

Definition 2. *We say that a continuous function $f : \Omega \to \mathbb{R}^n$ has a* robust zero *in $\Omega' \subset \Omega$, iff there exists an $\epsilon > 0$ such that for every continuous $g : \Omega' \to \mathbb{R}^n$, $||g|| \leq \epsilon$ implies that $(f + g)(x) = 0$ has a solution in Ω'.*

We say that a continuous function $f : \Omega \to \mathbb{R}^n$ is robust *iff it either has a robust zero in Ω or there exists an $\epsilon > 0$ such that for every continuous $g : \Omega \to \mathbb{R}^n$, $||g|| \leq \epsilon$, $(f + g)(x) = 0$ does not have a solution in Ω.*

The definition of robustness reflects the stability of the existence of a solution with respect to small perturbations of the function. For example, the functions $f(x) = x^2 + 1$ and $g(x) = x^2 - 1$ defined on the interval $[-1, 1]$ are robust, while the function $h(x) = x^2$, $x \in [-1, 1]$ is not robust.

Using the notion of robustness, we can formulate the main theorem of the paper:

Theorem 1. *Let Ω be a (closed) box in $\mathbb{R}^n$ with rational endpoints, $f : \Omega \to \mathbb{R}^n$ be interval-computable, analytic on Ω, $f \neq 0$ on $\partial\Omega$. Then the existence of a zero of f in Ω is quasi-decidable. That is, there exists a (possibly non-terminating) algorithm that, for such f and Ω, computes a value Z in $\{\mathbf{T}, \mathbf{F}\}$ such that*

- *if* $Z = \mathbf{T}$ *then there exists an* $x \in \Omega$ *s.t.* $f(x) = 0$,
- *if* $Z = \mathbf{F}$ *then there does not exist an* $x \in \Omega$ *s.t.* $f(x) = 0$, *and*
- *the algorithm terminates for all robust inputs.*

3 Algorithm

In this section we present the algorithm whose existence implies the main theorem (i.e., quasi-decidability). The algorithm builds on the notion of degree from the field of differential topology [9,13,16]. For a smooth function $f : \Omega \to \mathbb{R}^n$ and $p \notin f(\partial\Omega)$, the degree of f with respect to Ω and a point $p \in \mathbb{R}^n$ is denoted by $\deg(f, \Omega, p)$. For further details on the degree see Section 4 below.

Let B be a box and let $f : B \to \mathbb{R}^n$ be a function, nowhere zero on ∂B. We propose the following algorithm for proving that f has a robust zero in B.

$\epsilon \leftarrow 1$
loop
  divide B into a grid of boxes of maximal side-length ϵ
  for each closed $(n-1)$-dimensional face C_i of a box C in the grid
    if $0 \in I(f)(C_i)$
      merge all boxes containing C_i
  if there exist a grid element D for which $\deg(f, D, 0) \neq 0$
    return "robust zero exists"
  if for all grid elements D, $0 \notin I(f)(D)$
    return "no zero exists"
  $\epsilon \leftarrow \epsilon/2$

Due to the over-approximation property of interval arithmetic (first property of Definition 1), if the algorithm detects non-existence of a zero, this is indeed correct. This proves the second item of Theorem 1. Moreover, due to the fact that a non-zero degree implies the existence of a zero (see next section), if the algorithm detect existence of a zero, this is also correct. This proves the first item of Theorem 1. The main remaining problem is the third item, that is, to show that the algorithm will terminate (and hence detect zero existence/non-existence) for all robust inputs.

4 Degree of a Continuous Function

In this section we describe some basic properties of the degree, and show, how it can be computed. We have already mentioned in the introduction that in the one-dimensional case, the degree captures the information provided by the intermediate value theorem.

Let $\Omega \subset \mathbb{R}^n$ be open and bounded, $f : \bar{\Omega} \to \mathbb{R}^n$ continuous and smooth (i.e., infinitely often differentiable) in Ω, $p \notin f(\partial\Omega)$. For regular values $p \in \mathbb{R}^n$ (i.e., values p such that for all y with $f(y) = p$, $\det f'(y) \neq 0$), a generalization of

the directional information used in the one-dimensional case, is the sign of the determinant $\det f'(y)$. Adding up those signs results in the explicit definition [13] of $\deg(f, \Omega, p)$ by $\deg(f, \Omega, p) := \sum_{y \in f^{-1}(p)} \operatorname{sign} \det f'(y)$.

See standard textbooks for a generalization to non-regular values [13]. Here we give an alternative, axiomatic definition, that can be shown to be unique. In this approach $\deg(f, \Omega, p)$ is the unique integer satisfying the following properties [6,16, e.g.]:

1. For the identity function I, $\deg(I, \Omega, p) = 1$ iff $p \in \Omega$
2. If $\deg(f, \Omega, p) \neq 0$ then $f(x) = p$ has a solution in Ω
3. If there is a continuous function (a "homotopy") $h : [0,1] \times \bar{\Omega} \to \mathbb{R}^n$ such that $h(0) = f$, $h(1) = g$ and $p \notin h(t, \partial\Omega)$ for all t, then $\deg(f, \Omega, p) = \deg(g, \Omega, p)$
4. If $\Omega_1 \cap \Omega_2 = \emptyset$ and $p \notin f(\partial\Omega_1 \cup \partial\Omega_2)$, then $\deg(f, \Omega_1 \cup \Omega_2, p) = \deg(f, \Omega_1, p) + \deg(f, \Omega_2, p)$
5. $\deg(f, \Omega, p)$, as a function of p, is constant on any connected component of $\mathbb{R}^n \setminus f(\partial\Omega)$.

This can be extended to the case where Ω has dimension n but is is embedded into some higher-dimensional space (in topological terms, f is a differentiable function between two compact oriented manifolds of the same dimensions). For example, if f is a function from a segment c of a curve (i.e., a set of dimension 1) in $\mathbb{R}^k$ to another segment of a curve in $\mathbb{R}^k$, and if $f \neq 0$ on the endpoints of c, then $\deg(f, c, 0)$ is well defined.

The literature provides several articles [22,2,1, e.g.] that claim to provide an algorithm that automatically computes the topological degree. However, they either just contain informal recipes, or require real-number operations for which it is not clear how to implement them on computers, or whose correctness relies on unknown Lipschitz constants. In order to clarify the situation, we give an algorithm here that is based on ideas readily available in the literature, but that does not have those deficiencies.

The algorithm is based on a theorem that recursively reduces the computation of the degree wrt. to a k-dimensional box to the computation of the degree wrt. to a $(k-1)$-dimensional box. The theorem uses the notion of orientation that has a specific meaning in differential topology [9,13,16]. In order to make the material digestible to a more general audience, and in order to demonstrate algorithmic implementability, we describe here an equivalent, but simpler formalization for the special case of boxes (instead of general manifolds).

We define the orientation of a box in $\mathbb{R}^n$ to be a sign $s \in \{1, -1\}$. Let us consider a k-dimensional box B with orientation s. Observe that we can obtain faces of B by replacing one of the intervals constituting B by either its lower or upper bound (the resulting face is a $(k-1)$-dimensional box). Assume that this interval is the r-th (non-constant) interval of B (so $r \in \{1, \ldots, k\}$). Then, if the face results from taking a lower bound, we define the *induced* orientation of the face to be $(-1)^r s$, if it results from taking an upper bound, the orientation is $(-1)^{r+1} s$.

Let D be a finite union of oriented k-boxes. The orientation of a union of oriented boxes is, for our purposes, just the information about the orientation of each box in the union. The induced orientation of ∂D is the set ∂D, consisting of $k-1$-dimensional boxes with orientations induced from the boxes in D.

Theorem 2. *Let B be an oriented finite union of n-dimensional boxes with connected interior. Let $f : B \to \mathbb{R}^n$ be continuous such that $f \neq 0$ on the boundary of B. Let $D_1, \ldots, D_k$ be disjoint subsets of the boundary of B such that D_i is a finite union of $(n-1)$-dimensional boxes and the interior of D_i in ∂B is connected. We denote the boundary of D_i in the topology of ∂B by ∂D_i. The orientation of B induces an orientation of each D_i, $i \in \{1, \ldots, k\}$.*

For $r \in \{1, \ldots, k\}$ we denote by f_r the r-th component of f and $f_{\neg r} := (f_1, \ldots, f_{r-1}, f_{r+1}, \ldots, f_n)$.

Now let $r \in \{1, \ldots, n\}$, and $s \in \{-1, 1\}$ such that

- *for all $i \in \{1, \ldots, k\}$, f_r has constant sign s in D_i,*
- *$\bigcup_{i \in \{1,\ldots,k\}} D_i$ contains all zeros of $f_{\neg r}$ for which f_r has sign s, and*
- *for all $i \in \{1, \ldots, k\}$, $0 \notin f_{\neg r}(\partial D_i)$.*

Then

$$deg(f, B, 0) = (-1)^{r-1} s \sum_{i \in \{1,\ldots,k\}} deg(f_{\neg r}, D_i, 0).$$

Finally, for a one dimensional closed, connected union of oriented boxes D^1 of B with left-most face l and right-most face r (according to orientation)

$$\deg(f, D^1, 0) = \begin{cases} 1 & \text{if } f(l) < 0 < f(r) \\ -1 & \text{if } f(r) < 0 < f(l) \\ 0 & \text{if } f(l) f(r) > 0 \end{cases}$$

Observe that one-dimensional boxes have two faces (that are points) with opposite induced orientation. We define the *left* face to be the face with the opposite induced orientation as the original box, and the *right* face to be the face with the same induced orientation.

When starting the recursion in the above theorem with an n-dimensional box in $\mathbb{R}^n$ of orientation 1, if in the base case D^1 consists of more than one box, then every left face of a box in D^1 is either a boundary point, or the right face of another box. This makes the notion of a left-most face l and right-most face r of D^1 well-defined.

The theorem follows directly from Kearfott [10, Theorem 2.2], which again is a direct consequence of some results of Stenger [22].

Now the algorithm just recursively reduces the computation of the topological degree in dimension n to lower dimension using the above theorem. In each recursive step, for an arbitrary choice for r and s, it computes sets $D_1, \ldots, D_k$ fulfilling the necessary conditions. If f is analytic, $\{f_r = 0\}$ and $\{f_{\neg r} = 0\}$ have a finite number of connected components (see next Section) and the sets D_i can be found by computing an increasingly fine decomposition of the boundary of B

into boxes, and checking the necessary conditions using interval arithmetic. Due to the second property in Definition 1 this procedure will eventually approximate f on the boundary of B arbitrarily closely, and hence it will eventually find such a decomposition.

5 Zeros of Analytic Functions

A function $f : \Omega \to \mathbb{R}^n$ defined on an open set Ω is analytic, iff each of its components $f_1, \ldots, f_n$ is an analytic function, that is, iff for each $i \in \{1, \ldots, n\}$, for each point $x_0 \in \Omega$, there exists a neighborhood U of x_0 and a sequence of numbers $\{c_j\}_{j \in \mathbb{N}_0}$ such that $f_i = \sum_{j \in \mathbb{N}_0} c_j (x - x_0)^j$ on U. The set of analytic functions is closed with respect to addition, multiplication, division by nonzero function, composition and differentiation.

We will need the following statement later:

Theorem 3. *For an analytic function $f : \Omega \to \mathbb{R}^n$, $0 \notin f(\partial\Omega)$, the set $\{f = 0\}$ consists of a finite number of connected components.*

Proof. It follows from Lojasiewicz's theorem [12] that the zero set of a real valued analytic functions is locally a union of a finite number of manifolds of various dimensions. So, the zero set of a real valued analytic function defined on a compact set $\bar{\Omega}$ has a finite number of connected components and the set $\{f = 0\}$ coincides with the zero set of the real valued analytic function $\sum_i f_i^2$ on $\bar{\Omega}$. □

An analogous statement for smooth (but not analytic) functions f does not hold. One can easily construct a smooth function $f : [0, 1] \to \mathbb{R}$ such that $\{f = 0\}$ is the Cantor set which is totally disconnected.

6 Degree and Robustness

In this section, we will clarify the connection between robust solution of a function f and the degree of f. First, we prove that if $\deg(f, \Omega, 0) \neq 0$, then f has a robust zero in Ω. We will use the rest of the section to prove a partial converse of this. We will show that if the degree is zero and the set of solution $f = 0$ is connected, then f does *not* have a robust zero in Ω. This will be used as a main ingredient in the proof of the main theorem, given in the next section.

Theorem 4. *Let $\Omega \subset \mathbb{R}^n$ be an open, and bounded set. Let $f : \bar{\Omega} \to \mathbb{R}^n$ be continuous and smooth on Ω, $0 \notin f(\partial\Omega)$ and let $\deg(f, \Omega, 0) \neq 0$. Then f has a robust zero in Ω.*

Proof. Let $\epsilon < \min_{x \in \partial\Omega} |f|$. For any g such that $||g - f|| < \epsilon$, we define a homotopy $h(t, x) = tf(x) + (1-t)g(x)$ between f and g. We see that for $x \in \partial\Omega$ and $t \in [0, 1]$,

$$|h(t, x)| = |tf(x) + (1 - t)g(x)| = |f(x) + (1 - t)(g(x) - f(x))| \geq |f(x)| - \epsilon > 0$$

so that $h(t, x) \neq 0$ for $x \in \partial\Omega$. From Section 4, properties 2 and 3, we see that $g(x) = 0$ has a solution. □

We will now consider the case when the degree is zero.

Lemma 1. *Let B be homeomorphic to an $n-$dimensional ball, $f : B \to \mathbb{R}^n$ be continuous, nowhere zero on ∂B and let $\deg(f, B, 0) = 0$. Then there exists a continuous nowhere zero function $g : B \to \mathbb{R}^n$ such that $g = f$ on ∂B and $||g|| \leq 2||f||$.*

Proof. We may assume, without loss of generality, that $B = \{x \in \mathbb{R}^n; |x| = 1\}$ and $\partial B = S^{n-1}$ is the $(n-1)$-sphere. Let $R : \mathbb{R}^n \backslash \{0\} \to S^{n-1}$ be defined by $R(x) := x/|x|$. The degree $\deg(f, B, 0) = 0$ is equal to the degree of the function $R \circ f|_{S^{n-1}} : S^{n-1} \to S^{n-1}$ The Hopf theorem [13, pp. 51]) states that the degree classifies continuous self-functions of a sphere up to homotopy. So, $R \circ f|_{S^{n-1}}$ is homotopy equivalent to a constant map. So, there exists a homotopy $F : [0,1] \times S^{n-1} \to S^{n-1}$ such that $F(1, x) = (R \circ f)(x)$ and $F(0, x) = c \in S^{n-1}$. Let $r \in [0,1]$ and $x \in S^{n-1}$. Define the function $g : B \to \mathbb{R}^n$ by

$$g(rx) = F(r, x)(r|f(x)| + (1 - r)||f||).$$

This function is continuous, nowhere zero and well defined because $g(0x) = g(0) = c||f||$ is independent of $x \in S^{n-1}$. Clearly, $g(x) = f(x)$ on S^{n-1} and $|g(rx)| \leq r|f(x)| + (1 - r)||f|| \leq 2||f||$. □

Further, we will need the following technical lemma:

Lemma 2. *Let $\Omega \subset \mathbb{R}^n$ be open and bounded, $f : \bar{\Omega} \to \mathbb{R}^n$ continuous and smooth on Ω, $0 \notin f(\partial\Omega)$. Then there exists a continuous function $\tilde{f} : \bar{\Omega} \to \mathbb{R}^n$, smooth on Ω, with the following properties:*

1. *$\tilde{f} = f$ on $\partial\Omega$,*
2. *0 is a regular value of $\tilde{f}$,*
3. *$||\tilde{f}|| \leq 2||f||$,*
4. *$\tilde{f}$ is homotopy equivalent to f due to 0a homotopy $h(t)$ such that $0 \notin h(t)(\partial\Omega)$.*

Proof. Let U be an open neighborhood of $\partial\Omega$ in $\bar{\Omega}$ such that $f \neq 0$ on U and $\min\{|f(x)|; x \in \bar{U}\} = \epsilon > 0$. From Sard's theorem ([13]), there exists a regular value x_0 of f with $|x_0| < \epsilon/2$. It follows that 0 is a regular value of the function $f(x) - x_0$. Consider a covering of $\partial\Omega$ by open sets $\{U_\alpha; \alpha \in \Lambda_1\}$ such that $U_\alpha \subset U$ for all $\alpha \in \Lambda_1$. As $\partial\Omega$ is compact, we may assume that Λ_1 is finite. Further, define a covering of Ω by open sets $\{U_\beta; \beta \in \Lambda_2\}$. For $\Lambda = \Lambda_1 \cup \Lambda_2$, we have a covering $\{U_\alpha; \alpha \in \Lambda\}$ of the compact space $\bar{\Omega}$. Let $\{\rho_\alpha; \alpha \in \Lambda\}$ be the subordinate partition of unity consisting of continuous functions smooth in Ω and define $\phi(x) := \sum_{\alpha \in \Lambda_1} \rho_\alpha$. Then ϕ is a smooth function supported in $\bar{U}$ such that $\phi = 1$ on $\partial\Omega$ and $\phi = 0$ on $\bar{\Omega} \backslash U$. Define $\tilde{f}(x) = f(x) - (1 - \phi(x))x_0$. Clearly, $\tilde{f} = f$ on $\partial\Omega$. The function $\tilde{f}$ is nowhere zero on $\bar{U}$, because $|\tilde{f}(x)| \geq |f(x)| - |x_0| \geq \epsilon/2$ on $\bar{U}$. On $\bar{\Omega} \backslash \bar{U}$, $\tilde{f}(x) = f(x) - x_0$. In particular, 0 is a regular value of $\tilde{f}$ and $||\tilde{f}|| \leq ||f|| + |x_0| \leq 2||f||$. Finally, a homotopy between f and $\tilde{f}$ may be given by $h(t) = f(x) + (1 - t)\tilde{f}(x)$. □

We have seen in Lemma 1 that if the degree of f on a ball is zero, then we may define a nowhere zero function on the ball that coincides with f on the boundary. We will now see that this is true not only for a ball, but for any connected bounded and open set $\Omega \in \mathbb{R}^n$.

Lemma 3. *Let Ω be a connected, open, bounded subset of $\mathbb{R}^n$, $f : \bar{\Omega} \to \mathbb{R}^n$ continuous and smooth on Ω, $0 \notin f(\partial\Omega)$ and $\deg(f, \Omega, 0) = 0$. Then there exists a continuous nowhere zero function $g : \bar{\Omega} \to \mathbb{R}^n$ such that $g = f$ on $\partial\Omega$ and $||g|| \leq 4||f||$.*

Proof. If the dimension $n = 1$, the function f is defined on an interval $[a, b]$ and the degree assumption implies that $f(a)$ and $f(b)$ have equal signs. So, we may define $g = f(a) + (x - a)(f(b) - f(a))/(b - a)$ and the lemma is proved. Assume further that $n \geq 2$. From lemma 2, we construct a continuous function $\tilde{f} : \bar{\Omega} \to \mathbb{R}^n$ smooth on Ω, $\tilde{f} = f$ on $\partial\Omega$, $||\tilde{f}|| \leq 2||f||$, having 0 as a regular value, homotopy equivalent to f. In particular, $\deg(\tilde{f}, \bar{\Omega}, 0) = 0$.

The compactness of $\bar{\Omega}$ implies that $\tilde{f}^{-1}(0)$ is finite. Because $\deg(\tilde{f}, \Omega, 0) = 0$, we may enumerate the points in $\tilde{f}^{-1}(0)$ as $\{x_1, \ldots, x_{2m}\}$ so that $\tilde{f}$ is orientation-preserving in the neighborhoods of $x_1, x_2, \ldots, x_m$ and orientation-reversing in the neighborhoods of $x_{m+1}, \ldots x_{2m}$.

Choose m smooth, pairwise disjoint, non-self-intersecting curves c_i in Ω connecting x_i and x_{m+i}. This is possible, because the dimension $n \geq 2$ and the complement of a smooth non-self-intersecting curve in an open connected set $\Omega \subset \mathbb{R}^n$ is still open and connected. For these smooth curves, there exist disjoint neighborhoods homeomorphic to balls $B_1, \ldots, B_m$ (see e.g. [13, Product neighborhood theorem]). Because $\deg(\tilde{f}, B_i, 0) = 1 - 1 = 0$, we may apply lemma 1 to construct nowhere zero functions $g_i : \bar{B}_i \to \mathbb{R}^n$ such that $g_i = \tilde{f}$ on ∂B_i and $|g_i(x)| \leq 2||\tilde{f}|| \leq 4||f||$. The resulting function $g(x)$ defined by $g = g_i$ on B_i and $\tilde{f}$ elsewhere is continuous and has the properties required. □

We now prove a partial conversion of Theorem 4.

Theorem 5. *Let Ω be open, connected, bounded, $f : \bar{\Omega} \to \mathbb{R}^n$ continuous and smooth on Ω, $0 \notin f(\partial\Omega)$. Let f have a robust zero in Ω and assume that the set $\{f = 0\} \subset \Omega$ is connected. Then $\deg(f, \Omega, 0) \neq 0$.*

Proof. Let $\epsilon > 0$. Since $\{f = 0\}$ is connected, is is contained in a single connected component Ω' of the open set $\{x; |f(x)| < \epsilon\}$. Let $\deg(f, \Omega, 0) = 0$. Applying lemma 2 to the set Ω', we construct a continuous function $\tilde{f} : \bar{\Omega}' \to \mathbb{R}^n$ smooth on Ω', homotopy equivalent to $f : \bar{\Omega}' \to \mathbb{R}^n$, $\tilde{f} = f$ on $\partial\Omega'$, having 0 as a regular value and $||\tilde{f}|| \leq 2||f_{|\bar{\Omega}'}|| \leq 2\epsilon$. Because the set $\{f = 0\}$ is connected and contained in Ω', we obtain that $\deg(f, \Omega, 0) = \deg(f, \Omega', 0) = \deg(\tilde{f}, \Omega', 0) = 0$. Using lemma 3, we obtain a continuous function $g : \bar{\Omega}' \to \mathbb{R}^n$ such that $g = \tilde{f}$ on $\partial\Omega'$, $g \neq 0$ on $\bar{\Omega}'$ and $|g| \leq 2|\tilde{f}| \leq 4\epsilon$ on $\bar{\Omega}'$. Extending g to all $\bar{\Omega}$ by $g = f$ on $\bar{\Omega}\backslash\Omega'$, we obtain an everywhere nonzero continuous function g such that $||g - f|| \leq 5\epsilon$. This can be done for any ϵ and it follows that f has no robust zero in Ω. □

7 Proof of the Main Theorem

As we have seen in Section 3, the algorithm presented there—if it terminates—will always correctly detect the existence of a zero. The main remaining problem for proving quasi-decidability, was termination of this algorithm for robust inputs. In the positive case of existence of a zero, we will actually prove that this holds in both directions:

Theorem 6. *Let B be a box in $\mathbb{R}^n$, $f : B \to \mathbb{R}^n$ be continuous in B and analytic in the interior of B, $f \neq 0$ on ∂B. Assume that f has a zero in B. Then the algorithm proposed in Section 3 terminates with the output "robust zero exists" if and only if f has a robust zero in B.*

Proof. First, if the algorithm terminates, then $\deg(f, D, 0) \neq 0$, for some union of boxes $D \subset B$ and it follows from Theorem 4 that f has a robust zero in D. So, f has a robust zero in B.

Suppose now that the function f has a robust zero in B. We know from section 5 that for an analytic function f, there exists a finite number of connected components of $\{f = 0\}$, so any two of them have positive distance. Let as denote the component by $Z_1, \ldots, Z_m$. Let $U_1, \ldots U_m$ be open connected neighborhoods of $Z_1, \ldots, Z_m$ such that $\bar{U}_1, \ldots \bar{U}_m$ are disjoint. If $\deg(f, \bar{U}_i, 0) = 0$, it follows from Theorem 5 that for any $\epsilon > 0$, there exists a continuous function $g_i : \bar{U}_i \to \mathbb{R}^n$ such that g_i is nowhere zero, $||g_i - f_{|\bar{U}_i}|| \leq \epsilon$ and $g_i = f$ on $\partial \bar{U}_i$. Replacing f by g_i on $\bar{U}_i$, we would obtain a continuous nowhere zero function $g : B \to \mathbb{R}^n$, $||g - f|| \leq \epsilon$, contradicting the assumption that f has a robust zero in Ω. So, at least for one i, $\deg(f, \bar{U}_i, 0) \neq 0$. Because $\partial \bar{U}_i$ is compact, it has positive distance $d > 0$ from Z_i. Let as assume that $\epsilon \leq d/(\sqrt{n})$. If we cover B with a grid of boxes of side-length smaller then ϵ, any box that has nonempty intersection with Z_i is contained in U_i. Possibly merging the boxes whose boundaries intersect Z_i (if there are any), we obtain a set $D \subset U_i$ such that the interior of D is a neighborhood of Z_i. Because the sets U_i are disjoint, the zero set of f in U_i is just Z_i. So, $\deg(f, D, 0) = \deg(f, \bar{U}_i, 0) \neq 0$ and the algorithm terminates. □

In the case where the input f does not have a zero, termination for robust inputs holds due to the second property of $I(f)$ in Definition 1. The main theorem of this paper follows as a corollary.

8 Possible Generalizations

We needed analyticity of f only to be sure that $\{f = 0\}$ decomposes into a finite number of connected components. If f has this property, (e.g. if $\{f = 0\}$ is discrete), the algorithm terminates for a robust input.

We will now discuss the generalization of the theorem to the case where the number of variables is not equal to the number of equations. That is, we have the situation where we have an n-dimensional box B in $\mathbb{R}^n$, $f : B \to \mathbb{R}^m$ analytic. In the case were $n < m$ (i.e. the number of variables is smaller then the number

of equations) it follows from Sard's theorem, that there exist regular values $y \in \mathbb{R}^m$ arbitrarily close to 0. So, some neighborhood U of $f^{-1}(y)$ is mapped homeomorphically to $f(U) \in \mathbb{R}^n$. But an n-dimensional open set can not be homeomorphic to an open set in $\mathbb{R}^m$ for $m \neq n$, so $f(U)$ does not contain a neighborhood of y. So, $0 \in \mathbb{R}^m$ is not in the interior of the image of f and f cannot have a robust zero in B.

On the other hand, for $n > m$ (more variables then equations), the situation is much more subtle. In some cases, we could fix some $n - m$ input variables in f to be constants $a \in \mathbb{R}^{n-m}$ and try to solve the equation $f(a, x) = 0$. This is a function from a subset of $\mathbb{R}^m$ to $\mathbb{R}^m$ and if it has a robust zero in $\{x \in \mathbb{R}^m;\ (a, x) \in B\}$, so, clearly, f has a robust zero in B. The converse, however, is not true. If $f(a, x)$ does not have a robust zero for any fixed choice of $a \in \mathbb{R}^{n-m}$ (a ranging from any subset of $m - n$ from the total number of n variables), f still may have a robust zero in B. This makes a straightforward generalization of our result difficult.

The ideas presented in Section 6 may be easily generalized to arbitrary dimensions, if the condition of a nonzero degree is replaced by the condition "the map $f/|f|$ from $\partial\Omega$ to the sphere S^{m-1} can be extended to a map from Ω to S^{m-1}". This is the extension problem in computational homotopy theory. If Ω is the unit ball in $\mathbb{R}^2$, $\partial\Omega = S^1$ and X is an arbitrary space, the question whether or not $f : \partial\Omega \to X$, presented algorithmically, can be extended to $f : \Omega \to X$ is equivalent to the word problem and there is no algorithm to solve it [21]. The question whether or not such an algorithm exists for X being the sphere and Ω arbitrary, is—up to our knowledge—an open problem.

9 Conclusion

In the paper, we have proved that the problem of checking satisfiability of systems of equations of real analytic functions in a box is quasi-decidable in the sense that there exists an algorithm that successfully can do this check in all robust cases. Hence, problems that correspond to application domains where perturbations in the form of modeling errors, manufacturing imprecision etc. occur, are solvable in practice (provided enough computing power is available).

The generalization to the full first-order case is an open problem.

References

1. Aberth, O.: Computation of topological degree using interval arithmetic, and applications. Mathematics of Computation 62(205), 171–178 (1994)
2. Boult, T.E., Sikorski, K.: Complexity of computing topological degree of lipschitz functions in n dimensions. J. Complexity 2, 44–59 (1986)
3. Caviness, B.F., Johnson, J.R. (eds.): Quantifer Elimination and Cylindrical Algebraic Decomposition. Springer, Heidelberg (1998)
4. Collins, P.: Computability and representations of the zero set. Electron. Notes Theor. Comput. Sci. 221, 37–43 (2008)

5. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. International Journal of Foundations of Computer Science (IJFCS) 18(1), 63–86 (2007)
6. Fonseca, I., Gangbo, W.: Degree Theory in Analysis and Applications. Clarendon Press, Oxford (1995)
7. Fränzle, M.: Analysis of hybrid systems: An ounce of realism can save an infinity of states. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 126–140. Springer, Heidelberg (1999)
8. Frommer, A., Lang, B.: Existence tests for solutions of nonlinear equations using Borsuk's theorem. SIAM Journal on Numerical Analysis 43(3), 1348–1361 (2005)
9. Hirsch, M.: Differential topology. Springer, Heidelberg (1976)
10. Kearfott, R., Dian, J., Neumaier, A.: Existence verification for singular zeros of complex nonlinear systems. SIAM J. Numer. Anal. 38(2), 360–379 (2000)
11. Kearfott, R.B.: On existence and uniqueness verification for non-smooth functions. Reliable Computing 8(4), 267–282 (2002)
12. Krantz, S., Parks, H.R.: A Primer of Real Analytic Functions. Birkhäuser, Basel (2002)
13. Milnor, J.W.: Topology from the Differential Viewpoint. Princeton Univ. Press, Princeton (1997)
14. Munkres, J.: Topology. Prentice Hall, Englewood Cliffs (1999)
15. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge Univ. Press, Cambridge (1990)
16. O'Regan, D., Cho, Y., Chen, Y.Q.: Topological Degree Theory and Applications. Chapman & Hall, Boca Raton (2006)
17. Ratschan, S.: Quantified constraints under perturbations. Journal of Symbolic Computation 33(4), 493–505 (2002)
18. Ratschan, S.: Efficient solving of quantified inequality constraints over the real numbers. ACM Transactions on Computational Logic 7(4), 723–748 (2006)
19. Ratschan, S.: Safety verification of non-linear hybrid systems is quasi-semidecidable. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 397–408. Springer, Heidelberg (2010)
20. Rump, S.M.: A note on epsilon-inflation. Reliable Computing 4, 371–375 (1998)
21. Sergeraert, F.: Introduction to combinatorial homotopy theory. In: Summer School and Conference Mathematics, Algorithms and Proofs, Lecture notes (2008), `http://www-fourier.ujf-grenoble.fr/~sergerar/Papers/Trieste-LectureNotes.pdf`
22. Stenger, F.: Computing the topological degree of a mapping in R^n. Numerische Mathematik 25, 23–38 (1975)
23. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. Univ. of California Press, Berkeley (1951) (also in [3])
24. Weihrauch, K.: Introduction to Computable Analysis. Texts in Theoretical Computer Science. Springer, Heidelberg (2000)

On Minimising Automata with Errors*

Paweł Gawrychowski[1,**], Artur Jeż[1,**], and Andreas Maletti[2,***]

[1] Institute of Computer Science, University of Wrocław
ul. Joliot-Curie 15, 50-383 Wrocław, Poland
{gawry,aje}@cs.uni.wroc.pl
[2] Institute for Natural Language Processing, Universität Stuttgart
Azenbergstraße 12, 70174 Stuttgart, Germany
andreas.maletti@ims.uni-stuttgart.de

Abstract. The problem of k-minimisation for a DFA M is the computation of a smallest DFA N (where the size $|M|$ of a DFA M is the size of the domain of the transition function) such that $L(M) \triangle L(N) \subseteq \Sigma^{<k}$, which means that their recognized languages differ only on words of length less than k. The previously best algorithm, which runs in time $\mathcal{O}(|M| \log^2 n)$ where n is the number of states, is extended to DFAs with partial transition functions. Moreover, a faster $\mathcal{O}(|M| \log n)$ algorithm for DFAs that recognise finite languages is presented. In comparison to the previous algorithm for total DFAs, the new algorithm is much simpler and allows the calculation of a k-minimal DFA for each k in parallel. Secondly, it is demonstrated that calculating the least number of introduced errors is hard: Given a DFA M and numbers k and m, it is NP-hard to decide whether there exists a k-minimal DFA N with $|L(M) \triangle L(N)| \leq m$. A similar result holds for hyper-minimisation of DFAs in general: Given a DFA M and numbers s and m, it is NP-hard to decide whether there exists a DFA N with at most s states such that $|L(M) \triangle L(N)| \leq m$.

Keywords: finite automaton, minimisation, lossy compression.

1 Introduction

Deterministic finite automata (DFAs) are one of the simplest devices recognising languages. The study of their properties is motivated by (i) their simplicity, which yields efficient operations, (ii) their wide-spread applications, (iii) their connections to various other areas in theoretical computer science, and (iv) the apparent beauty of their theory. A DFA M is a quintuple $\langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is its finite state-set, Σ is its finite alphabet, $\delta\colon Q \times \Sigma \to Q$ is its partial transition function, $q_0 \in Q$ is its starting state, and $F \subseteq Q$ is its set of accepting

* This work was partially done when A. Maletti was visiting Wrocław University thanks to the support of the "Visiting Professors" programme of the Municipality of Wrocław.
** Supported by MNiSW grant number N N206 492638, 2010–2012.
*** Supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760 and the German Research Foundation (DFG) grant MA/4959/1-1.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 327–338, 2011.

states. The DFA M is total if δ is total. The transition function δ is extended to $\delta\colon Q \times \Sigma^* \to Q$ in the standard way. The language $L(M)$ that is *recognised by the DFA* M is $L(M) = \{w \mid \delta(q_0, w) \in F\}$.

Two DFAs M and N are *equivalent* (written as $M \equiv N$) if $L(M) = L(N)$. A DFA M is *minimal* if all equivalent DFAs are at least as large. One of the classical DFA problems is the *minimisation problem*, which given a DFA M asks for the (unique) minimal equivalent DFA. The asymptotically fastest DFA minimisation algorithm runs in time $\mathcal{O}(|\Sigma|\, n \log n)$ and is due to Hopcroft [7,5], where $n = |Q|$; its variant for partial DFAs is known to run in time $\mathcal{O}(|M| \log n)$.

Recently, minimisation was also considered for hyper-equivalence [2], which allows a finite difference in the languages. Two languages L and L' are *hyper-equivalent* if $|L \triangle L'| < \infty$, where $\triangle$ denotes the symmetric difference of two sets. The DFAs M and N are hyper-equivalent if their recognised languages are. The DFA M is *hyper-minimal* if all hyper-equivalent DFAs are at least as large. The algorithms for hyper-minimisation were gradually improved over time to the currently best run-time $\mathcal{O}(|M| \log^2 n)$ [6,4], which can be reduced to $\mathcal{O}(|M| \log n)$ using a strong computational model (with randomisation or special memory access). Since classical DFA minimisation linearly reduces to hyper-minimisation [6], an algorithm that is faster than $\mathcal{O}(|M| \log n)$ seems unlikely. Moreover, according to the authors' knowledge, randomisation does not help Hopcroft's [3] or any other DFA minimisation algorithm. Thus, the randomised hyper-minimisation algorithm also seems to be hard to improve.

Already [2] introduces a stricter notion of hyper-equivalence. Two languages L and L' are *k-similar* if they only differ on words of length less than k. Analogously, DFAs are k-similar if their recognised languages are. A DFA M is *k-minimal* if all k-similar DFAs are at least as large, and the *k-minimisation problem* asks for a k-minimal DFA that is k-similar to the given DFA M. The known algorithm [4] for k-minimisation of total DFAs runs in time $\mathcal{O}(|M| \log^2 n)$, however it is quite complicated and fails for non-total DFAs.

In this contribution, we present a simpler k-minimisation algorithm for general DFAs, which still runs in time $\mathcal{O}(|M| \log^2 n)$. This represents a significant improvement compared to the complexity for the corresponding total DFA if the transition table of M is sparse. Its running time can be reduced if we allow a stronger computational model. In addition, the new algorithm runs in time $\mathcal{O}(|M| \log n)$ for every DFA M that recognises a finite language. Finally, the new algorithm can calculate (a compact representation of) a k-minimal DFA for each possible k in a single run (in the aforementioned run-time). Outputting all the resulting DFAs might take time $\Omega(n|M| \log^2 n)$.

Although k-minimisation can be efficiently performed, no uniform bound on the number of introduced errors is provided. In the case of hyper-minimisation, it is known [8] that the *optimal* (i.e., the DFA committing the least number of errors) hyper-minimal DFA and the number of its errors m can be efficiently computed. However, this approach does not generalise to k-minimisation. We show the reason. Already the problem of calculating the number m of errors of an optimal k-minimal automaton is NP-hard. Finally, for some applications

it would be beneficial if we could balance the number m of errors against the size $|N|$. Thus, we also consider the question whether given a DFA M and two integers s and m there is a DFA N with at most s states that commits at most m errors (i.e., $|L(M) \triangle L(N)| \leq m$). We show that this problem is also NP-hard.

2 Preliminaries

We usually use the two DFAs $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ and $N = \langle P, \Sigma, \mu, p_0, F'\rangle$. We also write $\delta(w)$ for $\delta(q_0, w)$. The size of DFA M is denoted by $|M|$ and is the number of its non-empty transitions, i.e., entries of δ. The *right-language* $L_M(q)$ of a state $q \in Q$ is the language $L_M(q) = \{w \mid \delta(q, w) \in F\}$ recognised by M starting in state q. Minimisation of DFAs is based on calculating the equivalence $\equiv$ between states, which is defined by $q \equiv p$ if and only if $L_M(q) = L_N(p)$. Similarly, the *left language* of q is the language $\delta^{-1}(q) = \{w \mid \delta(w) = q\}$ of words leading to q in M. For two languages L and L', we define their *distance* $d(L, L')$ as

$$d(L, L') = \min\{\ell \mid L \cap \Sigma^{\geq \ell} = L' \cap \Sigma^{\geq \ell}\} \ ,$$

where $\min \emptyset = \infty$. Actually, d is an ultrametric. The distance d can be extended to states: $d(q, p) = d(L_M(q), L_N(p))$ for $q \in Q$ and $p \in P$. It satisfies the simple recursive formula:

$$d(q, p) = \begin{cases} 0 & \text{if } q \equiv p, \\ 1 + \max\{d(\delta(q, a), \mu(p, a)) \mid a \in \Sigma\} & \text{otherwise.} \end{cases} \tag{1}$$

The minimal DFAs considered in this paper are obtained mostly by state merging. We say that the DFA N is the result of *merging state q to state p* (assuming $q \neq p$) in M if N is obtained from M by changing all transitions ending in q to transitions ending in p and deleting the state q. If q was the starting state, then p is the new starting state. Formally, $P = Q \setminus \{q\}$, $F' = F \setminus \{q\}$, and

$$\mu(r, a) = \begin{cases} p & \text{if } \delta(r, a) = q \\ \delta(r, a) & \text{otherwise,} \end{cases} \qquad p_0 = \begin{cases} p & \text{if } q_0 = q \\ q_0 & \text{otherwise.} \end{cases}$$

The process is illustrated in Fig. 1. Let $\text{in-level}_M(q)$ be the length of a longest word leading to q in M. If there is no such longest word, then $\text{in-level}_M(q) = \infty$. Formally, we have $\text{in-level}_M(q) = \sup\{|w| \mid w \in \delta^{-1}(q)\}$ for every $q \in Q$.

3 Efficient k-minimisation

3.1 k-similarity and k-minimisation

Two languages L and L' are *k-similar* if they only differ on words of length smaller than k, and the two DFAs M and N are k-similar if their recognised languages are. The DFA M is *k-minimal* if all k-similar DFAs are at least as large. In this section, we present a general simple algorithm k-Minimise that computes a k-minimal DFA that is k-similar to the input DFA M. Then we present a data structure that allows a fast, yet simple implementation of it.

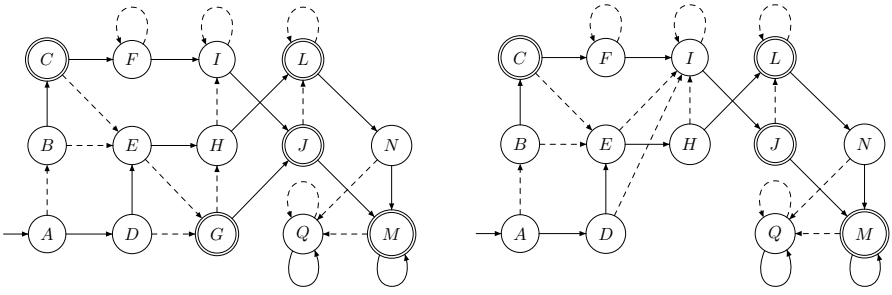


Fig. 1. Merging state G into I

Definition 1. *For two languages L and L', we let $L \sim_k L' \iff d(L, L') \leq k$.*

The hyper-equivalence relation [2] can now be defined as $\sim = \bigcup_k \sim_k$. Next, we extend k-similarity to states.

Definition 2. *Two states $q \in Q$ and $p \in P$ are k-similar, denoted by $q \sim_k p$, if*

$$d(q, p) + \min(k, \text{in-level}_M(q), \text{in-level}_N(p)) \leq k \ .$$

While $\sim_k$ is an equivalence relation on languages, it is, in general, only a compatibility relation (i.e., reflexive and symmetric) on states. On states the hyper-equivalence is not a direct generalisation of k-similarity. Instead, $p \sim q$ if and only if $L_M(q) \sim L_N(p)$. We use the k-similarity relation to give a simple algorithm k-Minimise(M), which constructs a k-minimal DFA (see Alg. 1). In Sect. 3.2 we show how to implement it efficiently.

Theorem 3. *k-Minimise returns a k-minimal DFA that is k-similar to M.*

Proof (sketch). There are two things to show: (i) that the obtained DFA N has the minimal number of states and (ii) that it is k-similar to M. The states of N are pairwise k-inequivalent (when considered in M) and using an approach similar to [4, Lemma 6] it naturally follows that each DFA that is k-similar to M has at least this number of states. For part (ii) we show that after each merge the current DFA N is k-similar to M. To this end, we first show that $\text{in-level}_N(p) \leq \text{in-level}_M(p)$ using a little more general induction hypothesis. Next, we estimate the distance between p regarded as a state in M and in N follows: $d(L_M(p), L_N(p)) \leq k - \text{in-level}_M(p)$. The rest of the proof are simple calculations using that d is an ultrametric. □

3.2 Distance Forests

In this section we define distance forests, which capture the information of the distance between states of a given minimal DFA M. We show that k-minimisation

Alg. 1. k-MINIMISE(M) with minimal M

```
1: calculate ∼_k on Q
2: N ← M
3: while q ∼_k p for some q, p ∈ P and q ≠ p do
4:     if in-level_M(q) ≥ in-level_M(p) then
5:         swap q and p
6:     N ← MERGE(N, q, p)
```

can be performed in linear time, when a distance forest for M is supplied. We start with a total DFA M because in this case the construction is fairly easy. In Sect. 3.3 we show how to extend the construction to non-total DFAs.

Let $\mathcal{F}$ be a forest (i.e., set of trees) whose leaves are enumerated by Q and whose edges are weighted by elements of $\mathbb{N}$. For convenience, we identify the leaf vertices with their label. For every $q \in Q$, we let $\mathrm{tree}(q) \in \mathcal{F}$ be the (unique) tree that contains q. The level 'level(v)' of a vertex v in $t \in \mathcal{F}$ is the maximal weight of all paths from v to a leaf, where the weights are added along a path. Finally, given two vertices v_1, v_2 of the same tree $t \in \mathcal{F}$, the lowest common ancestor of v_1 and v_2 is the vertex $\mathrm{lca}(v_1, v_2)$.

Definition 4 (Distance forest). *Let $\mathcal{F}$ be a forest whose leaves are enumerated by Q. Then $\mathcal{F}$ is a* distance forest *for M if for every $q, p \in Q$ we have*

$$d(q,p) = \begin{cases} \mathrm{level}(\mathrm{lca}(q,p)) & \text{if } \mathrm{tree}(q) = \mathrm{tree}(p), \\ \infty & \text{otherwise.} \end{cases}$$

To construct a distance forest we use (1) to calculate the distance. Since M is minimal, there are no states with distance 0. In phase ℓ, we merge all states at distance exactly ℓ into one state. Since we merged all states of distance at most $\ell - 1$ in the previous phases, we only need to identify the states of distance 1 in the merged DFA. Thus we simply group the states according to their vectors of transitions by letters from $\Sigma = \{a_1, \ldots, a_m\}$. To this end we store these vectors in a dictionary, which we organise as a trie of depth m. The leaf of a trie corresponding to a path $(q_1, \ldots, q_m)$ keeps a list of all states q such that $\delta(q, a_i) = q_i$ for every $1 \le i \le m$. For each node v in the trie we keep a *linear dictionary* that maps a state q into a child of v. We demand that this linear dictionary supports search, insertion, deletion, and enumeration of all elements.

Theorem 5. *Given a total DFA M, we can build a distance forest for M using $\mathcal{O}(|M| \log n)$ linear-dictionary operations.*

We now shortly discuss some possible implementations of the linear dictionary. An implementation using balanced trees would have linear space consumption and the essential operations would run in time $\mathcal{O}(\log n)$. If we allow randomisation, then we can use dynamic hashing. It has a worst-case constant time look-up and an amortised expected constant time for updates [9]. Since it is natural to assume that $\log n$ is proportional to the size of a machine word, we can hash

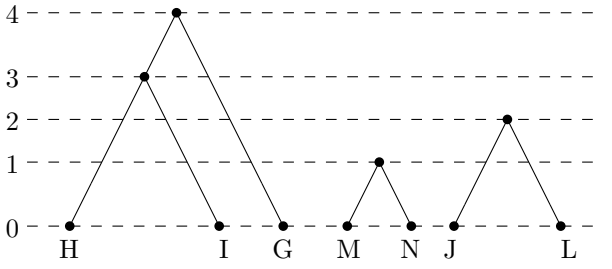


Fig. 2. A distance forest for the left DFA of Fig. 1. Single-node trees are omitted.

in constant time. We can obtain even better time bounds by turning to more powerful models. In the RAM model, we can use exponential search trees [1], whose time per operation is $\mathcal{O}(\frac{(\log\log n)^2}{\log\log\log n})$ in linear space. Finally, if we allow a quadratic space consumption, which is still possible in sub-quadratic time, then we can allocate (but not initialise) a table of size $|M| \times n$. Standard methods can be used to keep track of the actually used table entries, so that we obtain a constant run-time for each operation, but at the expense of $\Theta(|M|\, n)$ space; i.e., quadratic memory consumption.

We can now use a distance forest to efficiently implement k-Minimise. For each state q we locate its highest ancestor v_q with $\text{level}(v_q) \leq k - \text{in-level}(q)$. Then q can be merged into any state that occurs in the subtree rooted in v_q (assuming it has a smaller in-level). This can be done using a depth-first traversal on the trees of the distance forest. A more elaborate construction based on this approach yields the following.

Theorem 6. *Given a distance forest for M, we can compute the size of a k-minimal DFA that is k-similar to M for all k in time $\mathcal{O}(|M|)$. For a fixed k, we can also compute a k-minimal DFA in time $\mathcal{O}(|M|)$. Finally, we can run the algorithm in time $\mathcal{O}(|M| \log n)$ such that it has a k-minimal DFA stored in memory in its k-th phase.*

3.3 Finite Languages and Partial Transition Functions

The construction of a distance forest was based on a total transition function δ, and the run-time was bounded by the size of δ. We now show a modification for the non-total case. The main obstacle is the construction of a distance forest for an acyclic DFA. The remaining changes are relatively straightforward.

Theorem 7. *For every acyclic DFA M we can build a distance forest in time $\mathcal{O}(|M| \log n)$.*

Proof (sketch). Since $L(M)$ is finite, we have that $m(p) = \max\{|w| \,|\, w \in L_M(p)\}$ is a natural number for every state p. Let $Q_i = \{p \,|\, m(p) = i\}$ and $Q_{<\infty} = \bigcup_i Q_i$. Every state has a finite right-language, and thus every distance forest consists of a single tree. We iteratively construct the fragments of this tree by starting from

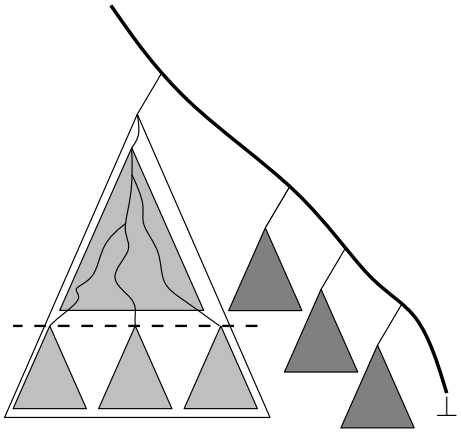


Fig. 3. Illustration for the construction of the distance tree. The spine is depicted using with a thicker line. Splitting one fragment into smaller recursive calls is shown.

a single leaf $\perp$, which represents the empty language and "undefinedness" of the transition function. Before we start to process Q_t, we have already constructed the distance tree for $\bigcup_{i<t} Q_i$. The constructed fragments are connected to a single path, called the *spine*, which ends at the leaf $\perp$ (see Fig. 3).

Let $Q_t = \{p_1, \ldots, p_s\}$, and let $v \in Q_t$. Moreover, let $\mathrm{f}(v)$ be the vector of states $\mathbf{v} = (\delta(v, a))_{a \in \Sigma}$, where the coordinates are sorted by a fixed order on Σ. Define the distance between those vectors as

$$d((p_a)_{a\in\Sigma}, (p'_a)_{a\in\Sigma}) = \max\{d(p_i, p'_i) + 1 \mid a \in \Sigma\} \ ,$$

where we know that $d(p_i, \perp) = m(p_i)$ and $d(\perp, p'_i) = m(p'_i)$. Similarly to the distance, we can define the father $\mathrm{f}(\mathbf{v})$ of a vector $\mathbf{v} = (p_a)_{a\in\Sigma}$ as $\mathrm{f}(\mathbf{v}) = (\mathrm{f}(p_a))_{a\in\Sigma}$. Then

$$\mathrm{f}^{\ell+1}(v) = \mathrm{f}^{\ell+1}(v') \iff \mathrm{f}^{\ell}(\mathbf{v}) = \mathrm{f}^{\ell}(\mathbf{v}').$$

We can now use a divide-and-conquer approach: First, for each vector we calculate its 2^k-th ancestor, where $k = \lceil \log s/2 \rceil$. Then all such vectors are sorted according to their ancestors, in particular they are partitioned into blocks with the same ancestors. After that we recurse onto those (bottom) blocks that have more than two entries and onto the upper block, which consists of the different 2^k-ancestors. The recursion ends for blocks containing at most two vectors, for which we calculate the distance tree directly. □

For every state $q \in Q$, its *signature* $\mathrm{sig}(q)$ is $\{a \mid L_M(\delta(q, a)) \text{ is infinite}\}$. If $\mathrm{sig}(q) \neq \mathrm{sig}(p)$, then $d(q, p) = \infty$, which allows us to keep a separate dictionary for each signature. Let us fix such a trie. To take into account also the transitions by letters outside the signature, we introduce a fresh letter \$, whose transitions are represented in the trie as well. We organize them such that in phase ℓ the \$-transitions for the states q and p are the same if and only if

$\max\{d(\delta(q,a),\delta(p,a)) \mid a \notin \mathrm{sig}(q)\} \leq \ell - 1$. This is easily organised if the distance forest for all states with a finite right-language is supplied.

Theorem 8. *Given a (non-total) DFA M we can build a distance forest for it using $\mathcal{O}(|M| \log n)$ linear-dictionary operations.*

4 Hyper-equivalence and Hyper-minimisation

When considering minimisation with errors, it is natural that one would like to impose a bound on the total number of errors introduced by minimisation. In this section, we investigate whether given $m, s \in \mathbb{N}$ and a DFA M we can construct a DFA N such that:

1. N is hyper-equivalent to M; i.e., $N \sim M$,
2. N has at most s states, and
3. N commits at most m errors compared to M; i.e., $|L(N) \mathbin{\triangle} L(M)| \leq m$.

Let us call the general problem 'error-bounded hyper-minimisation'. We show that this problem is intractable (NP-hard).

To show NP-hardness of the problem we reduce the 3-colouring problem to it. Roughly speaking, we construct the DFA M from a graph $G = \langle V, E\rangle$ as follows. Each vertex $v \in V$ is represented by a state $v \in Q$, and each edge $e \in E$ is represented by a symbol $e \in \Sigma$. We introduce additional states in a way such that their isomorphic copies are present in any minimal DFA that is hyper-equivalent to M. The additional states are needed to ensure that for every edge $e = \{v_1, v_2\} \in E$ the languages $L_M(\delta(v_1, e))$ and $L_M(\delta(v_2, e))$ differ. Now we assume that $m = |E| \cdot (|V| - 2)$ and $s = 14$. We construct the DFA M such that all vertices of $V \subseteq Q$ are hyper-equivalent to each other and none is hyper-equivalent to any other state. We can save $|V| - 3$ states by merging all states of V into at most 3 states. These merges will cause at least $|E| \cdot (|V| - 2)$ errors. Additionally, 3 states will become superfluous after the merges, so that we can save $|V|$ states. There are two cases:

- If the input graph G is 3-colourable by $c\colon V \to [3]$, then we can merge all states of $c^{-1}(i)$ into a single state for every $i \in [3]$. Since c is proper, we never merge states $v_1, v_2 \in Q$ with $\{v_1, v_2\} \in E$, which avoids further errors.
- On the other hand, if G is not 3-colourable, then we merge at least two states $v_1, v_2 \in Q$ such that $e = \{v_1, v_2\} \in E$. This merge additionally introduces 2 errors caused by the difference $L(\delta(v_1, e)) \mathbin{\triangle} L(\delta(v_2, e))$.

Consequently, a DFA that (i) is hyper-equivalent to M, (ii) has at most s states, and (iii) commits at most m errors exists if and only if G is 3-colourable. This shows that error-bounded hyper-minimisation is NP-hard.

Definition 9. *We construct a DFA $M = \langle Q, \Sigma, \delta, \top, F\rangle$ as follows:*

- $Q = \{\top, \bot, \infty, ☺, ☹\} \cup V \cup \{\bigcirc_j \mid \bigcirc \in \{☺, ☻, ☹\}, j \in [3]\}$,
- $\Sigma = \{a, b\} \cup V \cup E$,

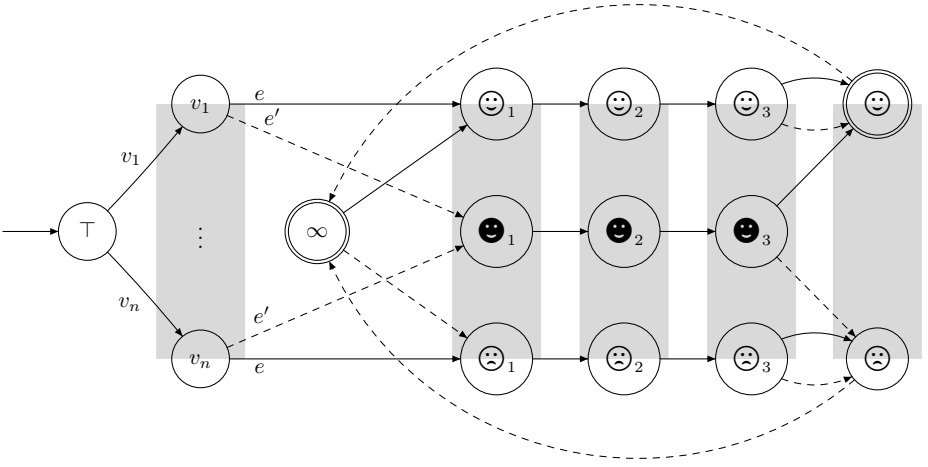


Fig. 4. DFA M constructed in Sect. 4, where a-transitions are represented by unbroken lines (unless noted otherwise), b-transitions by dashed lines, and $e = \{v_1, v_n\}$ and $e' = \{v_2, v_3\}$ with $v_1 < v_2 < v_3 < v_n$. The hyper-equivalence $\sim$ is indicated.

- $F = \{\infty, ☺\}$,
- *for every* $v \in V$, $e = \{v_1, v_2\} \in E$ *with* $v \notin e$ *and* $v_1 < v_2$, $○ \in \{☺, ☹\}$

$$\delta(\top, v) = v \qquad \delta(\infty, a) = ☺_1 \qquad \delta(\infty, b) = ☹_1$$
$$\delta(v, e) = ●_1 \qquad \delta(v_1, e) = ☺_1 \qquad \delta(v_2, e) = ☹_1$$
$$\delta(●_1, a) = ●_2 \quad \delta(●_2, a) = ●_3 \quad \delta(●_3, a) = ☺ \quad \delta(●_3, b) = ☹$$
$$\delta(○_1, a) = ○_2 \quad \delta(○_2, a) = ○_3 \quad \delta(○_3, a) = ○ \quad \delta(○_3, b) = ○ \quad \delta(○, b) = \infty$$

- *For all remaining cases, we set* $\delta(q, \sigma) = \bot$.

Consequently, the DFA M has $14 + |V|$ states (see Fig. 4). Next, we show how to collapse hyper-equivalent states using a proper 3-colouring $c\colon V \to [3]$ to obtain only 14 states.

Definition 10. *Let* $c\colon V \to [3]$ *be a proper* 3*-colouring for* G. *We construct the DFA* $c(M) = \langle P, \Sigma, \mu, \top, F\rangle$ *where*

- $P = \{\top, \bot, \infty, ☺, ☹\} \cup [3] \cup \{○_j \mid ○ \in \{☺, ☹\}, j \in [3]\}$,
- $\mu(p, \sigma) = \delta(p, \sigma)$ *for all* $p \in P \setminus \{\top, 1, 2, 3\}$ *and* $\sigma \in \Sigma$, *and*
- *for every* $v \in V$, $i \in [3]$, *and* $e = \{v_1, v_2\} \in E$ *with* $v_1 < v_2$

$$\mu(\top, v) = c(v) \qquad \mu(i, e) = \begin{cases} ☺_1 & \text{, if } c(v_2) \neq i \\ ☹_1 & \text{, otherwise.} \end{cases}$$

Lemma 11. *There exists a DFA that has at most* 14 *states and commits at most* $|E| \cdot (|V| - 2)$ *errors when compared to* M *if and only if* G *is* 3*-colourable.*

Corollary 12. *'Error-bounded hyper-minimisation' is NP-complete. More formally, given a DFA M and two integers $m, s \in \mathrm{poly}(|M|)$, it is NP-complete to decide whether there is a DFA N with at most s states and $|L(M) \triangle L(N)| \leq m$.*

5 Error-Bounded k-minimisation

In Sect. 3 the number of errors between M and the constructed k-minimal DFA was not calculated. In general, there is no unique k-minimal DFA for M and the various k-minimal DFAs for M can differ in the number of errors that they commit relative to M. Since several dependent merges are performed in the course of k-minimisation, the number of errors between the original DFA M and the resulting k-minimal DFA is not necessarily the sum of the errors introduced for each merging step. This is due to the fact that errors made in one merge may be cancelled out in a subsequent merge. It is natural to ask, whether it is nevertheless possible to *efficiently* construct an *optimal k-minimal* DFA for M (i.e., a k-minimal DFA with the least number of errors introduced). In the following we show that the construction of an optimal k-minimal DFA for M is NP-hard.

The intractability is shown by a reduction from the 3-colouring problem for a graph $G = \langle V, E\rangle$ in a similar, though much more refined, way as in Sect. 4. We again construct a DFA M with one state v for every vertex $v \in V$ and one letter e for each edge $e \in E$. We introduce three additional states $\{1_0, 2_0, 3_0\}$ (besides others) to represent the 3 colours. For the following discussion, let $N = \langle P, \Sigma, \mu, p_0, F'\rangle$ be a k-minimal DFA for M. Let us fix an edge $e = \{v_1, v_2\} \in E$. The DFA M is constructed such that the languages $L_M(\delta(v_1, e))$ and $L_M(\delta(v_2, e))$ have a large but finite symmetric difference; as in the previous section, if a proper 3-colouring $c\colon V \to [3]$ exists the DFA N can be obtained by merging each state v into $c(v)_0$. In addition, for every edge $e = \{v_1, v_2\} \in E$ and vertex $v \in e$, we let $\mu(c(v)_0, e) = \delta(v, e)$. On the other hand, if G admits no proper 3-colouring, then the DFA N is still obtained by state merges performed on M. However, because G has no proper 3-colouring, in the constructed DFA M there exist 2 states v_1, v_2 such that $e = \{v_1, v_2\} \in E$ and that both v_1 and v_2 are merged into the same state $p \in P$. Then the transition $\mu(p, e)$ cannot match both $\delta(v_1, e)$ and $\delta(v_2, e)$. In order to make such an error costly, the left languages of v and v' are designed to be large, but finite. In contrast, we can easily change the transitions of states $\{1_0, 2_0, 3_0\}$ by letters e because the left-languages of the states $\{1_0, 2_0, 3_0\}$ are small.

To keep the presentation simple, we will use two gadgets. The first one will enable us to make sure that two states cannot be merged: k-similar states are also hyper-equivalent, so we can simply avoid undesired merges by making states hyper-inequivalent. Another gadget will be used to increase the in-level of certain states to a desired value.

Lemma 13. *For every congruence $\simeq \subseteq Q \times Q$ on M, there exists a DFA N such that (i) $p_1 \not\sim p_2$ for every $p_1 \in P \setminus Q$ and $p_2 \in P$ with $p_1 \neq p_2$, and (ii) $q_1 \not\sim q_2$ in N for all $q_1 \not\simeq q_2$.*

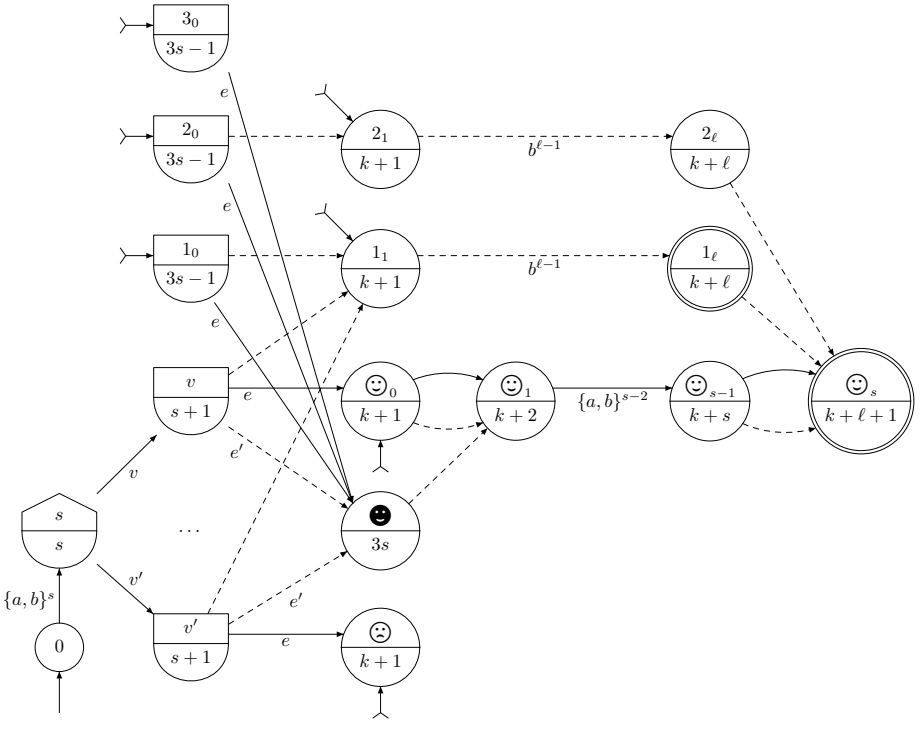


Fig. 5. Illustration of the DFA M of Section 5

In graphical illustrations, we use different shapes for q_1 and q_2 to indicate that $q_1 \not\sim q_2$, because of the gadget of Lemma 13. Note that states with the same shape need not be k-similar.

Lemma 14. *For every subset $S \subseteq Q \setminus \{q_0\}$ of states and map* min-level: $S \to \mathbb{N}$, *there exists a DFA $N = \langle Q \cup I, \Sigma \cup \Delta, \mu, q_0, F\rangle$ such that $|\mu^{-1}(i)| = 1$ for every $i \in I$ and* $\text{in-level}_N(s) = \max(\text{in-level}_M(s), \text{min-level}(s))$ *for every $s \in S$.*

We will indicate the level i below the state name in graphical illustrations. Moreover, we add a special feathered arrow to the state q, whenever the gadget is used for the state q to increase its level.

We now present the construction. Let $G = \langle V, E\rangle$ be an undirected graph. Select $k, s \in \mathbb{N}$ such that $s > \log(|V|) + 2$ and $k > 4s$. Moreover, let $\ell = k - 2s$.

Definition 15. *We construct the DFA $M = \langle Q, \Sigma, \delta, 0, F\rangle$ as follows:*

- $Q = \{\bot, \bullet, \ominus, 3_0\} \cup \{i_j \mid i \in [2], j \in [\ell]\} \cup V \cup [0, s] \cup \{\odot_i \mid 0 \le i \le s\}$,
- $\Sigma = \{a, b\} \cup V \cup E$,
- $F = \{\odot_s, 1_\ell\}$, *and*
- *for every $v \in V$, $e = \{v_1, v_2\} \in E$ with $v \notin e$ and $v_1 < v_2$, $i \in [s]$, and $j \in [\ell]$*

$$\delta(i-1, a) = i \qquad \delta(v_1, e) = \odot_0 \qquad \delta(1_0, e) = \bullet \qquad \delta(1_{j-1}, b) = 1_j$$

$$\begin{array}{llll} \delta(i-1,b)=i & \delta(v_2,e)=\text{☹} & \delta(2_0,e)=\text{☻} & \delta(2_{j-1},b)=2_j \\ \delta(\text{☹}_{i-1},a)=\text{☹}_i & \delta(v,e)=\text{☻} & \delta(3_0,e)=\text{☻} & \delta(1_\ell,b)=\text{☹}_s \\ \delta(\text{☹}_{i-1},b)=\text{☹}_i & \delta(v,a)=1_1 & & \delta(2_\ell,b)=\text{☹}_s \\ \delta(\text{☻},a)=\text{☹}_1 & \delta(s,v)=v & & \end{array}$$

– *For all remaining cases, we set* $\delta(q,\sigma)=\bot$.

Finally, we show how to collapse k-similar states using a proper 3-colouring $c\colon V \to [3]$. We obtain the k-similar DFA $c(M)=\langle P,\Sigma,\mu,0,F\rangle$ from M by merging each state v into $c(v)_0$. In addition, for every edge $e=\{v_1,v_2\}\in E$, we let $\mu(c(v_1)_0,e)=\delta(v_1,e)$ and $\mu(c(v_2)_0,e)=\delta(v_2,e)$. Since the colouring c is proper, we have that $c(v_1)\neq c(v_2)$, which yields that μ is well-defined. For the remaining $i\in[3]\setminus\{c(v_1),c(v_2)\}$, we let $\mu(i_0,e)=\text{☹}_0$. All equivalent states (i.e., $\bot$ and ☹) are merged. The gadgets that were added to M survive and are added to $c(M)$. Naturally, if a certain state does no longer exist, then all transitions leading to or originating from it are deleted too. This applies for example to ☻.

Lemma 16. *There exists a k-minimal DFA N for M with at most*

$$2^{2s-1}\cdot|E|\cdot(|V|-2)+3\cdot 2^{s-1}\cdot|E|+2^{s+1}\cdot|V|$$

errors if and only if the input graph G is 3-colourable.

Corollary 17. *'Error-bounded k-minimisation' is NP-complete.*

References

1. Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. J. ACM 54(3) (2007)
2. Badr, A., Geffert, V., Shipman, I.: Hyper-minimizing minimized deterministic finite state automata. RAIRO Theoret. Inform. Appl. 43(1), 69–94 (2009)
3. Castiglione, G., Restivo, A., Sciortino, M.: Hopcroft's algorithm and cyclic automata. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 172–183. Springer, Heidelberg (2008)
4. Gawrychowski, P., Jeż, A.: Hyper-minimisation made efficient. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 356–368. Springer, Heidelberg (2009)
5. Gries, D.: Describing an algorithm by Hopcroft. Acta Inf. 2(2), 97–109 (1973)
6. Holzer, M., Maletti, A.: An $n\log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. Theoret. Comput. Sci. 411(38–39), 3404–3413 (2010)
7. Hopcroft, J.E.: An $n\log n$ algorithm for minimizing states in a finite automaton. In: Kohavi, Z. (ed.) Theory of Machines and Computations, pp. 189–196. Academic Press, London (1971)
8. Maletti, A.: Better hyper-minimization— not as fast, but fewer errors. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 201–210. Springer, Heidelberg (2011)
9. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms 51(2), 122–144 (2004)

Contracting a Chordal Graph to a Split Graph or a Tree*

Petr A. Golovach[1], Marcin Kamiński[2], and Daniël Paulusma[1]

[1] School of Engineering and Computing Sciences, Durham University, Science Laboratories, South Road, Durham DH1 3LE, United Kingdom
{petr.golovach,daniel.paulusma}@durham.ac.uk
[2] Département d'Informatique, Université Libre de Bruxelles, Belgium
marcin.kaminski@ulb.ac.be

Abstract. The problems Contractibility and Induced Minor are to test whether a graph G contains a graph H as a contraction or as an induced minor, respectively. We show that these two problems can be solved in $|V_G|^{f(|V_H|)}$ time if G is a chordal input graph and H is a split graph or a tree. In contrast, we show that containment relations extending Subgraph Isomorphism can be solved in linear time if G is a chordal input graph and H is an arbitrary graph not part of the input.

1 Introduction

There are several natural and elementary algorithmic problems to test whether the structure of some graph H shows up as a *pattern* within the structure of another graph G. We focus on one such problem in particular, namely whether one graph contains some other graph as a contraction. Before we give a survey of existing work and present our results, we first state some basic terminology.

We consider undirected finite graphs that have no loops and no multiple edges. We denote the vertex set and edge set of a graph G by V_G and E_G, respectively. If no confusion is possible, we may omit the subscripts. We refer to Diestel [7] for any undefined graph terminology. Let $e = uv$ be an edge in a graph G. The *edge contraction* of e removes u and v from G, and replaces them by a new vertex adjacent to precisely those vertices to which u or v were adjacent. A graph H is a *minor* of a graph G if H can be obtained from G by a sequence of vertex deletions, edge deletions, and edge contractions. If only vertex deletions and edge contractions are allowed, then H is an *induced minor* of G. If only edge contractions are allowed, then H is a *contraction* of G. If only vertex deletions and edge deletions are allowed, then H is a *subgraph* of G. The corresponding decision problems are called Minor, Induced Minor, Contractibility, and Subgraph Isomorphism, respectively.

Matoušek and Thomas [21] showed that Contractibility, Induced Minor and Minor are NP-complete even on ordered input pairs (G, H) where G and H are trees of bounded diameter, or G and H are trees with at most one

* This work is supported by EPSRC (EP/G043434/1).

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 339–350, 2011.

vertex of degree more than 5. If H is a cycle on $|V_G|$ vertices, then Subgraph Isomorphism is equivalent to asking whether G is Hamiltonian. This is an NP-complete problem even when G is restricted to be chordal bipartite, as shown by Müller [22]. It is therefore natural to fix the graph H in an ordered input pair (G, H) and consider only the graph G to be part of the input. We indicate this by adding "H-" to the names of the decision problems.

A celebrated result by Robertson and Seymour [24] states that H-Minor can be solved in cubic time. It is straightforward that H-Subgraph can be solved in polynomial time for any fixed graph H. The computational complexity classifications of the problems H-Induced Minor and H-Contractibility are still open, although many partial results are known. Fellows, Kratochvíl, Middendorf, and Pfeiffer [9] gave both polynomial-time solvable and NP-complete cases for the H-Induced Minor problem. The smallest known NP-complete case is a graph H on 68 vertices [9]. A number of polynomial-time solvable and NP-complete cases for the H-Contractibility problem can be found in a series of papers started by Brouwer and Veldman [5] and followed by Levin, Paulusma and Woeginger [19,20] and van 't Hof et al. [17]. The smallest NP-complete cases are when H is a 4-vertex path or a 4-vertex cycle [5]. Because some of the open cases for both problems are notoriously difficult, special graph classes have been studied in the literature. Fellows, Kratochvíl, Middendorf, and Pfeiffer [9] showed that for every fixed graph H, the H-Induced Minor problem can be solved in polynomial time on planar graphs. Also the H-Contractibility problem can be solved in polynomial time for every fixed H on this graph class [18].

A graph G is a *split graph* if G has a *split partition*, which is a partition of its vertex set into a clique C_G and an independent set I_G. Split graphs were introduced by Foldes and Hammer [10] in 1977 and have been extensively studied since then; see e.g. the monographs of Brandstädt, Le and Spinrad [4], or Golumbic [14]. Belmonte, Heggernes, and van 't Hof [1] showed that Contractibility is NP-complete for ordered input pairs (G, H) where G is a split graph and H is a split graph of a special type, namely a threshold graph. They also showed that for every fixed graph H, the H-Contractibility problem can be solved in polynomial time for split graphs. As a matter of fact, H may be assumed to be a split graph in this result, because split graphs are closed under taking contractions. Golovach et al. [13] showed that Minor and Induced Minor are NP-complete for ordered input pairs (G, H) where G and H are split graphs. They also showed that Contractibility and Induced Minor are W[1]-hard for such input pairs (G, H) when parameterized by $|V_H|$. Hence, it is unlikely that these two problems can be solved in $|V_G|^{O(1)}$ time for such input pairs (G, H) with the constant in the exponent independent of H. The same authors [13] showed that H-Induced Minor is polynomial-time solvable on split graphs for any fixed graph H. Because split graphs are closed under taking induced minors, also in this result H may be assumed to be a split graph.

A graph is called *chordal* (or *triangulated*) if it contains no induced cycle on at least four vertices; note that every split graph is chordal. Heggernes, van 't Hof, Lévêque, and Paul [16] showed that H-Contractibility can be solved

in polynomial time for chordal graphs when H is an a fixed path by checking whether the diameter of the chordal input graph is greater than or equal to the length of the fixed path. We observe that testing whether a graph G contains a fixed path H as an induced minor is equivalent to testing whether G contains H as an induced subgraph. This means that H-Induced Minor is polynomial-time solvable for general graphs if H is a fixed path; this problem is open if H is a fixed tree.

Our Results. We extend the aforementioned results of Belmonte, Heggernes, and van 't Hof [1] and Golovach et al. [13] by showing that H-Contractibility and H-Induced Minor can be solved in polynomial time on chordal graphs for any fixed split graph H. We also show that H-Contractibility and H-Induced Minor are polynomial-time solvable for chordal graphs when H is any fixed tree. This extends the aforementioned result of Heggernes, van 't Hof, Lévěque, and Paul [16]. In contrast to the W[1]-hardness of Contractibility and Induced Minor for split graphs [13], we show that the problems Subgraph Isomorphism, Minor and the related problems Topological Minor and Immersion, which we define later, can be solved in linear time if G is a chordal graph and H is an arbitrary fixed graph not part of the input.

2 Preliminaries

Let $G = (V, E)$ be a graph. A subset $U \subseteq V$ is a *clique* if there is an edge in G between any two vertices of U, and U is an *independent set* if there is no edge in G between any two vertices of U. We write $G[U]$ to denote the subgraph of G *induced* by $U \subseteq V$, i.e., the graph on vertex set U and an edge between any two vertices whenever there is an edge between them in G. Two sets $U, U' \subseteq V$ are called *adjacent* if there exist vertices $u \in U$ and $u' \in U'$ such that $uu' \in E$. A vertex v is a *neighbor* of u if $uv \in E$. The *degree* $d_G(u)$ of a vertex u is its number of neighbors. A set $U \subset V$ is a *cut-set* if $G - U$ is disconnected; if $U = \{u\}$, then u is called a *cut-vertex*. A vertex $v \in V$ is called *simplicial* if its neighbors in G form a clique.

For our proofs the following global structure is useful. Let G and H be two graphs. An *H-witness structure* $\mathcal{W}$ is a vertex partition of G into $|V_H|$ (nonempty) sets $W(x)$ called *H-witness bags*, such that

(i) each $W(x)$ induces a connected subgraph of G;
(ii) for all $x, y \in V_H$ with $x \neq y$, bags $W(x)$ and $W(y)$ are adjacent in G if and only if x and y are adjacent in H.

By contracting all bags to singletons we observe that H is a contraction of G if and only if G has an H-witness structure such that conditions (i)-(ii) hold. Because every vertex $u \in V_G$ is in at most one bag, we can define $W_u = W(x)$ if u is in $W(x)$. We also use the shorthand notation $W(X) = \bigcup_{x \in X} W(x)$ for some $X \subseteq V_H$. We note that G may have more than one H-witness structure with respect to the same containment relation. Let H be an induced subgraph of G.

We say that G has a *subgraph contraction* to H if G has an H-witness structure $\mathcal{W}$ such that $x \in W(x)$ for all $x \in V_H$. We also say that a vertex $u \in V_G \setminus V_H$ is *contracted to* $x \in V_H$ if $u \in W(x)$.

A clique tree $\mathcal{T}_G$ of a (connected) graph G is a tree that has as vertices the maximal cliques of G and has edges such that each graph induced by those cliques that contain a particular vertex of G is a subtree. We let $\mathcal{K}_G$ denote the set of all maximal cliques of G. The following three lemmas are well-known and useful; from now on we implicitly assume that we can compute a clique tree of a chordal graph in linear time whenever we need such a tree for our algorithms.

Lemma 1 ([12]). *A connected graph is chordal if and only if it has a clique tree.*

Lemma 2 ([23]). *Let $G = (V, E)$ be a chordal graph. Then $\sum_{K \in \mathcal{K}_G} |K| = O(|V| + |E|)$.*

Lemma 3 ([3,11]). *A clique tree of a connected chordal graph $G = (V, E)$ can be constructed in $O(|V| + |E|)$ time.*

Let $G = (V, E)$ be a chordal graph. We refer to a set $K \in \mathcal{K}_G$ as a *node* of $\mathcal{T}_G$. We define the notions *root node*, *parent node*, *child node* and *leaf node* of a clique tree similar to the notions root, parent, child and leaf of a "normal" tree. If the bag $K_r \in \mathcal{K}_G$ is the root node of $\mathcal{T}_G$, then we say that T is *rooted at* K_r. A *descendant* of a node K is a node K^* such that K lies on the (unique) path from K^* to the root node K_r in $\mathcal{T}_G$; note that each node is its own descendant. Every node $K \neq K_r$ of a clique tree $\mathcal{T}_G$ has exactly one parent node K' in $\mathcal{T}_G$. We say that a vertex $v \in K$ is *given to the parent node* K' if $v \in K \cap K'$, i.e., if v is both in the child node K and in the parent node K'. We say that vertex $v \in K$ *stays behind* if $v \in K \setminus K'$, i.e., if v is in the child node K but not in the parent node K'. Bernstein and Goodman [2] showed that a tree $\mathcal{T}$ with vertex set $\mathcal{K}_G$ is a clique tree of G if and only if $\mathcal{T}$ is a maximum weight spanning tree of the *clique graph* $C(G)$ of G; this is the weighted graph that has as vertices the maximal cliques of G and that has an edge K_1K_2 with weight $|K_1 \cap K_2|$ whenever $K_1 \cap K_2 \neq \emptyset$. This leads to the following observation that we will implicitly use in the proofs of our results.

Observation 1. *Let G be a connected chordal graph with at least two maximal cliques. Let $\mathcal{T}_G$ be a clique tree of G rooted at K_r. At least one vertex of any node $K \neq K_r$ of T is given to the parent node of K and at least one vertex stays behind. Moreover, $|K| \geq 2$ for all $K \in \mathcal{K}_G$.*

3 Contracting to Split Graphs

Throughout this section, we assume that G denotes a chordal graph with set of maximal cliques $\mathcal{K}_G$ and that H denotes a split graph with a split partition (C_H, I_H), where $C_H = \{x_1, \ldots, x_p\}$ and $I_H = \{y_1, \ldots, y_q\}$. If G has an H-witness structure, then we call the bags corresponding to the vertices in C_H and I_H *clique bags* and *independent bags*, respectively.

If $q = 0$, then G has H as a contraction if and only if G has H as a minor, and we can use Robertson and Seymour's theorem [24] to test this in polynomial time. Hence, we assume that $p \geq 1$ and $q \geq 1$. Because H is connected, we may assume without loss of generality that G is connected. We then do as follows. We first show in Lemma 4 that G has an H-witness structure where every maximal clique is distributed over the bags in a very restricted way, should G contain H as a contraction. This enables us to branch in a specific way leading to a new set of connected chordal graphs. This is Phase 1 of the algorithm. In Phase 2 we process each of the obtained graphs by applying a dynamic programming procedure. We start with the following observation, which follows from the definition of an H-witness structure.

Observation 2. *If G contains H as a contraction, then no clique of G has two vertices belonging to two non-adjacent bags of some H-witness structure of G.*

Lemma 4. *If G contains H as a contraction, then G has an H-witness structure $\mathcal{W}$ such that for every maximal clique K of G exactly one of the following statements hold:*

(i) $K \subseteq W(C_H)$, *or*
(ii) $K \subseteq W(y_j)$ *for some* $y_j \in I_H$, *or*
(iii) there is a vertex $u \in K$ *with* $K \setminus \{u\} \subseteq W(C_H)$ *and* $u \in W(y_j)$ *for some* $y_j \in I_H$.

Moreover, for every $y_j \in I_H$, there exists at least one maximal clique K of G that contains a vertex u with $u \in W(y_j)$ and $K \setminus \{u\} \subseteq W(C_H)$.

Proof. Suppose that G contains H as a contraction. Let $\mathcal{W}$ be an H-witness structure of G such that $W(C_H)$ is maximal. Let K be a maximal clique of G. Suppose that (i) and (ii) do not hold. By Observation 2, we then find that K contains three vertices u_1, u_2 and v, such that u_1 and u_2 belong to some independent bag $W(y_j)$ and v belongs to $W(C_H)$.

Let U_1 be the component of $G[W(y_j) \setminus \{u_2\}]$ that contains u_1, and let U_2 be the component of $G[W(y_j) \setminus \{u_1\}]$ that contains u_2. If U_2 is adjacent to every clique bag that is adjacent to U_1, then we move U_1 from $W(y_j)$ to the clique bag that contains v. We do the same with U_2 in the case that U_1 is adjacent to every clique bag that is adjacent to U_2. In both cases, the maximality of $W(C_H)$ is violated. Hence, there exists a clique bag $W(x_h)$ that is adjacent to U_1 but not to U_2, and a clique bag $W(x_i)$ that is adjacent to U_2 but not to U_1. We let s be a vertex in U_1 that has a neighbor t in $W(x_h)$ and s' be a vertex in U_2 that has a neighbor t' in $W(x_i)$. Here, we choose s as close as possible to u_1 and s' as close as possible to u_2; note that s and s' might be equal to u_1 and u_2, respectively. Let P be a shortest path from s to s' in $G[W(y_j)]$. Note that P passes through u_1 and u_2. By definition, s is the only vertex on P that has a neighbor, namely t, in $W(x_h)$, and s' is the only vertex on P that has a neighbor, namely t', in $W(x_i)$. In $G[W(x_h) \cup W(x_i)]$, we choose a shortest path Q from t to t'. Then the paths P and Q together with the two edges st and $s't'$ form an induced cycle in G on at least four vertices. This is not possible, because G is chordal.

To prove the second statement of Lemma 4, consider a vertex $y_j \in I_H$. Let $x_i \in C_H$ be a neighbor of y_j. Then G must contain an edge tu with $t \in W(x_i)$ and $u \in W(y_j)$. Let K be a maximal clique of G that contains t and u. Because $\mathcal{W}$ satisfies the first statement of Lemma 4, we find that $K \setminus \{u\} \subseteq W(C_H)$, as desired. Hence, we have proven Lemma 4. □

We start our algorithm. In Phase 1 we choose a pair (K_j, u_j) for each $y_j \in I_H$. This gives us an ordered set S of q different pairs. We determine all possible choices of sets S. If G contains H as a contraction, then G has an H-witness structure $\mathcal{W}$ that satisfies Lemma 4. Then one of our chosen sets S will correspond to a set of q pairs satisfying the second statement of Lemma 4. To determine this, we consider every set S and each time we may modify G into a new graph G'. If in the end we have not discarded a set S then we put the corresponding graph G' in $\mathcal{G}$. This leads to Lemma 5; we omit its proof.

Lemma 5. *We can obtain in polynomial time a set $\mathcal{G}$ of chordal graphs, such that G contains H as a contraction if and only if there exists a graph $G' \in \mathcal{G}$ that has an H-witness structure $\mathcal{W}'$ with $W'(y_j) = \{u_j(G')\}$ for $j = 1, \ldots, q$, where $u_1(G'), \ldots, u_q(G')$ are q specified mutually non-adjacent vertices of G' that are not cut-vertices of G'.*

Suppose that we have obtained a set $\mathcal{G}$ that satisfies Lemma 5. Because $\mathcal{G}$ has polynomial size, we process each graph in it one by one. This is Phase 2 of our algorithm. For simplicity, we will denote such a graph by G again and its set of q specified vertices by $u_1, \ldots, u_q$. Let G^* be the graph obtained from G after removing $u_1, \ldots, u_q$. We make the following observation, which follows from the property of G that the intersection of two intersecting maximal cliques contain a vertex from $V_G \setminus \{u_1, \ldots, u_q\}$ as the vertices $u_1, \ldots, u_q$ are no cut-vertices and form an independent set in G.

Observation 3. *The graph G^* is a connected chordal graph.*

What is left to decide is if and how the vertices of G^* can be distributed over the clique bags of a witness structure $\mathcal{W}$ of G with $W(y_j) = \{u_j\}$ for $j = 1, \ldots, q$. In order to do this we follow a dynamic programming approach over a clique tree of G. For this purpose we must first decide how to root this tree, and in order to do that we need the following lemma.

Lemma 6. *Let C be a cut-set of G^*. If G has an H-witness structure $\mathcal{W}$ with $W(y_j) = \{u_j\}$ for $j = 1, \ldots, q$, then for all pairs of vertices $v_1, v_2 \in V_{G^*}$ that are in two different components of $G^* - C$, there exists a vertex $t \in C$ such that $W_{v_1} = W_t$ or $W_{v_2} = W_t$.*

Proof. Suppose that there exist two vertices $v_1, v_2 \in V_{G^*}$ that are in two different components of $G^* - C$ such that there is no vertex $t \in C$ with $W_{v_1} = W_t$ or $W_{v_2} = W_t$. Let $W_{v_1} = W(x_h)$ and $W_{v_2} = W(x_i)$. Then either $G[W(x_h)]$ is disconnected if $h = i$, or $W(x_h)$ and $W(x_i)$ are not adjacent if $h \neq i$. Both cases are not possible. □

Lemma 6 helps us to deduce a useful property for our dynamic programming; we describe this property in Lemma 7.

Lemma 7. *If G has an H-witness structure $\mathcal{W}$ with $W(y_j) = \{u_j\}$ for $j = 1, \ldots, q$, then there exists a node K_r^* such that $\mathcal{T}_{G^*}$ can be rooted at K_r^* with the following property valid for every parent node K_p^* with child node K_c^*: for all $v \in K_c^* \setminus K_p^*$ there exists a vertex $t \in K_c^* \cap K_p^*$ with $W_t = W_v$.*

Proof. Let K be a node of $\mathcal{T}_{G^*}$. Suppose that the property in the statement of Lemma 7 does not hold if K is to be the root. Then there is a node K_1 that has a child node K_2 such that $K_2 \setminus K_1$ has a vertex v for which there does not exist a vertex $t \in K_1 \cap K_2$ with $W_t = W_v$. We apply Lemma 6 for $C = K_1 \cap K_2$ and find that for every vertex v' in the component of $G^* - C$ that contains $K_2 \setminus K_1$ there exists a vertex $t' \in K_1 \cap K_2$ with $W_{t'} = W_{v'}$. We now choose as new root the node K_2. If K_2 satisfies the property in the statement of Lemma 7, then we are done. Otherwise, there exists a node K_3 with a child node K_4 as before. However, K_4 cannot be in the component of $G^* - C$ that contains the vertices of $K_1 \setminus K_2$. Hence, as G^* is finite, repeatedly applying the same argument, eventually yields a root node satisfying the statement of Lemma 7. □

From now we assume that $\mathcal{T}_{G^*}$ is rooted in a node K_r^* in such a way that the property described in Lemma 7 holds if G contains H as a contraction. We may do this, because we will consider if necessary all nodes of $\mathcal{T}_{G^*}$ to be the root node, and consequently repeat the algorithm a number of times. This number is polynomially bounded, because the total number of different nodes in $\mathcal{T}_{G^*}$ is polynomially bounded due to Lemma 2. Our next aim is to delete simplicial vertices of G that are in $V_{G^*} \setminus K_r^*$.

Lemma 8. *Let $v \in V_{G^*} \setminus K_r^*$ be a simplicial vertex of G. If G has an H-witness structure for G with $W(y_j) = \{u_j\}$ for $j = 1, \ldots, q$ where $v \in W(x_i)$ is adjacent to some vertex of $W(x_j)$ for some $x_i, x_j \in C_H$, then G has an H-witness structure $\mathcal{W}'$ with*

(i) $W'(z) = W(z)$ *for* $z \in V_H \setminus \{x_i, x_j\}$, *and*
(ii) $W'(x_i) = W(x_i) \setminus \{v\}$, *and*
(iii) $W'(x_j) = W(x_j) \cup \{v\}$.

Proof. Because $v \notin K_r^*$, we find that $W'(x_i) \neq \emptyset$ due to Lemma 7. Because v is simplicial, $W'(x_i)$ is connected. Because v is adjacent to a vertex from $W(x_j)$, $W'(x_j)$ is connected. Because v is a simplicial vertex of G, any neighbor of v in $\{u_1, \ldots, u_q\}$ is adjacent to a vertex of $W'(x_i)$ and to a vertex of $W(x_j)$. □

Lemma 8 implies that simplicial vertices in $V_{G^*} \setminus K_r^*$ can be included in any adjacent bag. This means that we can exclude them from the graph recursively, and from now we assume that G has no simplicial vertices in $V_{G^*} \setminus K_r^*$.

We now apply dynamic programming over $\mathcal{T}_{G^*}$. Our algorithm returns `Yes` if G has an H-witness structure $\mathcal{W}$ with $W(y_j) = \{u_j\}$ for $j = 1, \ldots, q$ that satisfies the property given in Lemma 7. It returns `No` otherwise. Due to space restriction we exclude the correctness proof and a running time analysis of our algorithm. This brings us to the main result of this section.

Theorem 1. *For any fixed split graph H, the H-*Contractibility *problem can be solved in polynomial time for chordal graphs.*

4 Contracting to Trees

Throughout this section, we assume that G denotes a chordal graph with set of maximal cliques $\mathcal{K}_G$ and that H denotes a tree with leaves $\{z_1,\ldots,z_r\}$. We also let $\mathcal{T}_G$ denote a clique tree of G. Due to Theorem 1, we may assume that H is not a split graph. Hence, H has at least four vertices. We will present an algorithm that decides in $|V(G)|^{|O(|V_H|)}$ time whether G contains H as a contraction.

If G has an H-witness structure $\mathcal{W}$ then we call the bags $W(z_i)$ the *leaf bags* of $\mathcal{W}$. We define *parent bag* and *child bag* of $\mathcal{W}$ analogously. We start with the following lemma.

Lemma 9. *If G contains H as a contraction, then G has an H-witness structure $\mathcal{W}$ with $W(z_i)=\{u_i\}$, $i=1,\ldots,r$ for some set of vertices $u_1,\ldots,u_r$.*

Proof. Suppose that G contains H as a contraction. Let $\mathcal{W}$ be an H-witness structure of G. Suppose that $|W(z_i)|\geq 2$ for some $1\leq i\leq r$. Let y_i be the parent of z_i in H. We choose a vertex $u_i\in W(z_i)$ that is not a cut-vertex of $G[W(z_i)]$ and move all vertices in $W(z_i)\setminus\{u_i\}$ to $W(y_i)$. This leads to a new H-witness structure of G, in which the leaf bag corresponding to z_i only contains u_i. If necessary, we apply this operation on every other leaf bag that contains more than one vertex. □

We call the vertices $u_1,\ldots,u_r$ in Lemma 9 the *leaf bag vertices* of the witness structure $\mathcal{W}$ and call $\mathcal{W}$ a *simple* witness structure. We can now show the following lemma, the proof of which has been omitted.

Lemma 10. *We can obtain in polynomial time a set $\mathcal{G}$ of graphs, such that G contains H as a contraction if and only if there exists a graph $G'\in\mathcal{G}$ that has a simple H-witness structure $\mathcal{W}'$ with as leaf bag vertices r specified vertices $u_1(G'),\ldots,u_r(G')$ that are of degree one in G' and that together form the set of all simplicial vertices of G'.*

From now on suppose that we have obtained a set $\mathcal{G}$ that satisfies Lemma 10. Because $\mathcal{G}$ has polynomial size, we can process each graph in it. Therefore, we may without loss of generality assume that we have a connected chordal input graph with a set of r specified vertices $u_1,\ldots,u_r$ that are of degree one and that together form the set of all simplicial vertices of the graph. For simplicity, we will denote this chordal input graph by G again.

If G allows an H-witness structure $\mathcal{W}$, then we call a maximal clique K of G a *vertex clique* of $\mathcal{W}$ if K contains vertices from exactly one witness bag from $\mathcal{W}$, and we call K an *edge clique* of $\mathcal{W}$ if K contains vertices from exactly two (adjacent) witness bags from $\mathcal{W}$. For a vertex $x\in V_H$, we call a vertex-clique

an x-vertex-clique if it is a subset of $W(x)$. For an edge $xy \in E_H$, we call an edge-clique an xy-edge-clique if it contains one or more vertices of $W(x)$ and one or more vertices of $W(y)$.

We make two useful observations. The first observation holds, because H is a tree. The second observation follows from Observation 4 and the property that every bag of a witness structure is connected.

Observation 4. *If G contains H as a contraction, then every maximal clique of G is either a vertex clique or an edge clique in every H-witness structure of G.*

Observation 5. *If G contains H as a contraction, then for all $xy \in E_H$ the xy-edge cliques in every H-witness structure of G are the nodes of a connected subtree of $\mathcal{T}_G$.*

We are now ready to show our last lemma. Recall that G contains exactly r vertices $u_1, \ldots, u_r$ of degree 1 and that G contains no other simplicial vertices.

Lemma 11. *There exists a polynomial-time algorithm that tests whether G has an H-witness structure $\mathcal{W}$ such that $W(y_i) = \{u_i\}$ for $i = 1, \ldots, r$.*

Proof. By Observation 4, all maximal cliques of G are either vertex-cliques or edge-cliques for any H-witness structure (if it exists). By Observation 11, the xy-edge cliques are in that case the nodes of a connected subtree $\mathcal{T}_{xy}$ of $\mathcal{T}_G$. We do as follows. For every $xy \in E_H$, we guess the nodes of $\mathcal{T}_{xy}$. This leads to a collection of $|E_H|$ subtrees that are to correspond to the subtrees $\mathcal{T}_{xy}$ of $\mathcal{T}_G$. Afterwards, we must perform a number of checks to see if such a family of subtrees is a "good" guess, i.e., leads to an H-witness structure $\mathcal{W}$ with the required properties.

Firstly, we must check if there is no node of $\mathcal{T}_G$ that is contained in two different subtrees. If so, then we discard our guessed family of subtrees. Otherwise we continue as follows. Because we have chosen nodes to correspond to all edge-cliques, all other nodes of $\mathcal{T}_G$ must correspond to the vertex-cliques. We may therefore contract these nodes to single vertices in G. Because single vertices are not maximal cliques, this leads to a new graph G^* with clique tree $\mathcal{T}_{G^*}$, the nodes of which are to correspond to the edge-cliques.

If there exist two maximal cliques K and K' of G^* with $K \cap K' \neq \emptyset$ that are nodes of two subtrees that are to correspond to subtrees $\mathcal{T}_{xy}$ and $\mathcal{T}_{x'y'}$ with $\{x, y\} \cap \{x', y'\} = \emptyset$, then we discard our guessed family of subtrees; every vertex in $K \cap K'$ must go to $W(x) \cup W(y)$ and $W(x') \cup W(y')$ simultaneously and this is not possible. Suppose that such a pair of maximal cliques does not exist. Then for every intersecting pair of maximal cliques from two different guessed subtrees corresponding to subtrees T_{xy} and $T_{x'y'}$ we have without loss of generality $x = x'$. This means that the vertices in this intersection must go into $W(x)$. Because of this, we contract the edges in the intersection of two such cliques into one vertex which we put in a new set A_x. In the end we must check for every $xy \in E_H$, if the vertices of subtree T'_{xy} can be partitioned into two sets B_1 and B_2

such that (i) $A_x \subseteq B_1$, (ii) $A_y \subseteq B_2$ and (iii) $G[B_1]$ and $G[B_2]$ are both connected. If so, then we have our desired H-witness structure of G. Otherwise we must consider some other family of guessed subtrees.

This algorithm runs in polynomial time for the following reasons. For every $xy \in E_H$, we actually only have to guess the nodes in $\mathcal{T}_G$ that correspond to the leaves of $\mathcal{T}_{xy}$. Because every leaf node of $\mathcal{T}_G$ has a simplicial vertex, $\mathcal{T}_G$ contains exactly r leaf nodes. This means that the number of nodes we must guess for $\mathcal{T}_G$ is at most r, which is a constant because we assume that r is fixed. We can process each set of guessed subtrees in polynomial time as well. In particular, we can check if A_x and A_y can be made connected in a subtree $\mathcal{T}_G$ by applying Robertson and Seymour's algorithm [24] for this problem, which is called the 2-Disjoint Connected Subgraphs problem. Their algorithm runs in polynomial time as long as the total number of specified vertices is bounded. This is the case in our setting, because the number of leaves of every guessed subtree is bounded and the intersection with other subtrees $\mathcal{T}_{x'y'}$ must always contain vertices from the leaves or the root of the guessed subtrees. This completes the proof of Lemma 11. □

By Lemmas 10 and 11 we obtain the main result of this section.

Theorem 2. *For any fixed tree H, the H-Contractibility problem can be solved in polynomial time for chordal graphs.*

5 Induced Minors and Extensions of Subgraphs

We let $P_1 \bowtie G$ denote the graph obtained from a graph G after adding a new vertex and making it adjacent to all vertices of G.

Lemma 12 ([17]). *Let H and G be two graphs. Then G has H as an induced minor if and only if $P_1 \bowtie G$ is $(P_1 \bowtie H)$-contractible.*

We observe that $P_1 \bowtie G$ is a chordal graph if G is chordal. Similarly, $P_1 \bowtie G$ is a split graph if G is a split graph. Then, combining Lemma 12 with Theorem 1 yields the following result.

Theorem 3. *For any fixed split graph H, the H-Induced Minor problem can be solved in polynomial time for chordal graphs.*

We also have the following result, which can be proven by using the same arguments as the proof of Theorem 2 after making a minor modification: if there exist two maximal cliques K and K' of the graph G^* in the proof of Lemma 11 with $K \cap K' \neq \emptyset$ that are nodes of two subtrees that are to correspond to subtrees $\mathcal{T}_{xy}$ and $\mathcal{T}_{x'y'}$ with $\{x, y\} \cap \{x', y'\} = \emptyset$, then we remove these vertices instead of discarding the family of guessed subtrees.

Theorem 4. *For any fixed tree H, the H-Induced Minor problem can be solved in polynomial time for chordal graphs.*

We conclude the paper with an observation that containment relations extending Subgraph Isomorphism can be decided in linear time on a chordal input graph G if the graph H is not considered to be part of the input. The H-Subgraph Isomorphism problem most likely cannot be solved in $|V_G|^{O(1)}$ time for a general graph G when the constant in the exponent is independent of H, as shown by Downey and Fellows [8]. Besides minors, we consider the following of such containment relations. A graph G contains H as a *topological minor* if H can be obtained from the input graph by deleting vertices and edges and contracting edges incident with a vertex of degree 2. Given a (not necessarily induced) path abc on three vertices, a *lift* is to remove the edges ab and bc and add the edge ac (if is not already present in the graph). A graph G contains H as a *immersion* if H can be obtained from G by vertex and edge deletions and lifts. The corresponding decision problems are called Topological Minor and Immersion. For any fixed graph H, both H-Topological Minor and H-Immersion have recently been shown to be solvable in $O(|V(G)|^3)$ time by Grohe, Kawarabayashi, Marx and Wollan [15].

Theorem 5. *For any fixed graph H, the four problems H-Subgraph Isomorphism, H-Minor, H-Topological Minor, and H-Immersion can be solved in $O(|V| + |E|)$ time for chordal graphs.*

Proof. Let G be the input chordal graph. We construct a clique tree $\mathcal{T}_G$ of G in linear time due to Lemma 3. In the same time, we can find a largest maximal clique in $\mathcal{T}_G$. We denote its size by c. If $c \geq |V_H|$, then any largest clique contains H as a subgraph and thus as a minor, topological minor, and immersion. Hence, G contains H as a subgraph, minor, topological minor, and immersion. Otherwise, the size of all cliques in $\mathcal{T}_G$ is at most c and $\mathcal{T}_G$ is a tree-decomposition of G of width $c-1$. All four relations H-Subgraph Isomorphism, H-Minor, H-Topological Minor, and H-Immersion can be decided in $O(|V| + |E|)$ time for graphs of bounded treewidth [6,15].

References

1. Belmonte, R., Heggernes, P., van 't Hof, P.: Edge contractions in subclasses of chordal graphs. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 528–539. Springer, Heidelberg (2011)
2. Bernstein, P.A., Goodman, N.: Power of natural semijoins. SIAM Journal on Computing 10, 751–771 (1981)
3. Blair, J.R.S., Peyton, B.: An introduction to chordal graphs and clique trees. Graph Theory and Sparse Matrix Computation 56, 1–29 (1993)
4. Brandstädt, A., Le, V.B., Spinrad, J.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications (1999)
5. Brouwer, A.E., Veldman, H.J.: Contractibility and NP-completeness. Journal of Graph Theory 11, 71–79 (1987)
6. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
7. Diestel, R.: Graph Theory, Electronic edn. Springer, Heidelberg (2005)

8. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for W[1]. Theoretical Computer Science 141, 109–131 (1995)
9. Fellows, M.R., Kratochvíl, J., Middendorf, M., Pfeiffer, F.: The Complexity of Induced Minors and Related Problems. Algorithmica 13, 266–282 (1995)
10. Foldes, S., Hammer, P.L.: Split graphs. In: Proceedings of the Eighth Southeastern Conference on Combinatorics, Graph Theory and Computing (Louisiana State Univ., Baton Rouge, La., 1977), Congressus Numerantium, vol. (XIX), Utilitas Math., Winnipeg, Man, pp. 311–315 (1977)
11. Galinier, P., Habib, M., Paul, C.: Chordal Graphs and Their Clique Graphs. In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 358–371. Springer, Heidelberg (1995)
12. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. Journal of Combinatorial Theory, Series B 16, 47–56 (1974)
13. Golovach, P.A., Kaminski, M., Paulusma, D., Thilikos, D.M.: Containment relations in split graphs (manuscript)
14. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics 57 (2004)
15. Grohe, M., Kawarabayashi, K., Marx, D., Wollan, P.: Finding topological subgraphs is fixed-parameter tractable. To appear in Symposium on Theory of Computing, STOC 2011 (2011)
16. Heggernes, P., van 't Hof, P., Lévêque, B., Paul, C.: Contracting chordal graphs and bipartite graphs to paths and trees. In: Proceedings of LAGOS 2011. Electronic Notes in Discrete Mathematics, Electronic Notes in Discrete Mathematics (to appear, 2011)
17. van 't Hof, P., Kamiński, M., Paulusma, D., Szeider, S., Thilikos, D.M.: On graph contractions and induced minors. Discrete Applied Mathematics (to appear)
18. Kamiński, M., Paulusma, D., Thilikos, D.M.: Contractions of planar graphs in polynomial time. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 122–133. Springer, Heidelberg (2010)
19. Levin, A., Paulusma, D., Woeginger, G.J.: The computational complexity of graph contractions I: polynomially solvable and NP-complete cases. Networks 51, 178–189 (2008)
20. Levin, A., Paulusma, D., Woeginger, G.J.: The computational complexity of graph contractions II: two tough polynomially solvable cases. Networks 52, 32–56 (2008)
21. Matoušek, J., Thomas, R.: On the complexity of finding iso- and other morphisms for partial k-trees. Discrete Mathematics 108, 343–364 (1992)
22. Müller, H.: Hamiltonian circuits in chordal bipartite graphs. Discrete Mathematics 156, 291–298 (1996)
23. Okamoto, Y., Takeaki, U., Uehara, R.: Counting the number of independent sets in chordal graphs. Journal of Discrete Algorithms 6, 229–242 (2008)
24. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. Journal of Combinatorial Theory, Series B 63, 65–110 (1995)

Quantum Finite Automata and Probabilistic Reversible Automata: R-trivial Idempotent Languages*

Marats Golovkins, Maksim Kravtsev, and Vasilijs Kravcevs

Faculty of Computing, University of Latvia, Raiņa bulv. 19, Riga LV-1586, Latvia
marats@latnet.lv, maksims.kravcevs@lu.lv,
kvasilijs@gmail.com

Abstract. We study the recognition of $\mathcal{R}$-trivial idempotent ($\mathcal{R}_1$) languages by various models of "decide-and-halt" quantum finite automata (QFA) and probabilistic reversible automata (DH-PRA). We introduce *bistochastic* QFA (MM-BQFA), a model which generalizes both Nayak's enhanced QFA and DH-PRA. We apply tools from algebraic automata theory and systems of linear inequalities to give a complete characterization of $\mathcal{R}_1$ languages recognized by all these models. We also find that "forbidden constructions" known so far do not include all of the languages that cannot be recognized by measure-many QFA.

1 Introduction

Measure-many quantum finite automata (MM-QFA) were defined in 1997 [11] and their language class characterization problem remains open still. The difficulties arise because the language class is not closed under Boolean operations like union and intersection [3]. The results by Brodsky and Pippenger [5] combined with the non-closure property imply that the class of languages recognized by MM-QFA is a proper subclass of the language variety corresponding to the **ER** monoid variety. The same holds for DH-PRA and for EQFA [8,14]. In [1], it is stated that MM-QFA recognize any regular language corresponding to the monoid variety **EJ**. Since any syntactic monoid of a unary regular language belongs to **EJ**, the results in [1] imply that MM-QFA recognize any unary regular language. In [4], a new proof of this result is given by explicitly constructing MM-QFA recognizing unary languages. In the paper, we consider a sub-variety of **ER**, the variety of $\mathcal{R}$-trivial idempotent monoids $\mathbf{R_1}$ and determine which $\mathcal{R}$-trivial idempotent languages ($\mathcal{R}_1$ languages) are recognizable by MM-QFA and other "decide-and-halt" models. Since $\mathbf{R_1}$ shares a lot of the characteristic properties with **ER**, the obtained results may serve as an insight to solve the general problem. The paper is structured as follows. Section 2 describes the

* Supported by the Latvian Council of Science, grant No. 09.1570 and by the European Social Fund, contract No. 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044. Unabridged version of this article is available at http://arxiv.org/abs/1106.2530.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 351–363, 2011.

algebraic tools - monoids, morphisms and varieties. Section 3 considers completely positive maps. We apply the result by Kuperberg to obtain Theorem 3.1, which is essential to prove the limitations of QFA in terms of language recognition. Sections 4 to 7 present the main results of the paper: (1) We introduce MM-BQFA, a model which generalizes the earlier "decide-and-halt" automata models (MM-QFA, DH-PRA, EQFA) and give some characteristics of the corresponding language class. We also obtain the class of languages recognized by MO-BQFA; (2) We define how to construct a system of linear inequalities for any $\mathcal{R}_1$ language and prove that if the system is not consistent the language cannot be recognized by MM-BQFA (and MM-QFA, DH-PRA, EQFA); (3) We construct DH-PRA (this presumes also EQFA and MM-BQFA) and MM-QFA for any $\mathcal{R}_1$ language having a consistent system of inequalities. Thus, we obtain that an $\mathcal{R}_1$ language is recognizable by "decide-and-halt" models if and only if the corresponding system of linear inequalities is consistent; (4) We show that "forbidden constructions" known from [3] do not give all of the languages that cannot be recognized by MM-QFA.

2 Monoids and Varieties

Given an alphabet A, let A^* be the set of words over alphabet A. Given a word $\mathbf{x}$, let $|\mathbf{x}|$ be the length of $\mathbf{x}$. Introduce a partial order $\leqslant$ on A^*, let $\mathbf{x} \leqslant \mathbf{y}$ if and only if there exists $\mathbf{z} \in A^*$ such that $\mathbf{xz} = \mathbf{y}$. Let $\mathcal{P}(A)$ be the set of subsets of A, including the empty set $\emptyset$. Note that there is a natural partial order on $\mathcal{P}(A)$, i.e., the subset order. Given a word $\mathbf{s} \in A^*$, let $\mathbf{s}\omega$ be the set of letters of the word $\mathbf{s}$. We say that $\mathbf{u}, \mathbf{v} \in A^*$ are equivalent with respect to ω, $\mathbf{u} \sim_\omega \mathbf{v}$, if $\mathbf{u}\omega = \mathbf{v}\omega$ (that is, $\mathbf{u}$ and $\mathbf{v}$ consist of the same set of letters). Let $\mathcal{F}(A)$ be the set of all words over the alphabet A that do not contain any repeated letters. The empty word ε is an element of $\mathcal{F}(A)$. Let τ be a function such that for every $\mathbf{s} \in A^*$, any repeated letters in $\mathbf{s}$ are deleted, leaving only the first occurrence. Given $\mathbf{u}, \mathbf{v} \in A^*$, we say that $\mathbf{u} \sim_\tau \mathbf{v}$, if $\mathbf{u}\tau = \mathbf{v}\tau$. Introduce a partial order $\leqslant$ on $\mathcal{F}(A)$, let $\mathbf{v_1} \leqslant \mathbf{v_2}$ if and only if there exists $\mathbf{v} \in \mathcal{F}(A)$ such that $\mathbf{v_1 v} = \mathbf{v_2}$. Note that $\sim_\omega$ and $\sim_\tau$ are equivalence relations. The functions ω and τ are morphisms; $(\mathbf{uv})\omega = \mathbf{u}\omega \cup \mathbf{v}\omega$ and $(\mathbf{uv})\tau = \mathbf{u}\tau \cdot \mathbf{v}\tau$. Moreover, ω (and τ) preserves the order relation since $\mathbf{u} \leqslant \mathbf{v}$ implies $\mathbf{u}\omega \subseteq \mathbf{v}\omega$ ($\mathbf{u} \leqslant \mathbf{v}$ implies $\mathbf{u}\tau \leqslant \mathbf{v}\tau$).

A general overview on varieties of finite semigroups, monoids as well as operations on them is given in [17]. Unless specified otherwise, the monoids discussed in this section are assumed to be finite. An element e of a monoid $\mathcal{M}$ is called an *idempotent*, if $e^2 = e$. If x is an element of a monoid $\mathcal{M}$, the unique idempotent of the subsemigroup of $\mathcal{M}$ generated by x [17] is denoted by x^ω. Given a regular language $L \subseteq A^*$, words $\mathbf{u}, \mathbf{v} \in A^*$ are called *syntactically congruent*, $\mathbf{u} \sim_L \mathbf{v}$, if for all $\mathbf{x}, \mathbf{y} \in A^*$ $\mathbf{xuy} \in L$ if and only if $\mathbf{xvy} \in L$. The set of equivalence classes $A^*/\sim_L$ is a monoid, called *syntactic monoid* of L and denoted $\mathcal{M}(L)$. The morphism φ from A^* to $A^*/\sim_L$ is called *syntactic morphism*. Given a monoid variety $\mathbf{V}$, the corresponding language variety is denoted by $\boldsymbol{\mathcal{V}}$. The set of languages over A recognized by monoids in $\mathbf{V}$ is denoted by $A^*\boldsymbol{\mathcal{V}}$.

Varieties Definitions. In this paper, we refer to the following monoid varieties. The definitions for $\mathbf{G}, \mathbf{J_1} = [\![x^2 = x, xy = yx]\!], \mathbf{R_1} = [\![xyx = xy]\!], \mathbf{ER_1}, \mathbf{J}, \mathbf{R}$ may be found in [9]. The definition for **EJ** is in [18]. Also, $\mathbf{ER} = [\![(x^\omega y^\omega)^\omega x^\omega = (x^\omega y^\omega)^\omega]\!]$, the variety considered in [6]. The respective language varieties corresponding to the monoid varieties above are denoted $\boldsymbol{\mathcal{G}}$ (group languages), $\boldsymbol{\mathcal{J}_1}$ (semilattice languages), $\boldsymbol{\mathcal{R}_1}$ ($\mathcal{R}$-trivial idempotent languages, or $\mathcal{R}_1$ languages), $\boldsymbol{\mathcal{ER}_1}$, $\boldsymbol{\mathcal{J}}$, $\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{EJ}}$, $\boldsymbol{\mathcal{ER}}$. It is possible to check that $\mathbf{J_1} \subset \mathbf{J} \subset \mathbf{EJ}$, $\mathbf{R_1} \subset \mathbf{R} \subset \mathbf{ER}$, $\mathbf{R_1} \subset \mathbf{ER_1} \subset \mathbf{ER}$, $\mathbf{J_1} \subset \mathbf{R_1}$, $\mathbf{J} \subset \mathbf{R}$ and $\mathbf{G} \subset \mathbf{EJ} \subset \mathbf{ER}$.

Semilattice Languages and Free Semilattices. A free semilattice over an alphabet A is defined as a monoid $(\mathcal{P}(A), \cup)$, where $\cup$ is the ordinary set union. For any alphabet A, the free semilattice $\mathcal{P}(A)$ satisfies the identities of $\mathbf{J_1}$, therefore $\mathcal{P}(A) \in \mathbf{J_1}$. Given a free semilattice $\mathcal{P}(A)$, one may represent it as a deterministic finite automaton $(\mathcal{P}(A), A, \emptyset, \cdot)$, where for every $X \in \mathcal{P}(A)$ and for every $a \in A$, $X \cdot a = X \cup \{a\}$. It is implied by results in [17] that for any semilattice language L over alphabet A, $L\omega$ is a set of final states, such that the automaton recognizes the language. Therefore, in order to specify a particular language $L \in A^*\boldsymbol{\mathcal{J}_1}$, one may identify it by indicating a particular subset of $\mathcal{P}(A)$. A free semilattice over $\{a, b, c\}$ represented as a finite automaton is depicted in Figure 1 on the left side. The states of $(\mathcal{P}(A), A, \emptyset, \cdot)$ can be separated into several levels, i.e., a state is at level k if it corresponds to an element in $\mathcal{P}(A)$ of cardinality k.

$\boldsymbol{\mathcal{R}_1}$ languages and Free Left Regular Bands. A free left regular band over an alphabet A is defined as a monoid $(\mathcal{F}(A), \cdot)$, where $\mathbf{x} \cdot \mathbf{y} = (\mathbf{xy})\tau$, i.e., concatenation followed by the application of τ. For any alphabet A, the free left regular band $\mathcal{F}(A)$ satisfies the identities of $\mathbf{R_1}$, therefore $\mathcal{F}(A) \in \mathbf{R_1}$. Given a free left regular band $\mathcal{F}(A)$, one may represent it as a deterministic finite automaton $(\mathcal{F}(A), A, \varepsilon, \cdot_{\mathcal{F}(A)})$. A free left regular band over $\{a, b, c\}$ represented as

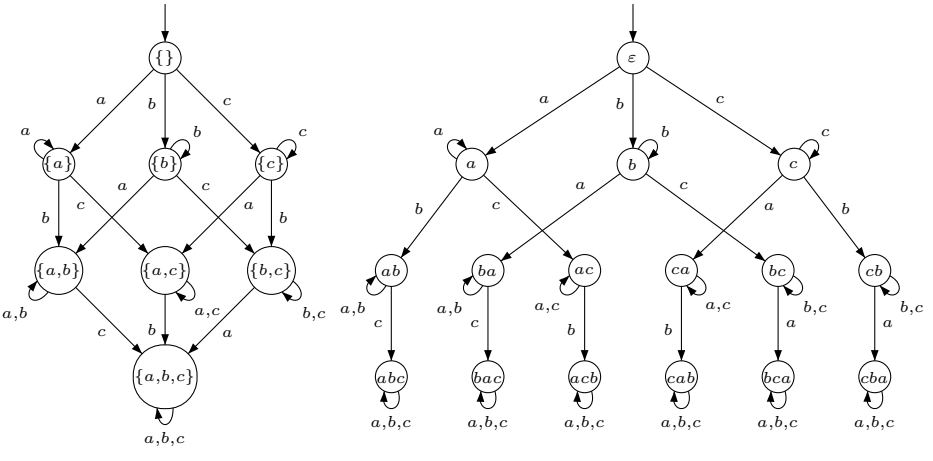


Fig. 1. Free semilattice and free left regular band over $\{a, b, c\}$

a finite automaton is depicted in Figure 1 on the right side. It is implied by [19] that for any $\mathcal{R}_1$ language L over alphabet A, $L\tau$ is a set of final states, such that the automaton recognizes the language. Therefore, in order to specify a particular language $L \in A^*\boldsymbol{\mathcal{R}_1}$, one may identify it by indicating a particular subset of $\mathcal{F}(A)$. For example, the semilattice language A^*aA^* may also be denoted as $\{\mathbf{a}, \mathbf{ab}, \mathbf{ba}, \mathbf{ac}, \mathbf{ca}, \mathbf{abc}, \mathbf{acb}, \mathbf{bac}, \mathbf{bca}, \mathbf{cab}, \mathbf{cba}\}$. We can also see that $\mathcal{P}(A)$ is a quotient of $\mathcal{F}(A)$. Indeed, let σ be a restriction of ω to $\mathcal{F}(A)$. The function σ is a surjective morphism from $\mathcal{F}(A)$ to $\mathcal{P}(A)$ which preserves the order relation.

Free left regular bands and free semilattices are key elements to prove that a quantum automaton may recognize a particular $\mathcal{R}_1$ language if and only if its system of linear inequalities is consistent.

3 Completely Positive Maps

In this section, we establish some facts about completely positive (CP) maps with certain properties, i.e., CP maps that describe the evolution of BQFA, defined in the next section. A comprehensive account on quantum computation and CP maps can be found in [16]. Following [16], we call a matrix $M \in \mathbb{C}^{n\times n}$ *positive*, if for any vector $X \in \mathbb{C}^n$, X^*MX is real and nonnegative. For arbitrary matrices M, N we may write $M \geqslant N$ if $M - N$ is positive. Let I_s be the identity map over $\mathbb{C}^{s\times s}$. Given Φ and Ψ, let $\Phi \bigotimes \Psi$ be the tensor product of those linear maps. A positive linear map Φ is called *completely positive*, if for any $s \geqslant 1$, $\Phi \bigotimes I_s$ is positive. Any CP map from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{m\times m}$ may be regarded as a linear operator in $\mathbb{C}^{n^2\times m^2}$. A CP map Φ is called *sub-tracial* iff for any positive M we have $\mathrm{Tr}(\Phi(M)) \leqslant \mathrm{Tr}(M)$. A CP map Φ from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{m\times m}$ is called *unital* if $\Phi(I_n) = I_m$. A CP map from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{m\times m}$ Φ is called *sub-unital* if $\Phi(I_n) \leqslant I_m$. A *composition* of CP maps $\Phi_0, ..., \Phi_m$ from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$ is a CP map $\Phi = \Phi_0 \circ \cdots \circ \Phi_m$ such that for any $M \in \mathbb{C}^{n\times n}$ $\Phi(M) = \Phi_0(\Phi_1(...(\Phi_m(M))...)$. A CP map Φ from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$ is called *bistochastic*, if it is both trace preserving and unital, i.e., for any positive M, $\mathrm{Tr}(\Phi(M)) = \mathrm{Tr}(M)$ and $\Phi(I_n) = I_n$. A CP map $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$ is called *sub-bistochastic*, if it is both sub-unital and sub-tracial. A composition of two sub-bistochastic CP maps is a sub-bistochastic CP map. We are interested about some properties of the asymptotic dynamics resulting from iterative application of a CP sub-bistochastic map. A CP map Φ from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$ is called *idempotent* if $\Phi \circ \Phi = \Phi$. It is said that a CP map Φ from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$ *generates a unique idempotent*, denoted Φ^ω, if there exists a sequence of positive integers n_s such that 1) exists the limit $\Phi^\omega = \lim_{s\to\infty} \Phi^{n_s}$; 2) the CP map Φ^ω is idempotent; 3) for any sequence of positive integers m_s such that the limit $\lim_{s\to\infty} \Phi^{m_s}$ exists and is idempotent, $\lim_{s\to\infty} \Phi^{m_s} = \Phi^\omega$. By Kuperberg [12], for any CP sub-bistochastic map Φ from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$, its idempotent Φ^ω exists, is unique and is a linear projection operator in $\mathbb{C}^{n^2\times n^2}$. That implies the subsequent theorem, which ultimately is the reason why certain models of quantum finite automata cannot recognize all regular languages.

Theorem 3.1. *Let $e_1, ..., e_k$ be idempotent CP sub-bistochastic maps from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{n\times n}$. Then for any i, $1 \leqslant i \leqslant k$, (1) $\lim_{n\to\infty}(e_1 \circ ... \circ e_k)^n = (e_1 \circ ... \circ e_k)^\omega = (e_{\pi(1)} \circ ... \circ e_{\pi(k)})^\omega$, where π is a permutation in $\{1, \ldots, k\}$; (2) $(e_1 \circ ... \circ e_k)^\omega = e_i \circ (e_1 \circ ... \circ e_k)^\omega = (e_1 \circ ... \circ e_k)^\omega \circ e_i$.*

Any finite quantum system at a particular moment of time (i.e., its *mixed state*) is described by a *density matrix*. By [16, Theorem 2.5], a matrix is a density matrix if and only if it is positive and its trace is equal to 1. Informally, an $n \times n$ density matrix describes a quantum system with n states. A completely positive trace-preserving map describes an evolution of a quantum system as allowed by quantum mechanics. It maps a density matrix to a density matrix.

4 Automata Models

For the formal definitions of other indicated automata models, the reader is referred to the following references. "Classical" models: Group Automata (GA, [21]), Measure-Once Quantum Finite Automata (MO-QFA, [15,5]), "Classical" Probabilistic Reversible Automata (C-PRA, [7,1]), Latvian Quantum Finite Automata (LQFA, [1]). "Decide-and-halt" models: Reversible Finite Automata (RFA, [2,9]), Measure-Many Quantum Finite Automata (MM-QFA, [11,5,3,1]), "Decide-and-halt" Probabilistic Reversible Automata (DH-PRA, [7,8]), Enhanced Quantum Finite Automata (EQFA, [14]). In case of classical acceptance, an automaton reads an input word until the last letter, and then accepts or rejects the word depending on whether the current state is final or non-final. In case of "decide-and-halt" acceptance, the automaton reads the input word until it enters a halting state. The input is accepted or rejected depending on whether the halting state is accepting or rejecting. Every word is appended with a special symbol, an end-marker, to ensure that any word is either accepted or rejected. We define MO and MM bistochastic QFA as a generalization of these models, which allows to prove the limitations of language recognition for all the models within single framework.

A bistochastic quantum finite automaton (BQFA) is a tuple $(Q, A \cup \{\#, \$\}, q_0, \{\Phi_a\})$, where Q is a finite set of states, A - a finite input alphabet, $\#, \$ \notin A$ - initial and final end-markers, q_0 - an initial state and for each $a \in A \cup \{\#, \$\}$ Φ_a is a CP bistochastic transition map from $\mathbb{C}^{|Q|\times|Q|}$ to $\mathbb{C}^{|Q|\times|Q|}$.

Regardless of which word acceptance model is used, each input word is enclosed into end-markers $\#, \$$. At any step, the mixed state of a BQFA may be described by a density matrix ρ. The computation starts in the state $|q_0\rangle\langle q_0|$.

Operation of a measure-once BQFA and word acceptance is the same as described for LQFA [1], only instead of sequences of unitary operations and orthogonal measurements we have arbitrary bistochastic CP maps. On input letter $a \in A$, ρ is transformed into $\Phi_a(\rho)$.

Operation of a measure-many BQFA and word acceptance is the same as described for EQFA [14], but arbitrary bistochastic CP maps are used. The set of states Q is partitioned into three disjoint subsets Q_{non}, Q_{acc} and Q_{rej} - non-halting, accepting and rejecting states, respectively. On input letter $a \in A$, ρ is

transformed into $\rho' = \Phi_a(\rho)$. After that, a measurement $\{P_{non}, P_{acc}, P_{rej}\}$ is applied to ρ', where for each $i \in \{non, acc, rej\}$ $P_i = \sum_{q \in Q_i} |q\rangle\langle q|$. To describe the probability distribution $S_{\#\mathbf{u}}$ of a MM-BQFA $\mathcal{A}$ after reading some prefix $\#\mathbf{u}$, it is convenient to use density matrices ρ scaled by p, $0 \leqslant p \leqslant 1$. So the probability distribution $S_{\#\mathbf{u}}$ of $\mathcal{A}$ is a triple (ρ, p_{acc}, p_{rej}), where $\mathrm{Tr}(\rho) + p_{acc} + p_{rej} = 1$, $\rho/\mathrm{Tr}(\rho)$ is the current mixed state and p_{acc}, p_{rej} are respectively the probabilities that $\mathcal{A}$ has accepted or rejected the input. So the scaled density matrix ρ may be called a *scaled mixed state*. For any $a \in A \cup \{\#, \$\}$, let $\Psi_a(\rho) = P_{non}\Phi_a(\rho)P_{non}$. After reading the next input letter a, the probability distribution is $S_{\#\mathbf{u}a} = (\Psi_a(\rho), p_{acc} + \mathrm{Tr}(P_{acc}\Phi_a(\rho)P_{acc}), p_{rej} + \mathrm{Tr}(P_{rej}\Phi_a(\rho)P_{rej}))$. For any word $\mathbf{a} = a_1 \dots a_k$, define $\Psi_{\mathbf{a}} = \Psi_{a_k} \circ \dots \circ \Psi_{a_1}$. Hence $\rho = \Psi_{\#\mathbf{u}}(|q_0\rangle\langle q_0|)$. Note that $\Psi_{\mathbf{a}}$ is a CP sub-bistochastic map.

Language recognition is defined in the same way as in Rabin's [20]. Suppose that an automaton $\mathcal{A}$ is one of the models from the list above. By $p_{\mathbf{x},\mathcal{A}}$ (or $p_{\mathbf{x}}$, if no ambiguity arises) we denote the probability that an input $\mathbf{x}$ is accepted by the automaton $\mathcal{A}$. We consider only bounded error language recognition.

BQFA as a generalization of other models. Since unitary operations and orthogonal measurements are bistochastic operations, MO-BQFA is a generalization of LQFA and MM-BQFA is a generalization of EQFA. Also, the Birkhoff theorem [22, Theorem 4.21] implies that MO-BQFA and MM-BQFA are generalizations of C-PRA and DH-PRA, respectively. On the other hand, BQFA are a special case of one-way general QFA, which admit any CP trace-preserving transition maps. One-way general QFA recognize with bounded error exactly the regular languages [10,13]. So the recognition power of BQFA is also limited to regular languages only.

Comparison of the language classes. Having a certain class of automata $\boldsymbol{\mathcal{A}}$, let us denote by $\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{A}})$ the respective class of languages. Thus $\boldsymbol{\mathcal{L}}(\mathrm{GA}) = \boldsymbol{\mathcal{L}}(\mathrm{MO\text{-}QFA}) = \boldsymbol{\mathcal{G}}$, $\boldsymbol{\mathcal{L}}(\mathrm{C\text{-}PRA}) = \boldsymbol{\mathcal{L}}(\mathrm{LQFA}) = \boldsymbol{\mathcal{L}}(\mathrm{MO\text{-}BQFA}) = \boldsymbol{\mathcal{EJ}}$, $\boldsymbol{\mathcal{G}} \subsetneq \boldsymbol{\mathcal{L}}(\mathrm{RFA}) \subsetneq \boldsymbol{\mathcal{ER}_1}$, $\boldsymbol{\mathcal{EJ}} \subsetneq \boldsymbol{\mathcal{L}}(\mathrm{MM\text{-}QFA}) \stackrel{?}{=} \boldsymbol{\mathcal{L}}(\mathrm{DH\text{-}PRA}) \stackrel{?}{=} \boldsymbol{\mathcal{L}}(\mathrm{EQFA}) \stackrel{?}{=} \boldsymbol{\mathcal{L}}(\mathrm{MM\text{-}BQFA}) \subsetneq \boldsymbol{\mathcal{ER}}$. Relations concerning BQFA depend on the theorem below. All the other relations are known from the references given in the list of automata models above.

Theorem 4.1. *$\boldsymbol{\mathcal{L}}(MO\text{-}BQFA) = \boldsymbol{\mathcal{EJ}}$ and $\boldsymbol{\mathcal{L}}(MM\text{-}BQFA) \subseteq \boldsymbol{\mathcal{ER}}$. $\boldsymbol{\mathcal{L}}(MM\text{-}BQFA)$ is closed under complement, inverse free monoid morphisms, and word quotient.*

The proof relies on Theorem 3.1 and ideas in [1,5] used for LQFA and MM-QFA.

We find that $\boldsymbol{\mathcal{L}}$(MM-BQFA) shares a lot of properties with the language classes of other "decide-and-halt" word acceptance models. In Section 7 it is noted that MM-BQFA does not recognize any of the languages corresponding to "forbidden constructions" from [3, Theorem 4.3]. As other "decide-and-halt" models, $\boldsymbol{\mathcal{L}}(\mathrm{MM\text{-}BQFA}) \subsetneq \boldsymbol{\mathcal{ER}}$ and $\boldsymbol{\mathcal{L}}$(MM-BQFA) is not closed under union and intersection (Corollary 6.3).

5 Linear Inequalities

In this section, we define a system of linear inequalities that an $\mathcal{R}_1$ language recognized by a MM-BQFA must satisfy. Let L be an $\mathcal{R}_1$ language and $\mathcal{S}$ - a

MM-BQFA, both over alphabet A. Let $\{\mathbf{v_0}, \mathbf{v_1}, ..., \mathbf{v_R}\} = \mathcal{F}(A)$. Assume $\mathbf{v_0} = \varepsilon$. For any $\mathbf{v}$ in $\mathcal{F}(A)$, where $\mathbf{v} = a_1...a_k$ (a_i are distinct letters of A), denote by $\mathbf{v}[i]$ a prefix of $\mathbf{v}$ of length i, i.e. $\mathbf{v}[0] = \varepsilon$ and for all i, $1 \leqslant i \leqslant k$, $\mathbf{v}[i] = a_1...a_i$. Recall that $\mathcal{F}(A)$ can be viewed as an automaton that recognizes an $\mathcal{R}_1$ language L, provided $L\tau$ is its set of final states (see Section 2).

Now define a linear system of inequalities $\mathfrak{L}$ as follows: *(1) For every* $\mathbf{v}$ *in* $\mathcal{F}(A)$*, where* $\mathbf{v} = a_1...a_k$ *take the formal expression* $\mathfrak{L}(\mathbf{v}) = x_0 + x_{\mathbf{v}[0]\omega,a_1} + x_{\mathbf{v}[1]\omega,a_2} + x_{\mathbf{v}[2]\omega,a_3} + ... + x_{\mathbf{v}[k-1]\omega,a_k} + y_{\mathbf{v}\omega}$*; (2) Introduce two another variables* p_1 *and* p_2*. For any* $\mathbf{v} \in \mathcal{F}(A)$*, if* $\mathbf{v} \in L\tau$*, construct an inequality* $\mathfrak{L}(\mathbf{v}) \geqslant p_2$*, otherwise construct an inequality* $\mathfrak{L}(\mathbf{v}) \leqslant p_1$*; (3) Append the system by an inequality* $p_1 < p_2$*.*

Example 5.1. Consider an $\mathcal{R}_1$ language $L = \{\mathbf{ab}, \mathbf{bac}\}$ over alphabet $A = \{a, b, c\}$. Among others, the system $\mathfrak{L}(L)$ has the following inequalities:

$$\begin{aligned}
\mathfrak{L}(\mathbf{ab}) &= x_0 + x_{\{\},a} + x_{\{a\},b} + y_{\{a,b\}} &&\geqslant p_2\\
\mathfrak{L}(\mathbf{bac}) &= x_0 + x_{\{\},b} + x_{\{b\},a} + x_{\{a,b\},c} + y_{\{a,b,c\}} &&\geqslant p_2\\
\mathfrak{L}(\mathbf{ba}) &= x_0 + x_{\{\},b} + x_{\{b\},a} + y_{\{a,b\}} &&\leqslant p_1\\
\mathfrak{L}(\mathbf{abc}) &= x_0 + x_{\{\},a} + x_{\{a\},b} + x_{\{a,b\},c} + y_{\{a,b,c\}} &&\leqslant p_1\\
& && p_1 < p_2
\end{aligned}$$

Informally, an inequality $\mathfrak{L}(\mathbf{v})$ represents the probability of accepting a specifically defined input word $\mathbf{u}$, such that $\mathbf{u}\tau = \mathbf{v} = a_1a_2...a_k$. The variable x_0 represents the probability to accept the input after reading the initial end-marker #. The variable $y_{\mathbf{v}\omega}$ represents the cumulative probability to stay in non-halting states before reading the final end-marker \$ and accept the input after reading it. The variable $x_{\mathbf{v}[i-1]\omega,a_i}$ represents the cumulative probability to stay in non-halting states after reading a (specifically defined) prefix $\mathbf{u}[i-1]$ (of $\mathbf{u}$) such that $\mathbf{u}[i-1]\tau = \mathbf{v}[i-1]$ and to accept input after reading a (specifically defined) prefix $\mathbf{u}[i]$ such that $\mathbf{u}[i]\tau = \mathbf{v}[i]$.

Theorem 5.2. *Suppose* L *is an* $\mathcal{R}_1$ *language. If the linear system* $\mathfrak{L}$ *is not consistent, then* L *cannot be recognized by any MM-BQFA.*

Proof. Let m_l ($l = 1, 2, ...$) be a sequence of positive integers such that for all letters $a \in A$ $\lim_{l\to\infty} \Psi_a^{m_l} = \Psi_a^{\omega}$ (existence is proved in the same way as the Kuperberg's result quoted in Section 3). Let μ be a function that assigns to any word in A^* the same word (of the same length) with letters sorted in alphabetical order. Let $\varkappa_i$, $i \in \mathbb{N}$, a morphism from A^* to A^* such that for any $a \in A$ $a\varkappa_i = a^i$. Let $\xi = \xi_l$ be an everywhere defined function from $\mathcal{F}(A)$ to A^*, such that $\varepsilon\xi = \varepsilon$ and for all $\mathbf{v} \in \mathcal{F}(A)$, if $|\mathbf{v}| = 1$ then $\mathbf{v}\xi = \mathbf{v}^{m_l}$ and otherwise, if $|\mathbf{v}| \geqslant 2$ then $\mathbf{v}\xi = (\mathbf{v}\mu\varkappa_{m_l})^l$.

Let us define the function θ (which depends on the parameter l) as follows. For any $\mathbf{v} = a_1...a_k$ let $\mathbf{v}\theta = \mathbf{v}[1]\xi...\mathbf{v}[k]\xi = a_1^{m_l}((a_1^{m_l}a_2^{m_l})\mu)^l...((a_1^{m_l}a_2^{m_l}\dots a_k^{m_l})\mu)^l$. Let $\varepsilon\theta = \varepsilon$. Note that $\mathbf{v}[i]\theta = (\mathbf{v}[i-1]\theta)(\mathbf{v}[i]\xi)$. In the discussion preceding this theorem, the word $\mathbf{u}$ corresponding to $\mathbf{v}$ is $\mathbf{v}\theta$ and the prefix $\mathbf{u}[i]$ corresponding to $\mathbf{v}[i]$ is $\mathbf{v}[i]\theta$. Note that $\mathbf{v}\theta\tau = \mathbf{v}$.

Now take the set $\mathcal{F}(A)\theta = \{\mathbf{u_k} \mid \mathbf{u_k} = \mathbf{v_k}\theta \text{ and } 0 \leqslant k \leqslant R\}$. Let us take a positive integer i and any two words $\mathbf{u}$ and $\mathbf{u}'$ in $\mathcal{F}(A)\theta$, such that $\mathbf{u}[i-1]\omega = \mathbf{u}'[i-1]\omega$. Let $\mathbf{v} = \mathbf{u}\tau$ and $\mathbf{v}' = \mathbf{u}'\tau$. If the parameter l is sufficiently large, Theorem 3.1 implies that after reading $\mathbf{u}[i-1]$ and $\mathbf{u}'[i-1]$ the automaton $\mathcal{S}$ has essentially the same scaled density matrices (which represent the non-halting states). Suppose that $\mathbf{v}[i] = \mathbf{v}'[i] = a_1...a_i$. The automaton $\mathcal{S}$ finishes reading the prefixes $\mathbf{u}[i]$ and $\mathbf{u}'[i]$ after reading the next symbols forming the sub-word $\mathbf{v}[i]\xi$. The cumulative probabilities to stay in non-halting states while reading $\mathbf{u}[i-1]$, $\mathbf{u}'[i-1]$ and to accept input after reading $\mathbf{v}[i]\xi = \mathbf{v}'[i]\xi$ will be essentially the same. (They converge to the same value as l tends to infinity). Hence those probabilities are reflected in the system of linear inequalities by the same variable $x_{\mathbf{v}[i-1],a_i}$. Thus, if a MM-BQFA $\mathcal{S}$ recognizes an $\mathcal{R}_1$ language L, then the linear system of inequalities $\mathfrak{L}$ has to be consistent. □

If the linear system $\mathfrak{L}(L)$ is not consistent, then L cannot be recognized by any MM-QFA, DH-PRA or EQFA as well. The statement converse to Theorem 5.2 is provided in Section 6 (Theorem 6.2).

Consider the inequalities in the system $\mathfrak{L}(L)$. The only possible coefficients of variables in any linear inequality are -1, 0 and 1. Denote by $Z = \{x_0, z_1, ..., z_s, y_1, ..., y_t, p_1, p_2\}$ the set of all the variables in the system $\mathfrak{L}$, where z_i are variables of the form $x_{\mathbf{v}[i-1]\omega,a_i}$, and y_i are variables of the form $y_{\mathbf{v}\omega}$.

Proposition 5.3. *The system $\mathfrak{L}$ is consistent if and only if it has a solution where $x_0 = 0$, $y_A = 0$, $0 \leqslant p_1, p_2 \leqslant 1$ and all the other variables $z_1, ..., z_s, y_1, ..., y_{t-1}$ are assigned real values from 0 to $1/|A|$.*

6 Construction of DH-PRA and MM-QFA for $\mathcal{R}_1$ Languages

Preparation of a linear programming problem. Consider an $\mathcal{R}_1$ language L over alphabet A. Construct the respective system of linear inequalities $\mathfrak{L}$. Obtain a system $\mathfrak{L}_1$ by supplementing $\mathfrak{L}$ with additional inequalities that enforce the constraints expressed in Proposition 5.3, according to which $\mathfrak{L}$ is consistent if and only if $\mathfrak{L}_1$ is consistent. Obtain a system $\mathfrak{L}'_1$ by replacing in $\mathfrak{L}_1$ the inequality $p_1 < p_2$ by $p_1 \leqslant p_2$. The linear programming problem, denoted $\mathfrak{P}$, is to maximize $p_2 - p_1$ according to the constraints expressed by $\mathfrak{L}'_1$. Since $\mathfrak{L}'_1$ is homogenous, it always has a solution where $p_1 = p_2$. Since the solution polytope of $\mathfrak{L}'_1$ is bounded, $\mathfrak{P}$ always has an optimal solution. Obviously, if the optimal solution yields $p_1 = p_2$, then $\mathfrak{L}_1$ is not consistent and therefore, by Theorem 5.2, a DH-PRA that recognizes L does not exist. Otherwise, if the optimal solution yields $p_1 < p_2$, then $\mathfrak{L}_1$ is consistent.

Automata derived from the free semilattice $\mathcal{P}(A)$. Assume $\mathfrak{L}_1$ is consistent, so we are able to obtain a solution of $\mathfrak{P}$ where $p_1 < p_2$. Given any expression Z of variables from $\mathfrak{L}_1$, let $\mathfrak{P}(Z)$ - the value which is assigned to Z by solving $\mathfrak{P}$. First, we use the obtained solution to construct probabilistic automata $\mathcal{A}_i$, $1 \leqslant i \leqslant |A|$. Those automata are not probabilistic reversible. Similarly as in the

"decide-and-halt" model, the constructed automata have accepting, rejecting and non-halting states. Any input word is appended by the end-marker \$. The initial end-marker # is not used for those automata themselves. Any automaton $\mathcal{A}_i$ is a tuple $(Q_i, A \cup \{\$\}, s_i, \delta_i)$, where Q_i is a set of states, s_i - an initial state and δ_i - a transition function $Q \times A \times Q \longrightarrow [0,1]$, so $\delta_i(q,a,q')$ is a probability of transit from q to q' on reading input letter a. $\mathcal{A}_i$ is constructed as follows: (1) Take the deterministic automaton $(\mathcal{P}(A), A, \emptyset, \cdot)$, remove all the states at level greater or equal to i. The remaining states are defined to be non-halting. The state $\emptyset$ is initial, it is the only state of $\mathcal{A}_i$ at level 0. For any a in A and state $\mathbf{s}$ at levels $\{0,\ldots,i-2\}$, $\delta_i(\mathbf{s},a,\mathbf{s}\cdot a) = 1$. For any state $\mathbf{s}$ at level $i-1$ and any a in $\mathbf{s}$, $\delta_i(\mathbf{s},a,\mathbf{s}) = 1$; (2) For any non-halting state $\mathbf{s}$ at levels $\{0,\ldots,i-2\}$, add a rejecting state $(\mathbf{s}\$)_{rej}$. Let $\delta_i(\mathbf{s},\$,(\mathbf{s}\$)_{rej}) = 1$; (3) For any state $\mathbf{s}$ at level $i-1$, add $|A|-|\mathbf{s}|+1$ accepting states $(\mathbf{s}a)_{acc}$, $a \in (A \setminus \mathbf{s}) \cup \{\$\}$. Also add $|A|-|\mathbf{s}|+1$ rejecting states $(\mathbf{s}a)_{rej}$, $a \in (A \setminus \mathbf{s}) \cup \{\$\}$; (4) If $a \in A \setminus \mathbf{s}$, any element $\mathbf{s}'a$ in $(\mathbf{s}\sigma^{-1})a$ defines the same variable $x_{\mathbf{s},a}$ in the system of linear inequalities (σ is defined in Section 2). Let $c_{\mathbf{s},a} = \mathfrak{B}(x_{\mathbf{s},a})$. Any element $\mathbf{s}'$ in $\mathbf{s}\sigma^{-1}$ defines the same variable $y_{\mathbf{s}}$. Let $d_{\mathbf{s}} = \mathfrak{B}(y_{\mathbf{s}})$; (5) Define missing transitions for the states at level $i-1$. For any state $\mathbf{s}$ at level $i-1$ and any a in $A \setminus \mathbf{s}$, let $t_{\mathbf{s},a} = \delta_i(\mathbf{s},a,(\mathbf{s}a)_{acc}) = c_{\mathbf{s},a}|A|$ and $\delta_i(\mathbf{s},a,(\mathbf{s}a)_{rej}) = 1 - t_{\mathbf{s},a}$. Let $v_{\mathbf{s}} = \delta_i(\mathbf{s},\$,(\mathbf{s}\$)_{acc}) = d_{\mathbf{s}}|A|$ and $\delta_i(\mathbf{s},\$,(\mathbf{s}\$)_{rej}) = 1 - v_{\mathbf{s}}$; (6) The formally needed transitions outgoing the halting states for now are left undefined.

Consider an automaton $\mathcal{A}$ (Figure 2), which with the same probability $1/|A|$ executes any of the automata $\mathcal{A}_1, ..., \mathcal{A}_{|A|}$ (i.e., it uses the initial end-marker # to transit to initial states of any of those automata). By construction of $\mathcal{A}_1, ..., \mathcal{A}_{|A|}$, the automaton $\mathcal{A}$ accepts any word $\mathbf{u} \in A^*$ with probability $\mathfrak{P}(\mathfrak{L}(\mathbf{u}\tau))$. Since for any word $\mathbf{u} \in L$, $\mathfrak{P}(\mathfrak{L}(\mathbf{u}\tau)) \geqslant \mathfrak{P}(p_2)$, and for any word $\mathbf{w} \notin L$, $\mathfrak{P}(\mathfrak{L}(\mathbf{w}\tau)) \leqslant \mathfrak{P}(p_1)$, the automaton $\mathcal{A}$ recognizes the language L.

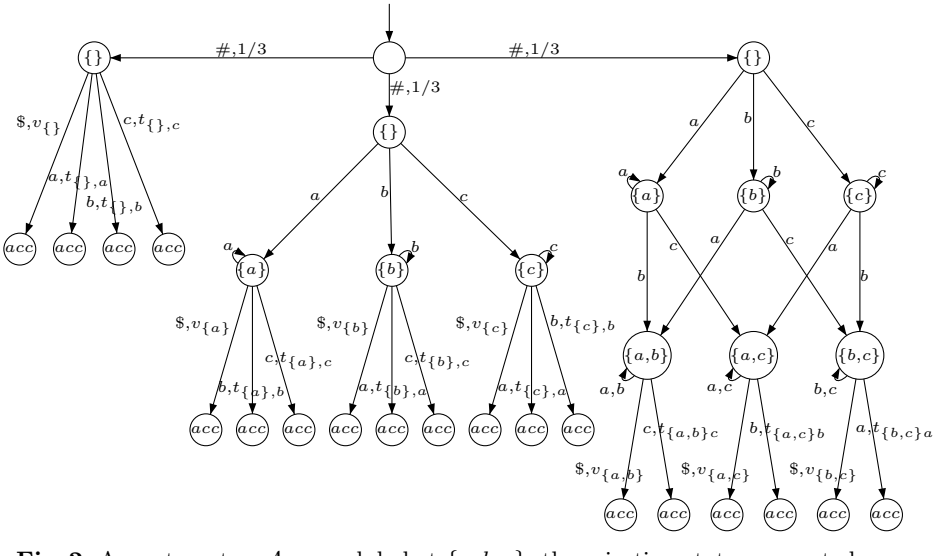


Fig. 2. An automaton $\mathcal{A}$ over alphabet $\{a,b,c\}$, the rejecting states are not shown

Construction of a DH-PRA. In order to construct a DH-PRA recognizing L, it remains to demonstrate that any of the automata $\mathcal{A}_1, ..., \mathcal{A}_{|A|}$ may be simulated by some DH probabilistic reversible automata, that is, for any automaton $\mathcal{A}_i$, it is possible to construct a sequence of DH-PRA $\mathcal{S}_{i,n}$, where $n \geqslant 1$, such that $p_{\mathbf{w},\mathcal{S}_{i,n}}$ converges uniformly to $p_{\mathbf{w},\mathcal{A}_i}$ on A^* as $n \to \infty$. An automaton $\mathcal{A}_i = (Q_i, A \cup \{\$\}, s_i, \delta_i)$ is used to construct a DH-PRA $\mathcal{S}_{i,n} = (Q_{i,n}, A \cup \{\$\}, s_i, \delta_{i,n})$ as described next. Initially $Q_{i,n}$ is empty. (1) For any non-halting state $\mathbf{s}$ at level j, $0 \leqslant j \leqslant i-1$, supplement $\mathcal{S}_{i,n}$ with non-halting states denoted $\mathbf{s}_k$, where $1 \leqslant k \leqslant n^j$; (2) For any non-halting state $\mathbf{s}$ at level j, $0 \leqslant j < i-1$, supplement $\mathcal{S}_{i,n}$ with rejecting states $(\mathbf{s}\$)_{rej,k}$, where $1 \leqslant k \leqslant n^j$; (3) For any non-halting state $\mathbf{s}$ at level $i-1$, accepting state $(\mathbf{s}a)_{acc}$ and rejecting state $(\mathbf{s}a)_{rej}$, where $a \in (A \setminus \mathbf{s}) \cup \{\$\}$, supplement $\mathcal{S}_{i,n}$ with accepting states $(\mathbf{s}a)_{acc,k}$ and rejecting states $(\mathbf{s}a)_{rej,k}$, where $1 \leqslant k \leqslant n^{i-1}$.

It remains to define the transitions. For any non-halting state $\mathbf{s}$ of $\mathcal{A}_i$ at level j, $1 \leqslant j \leqslant i-1$, the states in $\{\mathbf{s}_k\}$ are grouped into n^{j-1} disjoint subsets with n states in each, so that any state in $\{\mathbf{s}_k\}$ may be denoted as $\mathbf{s}_{l,m}$, where $1 \leqslant l \leqslant n^{j-1}$ and $1 \leqslant m \leqslant n$. For any letter a in A, consider all pairs of non-halting states $\mathbf{s}, \mathbf{t}$ of $\mathcal{A}_i$ such that $\mathbf{s} \neq \mathbf{t}$ and $\delta_i(\mathbf{s}, a, \mathbf{t}) = 1$. For any fixed k and any l and m, $1 \leqslant l, m \leqslant n$, define $\delta_{i,n}(\mathbf{s}_k, a, \mathbf{s}_k) = \delta_{i,n}(\mathbf{s}_k, a, \mathbf{t}_{k,m}) = \delta_{i,n}(\mathbf{t}_{k,m}, a, \mathbf{s}_k) = \delta_{i,n}(\mathbf{t}_{k,l}, a, \mathbf{t}_{k,m}) = 1/(n+1)$. For any non-halting state $\mathbf{s}$ of $\mathcal{A}_i$ at level j, $0 \leqslant j < i-1$, $\delta_{i,n}(\mathbf{s}_k, \$, (\mathbf{s}\$)_{rej,k}) = 1$, $\delta_{i,n}((\mathbf{s}\$)_{rej,k}, \$, \mathbf{s}_k) = 1$. For the same $(\mathbf{s}\$)_{rej,k}$ and any other letter b in $A \cup \{\$\}$, define $\delta_{i,n}((\mathbf{s}\$)_{rej,k}, b, (\mathbf{s}\$)_{rej,k}) = 1$. For any non-halting state $\mathbf{s}$ of $\mathcal{A}_i$ at level $i-1$ and $a \in (A \setminus \mathbf{s}) \cup \{\$\}$, the transitions induced by a among $\mathbf{s}_k$, $(\mathbf{s}a)_{acc,k}$, $(\mathbf{s}a)_{rej,k}$ are defined by the matrix $\begin{pmatrix} 0 & 0 & 1 \\ r_1 & r_2 & 0 \\ r_2 & r_1 & 0 \end{pmatrix}$, where $r_1 = \delta_i(\mathbf{s}, a, (\mathbf{s}a)_{acc})$, $r_2 = \delta_i(\mathbf{s}, a, (\mathbf{s}a)_{rej})$. The first, second and third rows and columns are indexed by $\mathbf{s}_k$, $(\mathbf{s}a)_{acc,k}$, $(\mathbf{s}a)_{rej,k}$, respectively. Note that $r_1 + r_2 = 1$. For the same $(\mathbf{s}a)_{acc,k}, (\mathbf{s}a)_{rej,k}$ and any other letter b in $A \cup \{\$\}$, define $\delta_{i,n}((\mathbf{s}a)_{acc,k}, b, (\mathbf{s}a)_{acc,k}) = \delta_{i,n}((\mathbf{s}a)_{rej,k}, b, (\mathbf{s}a)_{rej,k}) = 1$. We have defined all the non-zero transitions for $\mathcal{S}_{i,n}$. By construction, the transition matrices induced by any letter a in $A \cup \{\$\}$ are doubly stochastic.

Lemma 6.1. *For any i, $1 \leqslant i \leqslant |A|$, $p_{\mathbf{w},\mathcal{S}_{i,n}}$ converges uniformly to $p_{\mathbf{w},\mathcal{A}_i}$ on A^* as $n \to \infty$.*

Now it is possible to construct a DH-PRA $\mathcal{S} = (Q, A \cup \{\#, \$\}, s, \delta)$, which with the same probability $1/|A|$ executes the automata $\mathcal{S}_{1,n}, \ldots, \mathcal{S}_{|A|,n}$. The set of states Q is a disjoint union of $Q_1, ..., Q_{|A|}$. Take the initial state s_i of any $\mathcal{S}_{i,n}$ as the initial state s. For any $a \in A \cup \{\$\}$ and $q_1, q_2 \in Q_i$, $\delta(q_1, a, q_2) = \delta_i(q_1, a, q_2)$. For any initial states s_i and s_j of $\mathcal{S}_{i,n}$ and $\mathcal{S}_{j,n}$, $\delta(s_i, \#, s_j) = 1/|A|$. For any other state q, $\delta(q, \#, q) = 1$. So the transition matrices of $\mathcal{S}$ induced by any letter are doubly stochastic. By Lemma 6.1, $\mathcal{S}$ recognizes L if n is sufficiently large. Hence we have established one of the main results of this section:

Theorem 6.2. *Suppose L is an $\mathcal{R}_1$ language. If the linear system $\mathfrak{L}$ is consistent, then L can be recognized by a DH-PRA.*

Therefore, if the linear system $\mathfrak{L}$ is consistent, then L can be recognized by a MM-BQFA as well. Moreover, since all of the transition matrices of the constructed DH-PRA are also unitary stochastic, by [7, Theorem 5.2] L can be recognized by an EQFA.

Corollary 6.3. *The class $\boldsymbol{\mathcal{L}}$(MM-BQFA) is not closed under union and intersection. Moreover, $\boldsymbol{\mathcal{L}}$(MM-BQFA) $\subsetneq \boldsymbol{\mathcal{ER}}$.*

Proof. See the language $L = \{\mathbf{ab}, \mathbf{bac}\}$ in Example 5.1; the languages $\{\mathbf{ab}\}$ and $\{\mathbf{bac}\}$ can be recognized since their systems are consistent, while the system $\mathfrak{L}(L)$ is inconsistent. □

The construction of MM-QFA for $\mathcal{R}_1$ languages has some peculiarities which have to be addressed separately. Specifically, there exist semilattice languages that MM-QFA do not recognize with probability $1-\epsilon$ [2, Theorem 2] and therefore they can't simulate with the same accepting probabilities the automata $\mathcal{A}_1, ..., \mathcal{A}_{|A|}$. Nevertheless, since the matrices used in the construction of DH-PRA are unitary stochastic, a modified construction is still possible.

Theorem 6.4. *Suppose L is an $\mathcal{R}_1$ language. If the linear system $\mathfrak{L}$ is consistent, then L can be recognized by a MM-QFA.*

In summary, Theorems 5.2, 6.2 and 6.4 imply that an $\mathcal{R}_1$ language L can be recognized by MM-QFA if and only if the linear system $\mathfrak{L}(L)$ is consistent. MM-QFA, DH-PRA, EQFA and MM-BQFA recognize exactly the same $\mathcal{R}_1$ languages.

7 "Forbidden Constructions"

In [3, Theorem 4.3], Ķikusts has proposed "forbidden constructions" for MM-QFA; any regular language whose minimal deterministic finite automaton contains any of these constructions cannot be recognized by MM-QFA. It is actually implied by Theorem 3.1 that the same is true for MM-BQFA and other "decide-and-halt" models from Section 4. Also, by Theorem 4.1 any language that is recognized by a MM-BQFA is contained in $\boldsymbol{\mathcal{ER}}$. Therefore it is legitimate to ask whether all the $\boldsymbol{\mathcal{ER}}$ languages that do not contain any of the "forbidden constructions" can be recognized by MM-BQFA. The answer to this question is *negative*.

Theorem 7.1. *There exists an $\boldsymbol{\mathcal{ER}}$ language that does not contain any of the "forbidden constructions" and still cannot be recognized by MM-BQFA.*

Proof. It is sufficient to indicate a language in $\boldsymbol{\mathcal{R}_1}$ which satisfies the required properties and for which the system of linear inequalities is inconsistent. Such a language is $L = \{\mathbf{aedbc}, \mathbf{beca}, \mathbf{beda}, \mathbf{bedac}, \mathbf{eacb}, \mathbf{eacbd}, \mathbf{eadbc}, \mathbf{ebca}\}$ over alphabet $A = \{a, b, c, d, e\}$. □

8 Conclusion

In the paper we show which $\mathcal{R}_1$ languages can be recognized by "decide-and-halt" quantum automata. It is expected that these results can be first generalized to include any $\mathcal{R}$-trivial language, and finally, any language in $\boldsymbol{\mathcal{ER}}$, thus obtaining the solution of the language class problem for MM-QFA. To apply the same approach for $\mathcal{R}$-trivial languages, one would need to find convenient sets of finite $\mathcal{R}$-trivial and $\mathcal{J}$-trivial monoids that generate the varieties $\mathbf{R}$ and $\mathbf{J}$ and a function resembling θ (in the proof of Theorem 5.2). That is a subject for further research.

References

1. Ambainis, A., Beaudry, M., Golovkins, M., Ķikusts, A., Mercer, M., Thérien, D.: Algebraic Results on Quantum Automata. Theory of Computing Systems 39(1), 165–188 (2006)
2. Ambainis, A., Freivalds, R.: 1-Way Quantum Finite Automata: Strengths, Weaknesses and Generalizations. In: Proc. 39th FOCS, pp. 332–341 (1998)
3. Ambainis, A., Ķikusts, A., Valdats, M.: On the Class of Languages Recognizable by 1-Way Quantum Finite Automata. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 75–86. Springer, Heidelberg (2001)
4. Bianchi, M.P., Palano, B.: Behaviours of Unary Quantum Automata. Fundamenta Informaticae 104, 1–15 (2010)
5. Brodsky, A., Pippenger, N.: Characterizations of 1-Way Quantum Finite Automata. SIAM Journal on Computing 31(5), 1456–1478 (2002)
6. Eilenberg, S.: Automata, Languages and Machines, vol. B. Academic Press, New York (1976)
7. Golovkins, M., Kravtsev, M.: Probabilistic Reversible Automata and Quantum Automata. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 574–583. Springer, Heidelberg (2002)
8. Golovkins, M., Kravtsev, M., Kravcevs, V.: On a Class of Languages Recognizable by Probabilistic Reversible Decide-and-Halt Automata. Theoretical Computer Science 410(20), 1942–1951 (2009)
9. Golovkins, M., Pin, J.E.: Varieties Generated by Certain Models of Reversible Finite Automata. Chicago Journal of Theoretical Computer Science 2010, Article 2 (2010)
10. Hirvensalo, M.: Quantum Automata with Open Time Evolution. International Journal of Natural Computing Research 1(1), 70–85 (2010)
11. Kondacs, A., Watrous, J.: On The Power of Quantum Finite State Automata. In: Proc. 38th FOCS, pp. 66–75 (1997)
12. Kuperberg, G.: The Capacity of Hybrid Quantum Memory. IEEE Transactions on Information Theory 49-6, 1465–1473 (2003)
13. Li, L., Qiu, D., Zou, X., Li, L., Wu, L., Mateus, P.: Characterizations of One-Way General Quantum Finite Automata (2010), http://arxiv.org/abs/0911.3266
14. Mercer, M.: Lower Bounds for Generalized Quantum Finite Automata. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 373–384. Springer, Heidelberg (2008)
15. Moore, C., Crutchfield, J.P.: Quantum Automata and Quantum Grammars. Theoretical Computer Science 237(1-2), 275–306 (2000)

16. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
17. Pin, J.E.: Varieties of Formal Languages, North Oxford, London and Plenum, New-York (1986)
18. Pin, J.E.: BG = PG, a Success Story. In: Fountain, J. (ed.) NATO Advanced Study Institute. Semigroups, Formal Languages and Groups, pp. 33–47. Kluwer Academic Publishers, Boston (1995)
19. Pin, J.E., Straubing, H., Thérien, D.: Small Varieties of Finite Semigroups and Extensions. J. Austral. Math. Soc (Series A) 37, 269–281 (1984)
20. Rabin, M.O.: Probabilistic Automata. Information and Control 6(3), 230–245 (1963)
21. Thierrin, G.: Permutation Automata. Mathematical Systems Theory 2(1), 83–90 (1968)
22. Zhang, F.: Matrix Theory: Basic Results and Techniques. Springer, Heidelberg (1999)

A Universally Defined Undecidable Unimodal Logic*

Edith Hemaspaandra[1] and Henning Schnoor[2]

[1] Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623, U.S.A
eh@cs.rit.edu
[2] Institute for Computer Science, Christian-Albrechts-Universität zu Kiel, Kiel, Germany
schnoor@ti.informatik.uni-kiel.de

Abstract. Modal logics are widely used in computer science. The complexity of their satisfiability problems has been an active field of research since the 1970s. We prove that even very "simple" modal logics can be undecidable: We show that there is an undecidable unimodal logic that can be obtained by restricting the allowed models with an equality-free first-order formula in which only universal quantifiers appear.

Keywords: modal logic, satisfiability problems, decidability.

1 Introduction

Modal logics are widely used in many areas of computer science. See, for example, [LR86, HMT88, FHJ02, Moo03, BG04, ABvdT10]. The complexity of modal satisfiability problems has been an active field of research since Ladner's work in the 1970s [Lad77]. Early work focused on the complexity of single logics, but more recent work has focused on handling the computability and complexity for an infinite number of logics, see, e.g., [Fin85, DK98, AvBN98, HS08, HSS10].

Our ultimate goal is to classify the complexity of "all" modal logics. We are particularly interested in elementary modal logics, i.e., modal logics whose models are defined by a first-order formula, since modal logics used in practice are often defined in this way. An important first step towards classifying the complexity of all elementary modal logics is to determine which of them are decidable. The goal of this paper is to find a "simplest" undecidable elementary modal logic.

A particularly interesting simple class of elementary modal logics are the universal elementary modal logics, in which the class of models is defined by a universal first-order formula. Not only do many common modal logics belong to this class, it is also a class that is on the borderline of being decidable. In

* Supported in part by NSF grant IIS-0713061, the DAAD postdoc program, and by a Friedrich Wilhelm Bessel Research Award. Work done in part while Henning Schnoor was at the Rochester Institute of Technology.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 364–375, 2011.

particular, it is known that many universal Horn elementary unimodal logics (i.e., logics whose frame classes can be described by a universal Horn formula and that use a single modal operator) are in PSPACE, and it is conjectured that all these logics are decidable [HS08]. On the other hand, it is known that there is a universal elementary unimodal logic such that *global* satisfiability, i.e., satisfiability in all worlds in a model, is undecidable [Hem96].

To show undecidability for modal logics, we typically make models look like a grid, and need a way to access all the worlds in the grid as well as the two direct successors of a world. An early example of this is that 2-dimensional temporal logic on $\mathbb{N} \times \mathbb{N}$ with modal operators $\bigcirc_u$ and $\bigcirc_r$ that access the direct "up" and "right" successors of a world, and $\Box_u$ and $\Box_r$ that access all worlds that are above or to the right of the current world is undecidable [Har83]. The undecidability also holds if $\Box_u$ and $\Box_r$ are the only modal operators [Spa90]. Other examples of undecidable modal logics include logics of knowledge and time [HV89, LR86, Spa90], logics allowing identification of worlds [BS93, GG93], and most products of transitive modal logics [GKWZ05]. All these logics are multimodal and certainly not universal. As will be explained in Section 5.1, universal first-order formulas are not enough to enforce a grid-like structure. Still, as mentioned above, there exists a first-order universal formula such that *global* satisfiability for unimodal logics is undecidable [Hem96]. Section 4 will explain the idea behind the construction from [Hem96] and why this construction cannot be used to show that satisfiability for a universal elementary modal logic is undecidable.

However, using a more complicated construction, we show that there exists an undecidable universal elementary unimodal logic. This result holds even if we are not allowed to use the equality predicate in the first-order universal formula describing the models. And so we have indeed found a "simplest" undecidable modal logic in the sense that the syntactical fragment of first-order logic used to define the allowed frames is very limited. The complete proofs can be found in the technical report [HS11].

2 Preliminaries

(Uni)modal logic extends propositional logic with a unary operator $\Box$ (a dual operator $\Diamond$ abbreviates $\neg\Box\neg$). The *modal depth* $\mathsf{md}(\varphi)$ of a formula φ is the maximal nesting degree of the $\Box$-operator in φ. A *frame* F is a directed graph (W, R), where the vertices are called "worlds." A *model* $M = (W, R, \pi)$ consists of a frame (W, R) and a function π assigning to each variable the set of worlds in which x is true. We say that M is *based* on (W, R). For a class of frames $\mathcal{F}$, a model is an $\mathcal{F}$-model if it is based on a frame in $\mathcal{F}$. We often write $w \in M$ instead of $w \in W$. For a world $w \in M$, we define when a formula ϕ is *satisfied* at (or *holds* at) w in M (written $M, w \models \phi$). If ϕ is a variable x, then $M, w \models \phi$ if and only if $w \in \pi(x)$. Boolean operators are treated as usual. For the modal operator, $M, w \models \Box\phi$ iff $M, w' \models \phi$ for all worlds w' with $(w, w') \in R$. A modal formula is *globally* satisfied in M if it holds at every world of M.

First-order formulas are a standard way to define classes of frames. The *frame language* is the first-order language containing (in addition to Boolean operators) the binary relation R, interpreted as the edge relation in the graph, and the equality relation $=$. The semantics are defined in the obvious way. For example, the formula $\hat{\varphi}_{\text{trans}} := \forall xyz(xRy \wedge yRz \rightarrow xRz)$ is satisfied exactly by the transitive frames. We use $\hat{}$ to denote first-order formulas, e.g., $\hat{\psi}$ is a first-order formula, while ϕ is a modal formula. A frame is a $\hat{\psi}$-frame if it satisfies $\hat{\psi}$, a model M is a $\hat{\psi}$-model (we write $M \models \hat{\psi}$) if M is based on a $\hat{\psi}$-frame. The *basic frame language* is the frame language without equality. Following [HS08], $\mathsf{K}(\hat{\psi})$ is the logic in which a modal formula ϕ is satisfiable iff there is a $\hat{\psi}$-model M and a world $w \in M$ such that $M, w \models \phi$. Such logics are called elementary modal logics. For a first-order formula $\hat{\psi}$, we study the following problem:

Problem: $\mathsf{K}(\hat{\psi})$-SAT
Input: A modal formula ϕ
Question: Is there a $\hat{\psi}$-model M and a world $w \in M$ with $M, w \models \phi$?

For example, $\mathsf{K}(\hat{\varphi}_{\text{trans}})$-SAT is the satisfiability problem for the logic K4, i.e., the problem of deciding if a modal formula can be satisfied in a model based on a transitive frame.

3 Main Result

We show that modal satisfiability is undecidable already for a formula over the basic frame language in which every appearing variable is universally quantified:

Theorem 3.1. *There exists a universal first-order formula $\hat{\varphi}^{final}$ over the basic frame language such that $\mathsf{K}(\hat{\varphi}^{final})$-SAT is* coRE*-complete.*

Since satisfiability for elementary modal logics can be phrased as the negation of a first-order implication (see, e.g., [BdRV01, Lemma 6.32]), the problem is in coRE. It thus remains to construct $\hat{\varphi}^{\text{final}}$ such that $\mathsf{K}(\hat{\varphi}^{\text{final}})$-SAT is coRE-hard. As a direct corollary of the above theorem, we also obtain the following result: The *uniform* satisfiability problem, where both the modal and the first-order formula are given in the input, is coRE-complete.

4 Relation with Previous Work

The result from the literature closest to ours is [Hem96, Theorem 3.2], which shows coRE-hardness of *global* modal satisfiability for the class of frames defined by the following first-order universal formula (we omit quantification from first-order formulas; in this paper first-order variables are always universally quantified):

$$((xRy_1 \wedge xRy_2 \wedge xRy_3) \rightarrow ((y_1 = y_2) \vee (y_1 = y_3) \vee (y_2 = y_3))) \quad \wedge$$
$$\left(\bigwedge_{1 \leq i \leq 4} (xRy_i \wedge y_i Rz_i) \rightarrow \bigvee_{1 \leq i < j \leq 4} (z_i = z_j) \right).$$

We strengthen the above result in several ways: We prove undecidability already for the (local) satisfiability problem, i.e., given a modal formula φ, decide whether there is a model and *some* world in it that satisfies φ. Satisfiability is often much easier than global satisfiability: In the above example from [Hem96], satisfiability is in NP, while global satisfiability is undecidable. Intuitively, proving undecidability of the global satisfiability problem by encoding the grid is easier than undecidability for the local problem due to the following: A key property of the grid is that every world in a model has exactly two successors reachable in one step, and three worlds reachable in two steps. Clearly, enforcing the existence of successors is impossible using only universally quantified formulas. With a modal formula however, requiring two direct successors can be enforced easily using $\Diamond u \wedge \Diamond \overline{u}$. Requiring this formula to *globally* hold in a model hence enforces the grid structure relatively easily. In the satisfiability problem however, there is no way to require this formula to hold globally. This is the main reason why the global satisfiability problem allows us to express "positive" conditions (e.g., existence of two successors) more easily than the satisfiability problem.

In our proof, we employ some techniques from [Hem96]. In particular, our first step establishes undecidability of the global satisfiability problem over a class of frames similar to the one defined by the above formula. However, this step does much more than simply reproving the result from [Hem96]: The class of frames we construct here is considerably less restrictive than the one defined above. One reason for this is that with the more restricted *basic frame language* we use, we cannot restrict our frames as strongly as with the above formula from [Hem96]. More importantly however, we construct a class of frames that is tailor-made for being able to prove our main result—hardness of satisfiability—later. A key issue here is that of reflexivity: In contrast to the grid model, our class includes reflexive frames—in fact, the reflexive frames are those which later allow us to reduce the global satisfiability problem to the satisfiability problem. In particular, directly reducing global satisfiability for the class of frames defined by the formula above to satisfiability for a class of frames defined by a universal first-order formula over the basic frame language does not appear to be easier than our approach.

Technically, in addition to our first-order formulas being more complex than those used in [Hem96], a main difference with [Hem96] is the use of what we call the *abstraction* of a model: Essentially, our first-order formulas enforce the relevant conditions of the grid not in a model M itself, but in a model ${}^{M}/_{\sim}$ obtained from M by compressing cliques of worlds into a single world. Similar techniques were used in [Spa90, Spa93, RZ01, GKWZ05].

5 Proof of the Main Result

5.1 Global Satisfiability in the Grid Model

We prove coRE-hardness by a reduction from the global grid satisfiability problem: The *grid frame* has world set $\mathbb{N} \times \mathbb{N}$, and the accessibility relation

$R = \{((i,j),(i+1,j)),((i,j),(i+1,j)) \mid i,j \in \mathbb{N}\}$. A *grid model* is a model based on the grid frame. The *global grid satisfiability problem* is the following:

Problem: Global-Grid-Sat
Input: A modal formula ψ
Question: Is there a grid-model M that globally satisfies ψ?

In [Hem96], the following theorem was proven:

Theorem 5.1. Global-Grid-Sat *is* coRE*-hard, even restricted to inputs* ψ *with* $\mathsf{md}(\psi) \leq 1$.

Our proof forces models to be "essentially grid-like." However, universal first-order formulas are clearly not expressive enough to accomplish this: It is easy to see that if F' is a subframe of a frame F (i.e., an induced subgraph), then F' satisfies every universal first-order formula that F satisfies. In fact, universal first-order formulas *with equality* can express exactly the "forbidden subgraph properties." Hence with equality, we can express grid features as "each world has at most two direct successors," using the formula $(xRy_1 \wedge xRy_2 \wedge xRy_3) \rightarrow ((y_1 = y_2) \vee (y_1 = y_3) \vee (y_2 = y_3))$. As mentioned in Section 4, this formula was used in the proof of [Hem96, Theorem 3.2].

As mentioned above, requiring "positive" conditions, e.g., that every world has direct successors with certain properties, is reasonably easy in the global satisfiability setting, but considerably more difficult in the local setting. Further, even the above property of having at most two distinct successors cannot be expressed by a universal formula in the *basic* (equality-free) frame language: It is easy to see that no such formula can distinguish the singleton with two successors from the singleton with three successors. Hence, to simulate the grid, we exploit the interplay between the first-order and the modal aspect of the satisfiability problem. The main ingredients to our proof are the following:

1. To circumvent having to use the equality predicate, we use a first-order formula similar to the one above, except that instead of requiring $y_i = y_j$, we demand that there is a symmetric edge between these worlds with the first-order formula $(y_iRy_j) \wedge (y_jRy_i)$. We then ensure that the relation $x \sim y$ ("there is a symmetric edge between x and y") is an equivalence relation. We then "abstract" modal models to $\sim$-equivalence classes, which allows us to "simulate" equality in the basic frame language. On the abstraction we can then express all required "forbidden subgraph" properties, since as mentioned above, these can be expressed using universal first-order formulas with equality.
2. To ensure that this abstraction is sound, we use modal formulas to ensure that the relation respects propositional assignments: For the relevant variables, $\sim$-equivalent worlds will have the same valuation. This allows us to regard equivalence classes as single worlds in the abstracted model.
3. For the grid-structure, it remains to express the "positive" properties of the grid, for example that every world indeed has two distinct successors. While existence of successor worlds can be required with the modal operator $\Diamond$

(which in this step of the proof we can use "globally" as we still only deal with the global satisfiability problem), we need to ensure that there exist $\sim$-inequivalent successors—i.e., successors still present in the abstraction.

4. The main issue is to express the global nature of Global-Grid-Sat: The formula ψ is required to globally hold in the grid, whereas modal satisfiability is an existential property. To solve this, we force the existence of a "universal" world w_u that is connected to every other world. In w_u, requiring ψ to hold globally can be expressed with the (local) $\Box$ operator. This part of the proof crucially relies on features of the class of models considered in the first part.

5.2 Expressing the Grid: Universal First-Order Aspects

Following the strategy outlined above, we start by defining a set of universal first-order formulas that force the "abstraction" of a model to obey some essential grid properties. As mentioned above, we use $x \sim y$ to express that there is a symmetric edge from x to y (the frame will always be clear from context). We also use $x \sim y$ as an abbreviation for $(xRy \wedge yRx)$ in formulas.

Definition 5.2. *Let $\hat{\varphi}_{1\text{-}step}$ be the universal first-order formula*

$$(xRy_1 \wedge xRy_2 \wedge xRy_3) \rightarrow \left(\bigvee_{1 \leq i \leq 3} (x \sim y_i) \vee \bigvee_{1 \leq i < j \leq 3} (y_i \sim y_j) \right).$$

This expresses that in the grid, each world has only two distinct successors. When reading $\sim$ as equality, the formula exactly captures this requirement. Similarly, we enforce another important grid feature: While each world has two distinct successors, say y_1 and y_2, and each of these again has two distinct successors, say z_1^1, z_2^1, z_1^2 and z_2^2, each world can reach only three distinct worlds in two steps. Hence, two of the z_j^i must coincide. This is expressed as follows:

Definition 5.3. *Let $\hat{\varphi}_{2\text{-}step}$ be the universal first-order formula*

$$\left(\bigwedge_{1 \leq i \leq 4} (xRy_i \wedge y_iRz_i) \right) \rightarrow \left(\bigvee_{1 \leq i \leq 4} ((x \sim y_i) \vee (y_i \sim z_i)) \vee \bigvee_{1 \leq i < j \leq 4} (z_i \sim z_j) \right).$$

When reading $\sim$ as equality, this formula states that if z_1, z_2, z_3, z_4 are worlds reachable from x via intermediate worlds y_i (different from both x and z_i), then two of the z_i coincide. The formulas introduced up to now closely mirror the formulas in [Hem96]. The major differences and additions follow now.

Definition 5.4. *Let $\hat{\varphi}_{eq}^{\sim}$ be the first-order formula*

$$\big((x \sim y) \wedge yRz\big) \rightarrow xRz\big) \wedge \big((x \sim y) \wedge zRy\big) \rightarrow zRx\big).$$

This formula ensures that $\sim$-equivalent worlds have the exact same in- and outgoing edges, hence it forces $\sim$ to be an equivalence relation in reflexive models. The conjunction of the formulas above combines the first-order aspects of the grid that we can force with universal formulas over the basic frame language:

Definition 5.5. *Let $\hat{\varphi}^{grid} := \hat{\varphi}_{1\text{-}step} \wedge \hat{\varphi}_{2\text{-}step} \wedge \hat{\varphi}_{eq}^{\sim}$.*

5.3 Properties of Abstracted Frames

We now formally define abstractions of frames, which as mentioned before are obtained by compressing $\sim$-equivalence classes into a single world.

Definition 5.6. *For a reflexive frame $F = (W, R)$, where $F \models \hat{\varphi}^{\sim}_{eq}$, we define the* abstraction of F*, denoted with ${}^{F}/_{\sim}$, to be the frame $({}^{W}/_{\sim}, {}^{R}/_{\sim})$, where ${}^{W}/_{\sim}$ is the set of $\sim$-equivalence classes of W, and $[w]\,{}^{R}/_{\sim}\,[w']$ if and only if wRw'.*

The relation ${}^{R}/_{\sim}$ above is well-defined, since if $w \sim w'$ and $\hat{w} \sim \hat{w}'$, then $wR\hat{w}$ implies $w'R\hat{w}$, and this implies $w'R\hat{w}'$ (by $\hat{\varphi}^{\sim}_{\text{eq}}$). The following lemma summarizes the features of abstractions of frames satisfying $\hat{\varphi}^{\text{grid}}$.

Lemma 5.7. *If F is a reflexive $\hat{\varphi}^{grid}$-frame, then ${}^{F}/_{\sim}$ satisfies the following:*

1. *${}^{F}/_{\sim}$ is reflexive,*
2. *each world $[w]$ in ${}^{F}/_{\sim}$ has at most two direct successors different from $[w]$,*
3. *for each world $[w]$ in ${}^{F}/_{\sim}$, there are at most three worlds that can be reached on a path from $[w]$ of length two that does not use any reflexive edge.*

There are two main differences between the grid frame and the class of abstractions of $\hat{\varphi}^{\text{grid}}$-frames: First, worlds in abstractions of $\hat{\varphi}^{\text{grid}}$-frames corresponding to more than one original world are always reflexive. Second, the class of $\hat{\varphi}^{\text{grid}}$-frames (and thus the class of its abstractions) is subframe-closed. Essentially, abstractions of $\hat{\varphi}^{\text{grid}}$-frames can be seen as subframes of the reflexive closure of the grid frame. Note that reflexivity does not help to prove undecidability for global satisfiability, but will make it easier to later move to the satisfiability problem. As mentioned before, we cannot hope to enforce other grid properties with universal first-order formulas. For the remaining properties, we therefore use modal formulas and propositional variables.

5.4 Expressing the Grid: Modal Aspects

In the formula $\hat{\varphi}^{\text{grid}}$, we have expressed the universal first-order properties of the grid frame. The second part of our abstraction process is the valuation of the propositional variables. We use the interplay between modal formulas and the frame properties ensured in the previous section to address the following issues:

- We ensure that the abstraction is "modally sound." The main issue is forcing the relevant variables to have the same value in $\sim$-equivalent worlds. This gives well-defined truth values for the abstraction, and ensures that truth of modal formulas is invariant under abstraction.
- The abstractions constructed in the previous section are necessarily reflexive. To simulate the (non-reflexive) grid, we replace the $\Box\psi$-operator with one that only requires ψ to be true in the successors $w' \neq w$ of a world w.
- We have to enforce the "positive" properties of the grid frame, i.e., that every world in fact does have two distinct successors.

Our abstraction does not take into account all variables, but only a subset denoted with P—this will contain all variables appearing in the input formula ψ for the Global-Grid-Sat problem that we will reduce from.

Definition 5.8. *Let P be a set of propositional variables, and let $M = (W, R, \pi)$ be a reflexive $\hat{\varphi}^{\sim}_{eq}$-model. We define the model ${}^{M}/_{\sim}$ as*

$$ {}^{M}/_{\sim} := ({}^{W}/_{\sim}, {}^{R}/_{\sim}, {}^{\pi}/_{\sim}) , $$

where the assignment ${}^{\pi}/_{\sim}$ makes a variable $p \in P$ true in an equivalence class $[w]$ if and only if p is true in all elements of $[w]$, and lets all propositional variables not in P be false everywhere. We call ${}^{M}/_{\sim}$ the abstraction *of M.*

We do not make P explicit, it will always be clear from the context. Later, we only consider models where the truth value of variables in P does not depend on the representative of a class $[w]$. We say that $\sim$ *respects P in a model M* if $w \sim w'$ implies that each variable $p \in P$ is true in w iff it is true in w'. We often omit M if clear from the context. This property implies that the abstraction is indeed sound, i.e., preserves truth of all modal formulas—at least for reflexive models that also satisfy the first-order formula $\hat{\varphi}^{\sim}_{\text{eq}}$:

Lemma 5.9. *Let P be a set of propositional variables, let M be a reflexive $\hat{\varphi}^{\sim}_{eq}$-model such that $\sim$ respects P in M. Then for all $w \in W$, and all modal formulas ψ with $\text{VAR}(\psi) \subseteq P$, $M, w \models \psi$ if and only if ${}^{M}/_{\sim}, [w] \models \psi$.*

We therefore obtain the following: If M is a reflexive modal model that satisfies $\hat{\varphi}^{\text{grid}}$ and in which $\sim$ respects P, then its abstraction is a subframe of the reflexive closure of the grid that satisfies the same formulas ψ as M does (for formulas ψ with $\text{VAR}(\psi) \subseteq P$). Therefore, if we can enforce that $\sim$ respects P, we can enforce the abstractions of our models to exhibit the "forbidden subgraph"-features of the grid, without changing the set of satisfied modal formulas. To enforce that $\sim$ respects P, we define the following:

Definition 5.10. *For a finite set P of propositional variables, define ψ^{P}_{resp} as*

$$\begin{aligned} \psi^{P}_{resp} = \textstyle\bigwedge_{d=0}^{7} \bigwedge_{p \in P} ((d_8 = d) \rightarrow \\ \Box((d_8 = d) \vee (d_8 = (d+2) \bmod 8) \vee (d_8 = (d+3) \bmod 8)) \\ \wedge\ (p \rightarrow \Box((d_8 = d) \rightarrow p)) \\ \wedge\ (\overline{p} \rightarrow \Box((d_8 = d) \rightarrow \overline{p}))) \end{aligned}$$

Here d_8 is a variable over the natural numbers $0, \ldots, 7$. This obviously can be represented by three propositional variables d^a_8, d^b_8, and d^c_8 as follows:

$$d_8 = 0 \leftrightarrow (\overline{d^a_8} \wedge \overline{d^b_8} \wedge \overline{d^c_8}),\ d_8 = 1 \leftrightarrow (\overline{d^a_8} \wedge \overline{d^b_8} \wedge d^c_8), \ldots,\ d_8 = 7 \leftrightarrow (d^a_8 \wedge d^b_8 \wedge d^c_8).$$

For a world $w \in M$, with $d_8(w)$ we denote the unique value $d \in \{0, \ldots, 7\}$ such that $M, w \models (d_8 = d)$. For readability, we write $d_8 = d$ for $d_8 = d \bmod 8$. We now prove that the formula ψ^{P}_{resp} works as intended, if it is *globally* satisfied:

Lemma 5.11. *Let P be a finite set of propositional variables, let M be a reflexive $\hat{\varphi}^{grid}$-model that globally satisfies ψ^P_{resp}. Then $\sim$ respects $P \cup \{d_8\}$ in M.*

From the above and Lemma 5.9 we obtain the following:

Corollary 5.12. *Let M be a reflexive $\hat{\varphi}^{\sim}_{eq}$-model such that M globally satisfies ψ^P_{resp}. Then for all formulas ψ with $\mathrm{VAR}(\psi) \subseteq P \cup \{d_8\}$, and all worlds $w \in M$, $M, w \models \psi$ if and only if ${}^M/_{\sim}, [w] \models \psi$. In particular, ${}^M/_{\sim}$ globally satisfies ψ^P_{resp}.*

Using d_8 allows reasoning about direct successors of a world w that are not $\sim$-equivalent to w. We call these *non-symmetric successors of* w. Similarly, a *non-reflexive successor* of w is a direct successor $w' \neq w$ of w. We now force $\sim$-inequivalent worlds connected with a direct edge to have different d_8-values:

Definition 5.13. $\psi^{succ} = \bigwedge_{d=0}^{7} ((d_8 = d) \to \Diamond(d_8 = d+2) \wedge \Diamond(d_8 = d+3))$.

This expresses that a world w with $d_8(w) = d$ has direct successors with $d_8 = d+2$ and $d_8 = d+3$. We later identify "+2"/"+3"-successors with "upper"/"right" neighbors in the grid. If additionally the model globally satisfies ψ^P_{resp}, then from neither of these successors, the world w is reachable in one step: From ψ^P_{resp}, it follows that all direct successors of the two ones whose existence is forced by ψ^{succ} have d_8-values from $\{d+2, d+4, d+5, d+3, d+5, d+6\}$, none of which applies to w itself. More generally, every world $w' \neq w$ reachable from w with at most two steps has a different d_8-value than w. In addition, the successors with d_8-values of $d+2$ and $d+3$ cannot be connected with a direct edge in models satisfying ψ^P_{resp}. Hence, each world has two successors such that all of the three involved worlds give rise to different equivalence classes in the abstraction.

It follows from the above that in $\hat{\varphi}^{\mathrm{grid}}$-models globally satisfying the formula ψ^P_{resp}, the formula $\bigwedge_{d=0}^{7}((d_8 = d) \to \Box((d_8 \neq d) \to \psi))$ is true in a world w if and only if ψ is true in all non-symmetric, non-reflexive successors of w.

5.5 coRE-hardness of Global Satisfiability

We now show coRE-hardness for the global satisfiability problem on reflexive frames that satisfy $\hat{\varphi}^{\mathrm{grid}}$. In itself, this result is not stronger than what was already established in [Hem96], except for the fact that our formula only uses the basic frame language, i.e., does not use equality. However, the real benefit of this result will become apparent in the next section: The class that we define here allows us to easily reduce global satisfiability to satisfiability.

In Lemma 5.11, we have seen that if we can ensure that the formula ψ^P_{resp} is *globally* satisfied in a reflexive $\hat{\varphi}^{\mathrm{grid}}$-model, then $\sim$ respects P and in this case Lemma 5.9 tells us that our abstraction is sound, i.e., preserves truth values of modal formulas. Since the formulas ψ^P_{resp} and ψ^{succ} allow us to ensure that $\sim$ respects P and that every world has two distinct successors as in the grid frame, we therefore can use the construction from the previous section to prove coRE-hardness in the case that we are able to enforce $\psi^P_{\mathrm{resp}} \wedge \psi^{\mathrm{succ}}$ globally.

Recall that Global-Grid-Sat remains coRE-hard when the input is restricted to formulas ψ with $\mathsf{md}(\psi) \leq 1$. We therefore only consider such inputs for Global-Grid-Sat from now on, and define our reduction as follows:

Definition 5.14. *Let ψ be an input for* Global-Grid-Sat *with* $\mathsf{md}(\psi) \leq 1$. *Let* $P = \mathrm{VAR}(\psi)$, *and let $g(\psi)$ be defined inductively as follows:*

- *If ψ is a variable p, then $g(\psi) = p$,*
- $g(\neg\psi) = \neg g(\psi)$,
- $g(\psi \wedge \xi) = g(\psi) \wedge g(\xi)$,
- $g(\Box\psi) = \bigwedge_{d=0}^{7}((d_8 = d) \to \Box((d_8 \neq d) \to g(\psi)))$.

The reduction f is now defined as $f(\psi) = g(\psi) \wedge \psi^P_{resp} \wedge \psi^{succ}$.

The only non-obvious part is the treatment of the $\Box$-operator. As argued above, this choice of $g(\Box\psi)$ requires ψ to be true in all non-reflexive, non-symmetric successor worlds of the current one. This is crucial when we consider abstractions: The non-symmetric successors of a world w in a model M directly correspond to the non-reflexive successors of the class $[w]$ in the model $M/\!\sim$.

Theorem 5.15. *Let ψ be an instance of* Global-Grid-Sat *with* $\mathsf{md}(\psi) \leq 1$. *Then ψ is a positive instance of* Global-Grid-Sat *if and only if $f(\psi)$ is globally satisfiable on a reflexive $\hat{\varphi}^{grid}$-model.*

Clearly, one can force the models to be reflexive with the universal clause xRx. However, for later proving undecidability for satisfiability instead of global satisfiability, it is crucial to leave open the possibility of non-reflexive worlds.

5.6 Removing Globalness

Above, we showed hardness for global satisfiability for reflexive $\hat{\varphi}^{\mathrm{grid}}$ frames. To obtain the result for satisfiability, we now express this global quantification with only the first-order frame language and the modal language.

To do so, we force the existence of a "universal" world w_u that has an outgoing edge to every other world. Since we cannot express this "existential" property with a universal first-order formula, we proceed as follows: We require that from every irreflexive world w_u, there is an edge to every world w with any incoming edge at all. This ensures that an irreflexive world w_u is universal with respect to the submodel rooted at w_u. Additionally, we require that any world with an incoming edge is reflexive. Hence if there is an irreflexive world w_u, then every world reachable from w_u in any number of steps is connected to w_u directly, and every such world is reflexive. This is achieved with the following formula:

Definition 5.16. $\hat{\varphi}^{univ} = (xRy \to yRy) \wedge (\neg(w_uRw_u) \to (xRy \to w_uRy))$.

The existence of an irreflexive world w_u can easily be enforced with the modal formula $u \wedge \Box\overline{u}$, where u is a new variable. We then enforce the formula $\hat{\varphi}^{\mathrm{grid}}$ constructed in the previous section only on reflexive worlds, and can thus identify

the "reflexive part" of a model with a model of the type as considered in the previous section. In particular, we know that global satisfiability of a formula of the form $f(\psi)$ on the "reflexive part" of our models is coRE-hard, where f is the function used in the reduction from Theorem 5.15. We then use the universal world w_u to express the global satisfiability problem with a single $\Box$-operator.

We therefore obtain the following theorem—the formula $\hat{\varphi}^{\text{final}}$ mentioned in the theorem is obtained from $\hat{\varphi}^{\text{grid}}$ with the above-mentioned elements that handle the "global" aspect of Global-Grid-Sat.

Theorem 5.17. *There exists a universal first-order formula $\hat{\varphi}^{final}$ over the basic frame language such that $\mathsf{K}(\hat{\varphi}^{final})$-$\mathsf{SAT}$ is* coRE*-hard.*

6 Conclusion

We have constructed an undecidable elementary unimodal logic that is "simple" with respect to the logic needed to define it. An interesting open question, suggested by an anonymous reviewer, is whether there exists an undecidable unimodal transitive subframe logic. Due to [Fin85], such a logic is not finitely axiomatizable.

Acknowledgment. We thank the anonymous reviewers for very helpful comments on the paper, including the above-mentioned open question.

References

[ABvdT10] Aucher, G., Boella, G., van der Torre, L.: Privacy policies with modal logic: The dynamic turn. In: Governatori, G., Sartor, G. (eds.) DEON 2010. LNCS, vol. 6181, pp. 196–213. Springer, Heidelberg (2010)

[AvBN98] Andréka, H., van Benthem, J., Németi, I.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic 27, 217–274 (1998)

[BdRV01] Blackburn, P., de Rijke, M., Venema, Y.: Modal logic. Cambridge University Press, New York (2001)

[BG04] Bennett, B., Galton, A.: A unifying semantics for time and events. Artificial Intelligence 153(1-2), 13–48 (2004)

[BS93] Blackburn, P., Spaan, E.: A modal perspective on the computational complexity of attribute value grammar. Journal of Logic, Language, and Information 2(2), 129–169 (1993)

[DK98] Demri, S., Konikowska, B.: Relative similarity logics are decidable: Reduction to FO^2 with equality. In: Dix, J., Fariñas del Cerro, L., Furbach, U. (eds.) JELIA 1998. LNCS (LNAI), vol. 1489, pp. 279–293. Springer, Heidelberg (1998)

[FHJ02] Frendrup, U., Hüttel, H., Jensen, J.: Modal logics for cryptographic processes. In: Proceedings of EXPRESS 2002 (2002)

[Fin85] Fine, K.: Logics containing K4. Part II. J. Symb. Log. 50(3), 619–651 (1985)

[GG93] Garvon, G., Goranko, V.: Modal logic with names. Journal of Philosophical Logic 22(6), 607–636 (1993)

[GKWZ05] Gabelaia, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Products of 'transitive' modal logics. J. Symb. Log. 70(3), 993–1021 (2005)

[Har83] Harel, D.: Recurring dominoes: Making the highly undecidable highly understandable (preliminary report). In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 177–194. Springer, Heidelberg (1983)

[Hem96] Hemaspaandra, E.: The price of universality. Notre Dame Journal of Formal Logic 37(2), 174–203 (1996)

[HMT88] Halpern, J., Moses, Y., Tuttle, M.: A knowledge-based analysis of zero knowledge. In: STOC 1988: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 132–147. ACM Press, New York (1988)

[HS08] Hemaspaandra, E., Schnoor, H.: On the complexity of elementary modal logics. In: Albers, S., Weil, P. (eds.) STACS, Germany, LIPIcs, vol. 1, pp. 349–360. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2008)

[HS11] Hemaspaandra, E., Schnoor, H.: A simplest undecidable modal logic. CoRR, abs/1105.0845 (2011)

[HSS10] Hemaspaandra, E., Schnoor, H., Schnoor, I.: Generalized modal satisfiability. J. Comput. Syst. Sci. 76(7), 561–578 (2010)

[HV89] Halpern, J., Vardi, M.: The complexity of reasoning about knowledge and time. I. Lower bounds. J. Comput. Syst. Sci. 38(1), 195–237 (1989)

[Lad77] Ladner, R.: The computational complexity of provability in systems of modal propositional logic. SIAM Journal of Computation 6(3), 467–480 (1977)

[LR86] Ladner, R., Reif, J.: The logic of distributed protocols. In: Halpern, J.Y. (ed.) TARK, pp. 207–222. Morgan Kaufmann, San Francisco (1986)

[Moo03] Moody, J.: Modal logic as a basis for distributed computation. Technical Report CMU-CS-03-194, School of Computer Science, Carnegie Mellon University (October 2003)

[RZ01] Reynolds, M., Zakharyaschev, M.: On the products of linear modal logics. J. Log. Comput. 11(6), 909–931 (2001)

[Spa90] Spaan, E.: Nexttime is not necessary. In: Parikh, R. (ed.) TARK, pp. 241–256. Morgan Kaufmann, San Francisco (1990)

[Spa93] Spaan, E.: Complexity of Modal Logics. PhD thesis, Department of Mathematics and Computer Science, University of Amsterdam (1993)

On the Approximability of Minimum Topic Connected Overlay and Its Special Instances*

Jun Hosoda[1], Juraj Hromkovič[2], Taisuke Izumi[1], Hirotaka Ono[3], Monika Steinová[2], and Koichi Wada[1]

[1] Graduate School of Engineering, Nagoya Institute of Technology, Japan
{t-izumi,wada}@nitech.ac.jp
[2] Department of Computer Science, ETH Zurich, Switzerland
{juraj.hromkovic,monika.steinova}@inf.ethz.ch
[3] Department of Economic Engineering, Kyushu University, Japan
hirotaka@en.kyushu-u.ac.jp

Abstract. In the context of designing a scalable overlay network to support decentralized topic-based pub/sub communication, the Minimum Topic-Connected Overlay problem (Min-TCO in short) has been investigated: Given a set of t topics, collection of n users together with the lists of topics they are interested in, the aim is to connect these users to a network by a minimum number of edges such that every graph induced by users interested in a common topic is connected. It is known that Min-TCO is $\mathcal{NP}$-hard and approximable within $O(\log t)$ in polynomial time.

In this paper, we further investigate the problem and some of its special instances. We give various hardness results for instances where the number of users interested in a common topic is constant, and also for the instances where the number of topics in which an user is interested in is bounded by a constant. Furthermore, we close the gap of hardness of Min-TCO by showing its $\mathcal{LOGAPX}$-completeness. We also present a few polynomial-time algorithms for very restricted instances of Min-TCO.

1 Introduction

Recently, the spreading of social networks and other services based on sharing content allowed the development of many-to-many communication, often supported by these services. Publishers publish information through a logical channel that is consumed by subscribed users. This environment is often modeled by publish/subscribe (pub/sub) systems that can be classified into two categories. When the channels are associated with a collection of attributes and the messages are delivered to a subscriber only if their attributes match user-defined constraints we speak about *content-based* pub/sub systems. Each channel in

* This research is partly supported by the Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research, 21500013, 21680001, 22650004, 22700010, 23104511, 23310104, Foundation for the Fusion of Science Technology (FOST) and INAMORI FOUNDATION. The research is also partially funded by SNF grant 200021-132510/1.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 376–387, 2011.

topic-based pub/sub systems is associated with a single topic and the messages are distributed to the users via channels by his/her topic selection. There are numerous implementations of pub/sub systems, for details see [1, 4–6, 18, 19, 21].

In our paper, we focus on the topic-based peer-to-peer pub/sub systems. In such a system, subscribers interested in a particular topic have to be connected without the use of intermediate agents (such as servers). Many aspects of such a system can be studied (see [8, 16]). Minimizing the diameter of the overlay network can minimize the overall time in which a message is distributed to all the subscribers. When minimizing the (average) degree of nodes in the network, the subscribers need to maintain a smaller number of connections. In this paper, we study the minimization of the overall number of connections in the system. A small number of connections may be necessary due to maintenance requirements or may be helpful since thus information aggregated into single messages can be broadcasted to the network and thus amortize the head count of otherwise small messages.

We study here the hardness of *Minimum Topic-Connected Overlay* (Min-TCO) that was studied in different scenarios in [2, 8, 14, 15]. In Min-TCO we are given a collection of users, a set of topics and a user-interest assignment, we want to connect users in an overlay network G such that all users interested in a common topic are connected and the overall number of edges in G is minimal. The hardness of the problem was studied in [8] and [2]. In [8], the inapproximability by a constant was proved and a logarithmic-factor approximation algorithm was presented. In [2], lower bound on the approximability of Min-TCO was improved to $\Omega(\log(n))$, where n is the number of users.

Moreover, we focus here on the special instances of Min-TCO. We study the case where for each topic there is a constant number d of users interested in it. In [20], an $O(d^2)$-approximation algorithm for such instances is presented. We also consider the case where the number of topics in which any user is interested is bounded by a constant. We believe that such restrictions on the instances have wide practical applications such as when a publisher has a limited number of slots for users or the user's application limits the number of topics that he can follow.

In the study of the general Min-TCO we extend the method presented in [8] and design an approximation-preserving reduction from instances of the minimum hitting set problem to instances of Min-TCO. This reduction does not only prove a similar lower bound as in [2], but also shows that Min-TCO is $\mathcal{LOGAPX}$-complete and thus, concerning approximability, equivalent with such a famous problem as the minimum set cover. As our reduction is not blowing up the number of users interested in a common topic, the reduction is also an approximation-preserving reduction for the case when the number of users interested in a common topic is limited to a constant. This shows that the restriction of Min-TCO to such special instances is $\mathcal{APX}$-complete.

When the number of topics of Min-TCO is bounded from above by $(1+\varepsilon(n))^{-1} \cdot \log\log n$, for $\varepsilon(n) \geq \frac{3\log\log\log n}{\log\log n - 3\log\log\log n}$ (n is the number of users), we present a polynomial-time algorithm that computes the optimal solution.

In the study of instances, where the number of topics any user is interested in is restricted to a constant, we show that, if this number is at most 6, Min-TCO cannot be approximated within a factor 694/693 in polynomial time unless $\mathcal{P} = \mathcal{NP}$, even if any pair of two users is interested in at most three common topics.

The paper is organized as follows. Section 2 is devoted to the preliminaries and a summary of known results. The hardness results for instances of Min-TCO where we limit the number of users interested in a common topic by a constant are discussed in Section 3. The results related to the instances of Min-TCO, where the number of topics that each user is interested in is constant, are presented in Section 4. Section 5 contains a polynomial-time algorithm that solves Min-TCO when the number of topics is small. The conclusion is provided in Section 6.

2 Preliminaries

In this section, we define basic notions used throughout the paper. We assume that the reader is familiar with notions of graph theory. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Let $V(G)$ and $E(G)$ denote the set of vertices and the set of edges of G, respectively. We denote by $E[S]$ the set of edges of G in the subgraph induced by the vertices from $S \subseteq V$, i. e., $E[S] = \{\{u, v\} \in E \mid u, v \in S\}$. The graph induced by $S \subseteq V$ is denoted as $G[S] = (S, E[S])$. By $N[v]$ we denote the *closed neighborhood* of vertex v, i. e., $N[v] = \{u \in V \mid \{u, v\} \in E\} \cup \{v\}$. A graph G is called *connected*, if for any $u_1, u_\ell \in V$, there exists a path $(u_1, u_2, \ldots, v_\ell)$ such that $\{u_i, u_{i+1}\} \in E$, for all $1 \leq i < \ell$.

Let x be an instance of an optimization problem (in this paper, Min-TCO, Min-VC or Min-HS), then by $|x|$ we denote the size of this instance, i. e., the number of vertices and topics of an instance of Min-TCO and the number of elements and sets of an instance of Min-HS. For a set S, $|S|$ denotes the size of the set, i. e., the number of its elements.

The set of users or nodes of our network is denoted by $U = \{1, 2, \ldots, n\}$. The topics are $T = \{t_1, t_2, \ldots, t_m\}$. Each user subscribes to several topics. This relation is expressed by the user interest function $\mathrm{INT} : U \to 2^T$. The set of all vertices of U interested in a topic t is denoted as U_t. For instance, if user $u \in U$ is interested in topics t_1, t_3 and t_4, then we have $\mathrm{INT}(u) = \{t_1, t_3, t_4\}$ and $u \in U_{t_1}, U_{t_3}, U_{t_4}$. For a given set of users U, a set of topics T and an interest function INT, we say that a graph $G = (U, E)$ with $E \subseteq \{\{u, v\} \mid u, v \in U \wedge u \neq v\}$ is *t-topic-connected* for $t \in T$ if the subgraph $G[U_t]$ is connected. We call the graph *topic-connected* if it is t-topic-connected for all topics $t \in T$. Note that the topic-connectedness property implies that a message published for topic t is transmitted to all users interested in this topic without using non-interested users as intermediators.

The most general problem that we study in this paper is called *Minimum Topic Connected Overlay*:

Problem 1. Min-TCO is the following optimization problem:

Input: A set of users U, a set of topics T and user interest function $\mathrm{INT} : V \to 2^T$.
Feasible solutions: Any set of edges $E \subseteq \{\{u, v\} \mid u, v \in U \wedge u \neq v\}$ such that the graph (U, E) is topic-connected.
Costs: Size of E.
Goal: Minimization.

In this paper we study also some of its special instances. We restrict the number of users that are interested in a common topic, i. e., the size of U_t, to a constant. We also study the instances where each user is interested in a constant number of topics. The definitions necessary for these special instances are summarized in the beginning of the corresponding section.

We refer here to the famous *minimum hitting set problem* (Min-HS) and *minimum set cover problem* (Min-SC). In Min-HS, we are given a system of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ on n elements $X = \{x_1, \ldots, x_n\}$ (i. e., $S_j \subseteq X$). A feasible solution of this problem is a set $H \subseteq X$, such that $S_j \cap H \neq \emptyset$ for all j. Our goal is to minimize the size of H. The Min-SC is the dual problem to Min-HS. In this problem, we are given a system of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ on n elements $X = \{x_1, \ldots, x_n\}$, a feasible solution is a set $S \subseteq \mathcal{S}$ of sets such that for all i there exists j such that $x_i \in S_j \in S$ and the goal is the minimization of the size of S.

There are many modifications and subproblems of the hitting set problem that are intensively studied. In our paper, we refer to the d-HS problem – a restriction of Min-HS to instances where $|S_i| \leq d$ for all i.

The Min-HS is equivalent to the Min-SC ([3]), thus all the properties of Min-SC carry over to Min-HS. Following from these properties, we have $\mathcal{LOGAPX}$-completeness of Min-HS ([9]) and $\mathcal{APX}$-completeness of d-HS ([17]). There is a well known d-approximation algorithm for d-HS, it can be approximated with ratio $d - \frac{(d-1)\ln\ln n}{\ln n}$ ([11]), it is $\mathcal{NP}$-hard to approximate it within a factor $(d - 1 - \varepsilon)$ ([10]) and d-HS is not approximable within a factor better than d unless the unique game conjecture fails ([13]).

We use the standard definitions from the complexity theory (for details see [12]):

- For $\mathcal{NPO}$ problems in the class $\mathcal{PTAS}$, there exists an algorithm that, for arbitrary $\varepsilon > 0$, produces a solution in polynomial time that is within a factor $(1 + \varepsilon)$ from optimal.
- The $\mathcal{NPO}$ problems in the class $\mathcal{APX}$ are approximable by some constant-factor approximation algorithm in polynomial time.
- For $\mathcal{NPO}$ problems in the class $\mathcal{LOGAPX}$, there exists a polynomial-time logarithmic-factor approximation algorithm.

Thus

$$\mathcal{PTAS} \subseteq \mathcal{APX} \subseteq \mathcal{LOGAPX}.$$

Definition 1. *Let A and B be two $\mathcal{NPO}$ minimization problems. Let I_A and I_B be the sets of the instances of A and B, respectively. Let $S_A(x)$ and $S_B(y)$ be the sets of the feasible solutions and let $cost_A(x)$ and $cost_B(y)$ be polynomially*

computable measures of the instances $x \in I_A$ and $y \in I_B$, respectively. We say that A is AP-reducible *to B, if there exist functions f and g and a constant $\alpha > 0$ such that:*

1. *For any $x \in I_A$ and any $\varepsilon > 0$, $f(x, \varepsilon) \in I_B$.*
2. *For any $x \in I_A$, for any $\varepsilon > 0$, and any $y \in S_B(f(x, \varepsilon))$, $g(x, y, \varepsilon) \in S_A(x)$.*
3. *The functions f and g are computable in polynomial time with respect to the sizes of instances x and y, for any fixed ε.*
4. *The time complexity of computing f and g is nonincreasing with ε for all fixed instances of size $|x|$ and $|y|$.*
5. *For any $x \in I_A$, for any $\varepsilon > 0$, and for any $y \in S_B(f(x, \varepsilon))$*

$$\frac{cost_B(y)}{\min\{cost_B(z) \mid z \in S_B(f(x,\varepsilon))\}} \leq 1 + \varepsilon \text{ implies}$$
$$\frac{cost_A(g(x,y,\varepsilon))}{\min\{cost_A(z) \mid z \in S_A(x)\}} \leq 1 + \alpha \cdot \varepsilon.$$

3 Hardness of Min-TCO When the Number of Users Interested in a Common Topic Is a Constant

In this whole section, we denote by a triple (U, T, INT) an instance of Min-TCO. We focus here on the case where the number of users that share a topic t, i. e., $|U_t|$, is bounded. It is easy to see that, if $\max_{t \in T} |U_t| \leq 2$, then Min-TCO can be solved in linear time, because two users sharing a topic t should be directly connected by an edge, which is the unique minimum solution.

Theorem 1. *If $\max_{t \in T} |U_t| \leq 2$, then* Min-TCO *can be solved in linear time.*

We extend the methods from [8] and design an AP-reduction from d-HS to Min-TCO, where $\max_{t \in T} |U_t| \leq d + 1$. Due to this reduction and the already published statements, we directly obtain lower bounds on the approximability and $\mathcal{APX}$-completeness of these restricted instances of Min-TCO.

Theorem 2. *For arbitrary $d \geq 2$, there exists an AP-reduction from d-*HS *to* Min-TCO*, where $\max_{t \in T} |U_t| \leq d + 1$.*

Proof. Let $I_{\mathrm{HS}} = (X, \mathcal{S})$ be an instance of d-HS and let $\varepsilon > 0$ be arbitrary. We omit the subscript in functions $cost_{d-\mathrm{HS}}$ and $cost_{\mathrm{Min}-(d+1)-\mathrm{TCO}}$ as they are unambiguous. For the instance I_{HS}, we create an instance $I_{\mathrm{TCO}} = (U, T, \mathrm{INT})$ of Min-TCO with $\max_{t \in T} |U_t| \leq d + 1$ with $|X| + k$ users, where $k = |X|^2 \cdot \left\lceil \frac{1+\varepsilon}{\varepsilon} \right\rceil$, as follows (the function f in the definition of AP-reduction).

$$U = X \cup \{p_i \mid p_i \notin X \wedge 1 \leq i \leq k\},$$
$$T = \{t^i_{S_j} \mid S_j \in \mathcal{S} \wedge 1 \leq i \leq k\},$$
$$\mathrm{INT}(x) = \begin{cases} \{t^i_{S_j} \mid x \in S_j \wedge S_j \in \mathcal{S} \wedge 1 \leq i \leq k\} & \text{for } x \in X \\ \{t^i_{S_j} \mid S_j \in \mathcal{S}\} & \text{for } x = p_i \end{cases}$$

Observe that the instance contains $k \cdot |\mathcal{S}|$ topics and its size is polynomial in the size of I_{HS}. The users interested in a topic $t^i_{S_j}$ ($S_j \in \mathcal{S}$) are exactly the elements that are members of set S_j in d-HS plus a *special user* p_i ($1 \leq i \leq k$). Let Sol_{TCO} be a feasible solution of Min-TCO on instance I_{TCO}. We partition the solution into levels. Level i is a set L_i of the edges of Sol_{TCO} that are incident with the special user p_i. In addition, we denote by L_0 the set of edges of Sol_{TCO} that are not incident with any special user. Therefore, $Sol_{\mathrm{TCO}} = \bigcup_{i=0}^{k} L_i$ and $L_i \cap L_j = \emptyset$ ($0 \leq i < j \leq k$).

We claim that, for any L_i ($1 \leq i \leq k$), the set of the non-special users incident with edges of L_i is a feasible solution of the instance I_{HS} of d-HS. This is true since, if a set $S_j \in \mathcal{S}$ is not hit, none of the edges $\{x, p_i\}$ ($x \in S_j$) is in L_i. But then the users interested in topic $t^i_{S_j}$ are not interconnected as user p_i is disconnected.

Let j be chosen such that L_j is the smallest of all sets L_i, for $1 \leq i \leq k$. We construct Sol_{HS} by picking all the non-special users that are incident to some edge from L_j (the function g in the definition of AP-reduction). Denote an optimal solution of d-HS and Min-TCO for I_{HS} and I_{TCO} by Opt_{HS} and Opt_{TCO}, respectively.

If we knew Opt_{HS}, we would be able to construct a feasible solution of Min-TCO on I_{TCO} as follows. First, we pick the edges $\{x, p_i\}$, $x \in Opt_{\mathrm{HS}}$, for all special users p_i, and include them in the solution. This way, for any topic $t \in \mathrm{INT}(p_i)$, we connect p_i to some element of X that is interested in t, too. To have a feasible solution, we could miss some edges between some elements of X. So, we pick all the edges between elements from X. The feasible solution of Min-TCO on I_{TCO} that we obtain has roughly cost

$$k \cdot cost(Opt_{\mathrm{HS}}) + |X|^2 \geq cost(Opt_{\mathrm{TCO}}).$$

On the other hand, if we replace all levels L_i ($1 \leq i \leq k$) by level L_j in Sol_{TCO}, we still have a feasible solution of Min-TCO on I_{TCO}, with cost possibly smaller. Thus

$$k \cdot cost(Sol_{\mathrm{HS}}) \leq cost(Sol_{\mathrm{TCO}}).$$

We use these two inequalities to bound the size of Sol_{HS}:

$$k \cdot cost(Sol_{\mathrm{HS}}) \leq \frac{cost(Sol_{\mathrm{TCO}})}{cost(Opt_{\mathrm{TCO}})} \cdot \left(k \cdot cost(Opt_{\mathrm{HS}}) + |X|^2\right)$$

and thus

$$\frac{cost(Sol_{\mathrm{HS}})}{cost(Opt_{\mathrm{HS}})} \leq \frac{cost(Sol_{\mathrm{TCO}})}{cost(Opt_{\mathrm{TCO}})} \cdot \left(1 + \frac{|X|^2}{k}\right).$$

If $cost(Sol_{\mathrm{TCO}})/cost(Opt_{\mathrm{TCO}}) \leq 1 + \varepsilon$ and $\alpha := 2$, then we have

$$\frac{cost(Sol_{\mathrm{HS}})}{cost(Opt_{\mathrm{HS}})} \leq (1+\varepsilon) \cdot \left(1 + \frac{|X|^2}{k}\right) \leq (1+\varepsilon) \cdot \left(1 + \frac{\varepsilon}{1+\varepsilon}\right) = 1 + 2\varepsilon.$$

It is easy to see that the five conditions of Definition 1 are satisfied and thus we have an AP-reduction. □

Corollary 1. Min-TCO *with* $\max_{t \in T} |U_t| \leq d$ *is* $\mathcal{APX}$*-complete for arbitrary* $d \geq 3$.

Proof. The $\mathcal{APX}$-hardness follows from the $\mathcal{APX}$-hardness of d-HS ([17]). Due to [20], Min-TCO under the mentioned constraint is approximable by a factor in $O(d^2)$ and thus it belongs to the class $\mathcal{APX}$.

Corollary 2. *For any* $\delta > 0$ *and polynomial-time* α*-approximation algorithm of* Min-TCO *with* $\max_{t \in T} |U_t| \leq d + 1$*, there exists a polynomial-time* $(\alpha + \delta)$*-approximation algorithm of* d-HS.

Proof. The approximation algorithm for d-HS would use Theorem 2 with $k := \left\lceil \frac{\alpha \cdot |X|^2}{\delta} \right\rceil$. □

Corollary 3. Min-TCO *with* $\max_{t \in T} |U_t| \leq d$ $(d \geq 3)$ *is* $\mathcal{NP}$*-hard to approximate with a factor* $(d - 1 - \varepsilon)$*, for any* $\varepsilon > 0$*, and, if the unique game conjecture holds, there is no polynomial-time* $(d - \varepsilon)$*-approximation algorithm for it.*

Proof. Otherwise, the reduction described in the proof of Theorem 2 would imply an approximation algorithm for d-HS with a ratio better than $d - 1$ and d respectively. This would directly contradict theorems proven in [10] and [13]. □

The following corollary is an improvement of the already known results of [8] where an $O(\log |T|)$-approximation algorithm is presented, and of [2] where a lower bound of $\Omega(\log(n))$ on the approximability is shown. We close the gap by designing a reduction that can reduce *any* problem from class $\mathcal{LOGAPX}$ to Min-TCO that preserves the approximation ratio up to a constant.

Corollary 4. Min-TCO *is* $\mathcal{LOGAPX}$*-complete.*

Proof. Min-TCO is in the class $\mathcal{LOGAPX}$ since it admits a logarithmic approximation algorithm as presented in [8]. Our reduction from the proof of Theorem 2 is independent of d and thus an AP-reduction from $\mathcal{LOGAPX}$-complete Min-HS to Min-TCO. □

4 Hardness of Min-TCO When the Number of Connections of a User Is Constant

It is natural to consider Min-TCO with bounded number of connections per user, i. e., to bound $\max_{u \in U} |\mathrm{INT}(u)|$, since the number of topics in which one user is interested in is usually not too large. We show that, sadly, Min-TCO is $\mathcal{APX}$-hard even if $\max_{u \in U} |\mathrm{INT}(u)| \leq 6$. To show this, we design a reduction from *minimum vertex cover* (Min-VC) to Min-TCO. The minimum vertex cover problem is just a different name for d-HS with $d = 2$. For a better presentation, in this section, we refer to Min-VC instead of 2-HS.

Given a graph $G = (V', E')$ and a positive integer k as an instance of Min-VC where the goal is to decide whether the given graph has a solution of size at

most k. We construct an instance of Min-TCO as follows. Let $V = V^{(1)} \cup V^{(2)}$ be the set of vertices, where $V^{(1)} = \{v^{(1)} \mid v \in V'\}$ and $V^{(2)} = \{v^{(2)} \mid v \in V'\}$. For each edge $e \in E'$, we prepare three topics, $t_e^{(0)}, t_e^{(1)}$ and $t_e^{(2)}$. The set of topics is the union of all these topics, i. e., $T = \bigcup_{e \in E'} \{t_e^{(0)}, t_e^{(1)}, t_e^{(2)}\}$. The user interest function INT is defined as

$$\mathrm{INT}(u^{(1)}) = \bigcup_{e \in E'[N[u]]} \{t_e^{(0)}, t_e^{(1)}\}$$
$$\mathrm{INT}(u^{(2)}) = \bigcup_{e \in E'[N[u]]} \{t_e^{(0)}, t_e^{(2)}\}.$$

The following lemma shows the relation between the solutions of the two problems.

Lemma 1. *The instance* (V, T, INT) *of* Min-TCO *defined as above has an optimal solution of cost* $k + 2|E'|$ *if and only if the instance* (V', E') *of* Min-VC *has an optimal solution of cost* k.

Moreover, any feasible solution H *of* (V, T, INT) *can be transformed into a feasible solution of* (V', E') *of cost at most* $|H| - 2|E'|$.

We use the Min-VC on degree-bounded graphs, which is $\mathcal{APX}$-hard, to show lower bounds for our restricted Min-TCO. By the above reduction and the lemma, we prove the following theorem.

Theorem 3. Min-TCO *with* $\max_{v \in U} |\mathrm{INT}(v)| \leq 6$ *cannot be approximated within a factor of* $694/693$ *in polynomial time, unless* $\mathcal{P} = \mathcal{NP}$, *even if* $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| \leq 3$ *holds for every pair of different users* $u, v \in U$.

Proof. We prove the statement by contradiction. Suppose that there exists an approximation algorithm A for Min-TCO with the above stated restrictions that has the ratio $(1 + \delta)$.

Let $G = (V', E')$ be an instance of Min-VC and let G be cubic and regular (i. e., each vertex is incident with exactly three edges). We construct an instance I_{TCO} of Min-TCO as stated above and we apply our algorithm A to it to obtain a feasible solution Sol_{TCO}. From such a solution, by Lemma 1, we create a feasible solution of the original Min-VC instance Sol_{VC}. We denote by Opt_{TCO} and Opt_{VC} the optimal solutions of I_{TCO} and G, respectively.

Let d be a constant such that $d \cdot cost(Opt_{\mathrm{VC}}) = 3|V'|$. Since G is cubic and regular, $cost(Opt_{\mathrm{VC}}) \geq |E'|/3 = |V'|/2$ and thus $d \leq 6$.

Observe that, due to Lemma 1, $cost(Opt_{\mathrm{TCO}}) = cost(Opt_{\mathrm{VC}}) + 2|E'| = cost(Opt_{\mathrm{VC}}) + d \cdot cost(Opt_{\mathrm{VC}})$ and $cost(Sol_{\mathrm{TCO}}) \geq cost(Sol_{\mathrm{VC}}) + 2|E'| = cost(Sol_{\mathrm{VC}}) + d \cdot cost(Opt_{\mathrm{VC}})$. These two estimations give us the following bound

$$\frac{cost(Sol_{\mathrm{VC}}) + d \cdot cost(Opt_{\mathrm{VC}})}{cost(Opt_{\mathrm{VC}}) + d \cdot cost(Opt_{\mathrm{VC}})} \leq \frac{cost(Sol_{\mathrm{TCO}})}{cost(Opt_{\mathrm{TCO}})} \leq 1 + \delta.$$

The above inequality allows us to bound the ratio of our Min-VC solution Sol_{VC} and the optimal solution Opt_{VC}:

$$\frac{cost(Sol_{\mathrm{VC}})}{cost(Opt_{\mathrm{VC}})} \leq (1+\delta)\cdot(d+1) - d = 1 + \delta(d+1) \leq 1 + 7\delta.$$

For $\delta := \frac{1}{693}$, we obtain a $\frac{100}{99}$-approximation algorithm for Min-VC on 3-regular graphs which is directly in contradiction with theorem proven in [7]. □

Corollary 5. Min-TCO *with* $\max_{v\in U} |\mathrm{INT}(v)| \leq 6$ *is* $\mathcal{APX}$*-hard.*

Corollary 6. Min-TCO *with* $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| \leq 3$*, for all users* $u, v \in U$*, is* $\mathcal{APX}$*-hard.*

This result is almost tight, the case when $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| \leq 2$ is still open. The following theorem shows that Min-TCO with $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| \leq 1$, for every pair of distinct users $u, v \in U$, can be solved in linear time.

Theorem 4. Min-TCO *can be solved in linear time, if* $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| \leq 1$ *holds for every pair of users* $u, v \in U$*,* $u \neq v$*.*

Proof. We execute the following simple algorithm. First set the solution $E := \emptyset$. Then sequentially, for each topic t, choose its representative $v^* \in U$ ($t \in \mathrm{INT}(v^*)$) and add edges $\{\{v^*, u\} \mid u \in U_t \setminus \{v^*\}\}$ to the solution E. We show that, if $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| \leq 1$, for all distinct $u, v \in U$, then the solution E is optimal.

Observe that, in our case, any edge in any feasible solution is present because of a unique topic. We cannot find an edge $e = \{u, v\}$ of the solution that belongs to the subgraphs for two different topics. (Otherwise $|\mathrm{INT}(v) \cap \mathrm{INT}(u)| > 1$ and our assumption would be wrong for the two endpoints of the edge e.) Thus, any solution consisting of spanning trees for every topic is feasible and optimal. Note that its size is $|T| \cdot (|U| - 1)$. □

Corollary 7. Min-TCO *with* $\max_{u\in U} |\mathrm{INT}(u)| \leq 2$ *can be solved in linear time.*

5 A Polynomial-Time Algorithm for Min-TCO with Bounded Number of Topics

In this section, we present a simple brute-force algorithm that achieves a polynomial running time when the number of topics is bounded by $|T| \leq \log\log|U| - 3\log\log\log|U|$.

Theorem 5. *The optimal solution of* Min-TCO *can be computed in polynomial time if* $|T| \leq (1+\varepsilon(|U|))^{-1} \cdot \log\log|U|$*, for a function* $\varepsilon(n) \geq \frac{3\log\log\log n}{\log\log n - 3\log\log\log n}$*.*

Proof. Let (U, T, INT) be an instance of Min-TCO such that $|T| \leq (1+\varepsilon(|U|))^{-1} \cdot \log\log|U|$. We shorten the notation by setting $t = |T|$ and $n = |U|$.

First observe that, if $u, v \in U$ and $\mathrm{INT}(u) \subseteq \mathrm{INT}(v)$, instead of solving instance (U, T, INT), we can solve Min-TCO on instance $(U \setminus \{u\}, T, \mathrm{INT})$ and

add to such solution the direct edge $\{u, v\}$. Note that u has to be incident with at least one edge in any solution. Thus, the addition of the edge $\{u, v\}$ cannot increase the cost. Moreover, any other user that would be connected to u in some solution can be also connected to v. Thus, we can remove u, solve the smaller instance and then add u by a single edge. Such solution is feasible and its size is unchanged. We say that vertex u is *dominated* by the vertex v if $\mathrm{INT}(u) \subseteq \mathrm{INT}(v)$.

Therefore, before applying our simple algorithm, we remove from the instance all the users that are dominated by some other user. We denote the set of remaining users (i. e., those with incomparable sets of interesting topics) by M and $m = |M| \leq 2^t \leq (\log n)^{(1+\varepsilon(n))^{-1}}$.

Our simple algorithm exhaustively searches over all the possible solutions on instance (M, T, INT) and then reconnects each of the removed users $U \setminus M$ by a single edge. The transformation to set M and then connection of the removed users is clearly polynomial. Thus, we only need to show that our exhaustive search is polynomial.

Observe that the size of the optimal solution is at most $t(n-1)$ as merged spanning trees, for all the topics, form a feasible solution. Our algorithm exhaustively searches over all the possible solutions, i. e., it tries every possible set of i edges for $1 \leq i \leq t(m-1)$ and verifies the topic-connectivity requirements for such sets of edges. The verification of each set can be done in polynomial time. The number of sets it checks can be bounded as follows:

$$\begin{aligned}\sum_{i=1}^{t(m-1)} \binom{\binom{m}{2}}{i} &\leq \sum_{i=1}^{tm} \binom{m^2}{i} \\ &\leq tm \cdot m^{2tm} \\ &\leq (\log n)^{2\cdot(1+\varepsilon(n))^{-2}\log\log n\cdot(\log n)^{(1+\varepsilon(n))^{-1}}} \cdot O(\log^2 n)\end{aligned}$$

To check a polynomial number of sets, it is sufficient to bound the first factor by a polynomial, i. e., by at most n^c for some $c > 0$:

$$\begin{aligned}(\log n)^{2\cdot(1+\varepsilon(n))^{-2}\cdot\log\log n\cdot(\log n)^{(1+\varepsilon(n))^{-1}}} &\leq n^c \\ 2\cdot(1+\varepsilon(n))^{-2}\cdot\log\log^2 n\cdot(\log n)^{(1+\varepsilon(n))^{-1}} &\leq c\log n \\ 2\cdot(1+\varepsilon(n))^{-2}\cdot\log\log^2 n &\leq c(\log n)^{\frac{\varepsilon(n)}{1+\varepsilon(n)}} \\ (\log 2 - 2\log(1+\varepsilon(n))) + 2\log\log\log n &\leq \frac{\varepsilon(n)}{1+\varepsilon(n)}\cdot\log\log n + \log c\end{aligned}$$

If we find a function $\varepsilon(n)$ that satisfies the following inequality 1, it would imply that the inequalities above are satisfied. (Note that $2\log(1+\varepsilon(n)) > 0$ when n tends to infinity.)

$$\log 2 + 2\log\log\log n \leq \frac{\varepsilon(n)}{1+\varepsilon(n)}\cdot\log\log n \tag{1}$$

We are now able to estimate the function $\varepsilon(n)$.

$$\varepsilon(n) \geq \frac{\log 2 + 2 \log\log\log n}{\log\log n - 2\log\log\log n - \log 2} \tag{2}$$

For a nicer presentation, we use $\varepsilon(n) \geq \frac{3\log\log\log n}{\log\log n - 3\log\log\log n}$ that also satisfies (2). □

6 Conclusion

In this paper, we have closed the gap in the approximation hardness of Min-TCO by showing its $\mathcal{LOGAPX}$-completeness. We studied a subproblem of Min-TCO where the number of users interested in a common topic is bounded by a constant d. We showed that, if $d \leq 2$, the restricted Min-TCO is in $\mathcal{P}$ and, if $d \geq 3$, it is $\mathcal{APX}$-complete. The latter allows us to match lower bounds on approximability of these specially instances that match any lower bound known for any problem from class $\mathcal{APX}$. Furthermore, we studied instances of Min-TCO where the number of topics in which a single user is interested in is bounded by a constant d. We presented a reduction that shows that such instances are $\mathcal{APX}$-hard for $d = 6$. In this reduction, any two users have at most three common topics, thus the reduction shows also that latter restricted Min-TCO is $\mathcal{APX}$-hard. We also investigated Min-TCO with a bounded number of topics. Here we presented a polynomial-time algorithm for $|T| \leq (1 + \varepsilon(|U|))^{-1} \cdot \log\log|U|$ and a function $\varepsilon(n) \geq \frac{3\log\log\log n}{\log\log n - 3\log\log\log n}$.

References

1. Anceaume, E., Gradinariu, M., Datta, A.K., Simon, G., Virgillito, A.: A semantic overlay for self- peer-to-peer publish/subscribe. In: Proc. of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), p. 22 (2006)
2. Angluin, D., Aspnes, J., Reyzin, L.: Inferring social networks from outbreaks. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS, vol. 6331, pp. 104–118. Springer, Heidelberg (2010)
3. Ausiello, G., D'Atri, A., Protasi, M.: Structure preserving reductions among convex optimization problems. Journal of Computer and System Sciences 21(1), 136–153 (1980)
4. Baldoni, R., Beraldi, R., Quéma, V., Querzoni, L., Piergiovanni, S.T.: Tera: topic-based event routing for peer-to-peer architectures. In: Proc. of the 2007 Inaugural International Conference on Distributed Event-Based Systems (DEBS 2007). ACM International Conference Proceeding Series, vol. 233, pp. 2–13. ACM, New York (2007)
5. Carzaniga, A., Rutherford, M.J., Wolf, A.L.: A routing scheme for content-based networking. In: Proc. of IEEE INFOCOM 2004 (2004)
6. Chand, R., Felber, P.: Semantic peer-to-peer overlays for publish/Subscribe networks. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1194–1204. Springer, Heidelberg (2005)

7. Chlebík, M., Chlebíková, J.: Inapproximability results for bounded variants of optimization problems. In: Lingas, A., Nilsson, B.J. (eds.) FCT 2003. LNCS, vol. 2751, pp. 27–38. Springer, Heidelberg (2003)
8. Chockler, G., Melamed, R., Tock, Y., Vitenberg, R.: Constructing scalable overlays for pub-sub with many topics. In: Proc. of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC 2007), pp. 109–118. ACM, New York (2007)
9. Creignou, N., Khanna, S., Sudan, M.: Complexity classifications of boolean constraint satisfaction problems. Society for Industrial and Applied Mathematics, Philadelphia (2001)
10. Dinur, I., Guruswami, V., Khot, S., Regev, O.: A new multilayered PCP and the hardness of hypergraph vertex cover. In: Proc. of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003), pp. 595–601. ACM Press, New York (2003)
11. Halperin, E.: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. SIAM Journal on Computing 31(5), 1608–1623 (2002)
12. Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2003)
13. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2-epsilon. Journal of Computer and System Sciences 74(3), 335–349 (2008)
14. Korach, E., Stern, M.: The complete optimal stars-clustering-tree problem. Discrete Applied Mathematics 156(4), 444–450 (2008)
15. Korach, E., Stern, M.: The clustering matroid and the optimal clustering tree. Mathematical Programming 98(1-3), 385–414 (2003)
16. Onus, M., Richa, A.W.: Minimum maximum degree publish-subscribe overlay network design. In: Proc. of IEEE INFOCOM 2009, pp. 882–890. IEEE, Los Alamitos (2009)
17. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43(3), 425–440 (1991)
18. Ramasubramanian, V., Peterson, R., Sirer, E.G.: Corona: A high performance publish-subscribe system for the world wide web. In: Proc. of the 3rd Symposium on Networked Systems Design and Implementation (NSDI 2006), USENIX (2006)
19. Sandler, D., Mislove, A., Post, A., Druschel, P.: FeedTree: Sharing web micronews with peer-to-peer event notification. In: van Renesse, R. (ed.) IPTPS 2005. LNCS, vol. 3640, pp. 141–151. Springer, Heidelberg (2005)
20. Steinová, M.: On the hardness and approximation of minimum topic-connected overlay (brief announcement). In: Proc. of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2011), pp. 295–296. ACM, New York (2011)
21. Zhuang, S., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001), pp. 11–20. ACM, New York (2001)

Can Everybody Sit Closer to Their Friends Than Their Enemies?*

Anne-Marie Kermarrec[1] and Christopher Thraves[2]

[1] INRIA Rennes – Bretagne Atlantique, France
[2] LADyR, GSyC, Universidad Rey Juan Carlos, Spain

Abstract. *Signed graphs* are graphs with signed edges. They are commonly used to represent positive and negative relationships in social networks. While *balance theory* and *clusterizable graphs* deal with signed graphs, recent empirical studies have proved that they fail to reflect some current practices in real social networks. In this paper we address the issue of drawing signed graphs and capturing such social interactions. We relax the previous assumptions to define a drawing as a model in which every vertex has to be placed closer to its neighbors connected through a positive edge than its neighbors connected through a negative edge in the resulting space. Based on this definition, we address the problem of deciding whether a given signed graph has a drawing in a given ℓ-dimensional Euclidean space. We focus on the 1-dimensional case, where we provide a polynomial time algorithm that decides if a given *complete* signed graph has a drawing, and provides it when applicable.

Keywords: Signed graphs, graph embedding, graph drawing, structural balance.

1 Introduction

Social interactions may reflect a wide range of relations with respect to professional links, similar opinions, friendship. As anything related to feelings they may well capture opposite social interaction, e.g., like/dislike, love/hate, friend/enemy. Those social interactions are commonly referred as binary relations. Recent studies on social networks have shown the real existence of binary relations [1,3,12,15,13,16]. A natural abstraction of a network that involves binary relations is a graph with a sign assignment on their edges. Typically, vertices related by a positive interaction (friend) are connected via a positive edge in the graph. On the other hand, vertices socially interacting in a negative way (enemies), are connected via a negative edge in the graph. Such an abstraction is commonly known as *signed graph*.

To the best of our knowledge, the idea of signed graphs representing social networks was introduced in the fifties by Cartwright and Harary in [4]. In that

* This work has been supported by the ERC Starting research grant GOSSPLE number 204742, Comunidad de Madrid grant S2009TIC-1692 and Spanish MICINN grant TIN2008–06735-C02-01.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 388–399, 2011.

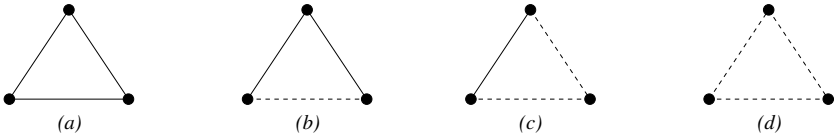


Fig. 1. Signed triangles. This figure depicts every possible combination of positive edges and negatives edges in a triangle. Dashed lines represent negative edges, while continuous lines represent positive edges.

seminal work, the notion of *balanced* signed graph was introduced, and was used to characterize *forbidden* patterns for social networks. Informally, a balanced signed graph is a signed graph that respects the following social rules: *my friend's friend is my friend, my friend's enemy is my enemy, my enemy's friend is my enemy, and my enemy's enemy is my friend.* Formally, a balanced signed graph was defined as a complete[1] signed graph that does not contain as subgraphs[2] neither triangle *(b)* nor triangle *(d)* as depicted in Fig. 1. A general definition of balanced signed graphs says that a signed graph is balanced if all its cycles are *positive*. Using that definition, in [7] it was proved that: "A signed graph is balanced if and only if vertices can be separated into two mutually exclusive subsets such that each positive edge joins two vertices of the same subset and each negative edge joins vertices from different subsets."

Even though, the definition of balanced signed graph intuitively makes sense to characterize social networks, one only has to consider her own set of relationships to find out that your friends are not always friends themseleves and this let us think that real life can hardly be represented by balanced structures. Davis in [5] gave a second characterization for social networks by introducing the notion of *clusterizable* graph. A signed graph is clusterizable if it shows a clustering, i.e., if there exists a partition of the vertices (may be in more than two subsets) such that each positive edge connects two vertices of the same subset and each negative edge connects vertices from different subsets. In the same work, it was proved that "a given signed graph has a clustering if and only if it contains no cycle having exactly one negative edge." Therefore, similar to the definition of balance for complete signed graphs, when a given signed graph is complete it has a clustering if and only if it does not contain triangle *(b)* in Fig. 1 as subgraph.

Balanced signed graphs and clusterizable signed graphs have been used in various studies about social networks [15,13]. Yet, the recent availability of huge databases of social networks enabled empirical studies on real data to check if social structures followed the balance definition [15,16]. Typically, those studies have been carried out to check the presence or the absence of forbidden triangles in social structures. The two common conclusions that can be extracted from those studies are: *(i)* first, the triangle with only positive edges (triangle *(a)* in

[1] A signed graph is *complete* if every pair of distinct vertices is connected by a unique signed edge.

[2] Through this work, the subgraph of a given signed graph G is a subgraph of G that matches the sign assignment.

Fig. 1) is the most likely triangle to be present in a social structure; *(ii)* also they show that the four possible triangles (triangle *(a-d)* Fig.1) are always present in a social structure. Consequently, we can arguably conclude that neither balanced signed graphs nor clusterizable signed graphs fully represent social structures and let us revisit the representation of social interactions.

Our contribution extends the notion of balanced and clusterizable signed graphs. We relax the previous definition and consider that a social structure should merely ensure that each vertex is able to have its friends closer than its enemies. In other words, if a signed graph is embedded in a metric space, each vertex should be placed in the resulting space closer to its neighbors connected via a positive edge than to its neighbors connected via a negative edge. In this paper, we tackle this issue introducing a formal definition of a *valid* graph drawing for signed graphs in a Euclidean metric space. Then we address several questions: *(i)* for a given signed graph and a given dimension, is such a valid drawing possible? given signed graph, what is the minimal dimension in which it can be drawn validly? *(ii)* in case such a drawing is possible, how can it be achieved and at which complexity?

The rest of the paper is structured as follows: After formalizing the notion of a valid drawing for signed graphs in Section 2 and placing them in perspective with balanced graphs, we briefly describe our contribution. In Section 3, we show examples of signed graphs without a valid drawing in dimension 1 and 2. We also introduce the notion of *minimal* signed graph without a valid drawing. That notion captures the idea of forbidden patterns in a social structure. We then focus on dimension 1 in Section 4. We visit the related works in Section 5. We finally close our work in Section 6 with a discussion about the problems this work left open.

2 Problem Statement

In this section we present the context of this work, the required definitions and notations and state the problem. We contextualize our definitions with respect to balanced graph theory. Finally, we provide a brief overview of our contributions.

Definitions and notations: We use $G = (V, E)$ to denote a graph where vertices are denoted with p_i, i ranges from 1 to n. A *signed graph* is defined as follows:

Definition 1. *A* signed graph *is a graph* $G = (V, E)$ *together with a sign assignment* $f : E \to \{-1, +1\}$ *to their edges.*

Equivalently, a signed graph can be defined as a graph G together with a bipartition of the set of edges E. Using Definition 1, the set of edges E is partitioned in $E^+ = \{e \in E : f(e) = +1\}$ and $E^- = \{e \in E : f(e) = -1\}$, such that $E = E^+ \bigcup E^-$ and $E^+ \bigcap E^- = \emptyset$. In the rest of the work, we use $G = (V, E^+ \bigcup E^-)$ to denote a signed graph composed by graph G, with E^+ and E^- as the bipartition of E. Given a signed graph $G = (V, E^+ \bigcup E^-)$, we define *positive* and *negative neighbors* for each vertex in G. Let p_i be a vertex in

G, and $P_i = \{p_j \in V : (p_i, p_j) \in E\}$ be the set of neighbors of p_i. Let us define the set of *positive neighbors* for p_i as follows: $P_i^+ := \{p_j \in P_i : (p_i, p_j) \in E^+\}$. Equivalently, let us define the set of *negative neighbors* for p_i as follows: $P_i^- := \{p_j \in P_i : (p_i, p_j) \in E^-\}$. Therefore, P_i^+ can be considered as the set of p_i's friends, while in the same case, vertices in P_i^- would represent p_i's enemies.

Let us now define graph drawing. Let $G = (V, E)$ be a graph, not necessarily signed. $D(G) = \{u_i \in \mathbb{R}^\ell\}_{i \in \{1,2,\ldots,n\}}$ is a *drawing* of G in the ℓ-dimensional Euclidean space $\mathbb{R}^\ell$ when $D(G)$ is an injection of V in $\mathbb{R}^\ell$.

Since this definition of graph drawing is not sufficient to capture signed graph for there is no distinction between positive and negative edges, we introduce a specific definition of *valid* graph drawing for signed graphs.

Definition 2. *Let $G = (V, E^+ \bigcup E^-)$ be a signed graph, and $D(G)$ be a drawing of G in $\mathbb{R}^\ell$. We say that $D(G)$ is a* valid drawing *of G if: $\forall p_i \in V \quad \forall p_j \in P_i^+ \quad$ and $\quad \forall p_k \in P_i^- \quad d(u_i, u_j) < d(u_i, u_k)$, where $d(\cdot, \cdot)$ denotes the Euclidean distance between two elements in $\mathbb{R}^\ell$.*

Definition 2 captures the fact that every vertex has to be placed closer to its positive neighbors than to its negative neighbors. In the case that there exists a valid drawing of a given signed graph G in $\mathbb{R}^\ell$, we say that G *has a valid drawing* in $\mathbb{R}^\ell$, or simply, we say that G has a valid drawing in dimension ℓ. Otherwise, we simply say that G is a signed graph without valid drawing in dimension ℓ. In this paper, we are interested in a *classification problem*, aiming at determining if a given signed graph has a valid drawing in a given $\mathbb{R}^\ell$. Particularly, we focus in the 1-dimensional case.

Valid drawing versus balanced signed graphs: Balanced signed graphs and clusterizable signed graphs are closely related. Indeed, it is straightforward to observe that if we denote the set of balanced signed graphs with $\mathcal{B}$ and the set of clusterizable graphs with $\mathcal{C}$, then $\mathcal{B}$ is a proper subset of $\mathcal{C}$. That comes from the characterization of balanced signed graphs as clusterizable signed graphs with at most two clusters.

On the other hand, if we denote with $\mathcal{D}^\ell$ the set of signed graphs that have a valid drawing in the ℓ-dimensional Euclidean space. Then, it is also straightforward to note the fact that $\mathcal{D}^\ell$ is a subset of $\mathcal{D}^{\ell'}$ if $\ell \leq \ell'$. Hence, there is a chain of set inclusions of the form $\mathcal{D}^1 \subset \mathcal{D}^2 \subset \mathcal{D}^3 \subset \cdots \subset \mathcal{D}^\ell \subset \cdots$.

In order to place our definition of valid drawing in the context of balanced and clusterizable signed graphs, we point out the fact that *if a graph is clusterizable, then it has a valid drawing in the Euclidean line.* To demonstrate that fact, let us draw a clusterizable signed graph in the following way: place every vertex of a cluster within an interval of length d, then every positive edge will have length at most d. Thereafter, place every pair of clusters at distance at least d' from each other, such that $d' > d$. Therefore, every negative edge will have length at least $d' > d$ and the drawing will be valid. Thus, we can complete the chain of set inclusions adding clusterizable signed graphs and balanced signed graphs as

follows: $\mathcal{B} \subset \mathcal{C} \subset \mathcal{D}^1 \subset \mathcal{D}^2 \subset \mathcal{D}^3 \subset \cdots \subset \mathcal{D}^\ell \subset \cdots$. Establishing this connection is important and helps us to put in context the $\mathcal{D}^1$ case, our main contribution in this work.

Contributions: Our first contribution is precisely the definition of a valid drawing that represents social structures. As far as we know, we are the first to formally define the notion of valid drawing for signed graphs to express the intuitive notion of *everybody sits closer to its friends than enemies*. Consequently, the classification problem is considered in this setting for the first time as well.

First, we show that some graphs do not have a valid drawing. To this end, we provide examples of signed graphs without valid drawing in the Euclidean line and plane. Also, we introduce the concept of *minimal signed graph without a valid drawing* in a given ℓ-dimensional Euclidean space. That definition helps us to find forbidden patterns when we have to decide whether a given graph has or not a valid drawing in a given dimension.

Thereafter, we focus on the specific issue of deciding whether a *complete* signed graph has a valid drawing in the Euclidean line. We characterize the set of complete signed graphs with a valid drawing in the line. We provide a polynomial time algorithm that decides whether a given complete signed graph has or not a valid drawing in the line. When a given complete signed graph has a valid drawing in the line, we provide a polynomial time algorithm that constructs one valid drawing for it.

3 Graphs without Valid Drawing

In this section we present examples of signed graphs without valid drawing in dimension 1 and 2. Specifically, we present three signed graphs, (resp. two), for which we prove they do not have a valid drawing in $\mathbb{R}$, (resp. $\mathbb{R}^2$). We conclude this section with the introduction of the concept of *minimal signed graph without valid drawing*, and with a discussion about its consequences.

Signed graphs without valid drawing in the line: Let us start presenting the counterexamples in increasing order with respect to the number of vertices.

Positive square: The first counterexample is a signed graph composed of four vertices connected in a square of positive edges, while the diagonals are assigned negative signs. Fig. 2*(a)* depicts the described signed graph, let us refer to this signed graph as *the positive square*.

Positive star: The second counterexample is also a signed graph composed of four vertices. In this case, there is a central vertex connected with positive edges to the other three vertices. While, the three other vertices are connected among them via a triangle of negative edges. Fig. 2*(b)* shows the described signed graph. Let us call this second graph *the positive star*.

Positive triangle: The last counterexample is a signed graph with six vertices, three of them are clustered with positive edges in a central triangle. Each external vertex, those that are not in the central triangle, is matched via a positive edge with one corner of the triangle, and connected via a negative edge with

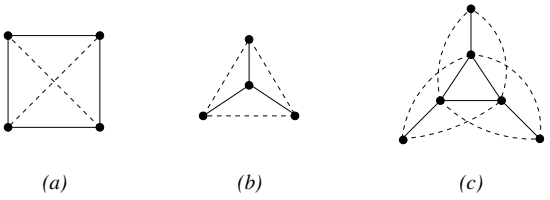


Fig. 2. This figure shows signed graphs *(a) the positive square*, *(b) the positive star*, *(c) the positive triangle*. Dashed lines represent negative edges and continuous lines represent positive edges.

the other two corners of the triangle. Fig. 2*(c)* shows precisely the described signed graph. Let us call this last graph *the positive triangle*.

Lemma 1. *Let G be a signed graph. If G is either the positive square or the positive star or the positive triangle, then G has not a valid drawing in the Euclidean line $\mathbb{R}$.*

Signed graphs without valid drawing in the plane: In the plane we have found two signed graphs without valid drawing.

Negative triangle: The *negative triangle* is a graph of five vertices, three of them, forming a triangle, being connected by negative edges. The other two vertices are connected by a negative edge. The other edges are positive. Fig. 3*(a)* depicts the negative triangle.

Negative cluster: The *negative cluster* is a signed graph with seven vertices. Six vertices out of the seven are connected in a cluster of negative edges. The seventh vertex, called central vertex, is connected with positive edges with each of the six vertices in the cluster. Fig. 3*(b)* depicts the negative cluster.

Lemma 2. *Let G be a signed graph. If G is either the negative triangle or the negative cluster, then G has not a valid drawing in the Euclidean plane $\mathbb{R}^2$.*

Minimal graphs without valid drawing: An interesting remark about graphs with valid drawing in a given dimension is the fact that the property of having a valid drawing in a given dimension is heritable through subgraphs. This can be stated as follows. *Let G be a signed graph. Then, G has no valid drawing in $\mathbb{R}^\ell$ if and only if there exists a subgraph of G that has no valid drawing in $\mathbb{R}^\ell$.* Pushing previous remark to the extreme case when the only subgraph without valid drawing is the same graph, then we obtain the definition of *minimal graph without valid drawing*.

Definition 3. *Let G be a signed graph without valid drawing in $\mathbb{R}^\ell$. We say that G is* minimal *when every proper subgraph of G has a valid drawing in $\mathbb{R}^\ell$.*

At this point, and using Definition 3, we state a complementary problem to the one stated in Section 2. Let us denote $\mathcal{M}^\ell$ the set of all minimal graphs without valid drawing in $\mathbb{R}^\ell$. The complementary problem consists in *give a complete characterization* of $\mathcal{M}^\ell$. The interest of this problem arises in the search of

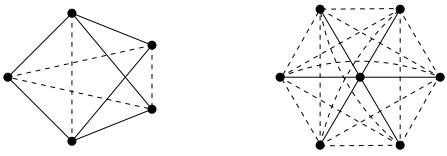


Fig. 3. This figure shows two signed graphs *(a) the negative triangle* and *(b) the negative cluster*. Dashed lines represent negative edges and continuous lines represent positive edges.

forbidden patterns in signed graphs such that they have a valid drawing. Since, the property of having a valid drawing is heritable through subrgaphs. Then, a signed graph G has not a valid drawing in $\mathbb{R}^\ell$ if and only if G does not contain as subgraph any G' in $\mathcal{M}^\ell$. Our contribution to the complementary problem follows.

Theorem 1. *Let G_1 be the positive square signed graph and G_2 be the positive star signed graph, then $\{G_1, G_2\}$ is a subset of $\mathcal{M}^1$.*

Theorem 2. *Let F_1 be the negative triangle signed graph and F_2 be the cluster signed graph, then $\{F_1, F_2\}$ is a subset of $\mathcal{M}^2$.*

4 Drawing Signed Graphs in the Line

In this section we focus in the case $\ell = 1$. We prove that when the signed graph G is complete, there exists a polynomial time algorithm that determines whether it has a valid drawing in the line. Moreover, in the case that such a valid drawing exists, we give a polynomial time algorithm that provides it.

We start by pointing out the fact that any drawing of a given signed graph G in the Euclidean line implies an ordering on the set of vertices V. Let $D(G)$ be a drawing of G in the Euclidean line. Then, $D(G)$ is a set of values $\{u_1, u_2, \ldots, u_n\}$ in $\mathbb{R}$, where u_i determines the position in the line for vertex p_i. The ordering in which we are interested follows: we say that $p_i < p_j \iff u_i < u_j$. W.l.o.g., we assume that $u_1 < u_2 < \cdots < u_n$. Hence, the implicit ordering on the set of vertices V given by $D(G)$ is $p_1 < p_2 < \cdots < p_n$.

Given a signed graph $G = (V, E^+ \bigcup E^-)$, and an ordering on the set of vertices V. We define p_i's adjacencies vector as follows: $\overrightarrow{p_i} = (p_{i1}, p_{i2}, \ldots, p_{in})$, where $p_{ij} = -1$ for $p_j \in P_i^-$, $p_{ij} = 1$ for $i = j$, $p_{ij} = 1$ for $p_j \in P_i^+$ and $p_{ij} = 0$ for $p_i p_j \notin E$.

Lemma 3. *Let $G = (V, E^+ \bigcup E^-)$ be a signed graph. Let $p_1 < p_2 < \cdots < p_n$ be an ordering on the set of vertices V given by a valid drawing of G in $\mathbb{R}$. Then, for all p_i we have that: (i) if $p_{ij} = -1$ and $p_j < p_i$ then $p_{ij'}$ is either equal to -1 or 0, $\forall \quad p_{j'} < p_j$, and (ii) if $p_{ij} = -1$ and $p_j > p_i$ then $p_{ij'}$ is either equal to -1 or 0, $\forall \quad p_{j'} > p_i$.*

Proof. The proof is by contradiction. Let us assume that, even if the ordering on the set of vertices V comes from a valid drawing of G in the line, the first condition does not happen, i.e., there exists $p_{j'} < p_j < p_i$ such that $p_{ij'} = 1$ and $p_{ij} = -1$. Then, since the ordering comes from a valid drawing of G in the line, we have that $d(p_i, p_{j'}) > d(p_i, p_j)$, fact that creates a contradiction because vertices p_i and $p_{j'}$ are connected by a positive edge, while vertices p_i and p_j are connected by a negative edge. The proof is equivalent if we assume that the second condition does not hold.

Let us denote $p_{l_i^-}$ the *closest smaller negative neighbor* of vertex p_i defined as follows: $p_{l_i^-}$ is the p_i's neighbor such that $p_{l_i^-} < p_i$, $p_{il_i^-} = -1$ and p_{ij} is either equal to 0 or 1 for all $p_{l_i^-} < p_j \leq p_i$. Equivalently, let us denote $p_{r_i^-}$ the *closest bigger negative neighbor* of vertex p_i defined as follows: $p_{r_i^-}$ is the p_i's neighbor such that $p_i < p_{r_i^-}$, $p_{ir_i^-} = -1$ and p_{ij} is equal to 0 or 1 for all $p_i \leq p_j < p_{r_i^-}$. To complete this definitions, and to be consistent when those definitions are applied to vertices p_1 and p_n, we include two artificial vertices p_0 and p_∞ such that $p_0 < p_1$, $p_n < p_\infty$, and $p_{i0} = p_{i\infty} = -1$ for all vertex p_i. Then, $p_{l_1^-} = p_0$ and $p_{r_n^-} = p_\infty$. We also define $p_{l_i^+}$ (resp. $p_{r_i^+}$) the *farthest smaller* (resp. *bigger*) *positive neighbor* of vertex p_i as the p_i's neighbor such that $p_{l_i^+} < p_i$, $p_{il_i^+} = 1$ and p_{ij} is equal to -1 or 0 for all $p_j < p_{l_i^+} < p_i$ (resp. $p_i < p_{r_i^+}$, $p_{ir_i^+} = 1$ and p_{ij} is equal to -1 or 0 for all $p_i < p_{r_i^+} < p_j$). We observe the fact that, in the special case when an ordering on the set of vertices V satisfies condition (i) and (ii), then $p_{l_i^-} < p_{l_i^+}$ and $p_{r_i^+} < p_{r_i^-}$. That is an important characteristic in what follows.

Lemma 4. *Let $G = (V, E^+ \bigcup E^-)$ be a signed graph. If there exists an ordering on the set of vertices V such that for all p_i, it is true that:*

(i) if $p_{ij} = -1$ and $p_j < p_i$ then $p_{ij'}$ is either equal to -1 or 0, $\forall \quad p_{j'} < p_j$, and
(ii) if $p_{ij} = -1$ and $p_j > p_i$ then $p_{ij'}$ is either equal to -1 or 0, $\forall \quad p_{j'} > p_i$.

then, there exists a valid drawing of G in the line.

Proof. Let G be a signed graph, and $p_1 < p_2 < p_3 < \cdots < p_n$ be an ordering on the set of vertices V that satisfies condition (i) and (ii). The proof is constructive, we assign real values to p_i's position u_i, for every i. The construction maintains the ordering, i.e., two vertices p_i and p_j in V are placed in the line at u_i and u_j respectively, such that if $p_i < p_j$ then $u_i < u_j$.

The fact that the construction follows the ordering and the fact that the ordering satisfies condition (i) and (ii) imply that for every vertex every smaller (resp. bigger) positive neighbor is placed closer than every smaller (resp. bigger) negative neighbor. This conclusion comes directly from the fact that $p_{l_i^-}$ is strictly smaller than $p_{l_i^+}$ and $p_{r_i^-}$ is strictly bigger than $p_{r_i^+}$, then $u_i - u_{l_i^-} > u_i - u_{l_i^+}$ and $u_{r_i^-} - u_i > u_{r_i^+} - u_i$.

Now, we give a construction that provides a drawing satisfying also for every vertex that every smaller (resp. bigger) positive neighbor is placed closer than

Algorithm 1. Construction of a valid drawing based on an ordering such that conditions (i) and (ii) are satisfied for every vertex

initialization
  Set $u_{l_1^-} = 0$; Set $condition(0) = u_1 < u_2 < u_3 < \cdots < u_n$;
while $i \leq n$ **do**
  Set u_i such that it satisfies conditions $condition(j)$ for $0 \leq j \leq i-1$;
  Set $i = i + 1$;
RETURN $\{u_1, u_2, \ldots, u_n\}$;

every bigger (resp. smaller) negative neighbor. In order to satisfy this property, it is required to assign positions u_i such that $u_i - u_{l_i^-} > u_{r_i^+} - u_i$ and $u_{r_i^-} - u_i > u_i - u_{l_i^+}$, or equivalently each u_i has to satisfy:

$$u_{r_i^+} + u_{l_i^-} < 2u_i < u_{r_i^-} + u_{l_i^+} \tag{1}$$

If values u_i are assigned sequentially from 1 to n following condition (1), then u_i assignment imposes a condition on some u_j for p_j strictly bigger than p_i. We denote $condition(i)$ the condition imposed by u_i's assignment. Hence the construction is presented in Algorithm 1.

The proof finishes showing that the set of conditions $condition(j)$ for $0 \leq j \leq i-1$ is satisfiable. We prove that last point by contradiction. Assume there are two conditions that contradict each other, i.e., for some $p_j < p_i$ such that $u_{r_i^+} = u_{r_j^-}$, $condition(i)$ implies $u_{r_i^+} < 2u_i - u_{l_i^-}$ and $condition(j)$ implies $2u_j - u_{l_j^+} < u_{r_i^-}$, but the assignment produces $2u_i - u_{l_i^-} < 2u_j - u_{l_j^+}$.

If we have $2u_i - u_{l_i^-} < 2u_j - u_{l_j^+}$, then also we have $2u_i - 2u_j < u_{l_i^-} - u_{l_j^+}$. On the other hand, assignment for u_i and u_j followed condition (1), hence $2u_i - 2u_j > u_{r_i^+} + u_{l_i^-} - u_{r_j^-} - u_{l_j^+}$. Since $u_{r_i^+} = u_{r_j^-}$, the previous equation is equivalent to $2u_i - 2u_j > u_{l_i^-} - u_{l_j^+}$, which generates a contradiction. Then, the set of conditions $condition(j)$ for $0 \leq j \leq i-1$ is satisfiable, and Algorithm 1 produces a proper drawing in the Euclidean line. Therefore, the Lemma is proved.

Until this point, we have characterized the property of having a valid drawing by the existence of an ordering on the set of vertices such that conditions (i) and (ii) are satisfied for every vertex. Therefore, determining whether a given signed graph G has a valid drawing in the Euclidean line is equivalent to determining if G has an ordering on its set of vertices such that conditions (i) and (ii) are satisfied for every vertex, in the following we focus on that task. In the sequel, we work with complete signed graphs. We define G^+ the *positive graph* of a given signed graph G as the subgraph of G composed by its positive edges. The positive graph of a given signed graph is not considered as a signed graph.

Lemma 5. *Let G be a complete signed graph, and G^+ be its positive graph. If there exists an ordering on the set of vertices V such that conditions (i) and (ii) are satisfied for every vertex, then G^+ is chordal.*

Let us remind you that a *chordal graph* is a graph where every induced cycle on four or more vertices has a chord. An ordering $p_1 < p_2 < \cdots < p_n$ of vertices is a *perfect elimination ordering* of graph G if the neighborhood of each vertex p_i forms a clique in the graph induced by vertices $p_i, \ldots, p_n$. It is known that a graph is chordal if and only if there exists *perfect elimination ordering* on its set of vertices. (See e.g. [6]).

Lemma 6. *Let G be a complete signed graph with more than four vertices and with a valid drawing in the Euclidean line. Let G^+ be G's positive graph. When G^+ is connected, every perfect elimination ordering on the set of vertices makes satisfiable conditions (i) and (ii) for every vertex.*

In [6], the authors present an algorithm that decides whether a graph is chordal and computes a perfect elimination ordering in case it exists in time $O(n + m)$. Therefore, using that algorithm and the previous results stated in this section, we can decide if a given complete signed graph has a valid drawing and compute such a drawing in polynomial time.

Theorem 3. *Given a complete signed graph G, deciding whether it has a valid drawing in the Euclidean line can be computed in polynomial time. Furthermore, if G has a valid drawing in the Euclidean line, computing such a drawing can be done in polynomial time.*

5 Related Work

To the best of our knowledge, the notion of balance of a signed graph is introduced by Harary in [7], where structural results are presented. Thereafter, Cartwright and Harary applied structural balance theory to social structures, and they compared it with Heidedr's theory in [4]. Later, Davis relaxed the definition of balanced signed graph in [5] to obtain clusterizable graphs, a more general structure on signed graphs. The aforementioned works represent the theoretical basis of clustering and structural balance in signed graphs. On top of that, we found several other works with interesting contributions to structural balance theory. Just as an example of them, in [8] Harary and Kabell gave a polynomial time algorithm that detects balance in signed graphs, whereas in [9,10] the authors counted balanced signed graphs using either marked graphs or Pólya enumeration theorem.

The clustering problem on signed graphs is studied by Bansal et al. in [2]. In that work, the authors considered an optimization problem where the set of vertices of a signed graph has to be partitioned such that the number of positive edges within clusters plus the number of negative edges between clusters is maximized. Clusterizable signed graphs defined by Davis, for instance, achieve the maximum of this value in the total number of edges. Bansal et al. proved that finding the optimal clustering is NP-hard, and they gave approximation algorithm that minimizes disagreements and that maximizes agreements.

As far as we know, the closest work to what we have proposed here is [11] by Kunegis et al. In that work, the authors applied spectral analysis to provide

heuristics for visualization of signed graphs, and link prediction. The visualization proposed is equivalent to the visualization we propose. Nevertheless, their work is only empirical and applied to 2D visualization, hence our contributions complement their contributions.

Recently, signed graphs have been used to study social networks. Antal et al. in [1] studied the dynamic of social networks. The authors studied the evolution of a social network represented by a signed graph under the dynamic situation when a link changes its sign if it is part of an imbalanced triangle. The authors proved the convergence to a balanced state where no imbalanced triangles remain. Leskovec et al. in [15] studied how binary relations affect the structure of on-line social networks. The author connected their analysis with structural balance theory and other structural theories on signed graphs. One of their conclusions says that structural balance theory captures some common patterns but fails to do so for some other fundamental observed phenomena. The structure of on-line social networks is also studied in [16], where the authors study a complete, multi-relational, large social network of a society consisting of the 300,000 players of a massive multiplayer on-line game. The authors present an empirical verification of the structural balance theory. Finally, prediction of signed links based on balance and status theory is studied by Leskovec et al. in [14]. The authors prove that signed links can be predicted with high accuracy in on-line social networks such as Epinions, Slashdot and Wikipedia.

6 Open Problems

This work opens many interesting research directions and this is precisely one of the strength of this piece of work. A number of general questions remain open such as: Given a signed graph G, what is the smallest dimension in which it has a valid drawing? Is it possible to find *efficiently* a valid drawing in such a smallest dimension?

More specifically, in the special case of the Euclidean plane, is it possible to decide in polynomial time whether a given signed graph has a valid drawing and the compute a drawing? Is it easier when the given signed graph is complete as well as it is in the case of the Euclidean line? One open question in the case of the Euclidean line is whether there is a polynomial time algorithm that computes a valid drawing in the Euclidean line when the given signed graph is not complete? Similarly when the given signed graph is complete.

Extensions to the problem might go along different directions. For instance, the metric space can be different, it might not be Euclidean. A more general question is to study the impact of the metric space, for instance, on the forbidden patterns. On the other hand, the value assignment to the edges can range in a larger set of values rather than being binary. The last proposed extension is interesting since it enables to match more closely recommendation systems with ratings. In conclusion, even though in this work an important part of the problem of drawing signed graphs is solved, still there are several open problems to study.

References

1. Antal, T., Krapivsky, P.L., Redner, S.: Dynamics of social balance on networks. Phys. Rev. E 72(3), 036121 (2005)
2. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Machine Learning 56(1-3), 89–113 (2004)
3. Brandes, U., Fleischer, D., Lerner, J.: Summarizing dynamic bipolar conflict structures. IEEE Trans. Vis. Comput. Graph. 12(6), 1486–1499 (2006)
4. Cartwright, D., Harary, F.: Structural balance: a generalization of heider's theory. Psychological Review 63(5), 277–293 (1956)
5. Davis, J.A.: Clustering and structural balance in graphs. Human Relations 20(2), 181 (1967)
6. Habib, M., McConnell, R.M., Paul, C., Viennot, L.: Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theor. Comput. Sci. 234(1-2), 59–84 (2000)
7. Harary, F.: On the notion of balance of a signed graph. Michigan Mathematical Journal 2(2), 143 (1953)
8. Harary, F., Kabell, J.A.: A simple algorithm to detect balance in signed graphs. Mathematical Social Sciences 1(1), 131–136 (1980)
9. Harary, F., Kabell, J.A.: Counting balanced signed graphs using marked graphs. In: Proceedings of the Edinburgh Mathematical Society, vol. 24(2), pp. 99–104 (1981)
10. Harary, F., Palmer, E.: On the number of balanced signed graphs. Bulletin of Mathematical Biology 29, 759–765 (1967)
11. Kunegis, J., Schmidt, S., Lommatzsch, A., Lerner, J., De Luca, E.W., Albayrak, S.: Spectral analysis of signed graphs for clustering, prediction and visualization. In: SDM, page 559 (2010)
12. Lauterbach, D., Truong, H., Shah, T., Adamic, L.A.: Surfing a web of trust: Reputation and reciprocity on couchsurfing.com. In: CSE (4), pp. 346–353 (2009)
13. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Governance in social media: A case study of the wikipedia promotion process. In: ICWSM 2010 (2010)
14. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Predicting positive and negative links in online social networks. In: WWW 2010, pp. 641–650 (2010)
15. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Signed networks in social media. In: CHI 2010, pp. 1361–1370 (2010)
16. Szell, M., Lambiotte, R., Thurner, S.: Multirelational organization of large-scale social networks in an online world. PNAS 107(31), 13636–13641 (2010)

Submodularity on a Tree: Unifying $L^\natural$-Convex and Bisubmodular Functions

Vladimir Kolmogorov

University College London, UK
v.kolmogorov@cs.ucl.ac.uk

Abstract. We introduce a new class of functions that can be minimized in polynomial time in the value oracle model. These are functions f satisfying $f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \sqcap \boldsymbol{y}) + f(\boldsymbol{x} \sqcup \boldsymbol{y})$ where the domain of each variable x_i corresponds to nodes of a rooted binary tree, and operations $\sqcap, \sqcup$ are defined with respect to this tree. Special cases include previously studied $L^\natural$-convex and bisubmodular functions, which can be obtained with particular choices of trees. We present a polynomial-time algorithm for minimizing functions in the new class. It combines Murota's steepest descent algorithm for $L^\natural$-convex functions with bisubmodular minimization algorithms.

Keywords: Submodularity, $L^\natural$-convexity, bisubmodularity, Valued Constraint Satisfaction Problem (VCSP).

1 Introduction

Let $f : \mathcal{D} \rightarrow \mathbb{R}$ be a function of n variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ where $x_i \in D_i$; thus $\mathcal{D} = D_1 \times \ldots \times D_n$. We call elements of D_i *labels*, and the argument of f a *labeling*. Denote $V = \{1, \ldots, n\}$ to be the set of nodes. We will consider functions f satisfying

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \sqcap \boldsymbol{y}) + f(\boldsymbol{x} \sqcup \boldsymbol{y}) \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{D} \tag{1}$$

where binary operations $\sqcap, \sqcup : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ (expressed component-wise via operations $\sqcap, \sqcup : D_i \times D_i \rightarrow D_i$) are defined below.

There are several known cases in which function f can be minimized in polynomial time in the value oracle model. The following two cases will be of particular relevance:

- $L^\natural$*-convex functions*[1]: $D_i = \{0, 1, \ldots, K_i\}$ where $K_i \geq 0$ is integer, $a \sqcap b = \lfloor \frac{a+b}{2} \rfloor$, $a \sqcup b = \lceil \frac{a+b}{2} \rceil$. Property (1) is then called *discrete midpoint convexity* [32].
- *Bisubmodular functions*: $D_i = \{-1, 0, +1\}$, $a \sqcup b = \mathtt{sign}(a + b)$, $a \sqcap b = |ab|\mathtt{sign}(a + b)$.

[1] Pronounced as "L-natural convex".

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 400–411, 2011.

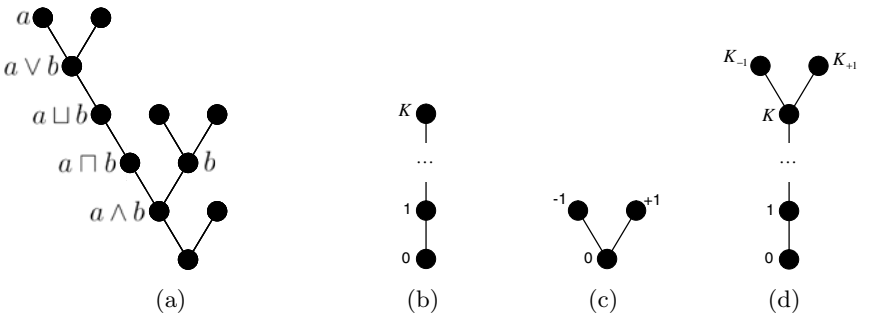


Fig. 1. Examples of trees. Roots are always at the bottom. (a) Illustration of the definition of $a \sqcap b$, $a \sqcup b$, $a \wedge b$ and $a \vee b$. (b) A tree for $L^\natural$-convex functions. (c) A tree for bisubmodular functions. (d) A tree for which a weakly tree-submodular function can be minimized efficiently (see section 4).

In this paper we introduce a new class of functions which includes the two classes above as special cases. We assume that labels in each set D_i are nodes of a tree T_i with a designated root $r_i \in D_i$. Define a partial order $\preceq$ on D_i as follows: $a \preceq b$ if a is an ancestor of b, i.e. a lies on the path from b to r_i $(a, b \in D_i)$. For two labels $a, b \in D_i$ let $\mathcal{P}[a \to b]$ be the unique path from a to b in T_i, $\rho(a, b)$ be the number of edges in this path, and $\mathcal{P}[a \to b, d]$ for integer $d \geq 0$ be the d-th node of this path so that $\mathcal{P}[a \to b, 0] = a$ and $\mathcal{P}[a \to b, \rho(a, b)] = b$. If $d > \rho(a, b)$ then we set by definition $\mathcal{P}[a \to b, d] = b$.

With this notation, we can now define $a \sqcap b$, $a \sqcup b$ as the unique pair of labels satisfying the following two conditions: (1) $\{a \sqcap b, a \sqcup b\} = \{\mathcal{P}[a \to b, \lfloor \frac{d}{2} \rfloor], \mathcal{P}[a \to b, \lceil \frac{d}{2} \rceil]\}$ where $d = \rho(a, b)$, and (2) $a \sqcap b \preceq a \sqcup b$ (Figure 1(a)). We call functions f satisfying condition (1) with such choice of $(\mathcal{D}, \sqcap, \sqcup)$ *strongly tree-submodular*. Clearly, if each T_i is a chain with nodes $0, 1, \ldots, K$ and 0 being the root (Figure 1(b)) then strong tree-submodularity is equivalent to $L^\natural$-convexity. Furthermore, if each T_i is the tree shown in Figure 1(c) then strong tree-submodularity is equivalent to bisubmodularity.

The main result of this paper is the following

Theorem 1. *If each tree T_i is binary, i.e. each node has at most two children, then a strongly tree-submodular function f can be minimized in time polynomial in n and* $\max_i |D_i|$.

Weak tree-submodularity. We will also study alternative operations on trees, which we denote as $\wedge$ and $\vee$. For labels $a, b \in D_i$ we define $a \wedge b$ as their highest common ancestor, i.e. the unique node on the path $\mathcal{P}[a \to b]$ which is an ancestor of both a and b. The label $a \vee b$ is defined as the unique label on the path $\mathcal{P}[a \to b]$ such that the distance between a and $a \vee b$ is the same as the distance between $a \wedge b$ and b (Figure 1(a)).

We say that function f is *weakly tree-submodular* if it satisfies

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \wedge \boldsymbol{y}) + f(\boldsymbol{x} \vee \boldsymbol{y}) \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{D} \tag{2}$$

We will show that strong tree-submodularity (1) implies weak tree-submodularity (2), which justifies the terminology. If all trees are chains shown in Figure 1(b) ($D_i = \{0, 1, \dots, K\}$ with 0 being the root) then $\wedge$ and $\vee$ correspond to the standard operations "meet" and "join" (min and max) on an integer lattice. It is well-known that in this case weakly tree-submodular functions can be minimized in time polynomial in n and K [39,32]. In section 4 we give a slight generalization of this result; namely, we allow trees shown in Figure 1(d).

1.1 Related Work

Studying operations $\langle \sqcap, \sqcup \rangle$ that give rise to tractable optimization problems received a considerable attention in the literature. Some known examples of such operations are reviewed below. For simplicity, we assume that domains D_i (and operations $\langle \sqcap, \sqcup \rangle$) are the same for all nodes: $D_i = D$ for some finite set D.

Submodular functions on lattices. The first example that we mention is the case when D is a distributive lattice and $\sqcap, \sqcup$ are the meet and join operations on this lattice. Functions that satisfy (1) for this choice of D and $\sqcap, \sqcup$ are called *submodular functions* on D; it is well-known that they can be minimized in strongly polynomial time [18,37,19].

Recently, researchers considered submodular functions on non-distributive lattices. It is known that a lattice is non-distributive if it contains as a sublattice either the pentagon $\mathcal{N}_5$ or the diamond $\mathcal{M}_3$. Krokhin and Larose [27] proved tractability for the pentagon case, using nested applications of a submodular minimization algorithm. The case of the diamond was considered by Kuivinen [28], who proved pseudo-polynomiality of the problem. The case of general non-distributive lattices is still open.

$L^\natural$-convex functions. The concept of $L^\natural$-convexity was introduced by Fujishige and Murota [16] as a variant of L-convexity by Murota [30]. $L^\natural$-convexity is equivalent to the combination of submodularity and integral convexity [13] (see [32] for details).

The fastest known algorithm for minimizing $L^\natural$-convex functions is the *steepest descent* algorithm of Murota [31,32,33]. Murota proved in [33] that algorithm's complexity is $O(n \min\{K, n \log K\} \cdot \texttt{SFM}(\texttt{n}))$ where $K = \max_i |D_i|$ and $\texttt{SFM}(n)$ is the complexity of a submodular minimization algorithm for a function with n variables. The analysis of Kolmogorov and Shioura [22] improved the bound to $O(\min\{K, n \log K\} \cdot \texttt{SFM}(\texttt{n}))$. In section 2 we review Murota's algorithm (or rather its version without scaling that has complexity $O(K \cdot \texttt{SFM}(n))$.)

Note, the class of $L^\natural$-convex functions is a subclass of submodular functions on a totally ordered set $D = \{0, 1, \dots, K\}$.

Bisubmodular functions. Bisubmodular functions were introduced by Chandrasekaran and Kabadi as rank functions of *(poly-)pseudomatroids* [7,21]. Independently, Bouchet [3] introduced the concept of Δ-matroids which is equivalent

to pseudomatroids. Bisubmodular functions and their generalizations have also been considered by Qi [35], Nakamura [34], Bouchet and Cunningham [4] and Fujishige [15].

It has been shown that some submodular minimization algorithms can be generalized to bisubmodular functions. Qi [35] showed the applicability of the ellipsoid method. Fujishige and Iwata [17] developed a weakly polynomial combinatorial algorithm for minimizing bisubmodular functions with complexity $O(n^5\texttt{EO}\log M)$ where $\texttt{EO}$ is the number of calls to the evaluation oracle and M is an upper bound on function values. McCormick and Fujishige [29] presented a strongly combinatorial version with complexity $O(n^7\texttt{EO}\log n)$, as well as a $O(n^9\texttt{EO}\log^2 n)$ fully combinatorial variant that does not use divisions. The algorithms in [29] can also be applied for minimizing a bisubmodular function over a *signed ring family*, i.e. a subset $\mathcal{R} \subseteq \mathcal{D}$ closed under $\sqcap$ and $\sqcup$.

Valued constraint satisfaction and multimorphisms. Our paper also fits into the framework of *Valued Constraint Satisfaction Problems* (VCSPs) [11]. In this framework we are given a *language* Γ, i.e. a set of cost functions $f : D^m \to \mathbb{R}_+ \cup \{+\infty\}$ where D is a fixed discrete domain and f is a function of arity m (different functions $f \in \Gamma$ may have different arities). A Γ-*instance* is any function $f : D^n \to \mathbb{R}_+ \cup \{+\infty\}$ that can be expressed as a finite sum of functions from Γ:

$$f(x_1, \ldots, x_n) = \sum_{t \in T} f_t(x_{i(t,1)}, \ldots, x_{i(t,m_t)})$$

where T is a finite set of terms, $f_t \in \Gamma$ is a function of arity m_t, and $i(t,k)$ are indexes in $\{1, \ldots, n\}$. A finite language Γ is called *tractable* if any Γ-instance can be minimized in polynomial time, and *NP-hard* if this minimization problem is NP-hard. These definitions are extended to infinite languages Γ as follows: Γ is called tractable if any finite subset $\Gamma' \subset \Gamma$ is tractable, and NP-hard if there exists a finite subset $\Gamma' \subset \Gamma$ which is NP-hard.

Classifying the complexity of different languages has been an active research area. A major open question in this line of research is the *Dichotomy Conjecture* of Feder and Vardi (formulated for the *crisp* case), which states that every constraint language is either tractable or NP-hard [14]. So far such dichotomy results have been obtained for some special cases, as described below.

A significant progress has been made in the **crisp** case, i.e. when Γ only contains functions $f : D^m \to \{0, +\infty\}$. The problem is then called *Constraint Satisfaction* (CSP). The dichotomy is known to hold for languages with a 2-element domain (Schaefer [36]), languages with a 3-element domain (Bulatov [6]), conservative languages[2] (Bulatov [5]), and languages containing a single relation without sources and sinks (Barto *et al.* [1]). All dichotomy theorems above have the following form: if all functions in Γ satisfy a certain condition given by one or more *polymorphisms* then the language is tractable, otherwise it is NP-hard.

[2] A crisp language Γ is called conservative if it contains all unary cost functions $f : D \to \{0, +\infty\}$ [5]. A general-valued language is called conservative if it contains all unary cost functions $f : D \to \mathbb{R}_+$ [23,24,25].

For general VCSPs the dichotomy has been shown to hold for Boolean languages, i.e. languages with a 2-element domain (Cohen *et al.* [11]), conservative languages (Kolmogorov and Živný [23,24,25], who generalized previous results by Deineko *et al.* [12] and Takhanov [38]), and $\{0,1\}$-valued languages with a 4-element domain (Jonsson *et al.* [20]). In these examples tractable subclasses are characterized by one or more *multimorphisms*, which are generalizations of polymorphisms. A multimorphism of arity k over D is a tuple $\langle \mathtt{OP}_1, \dots, \mathtt{OP}_k \rangle$ where $\mathtt{OP}_i$ is an operation $D^k \rightarrow D$. Language Γ is said to admit multimorphism $\langle \mathtt{OP}_1, \dots, \mathtt{OP}_k \rangle$ if every function $f \in \Gamma$ satisfies

$$f(\boldsymbol{x}_1) + \ldots + f(\boldsymbol{x}_k) \geq f(\mathtt{OP}_1(\boldsymbol{x}_1, \dots, \boldsymbol{x}_k)) + \ldots + f(\mathtt{OP}_k(\boldsymbol{x}_1, \dots, \boldsymbol{x}_k))$$

for all labelings $\boldsymbol{x}_1, \dots, \boldsymbol{x}_k$ with $f(\boldsymbol{x}_1) < +\infty$, ..., $f(\boldsymbol{x}_k) < +\infty$. (The pair of operations $\langle \sqcap, \sqcup \rangle$ used in (1) is an example of a binary multimorphism.) The tractable classes mentioned above (for $|D| > 2$) are characterized by *complementary pairs of STP and MJN* multimorphisms [24] (that generalized *symmetric tournament pair (STP)* multimorphisms [10]), and *1-defect chain* multimorphisms [20] (that generalized tractable weak-tree submodular functions in section 4 originally introduced in [26]).

To make further progress on classifying complexity of VCSPs, it is important to study which multimorphisms lead to tractable optimisation problems. Operations $\langle \sqcap, \sqcup \rangle$ and $\langle \wedge, \vee \rangle$ introduced in this paper represent new classes of such multimorphisms: to our knowledge, previously researchers have not considered multimorphisms defined on trees.

Combining multimorphisms. Finally, we mention that some constructions, namely *Cartesian products* and *Malt'stev products*, can be used for obtaining new tractable classes of binary multimoprhisms from existing ones [27]. Note, Krokhin and Larose [27] formulated these constructions only for lattice multimorphisms $\langle \sqcap, \sqcup \rangle$, but the proof in [27] actually applies to arbitrary binary multimorphisms $\langle \sqcap, \sqcup \rangle$.

2 Steepest Descent Algorithm

It is known that for $L^\natural$-convex functions local optimality implies global optimality [32]. We start by generalizing this result to strongly tree-submodular functions. Let us define the following "local" neighborhoods of labeling $\boldsymbol{x} \in \mathcal{D}$:

$$\begin{aligned} \mathtt{NEIB}(\boldsymbol{x}) &= \{\boldsymbol{y} \in \mathcal{D} \mid \rho(\boldsymbol{x}, \boldsymbol{y}) \leq 1\} \\ \mathtt{INWARD}(\boldsymbol{x}) &= \{\boldsymbol{y} \in \mathtt{NEIB}(\boldsymbol{x}) \mid \boldsymbol{y} \preceq \boldsymbol{x}\} \\ \mathtt{OUTWARD}(\boldsymbol{x}) &= \{\boldsymbol{y} \in \mathtt{NEIB}(\boldsymbol{x}) \mid \boldsymbol{y} \succeq \boldsymbol{x}\} \end{aligned}$$

where $\boldsymbol{u} \preceq \boldsymbol{v}$ means that $u_i \preceq v_i$ for all $i \in V$, and $\rho(\boldsymbol{x}, \boldsymbol{y}) = \max_{i \in V} \rho(x_i, y_i)$ is the l_∞-distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. Clearly, the restriction of f to $\mathtt{INWARD}(\boldsymbol{x})$ is

a submodular function, and the restriction of f to $\texttt{OUTWARD}(\boldsymbol{x})$ is bisubmodular assuming that each tree T_i is binary[3].

Proposition 1. *Suppose that* $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) | \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} = \min\{f(\boldsymbol{y}) | \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$. *Then* $\boldsymbol{x}$ *is a global minimum of* f.

Proof. First, let us prove that $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{NEIB}(\boldsymbol{x})\}$. Let $\boldsymbol{x}^*$ be a minimizer of f in $\texttt{NEIB}(\boldsymbol{x})$, and denote $\mathcal{D}^* = \{\boldsymbol{y} \in \mathcal{D} \mid y_i \in \mathcal{D}_i^* = \{x_i, x_i^*\}\} \subseteq \texttt{NEIB}(\boldsymbol{x})$. We treat set $\mathcal{D}_i^*$ as a tree with root $x_i \sqcap x_i^*$. Clearly, the restriction of f to $\mathcal{D}^*$ is an $L^\natural$-convex function under the induced operations $\sqcap$, $\sqcup$. It is known that for $L^\natural$-convex functions optimality of $\boldsymbol{x}$ in sets $\{\boldsymbol{y} \in \mathcal{D}^* \mid \boldsymbol{y} \preceq \boldsymbol{x}\}$ and $\{\boldsymbol{y} \in \mathcal{D}^* \mid \boldsymbol{y} \succeq \boldsymbol{x}\}$ suffices for optimality of $\boldsymbol{x}$ in $\mathcal{D}^*$ [32, Theorem 7.14], therefore $f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*)$. This proves that $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{NEIB}(\boldsymbol{x})\}$.

Let us now prove that $\boldsymbol{x}$ is optimal in $\mathcal{D}$. Suppose not, then there exists $\boldsymbol{y} \in \mathcal{D}$ with $f(\boldsymbol{y}) < f(\boldsymbol{x})$. Among such labelings, let us choose $\boldsymbol{y}$ with the minimum distance $\rho(\boldsymbol{x}, \boldsymbol{y})$. We must have $\boldsymbol{y} \notin \texttt{NEIB}(\boldsymbol{x})$, so $\rho(\boldsymbol{x}, \boldsymbol{y}) \geq 2$. Clearly, $\rho(\boldsymbol{x}, \boldsymbol{x} \sqcup \boldsymbol{y}) \leq \rho(\boldsymbol{x}, \boldsymbol{y}) - 1$ and $\rho(\boldsymbol{x}, \boldsymbol{x} \sqcap \boldsymbol{y}) \leq \rho(\boldsymbol{x}, \boldsymbol{y}) - 1$. Strong tree-submodularity and the fact that $f(\boldsymbol{y}) < f(\boldsymbol{x})$ imply that the cost of at least one of the labelings $\boldsymbol{x} \sqcup \boldsymbol{y}$, $\boldsymbol{x} \sqcap \boldsymbol{y}$ is smaller than $f(\boldsymbol{x})$. This contradicts to the choice of $\boldsymbol{y}$.

Suppose that each tree T_i is binary. The proposition shows that a greedy technique for computing a minimizer of f would work. We can start with an arbitrary labeling $\boldsymbol{x} \in \mathcal{D}$, and then apply iteratively the following two steps in some order:

(1) Compute minimizer $\boldsymbol{x}^{\texttt{in}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\}$ by invoking a submodular minimization algorithm, replace $\boldsymbol{x}$ with $\boldsymbol{x}^{\texttt{in}}$ if $f(\boldsymbol{x}^{\texttt{in}}) < f(\boldsymbol{x})$.
(2) Compute minimizer $\boldsymbol{x}^{\texttt{out}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$ by invoking a bisubmodular minimization algorithm, replace $\boldsymbol{x}$ with $\boldsymbol{x}^{\texttt{out}}$ if $f(\boldsymbol{x}^{\texttt{out}}) < f(\boldsymbol{x})$.

The algorithm stops if neither step can decrease the cost. Clearly, it terminates in a finite number of steps and produces an optimal solution. We will now discuss how to obtain a polynomial number of steps. We denote $K = \max_i |D_i|$.

2.1 $L^\natural$-Convex Case

For $L^\natural$-convex functions the *steepest descent* algorithm described above was first proposed by Murota [31,32,33], except that in step 2 a submodular minimization algorithm was used. Murota's algorithm actually computes both of $\boldsymbol{x}^{\texttt{in}}$ and $\boldsymbol{x}^{\texttt{out}}$ for the same $\boldsymbol{x}$ and then chooses a better one by comparing costs $f(\boldsymbol{x}^{\texttt{in}})$ and $f(\boldsymbol{x}^{\texttt{out}})$. A slight variation was proposed by Kolmogorov and Shioura [22], who allowed an arbitrary order of steps. Kolmogorov and Shioura also established a tight bound on the number of steps of the algorithm by proving the following theorem.

[3] If label x_i has less than two children in T_i then variable's domain after restriction will be a strict subset of $\{-1, 0, +1\}$. Therefore, we may need to use a bisubmodular minimization algorithm over a signed ring familiy $\mathcal{R} \subseteq \{-1, 0, +1\}^n$ [29].

Theorem 2 ([22]). *Suppose that each tree T_i is a chain. For a labeling $\boldsymbol{x} \in \mathcal{D}$ define*

$$\rho^-(\boldsymbol{x}) = \min\{\rho(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{y} \in OPT^-[\boldsymbol{x}]\}, OPT^-[\boldsymbol{x}] = \operatorname{argmin}\{f(\boldsymbol{y}) | \boldsymbol{y} \in \mathcal{D}, \boldsymbol{y} \preceq \boldsymbol{x}\} \quad (3a)$$

$$\rho^+(\boldsymbol{x}) = \min\{\rho(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{y} \in OPT^+[\boldsymbol{x}]\}, OPT^+[\boldsymbol{x}] = \operatorname{argmin}\{f(\boldsymbol{y}) | \boldsymbol{y} \in \mathcal{D}, \boldsymbol{y} \succeq \boldsymbol{x}\} \quad (3b)$$

(a) Applying step (1) or (2) to labeling $\boldsymbol{x} \in \mathcal{D}$ does not increase $\rho^-(\boldsymbol{x})$ and $\rho^+(\boldsymbol{x})$.
(b) If $\rho^-(\boldsymbol{x}) \geq 1$ then applying step (1) to $\boldsymbol{x}$ will decrease $\rho^-(\boldsymbol{x})$ by 1.
(c) If $\rho^+(\boldsymbol{x}) \geq 1$ then applying step (2) to $\boldsymbol{x}$ will decrease $\rho^+(\boldsymbol{x})$ by 1.

In the beginning of the algorithm we have $\rho^-(\boldsymbol{x}) \leq K$ and $\rho^+(\boldsymbol{x}) \leq K$, so the theorem implies that after at most K calls to step (1) and K calls to step (2) we get $\rho^-(\boldsymbol{x}) = \rho^+(\boldsymbol{x}) = 0$. The latter condition means that $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$, and thus, by proposition 1, $\boldsymbol{x}$ is a global minimum of f.

2.2 General Case

We now show that the bound $O(K)$ on the number of steps is also achievable for general strongly tree-submodular functions. We will establish it for the following version of the steepest descent algorithm:

S0 Choose an arbitrary labeling $\boldsymbol{x}^\circ \in \mathcal{D}$ and set $\boldsymbol{x} := \boldsymbol{x}^\circ$.
S1 Compute minimizer $\boldsymbol{x}^{\texttt{in}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\}$. If $f(\boldsymbol{x}^{\texttt{in}}) < f(\boldsymbol{x})$ then set $\boldsymbol{x} := \boldsymbol{x}^{\texttt{in}}$ and repeat step S1, otherwise go to step S2.
S2 Compute minimizer $\boldsymbol{x}^{\texttt{out}} \in \arg\min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$. If $f(\boldsymbol{x}^{\texttt{out}}) < f(\boldsymbol{x})$ then set $\boldsymbol{x} := \boldsymbol{x}^{\texttt{out}}$ and repeat step S2, otherwise terminate.

Note, one could choose x_i° to be the root of tree T_i for each node $i \in V$, then step `S1` would be redundant.

Theorem 3. *(a) Step* `S1` *is performed at most K times. (b) Each step* `S2` *preserves the following property:*

$$f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} \quad (4)$$

(c) Step `S2` *is performed at most K times. (d) Labeling $\boldsymbol{x}$ produced upon termination of the algorithm is a minimizer of f.*

Proof. For a labeling $\boldsymbol{x} \in \mathcal{D}$ denote $\mathcal{D}^-[\boldsymbol{x}] = \{\boldsymbol{y} \in \mathcal{D} \mid \boldsymbol{y} \preceq \boldsymbol{x}\}$. We will treat domain $\mathcal{D}^-[\boldsymbol{x}]$ as the collection of chains with roots r_i and leaves x_i. Let $\rho^-(\boldsymbol{x})$ be the quantity defined in (3a). There holds

$$f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{INWARD}(\boldsymbol{x})\} \quad \Leftrightarrow \quad \rho^-(\boldsymbol{x}) = 0 \quad (5)$$

Indeed, this equivalence can be obtained by applying proposition 1 to function f restricted to $\mathcal{D}^-[\boldsymbol{x}]$.

(a) When analyzing the first stage of the algorithm, we can assume without loss of generality that $\mathcal{D} = \mathcal{D}^-[\boldsymbol{x}^\circ]$, i.e. each tree T_i is a chain with the root

r_i and the leaf x_i°. Indeed, removing the rest of the tree will not affect the behaviour of steps `S1`. With such assumption, function f becomes $L^\natural$-convex. By theorem 2(b), steps `S1` will terminate after at most K steps.

(b,c) Property (4) (or equivalently $\rho^-(\boldsymbol{x}) = 0$) clearly holds after termination of steps `S1`. Let $\boldsymbol{z}$ be the labeling upon termination of steps `S2`. When analyzing the second stage of the algorithm, we can assume without loss of generality that $\mathcal{D} = \mathcal{D}^-[\boldsymbol{z}]$, i.e. each tree T_i is a chain with the root r_i and the leaf z_i. Indeed, removing the rest of the tree will not affect the behaviour of steps `S2`. Furthermore, restricting f to $\mathcal{D}^-[\boldsymbol{z}]$ does not affect the definition of $\rho^-(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{D}^-[\boldsymbol{z}]$.

By theorem 2(a), steps `S2` preserve $\rho^-(\boldsymbol{x}) = 0$; this proves part (b). Part (c) follows from theorem 2(c).

(d) When steps `S2` terminate, we have $f(\boldsymbol{x}) = \min\{f(\boldsymbol{y}) \mid \boldsymbol{y} \in \texttt{OUTWARD}(\boldsymbol{x})\}$. Combining this fact with condition (4) and using proposition 1 gives that upon algorithm's termination $\boldsymbol{x}$ is a minimizer of f.

3 Translation Submodularity

In this section we derive an alternative definition of strongly tree-submodular functions. As a corollary, we will obtain that strong tree submodularity (1) implies weak tree submodularity (2).

Let us introduce another pair of operations on trees. Given labels $a, b \in D_i$ and an integer $d \geq 0$, we define

$$a \uparrow^d b = \mathcal{P}[a \to b, d] \wedge b \qquad a \downarrow_d b = \mathcal{P}[a \to b, \rho(a \uparrow^d b, b)]$$

In words, $a \uparrow^d b$ is obtained as follows: (1) move from a towards b by d steps, stopping if b is reached earlier; (2) keep moving until the current label becomes an ancestor of b. $a \downarrow_d b$ is the label on the path $\mathcal{P}[a \to b]$ such that the distances $\rho(a, a \downarrow_d b)$ and $\rho(a \uparrow^d b, b)$ are the same, as well as distances $\rho(a, a \uparrow^d b)$ and $\rho(a \downarrow_d b, b)$. Note, binary operations $\uparrow^d, \downarrow_d$: $D_i \times D_i \to D_i$ (and corresponding operations $\uparrow^d, \downarrow_d$: $\mathcal{D} \times \mathcal{D} \to \mathcal{D}$) are in general non-commutative. One exception is $d = 0$, in which case $\uparrow^d, \downarrow_d$ reduce to the commutative operations defined in the introduction: $\boldsymbol{x} \uparrow^0 \boldsymbol{y} = \boldsymbol{x} \wedge \boldsymbol{y}$ and $\boldsymbol{x} \downarrow_0 \boldsymbol{y} = \boldsymbol{x} \vee \boldsymbol{y}$.

For fixed labels $a, b \in D_i$ it will often be convenient to rename nodes in $\mathcal{P}[a \to b]$ to be consecutive integers so that $a \wedge b = 0$ and $a \leq 0 \leq b$. Then we have $a = -\rho(a, a \wedge b)$, $b = \rho(a \wedge b, b)$ and

$$a \uparrow^d b = \max\{0, \min\{a + d, b\}\} \qquad a \downarrow_d b = a + b - (a \uparrow^d b)$$

Theorem 4. *(a) If f is strongly tree-submodular then for any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ and integer $d \geq 0$ there holds*

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \uparrow^d \boldsymbol{y}) + f(\boldsymbol{x} \downarrow_d \boldsymbol{y}) \tag{6}$$

(b) If (6) holds for any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{D}$ and $d \geq 0$ then f is strongly tree-submodular.

Note, this result is well-known for $L^\natural$-convex functions [32, section 7.1], i.e. when all trees are chains shown in Figure 1(b); inequality (6) was then written as $f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f((\boldsymbol{x} + d \cdot \mathbf{1}) \wedge \boldsymbol{y}) + f(\boldsymbol{x} \vee (\boldsymbol{y} - d \cdot \mathbf{1}))$, and was called *translation submodularity*. In fact, translation submodularity is one of the key properties of $L^\natural$-convex functions, and was heavily used, for example, in [22] for proving theorem 2.

Setting $d = 0$ in theorem 4(a) gives

Corollary 1. *A strongly tree-submodular function f is also weakly tree-submodular, i.e. (1) implies (2).*

A proof of theorem 6 is given in [26].

4 Weakly Tree-Submodular Functions

In this section we consider functions f that satisfy condition (2), but not necessarily condition (1). It is well-known [39,32] that such functions can be minimized efficiently if all trees T_i are chains rooted at an endpoint and $\max_i |D_i|$ is polynomially bounded. The algorithm utilizes Birkhoff's representation theorem [2] which says that there exists a *ring family* $\mathcal{R}$ such that there is an isomorphism between sets $\mathcal{D}$ and $\mathcal{R}$ that preserves operations $\wedge$ and $\vee$. (A subset $\mathcal{R} \subseteq \{0,1\}^m$ is a ring family if it is closed under operations $\wedge$ and $\vee$.) It is known that submodular functions over a ring family can be minimized in polynomial time, which implies the result. Note that the number of variables will be $m = O(\sum_i |D_i|)$.

Another case when f satisfying (2) can be minimized efficiently is when f is bisubmodular, i.e. all trees are as shown in Figure 1(c). Indeed, in this case the pairs of operations $\langle \sqcap, \sqcup \rangle$ and $\langle \wedge, \vee \rangle$ coincide.

An interesting question is whether there exist other classes of weakly tree-submodular functions that can be minimized efficiently. In this section we provide one rather special example. We consider the tree shown in Figure 1(d). Each T_i has nodes $\{0, 1, \ldots, K, K_{-1}, K_{+1}\}$ such that 0 is the root, the parent of k for $k = 1, \ldots, K$ is $k-1$, and the parent of K_{-1} and K_{+1} is K.

In order to minimize function f for such choice of trees, we create $K+1$ variables $y_{i0}, y_{i1}, \ldots, y_{iK}$ for each original variable $x_i \in D_i$. The domains of these variables are as follows: $\tilde{D}_{i0} = \ldots = \tilde{D}_{iK-1} = \{0,1\}$, $\tilde{D}_{iK} = \{-1, 0, +1\}$. Each domain is treated as a tree with root 0 and other nodes being the children of 0; this defines operations $\wedge$ and $\vee$ for domains $\tilde{D}_{i0}, \ldots \tilde{D}_{iK-1}, \tilde{D}_{iK}$. The domain $\tilde{\mathcal{D}}$ is set as the Cartesian product of individual domains over all nodes $i \in V$. Note, a vector $\boldsymbol{y} \in \tilde{\mathcal{D}}$ has $n(K+1)$ components.

For a labeling $\boldsymbol{x} \in \mathcal{D}$ let us define labeling $\boldsymbol{y} = \psi(\boldsymbol{x}) \in \tilde{\mathcal{D}}$ as follows:

$$\begin{array}{lcl}
x_i = k \in \{0, 1, \ldots, K\} & \Rightarrow & y_{i0} = \ldots = y_{ik-1} = 1,\ y_{ik} = \ldots = y_{iK} = 0 \\
x_i = K_{-1} & \Rightarrow & y_{i0} = \ldots = y_{iK-1} = 1,\ y_{iK} = -1 \\
x_i = K_{+1} & \Rightarrow & y_{i0} = \ldots = y_{iK-1} = 1,\ y_{iK} = +1
\end{array}$$

It is easy to check that mapping $\psi : \mathcal{D} \to \tilde{\mathcal{D}}$ is injective and preserves operations $\wedge$ and $\vee$. Therefore, $\mathcal{R} = \mathtt{Im}\ \psi$ is a *signed ring family*, i.e. a subset of $\tilde{\mathcal{D}}$ closed under operations $\wedge$ and $\vee$. It is known [29] that bisubmodular functions over ring families can be minimized in polynomial time, leading to

Proposition 2. *Functions that are weakly tree-submodular with respect to trees shown in Figure 1(d) can be minimized in time polynomial in n and $\max_i |D_i|$.*

5 Conclusions and Discussion

We introduced two classes of functions (strongly tree-submodular and weakly tree-submodular) that generalize several previously studied classes. For each class, we gave new examples of trees for which the minimization problem is tractable.

Our work leaves a natural open question: what is the complexity of the problem for more general trees? In particular, can we minimize efficiently strongly tree-submodular functions if trees are non-binary, i.e. if some nodes have three or more children? Note that the algorithm in section 2 and its analysis are still valid, but it is not clear whether the minimization procedure in step `S2` can be implemented efficiently. Also, are there trees besides the one shown in Figure 1(d) for which weakly tree-submodular functions can be minimized efficiently?

More generally, can one characterize for which operations $\langle \sqcap, \sqcup \rangle$ the minimization problem is tractable? Currently known tractable examples are distributive lattices, some non-distributive lattices [27,28], operations on trees introduced in this paper, and combinations of the above operations obtained via Cartesian product and Malt'sev product [27]. Are there tractable cases that cannot be obtained via lattice and tree-based operations?

References

1. Barto, L., Kozik, M., Niven, T.: The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). SIAM Journal on Computing 38(5), 1782–1802 (2009)
2. Birkhoff, G.: Rings of sets. Duke Mathematical Journal 3(3), 443–454 (1937)
3. Bouchet, A.: Greedy algorithm and symmetric matroids. Math. Programming 38, 147–159 (1987)
4. Bouchet, A., Cunningham, W.H.: Delta-matroids, jump systems and bisubmodular polyhedra. SIAM J. Discrete Math. 8, 17–32 (1995)
5. Bulatov, A.A.: Tractable Conservative Constraint Satisfaction Problems. In: Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS 2003), pp. 321–330 (2003)
6. Bulatov, A.A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. Journal of the ACM 53(1), 66–120 (2006)
7. Chandrasekaran, R., Kabadi, S.N.: Pseudomatroids. Discrete Math. 71, 205–217 (1988)

8. Cohen, D., Cooper, M., Jeavons, P.G.: A Complete Characterization of Complexity for Boolean Constraint Optimization Problems. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 212–226. Springer, Heidelberg (2004)
9. Cohen, D.A., Cooper, M., Jeavons, P.G., Krokhin, A.A.: Soft Constraints: Complexity and Multimorphisms. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 244–258. Springer, Heidelberg (2003)
10. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: Generalising submodularity and horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. Theoretical Computer Science 401, 36–51 (2008)
11. Cohen, D.A., Cooper, M.C., Jeavons, P.G., Krokhin, A.A.: The complexity of soft constraint satisfaction. Artificial Intelligence 170, 983–1016 (2006)
12. Deineko, V., Jonsson, P., Klasson, M., Krokhin, A.: The approximability of Max CSP with fixed-value constraints. Journal of the ACM 55(4) (2008)
13. Favati, P., Tardella, F.: Convexity in nonlinear integer programming. Ricerca Operativa 53, 3–44 (1990)
14. Feder, T., Vardi, M.: The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. SIAM Journal on Computing 28(1), 57–104 (1998)
15. Fujishige, S.: Submodular Functions and Optimization. North-Holland, Amsterdam (1991)
16. Fujishige, S., Murota, K.: Notes on L-/M-convex functions and the separation theorems. Math. Program. 88, 129–146 (2000)
17. Fujishige, S., Iwata, S.: Bisubmodular function minimization. SIAM J. Discrete Math. 19(4), 1065–1073 (2006)
18. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1988)
19. Iwata, S., Fleischer, L., Fujishige, S.: A combinatorial strongly polynomial algorithm for minimizing submodular functions. J. ACM 48, 761–777 (2001)
20. Jonsson, P., Kuivinen, F., Thapper, J.: Min CSP on Four Elements: Moving Beyond Submodularity. Tech. rep. arXiv:1102.2880 (February 2011)
21. Kabadi, S.N., Chandrasekaran, R.: On totally dual integral systems. Discrete Appl. Math. 26, 87–104 (1990)
22. Kolmogorov, V., Shioura, A.: New algorithms for convex cost tension problem with application to computer vision. Discrete Optimization 6(4), 378–393 (2009)
23. Kolmogorov, V., Živný, S.: The complexity of conservative finite-valued CSPs. Tech. rep. arXiv:1008.1555v1 (August 2010)
24. Kolmogorov, V., Živný, S.: Generalising tractable VCSPs defined by symmetric tournament pair multimorphisms. Tech. rep. arXiv:1008.3104v1 (August 2010)
25. Kolmogorov, V.: A dichotomy theorem for conservative general-valued CSPs. Tech. rep. arXiv:1008.4035v1 (August 2010)
26. Kolmogorov, V.: Submodularity on a tree: Unifying $L^\natural$-convex and bisubmodular functions. Tech. rep. arXiv:1007.1229v3, April 2011 (first version: July 2010)
27. Krokhin, A., Larose, B.: Maximizing supermodular functions on product lattices, with application to maximum constraint satisfaction. SIAM Journal on Discrete Mathematics 22(1), 312–328 (2008)
28. Kuivinen, F.: On the Complexity of Submodular Function Minimisation on Diamonds. Tech. rep. arXiv:0904.3183v1 (April 2009)
29. McCormick, S.T., Fujishige, S.: Strongly polynomial and fully combinatorial algorithms for bisubmodular function minimization. Math. Program., Ser. A 122, 87–120 (2010)

30. Murota, K.: Discrete convex analysis. Math. Program. 83, 313–371 (1998)
31. Murota, K.: Algorithms in discrete convex analysis. IEICE Transactions on Systems and Information E83-D, 344–352 (2000)
32. Murota, K.: Discrete Convex Analysis. SIAM Monographs on Discrete Mathematics and Applications 10 (2003)
33. Murota, K.: On steepest descent algorithms for discrete convex functions. SIAM J. Optimization 14(3), 699–707 (2003)
34. Nakamura, M.: A characterization of greedy sets: universal polymatroids (I). Scientific Papers of the College of Arts and Sciences 38(2), 155–167 (1998); The University of Tokyo
35. Qi, L.: Directed submodularity, ditroids and directed submodular flows. Mathematical Programming 42, 579–599 (1988)
36. Schaefer, T.: The Complexity of Satisfiability Problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978), pp. 216–226 (1978)
37. Schrijver, A.: A combinatorial algorithm minimizing submodular functions in strongly polynomial time. J. Combin. Theory Ser. B 80, 346–355 (2000)
38. Takhanov, R.: A Dichotomy Theorem for the General Minimum Cost Homomorphism Problem. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010), pp. 657–668 (2010)
39. Topkis, D.M.: Minimizing a submodular function on a lattice. Operations Research 26(2), 305–321 (1978)

Streaming Algorithms for Recognizing Nearly Well-Parenthesized Expressions

Andreas Krebs[1], Nutan Limaye[2], and Srikanth Srinivasan[3,⋆]

[1] University of Tübingen, Germany
mail@krebs-net.de
[2] Indian Institute of Technology, Bombay, India
nutan@cse.iitb.ac.in
[3] Institute for Advanced Study, USA
srikanth@math.ias.edu

Abstract. We study the streaming complexity of the membership problem of 1-turn-Dyck_2 and Dyck_2 when there are a few errors in the input string.

1-turn-Dyck_2 with errors: We prove that there exists a randomized one-pass algorithm that given x checks whether there exists a string $x' \in$ 1-turn-Dyck_2 such that x is obtained by flipping at most k locations of x' using:

- $O(k \log n)$ space, $O(k \log n)$ randomness, and $\mathrm{poly}(k \log n)$ time per item and with error at most $1/n^{\Omega(1)}$.
- $O(k^{1+\epsilon} + \log n)$ space for every $0 \leq \epsilon \leq 1$, $O(\log n)$ randomness, $O((\log^{O(1)} n + k^{O(1)}))$ time per item, with error at most $1/8$.

Here, we also prove that any randomized one-pass algorithm that makes error at most k/n requires at least $\Omega(k \log(n/k))$ space to accept strings which are exactly k-away from strings in 1-turn-Dyck_2 and to reject strings which are exactly $k+2$-away from strings in 1-turn-Dyck_2. Since 1-turn-Dyck_2 and the Hamming Distance problem are closely related we also obtain new upper and lower bounds for this problem.

Dyck_2 with errors: We prove that there exists a randomized one-pass algorithm that given x checks whether there exists a string $x' \in \mathsf{Dyck}_2$ such that x is obtained from x' by changing (in some restricted manner) at most k positions using:

- $O(k \log n + \sqrt{n \log n})$ space, $O(k \log n)$ randomness, $\mathrm{poly}(k \log n)$ time per element and with error at most $1/n^{\Omega(1)}$.
- $O(k^{1+\epsilon} + \sqrt{n \log n})$ space for every $0 < \epsilon \leq 1$, $O(\log n)$ randomness, $O((\log^{O(1)} n + k^{O(1)}))$ time per element, with error at most $1/8$.

1 Introduction

The data streaming model was introduced in the seminal work of Alon et al. [2]. This model naturally arises in situations where the input data is massive and rereading the input bits is expensive. The main parameters that play a role in

⋆ Supported by NSF grants DMS-0835373 and CCF-0832797.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 412–423, 2011.

designing algorithms in such situations are: the space used by the algorithm, and the number of passes made over the input. An algorithm is said to be an efficient data streaming algorithm, if the space used by the algorithm is substantially lesser than the length of the input (sublinear in the length of the input) and the number of passes is independent of the length of the input. Many variants of this basic model have been studied. (See for example [13] for a survey.)

The membership testing for well-paranthesized strings has been considered in the past. We denote the set of words with balanced parentheses of l different types by Dyck_l. It is known that there is a $O(\log n)$ space deterministic algorithm for testing membership in Dyck_l. (In fact the problem is known to be in TC^0 [5].) The problem has been considered from property testing perspective (see for example [1], [15]). Recently, the problem was considered in the streaming model by Magnify et al. [12]. It was proved that there is a randomized one-pass streaming algorithm that takes space $O(\sqrt{n \log n})$ and tests membership in Dyck_l. They also gave an efficient $O(\log^2 n)$ space algorithm which makes bidirectional pass (one forward and one backward pass) on the input. They also proved a lower bound of $\Omega(\sqrt{n})$ for any randomized streaming algorithm that makes a single unidirectional (only forward) pass. Chakrabarti et al. [7] and Jain et al. [10] considered the lower bound problem for unidirectional multi-pass randomized algorithms. In [7] it was proved that any T-pass (all passes made in the same direction) randomized algorithm requires $\Omega(\sqrt{n}/T \log \log n)$ space. Whereas [10] proved $\Omega(\sqrt{n}/T)$ space lower bound for the same. In [4] membership testing for other classes of languages was considered. In [3] it was proved that any randomized T pass algorithm (passes made in any direction) for testing membership in a deterministic context-free language requires $\Omega(n/T)$ space.

We consider a slightly general version of the membership testing problem for Dyck_l. Let Σ_l denote a set of l pairs of matching parentheses. We say that an opening parenthesis is *corrupted* if it is replaced by another opening parenthesis. Similarly, a closing parenthesis is *corrupted* if it is replaced by another closing parenthesis. For a language $L \in \Sigma_l^*$, let $\Delta^{\leq k}(L)$ be defined as the set of words over Σ_l^* obtained by corrupting at most k indices of any word in L. In this paper, we consider the membership problem for $\Delta^{\leq k}(\mathsf{Dyck}_l)$ and $\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_l)$, where $1\text{-turn-}\mathsf{Dyck}_2 = \{w\overline{w}^R \mid w \in \{(,[\}^n \ n \geq 1\}$ Here, $\overline{w}$ is the string obtained from w by replacing an opening parenthesis by its corresponding closing parenthesis and w^R is the reverse of w.

Accepting strings with at most k errors is a well-studied problem in many models of computation. In the streaming model, the problem has been studied in the past (see for example Cao et al. [6]). But we believe that the problem needs further investigation; this being the primary goal of this paper.

We observe that the membership testing problem for $\Delta^{\leq k}(\mathsf{Dyck}_l)$ ($\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_l)$) reduces to the membership testing problem of $\Delta^{\leq k}(\mathsf{Dyck}_2)$ ($\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_2)$, respectively). We give a simple fingerprinting algorithm for $\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_2)$ that uses $O(k \log n)$ bits of space and randomness. The space requirements of this algorithm are nearly optimal (because of a communication complexity lower bound of [9]) but the randomness requirements are not. We

consider the question of derandomizing the above. The question of derandomizing streaming algorithms has been considered in the past (see for example [8],[14],[16],[18]). We show that the algorithm can be modified to work with just $O(\log n)$ bits of randomness, incurring a small penalty in the amount of space used. We then consider similar questions for the more general problem $\Delta^{\leq k}(\mathsf{Dyck}_2)$.

The following table summarizes our algorithmic results:

Problem	Alg.	Space	Randomness	Error	Time (per element)
1-turn-Dyck_2	1	$O(k \log n)$	$O(k \log n)$	$1/\operatorname{poly}(n)$	$\operatorname{poly}(k \log n)$ [1]
	2	$O(k^{1+\epsilon} + \log n)$ [2]	$O(\log n)$	1/8	$O((\log n)^{O(1)} + k^{O(1)})$
Dyck_2	3	$O(k \log n + \sqrt{n \log n})$	$O(k \log n)$	$1/\operatorname{poly}(n)$	$\operatorname{poly}(k \log n)$ [1]
	4	$O(k^{1+\epsilon} + \sqrt{n \log n})$ [2]	$O(\log n)$	1/8	$O((\log n)^{O(1)} + k^{O(1)})$

In all the algorithms in the table above, we assume that the length of the input stream is known.

Using Algorithm 1, we can deduce the number of errors as well as their locations. Using a combination of the algorithm for membership testing of Dyck_2 due to [12] (which we refer to as MMN algorithm) and Algorithm 1, it is easy to get a membership testing algorithm for $\Delta^{\leq k}(\mathsf{Dyck}_2)$. However, such an algorithm uses $O(k\sqrt{n \log n})$ space. In order to achieve the claimed bound, we modify their algorithm for testing membership in Dyck_2 and use that in conjunction with Algorithm 1. In our algorithm, we do not need to store the partial evaluations of polynomials on the stack.

Algorithms 2 and 4 are inspired by the communication complexity protocols of Yao [20] and Huang et al. [9]. A mere combination of their ideas, however, is not enough to get the required bounds. The crucial observation here is that Yao's protocol can be derandomized by using a combination of small-bias distributions and distributions that fool DNF formulae. As this requires very few random bits, we get the desired derandomization. These algorithms are also better as compared to Algorithm 1 and 3 in terms of their time complexity. For Algorithm 2, we first prove that it suffices to give an efficient algorithm for $\mathsf{Ham}_{n,k}$, where $\mathsf{Ham}_{n,k}(x, y)$ for $x, y \in \{0,1\}^n$ is 1 if and only if the Hamming distance between x and y is at most k.

Finally, we consider the question of optimality. We prove that any algorithm that makes k/n error requires $\Omega(k \log(n/k))$ space to test membership in $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$ by proving a lower bound on $\mathsf{Ham}_{n,k}$. The two problems are related as follow: Let $w \in \Sigma^{2n}$ and let $w = uv$ where $u \in \{(,[\}^n$ and $v \in \{),]\}^n$. If (and) are both mapped to 0 and [and] are both mapped to 1 to obtain x, y from u, v then it is easy to see that $uv \in \Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$ if and only if $\mathsf{Ham}_{n,k}(x, y) = 1$.

[1] In the case of $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_l)$, this is the exact time per item. However, for $\Delta^{\leq k}(\mathsf{Dyck}_l)$ it is the time per item on average. In the latter case, the algorithm first reads a block and then uses $O(\operatorname{poly}(k \log n))$ time per element of the block. Therefore, the time per block is $O(\operatorname{poly}(k \log n)\sqrt{n/\log n})$. Both algorithms use an extra post-processing time of $n^{k+O(1)}$.

[2] for all $0 < \epsilon < 1$

The problem $\mathsf{Ham}_{n,k}$ was considered in [20], [9], in simultaneous message model. In [9], a lower bound (in fact, a quantum lower bound) of $\Omega(k)$ was proved for the problem. Their lower bound holds even for constant error protocols. To best of our knowledge no better lower bound is known for the problem. We improve on their lower bound by a $\log(n/k)$ factor under the assumption that the communication protocol is allowed to make small error. Our lower bound can be stated as follows:

Theorem 1. *Given two strings $x, y \in \{0,1\}^n$ such that either the Hamming distance between x, y is exactly k or exactly $k+2$, any randomized one-pass algorithm that makes error k/n requires space $\Omega(k \log(n/k))$ to decide which one of the two cases is true for the given x, y pair.*

For the lower bound, we use the result of Jayram et al. [17]. Intuitively, the hardest case seems to be to distinguish between exactly k and exactly $k+2$ errors. The main advantage of our lower bound proof is that it formalizes this intuition. Moreover, as our algorithm in Section 3 shows, this bound is tight up to a constant factor for $n \geq k^2$ (indeed, for $n \geq k^{1+\epsilon}$ for any $\epsilon > 0$). This bound is not tight in all situations though, for example when $n \gg k$ but the error is constant. Also, it does not apply to multi-pass algorithms. However, this is better than the earlier bounds [9] by a factor of $\log(n/k)$ for small error.

The rest of the paper is organized as follows: in the next section we give some basic definitions which will be used later in the paper. In Section 3 we give the two randomized one-pass algorithms for testing membership in $\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_2)$. In 4 we discuss our results regarding testing membership in $\Delta^{\leq k}(\mathsf{Dyck}_2)$. Our lower bound result is presented in Section 5. Due to lack of space, many proofs are omitted from this extended abstract.

2 Definitions and Preliminaries

Definition 1. *Given positive integers ℓ, n, m, and s, a function $F : \{0,1\}^s \to [m]^n$ is an ℓ-wise independent hash family if given any* distinct *$i_1, i_2, \ldots, i_\ell \in [n]$ and any (not necessarily distinct) $j_1, j_2, \ldots, j_\ell \in [m]$, we have $\Pr_{r \in \{0,1\}^s}[F(r)(i_1) = j_1 \wedge F(r)(i_2) = j_2 \wedge \cdots \wedge F(r)(i_\ell) = j_\ell] = \frac{1}{m^\ell}$ where $F(r)$ is interpreted as a function mapping $[n]$ to $[m]$ in the obvious way.*

Lemma 1. *[19] For any ℓ, n, m, there is an ℓ-wise independent hash family $F : \{0,1\}^s \to [m]^n$, with $s = O(\ell \log(n+m))$ with the property that there is a deterministic algorithm which, on input $r \in \{0,1\}^s$ and $i \in [n]$, computes $F(r)(i)$ in time* $\mathrm{poly}(s)$ *using space $O(s)$.*

3 Equivalence with Errors

In this section, we consider the problem of testing membership in $\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_l)$. Magniez et al. [12], showed that it suffices to design efficient streaming algorithms for testing membership in Dyck_2 in order to get efficient streaming algorithms for testing membership in Dyck_l. Formally,

Lemma 2 ([12]). *If there is a one-pass streaming algorithm for testing membership in* Dyck_2 *that uses space* $s(n)$ *for inputs of length* n*, then there is a one-pass streaming algorithm for testing membership in* Dyck_l *that uses space* $O(s(n \log l))$ *for inputs of length* n*.*

We first prove a lemma similar to Lemma 2, to state that it suffices to design an efficient streaming algorithm for $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$ ($\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$) in order to get an efficient streaming algorithms for $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_l)$ (respectively, $\Delta^{\leq k}(\mathsf{Dyck}_l)$).

Lemma 3. *If there is a one-pass streaming algorithm for testing membership in* $\Delta^{\leq 2k}(\text{1-}\textit{turn}\text{-}\mathsf{Dyck}_2)$ *(*$\Delta^{\leq 2k}(\mathsf{Dyck}_2)$*) that uses space* $s(n)$ *for inputs of length* n*, then there is a streaming algorithm for testing membership in* $\Delta^{\leq k}(\text{1-}\textit{turn}\text{-}\mathsf{Dyck}_l)$ *(*$\Delta^{\leq k}(\mathsf{Dyck}_l)$*) that uses space* $O(s(nl))$ *for inputs of length* n*.*

Let $D^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$ be the set of string obtained by changing at most k symbols of words in $\text{1-turn-}\mathsf{Dyck}_2$. Assuming that the length of the string is known, the membership testing for $D^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$ (which is more general than $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$) can also be handled by the techniques introduced in the paper. If the input string has opening parenthesis in the first half of the string, then it is considered to be an error. It is easy to keep track of such errors.

We now note that $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$ on inputs of length n reduces to the problem $\mathsf{Ham}_{n/2,k}$.

Lemma 4. *There is a deterministic one-pass streaming algorithm that uses space* $O(\log n)$ *and time* $O(n)$*, which given as input a string* $w \in \{(,[,),]\}^n$*, outputs a pair of strings* $x, y \in \{0,1\}^{n/2}$ *and either accepts or rejects. If the algorithm rejects, we have* $w \notin \Delta^{\leq k}(\text{1-}\textit{turn}\text{-}\mathsf{Dyck}_2)$*. Otherwise, we have* $\Delta(x, y^R) \leq k$ *iff* $w \in \Delta^{\leq k}(\text{1-}\textit{turn}\text{-}\mathsf{Dyck}_2)$*.*

The above lemma shows that it suffices to come up with a streaming algorithm for the Hamming distance problem to solve the problem $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$. Once we have such an algorithm, we simply run the above reduction on an input $w \in \{(,[,),]\}^n$, and obtain strings x, y^R, which we feed in as input to the algorithm for $\mathsf{Ham}_{n/2,k}$ (of course, if the reduction rejects, we reject the input). Though $\mathsf{Ham}_{n,k}$ is only a minor restatement of $\Delta^{\leq k}(\text{1-turn-}\mathsf{Dyck}_2)$, we prefer to work with this problem because of its cleaner definition.

Theorem 2. *For any* k *and any constant* $c > 0$*, there is a one-pass randomized streaming algorithm which, when given as input strings* $(x, y^R) \in \{0,1\}^n \times \{0,1\}^n$*, that accepts with probability* 1 *if* $\Delta(x, y) \leq k$ *and rejects with probability* $1 - 1/n^c$ *if* $\Delta(x, y) > k$*. The algorithm also detects the locations where* x *and* y *differ with probability at least* $1 - 1/n^c$ *if* $\Delta(x, y) \leq k$*. The algorithm uses* $O(k \log n)$ *space and* $O(k \log n)$ *randomness. The time required by the algorithm is* $\text{poly}(k \log n)$ *per item plus* $n^{k+O(1)}$ *for post-processing.*

Proof. The algorithm uses a fingerprinting strategy and is directly inspired by the standard randomized communication complexity protocol for the Equality

problem (see [11], for example). Fix a field $\mathbb{F}_{2^\ell}$, where the exact value of ℓ will be determined later. We call a polynomial $p(z) \in \mathbb{F}_{2^\ell}[z]$ *boolean* if all of its coefficients are 0 or 1. The *weight* of a boolean polynomial p will be the number of non-zero coefficients of p.

We think of $w \in \{0,1\}^n$ as defining a boolean polynomial $p_w(z) \in \mathbb{F}_{2^\ell}[z]$ as follows: $p_w(z) = \sum_{i=1}^{n} w_i z^{i-1}$, where w_i denotes the ith bit of w. Note that the polynomial $q_{x,y}(z) := p_x(z) + p_y(z)$ is a boolean polynomial of weight exactly $\Delta(x,y)$. We check that $\Delta(x,y) \leq k$ by evaluating $q_{x,y}(z)$ at a random $\alpha \in \mathbb{F}_{2^\ell}$. More formally, the algorithm is:

- Pick $\alpha \in \mathbb{F}_{2^\ell}$ uniformly at random.
- Check if $q_{x,y}(\alpha) = p(\alpha)$ for any boolean polynomial p of degree less than n and weight at most k. If not, REJECT.
- If the above does hold for *some* boolean polynomial of weight at most k, ACCEPT and pick *any* such polynomial $p(z) = \sum_i p_i z^i$. Let $S = \{i \mid p_i \neq 0\}$ be the support of p. Output S as the estimate of points where x and y differ.

Let us first establish the correctness of the above algorithm (assuming ℓ is large enough). Clearly, if $\Delta(x,y) \leq k$, then $q_{x,y}(z)$ is a polynomial of weight at most k and the algorithm always accepts. The algorithm can only err if: (a) $\Delta(x,y) > k$ or (b) $\Delta(x,y) \leq k$ but the algorithm outputs the wrong set of indices as its estimate of where x and y differ. In either case, there is a boolean polynomial $p(z)$ of degree less than n and weight at most k such that $q_{x,y}(z) \neq p(z)$ but $q_{x,y}(\alpha) = p(\alpha)$. For any fixed polynomial $p(z)$, this happens with probability at most $n/2^\ell$ by the Schwartz-Zippel Lemma. Since the number of polynomials of weight at most k is at most n^k, the probability that there exists *any* such polynomial p is bounded by $n^{k+1}/2^\ell$. Choosing $\ell = O(k \log n)$, we can reduce this error to $1/n^c$ as claimed.

Computing $q_{x,y}(\alpha)$ can easily be done in a one-pass fashion using space $O(\ell) = O(k \log n)$ and time $\mathrm{poly}(k \log n)$ per item. After reading the stream, we need to cycle through the n^k boolean polynomials p of weight at most k and compute the values they take at input $\alpha \in \mathbb{F}_{2^\ell}$, which can also be done in space $O(k \log n + \ell) = O(k \log n)$ and time $n^k \,\mathrm{poly}(k \log n) = n^{k+O(1)}$, as claimed above. This completes the proof of the theorem. □

3.1 A Randomness-Efficient Streaming Algorithm for Hamming Distance

Above, we showed that $\mathsf{Ham}_{n,k}$ can be computed using space $O(k \log n)$ and $O(k \log n)$ random bits. Are these parameters optimal? As we will show later in Section 5, the bound on space is nearly optimal. However, we show in this section that the number of random bits can be significantly reduced, if one is willing to use a small amount of additional space. The ideas in this section go back to the results of Yao [20] and Huang et al. [9], who designed efficient randomized communication complexity protocols for the two-party problem of checking if the Hamming Distance between x and y is at most k.

Let $\mathsf{PHam}_{n,k,l} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a partial function, defined as follows: On input (x, y^R) it evaluates to 0 if the hamming distance between x

and y is greater than or equal to l, it evaluates to 1 if the distance is less than or equal to k and is not defined on other inputs.

Theorem 3. *For every constant $0 \leq \epsilon \leq 1$ there is a randomized one-pass streaming algorithm that computes $\mathsf{Ham}_{n,k}$ using $O(k^{1+\epsilon} + \log n)$ space and $O(\log n)$ randomness and errs with probability bounded by $1/8$. The time taken by the algorithm is $O((\log n)^{O(1)} + k^{O(1)})$ per item.*

Proof Strategy: In order to prove the above theorem, we divide the problem into two parts. Assuming there are at most $2k$ errors, we design an algorithm that computes $\mathsf{Ham}_{n,k}$ correctly with high probability. We call this the inner algorithm. We design another randomized algorithm to compute $\mathsf{PHam}_{n,k,2k}$ with high probability. We call this algorithm the outer algorithm.

We output 1, that is we declare that the number of indices where x and y differ is at most k, if and only if both the inner and the outer algorithms output 1. If x and y differ on more than $2k$ indices, then the outer algorithm will output 0 with high probability. The answer of the inner algorithm will not be reliable in this case. Where as if they differ on more than k but less than $2k$ places then the inner algorithm will output 0 with high probability. Let γ_1, γ_2 be errors in inner and outer algorithms respectively. Then the overall error γ is bounded by $\gamma_1 + \gamma_2$. We prove that both γ_1 and γ_2 are bounded by $\gamma/2$ for a small constant γ.

Inner algorithm

Definition 2. *Given, $k, n \in \mathbb{N}$, we say that an element $w \in ([k] \times \{0,1\})^n$ is an* XOR representation of length *n of a string $a \in \{0,1\}^k$ if for each $j \in [k]$, we have $a_j = \bigoplus_{i:w_i=(j,u_i)} u_i$.*

We think of the XOR representation as streaming updates of a structure over $\mathbb{F}_2$.

Lemma 5. *There is a randomized one-pass streaming algorithm which given input $x, y^R \in \{0,1\}^n$ such that $\Delta(x,y) \leq 2k$ computes an XOR representation of length n of $a, b \in \{0,1\}^{16k^2/\gamma}$ such that with probability $1 - \gamma/4$, $\mathsf{Ham}_{n,k}(x,y) = \mathsf{Ham}_{16k^2/\gamma,k}(a,b)$ The algorithm uses $O(\log n)$ bits of randomness, $O(\log n)$ space, and $(\log(n/\gamma))^{O(1)}$ time per item.*

Proof. The proof is simple. We pick a random hash function h from a pairwise independent hash family of functions mapping $[n]$ to $[16k^2/\gamma]$. We think of h as dividing the n indices of x and y into $16k^2/\gamma$ buckets.

Given x, y such that $\Delta(x,y) \leq 2k$, call index i *good* if $x_i \neq y_i$. Given two good indices $i \neq j$, the probability that h maps both of them to the same bucket is at most $\gamma/16k^2$. A simple union bound tells us that with probability $1 - \gamma/4$, all the good indices are mapped to different buckets.

After having picked h, the streaming algorithm computes the XOR representations of a, b defined as follows: for any j, a_j is the XOR of the bits of x whose indices are in the jth bucket; formally, $a_j = \bigoplus_{i:h(i)=j} x_i$; the string b is similarly related to y. Clearly, if h maps the good indices to different buckets, then $a_j \neq b_j$ iff the jth bucket contains a good index and hence $\Delta(a,b) = \Delta(x,y)$. On reading

the input bit x_i, the algorithm computes the bucket $j = h(i)$ and writes down (j, x_i) which in effect updates the jth bit of a. In a similar way, when scanning y, the algorithm updates b.

The space and randomness requirements are easily analyzed. Picking a random hash function h from a pairwise independent family as above requires $O(\max\{\log n, \log(k^2/\gamma)\}) = O(\log(n/\gamma))$ random bits by Lemma 1. The algorithm needs to store these random bits only. Computing $h(j)$ for any j only requires space $O(\log n/\gamma)$. Finally, the processing time per element is $O(\text{poly}(\log(n/\gamma))$. □

We will use the above algorithm as a streaming reduction and solve the problem using the algorithm of Lemma 6.

Lemma 6. *For any n, k and every constant $0 < \epsilon < 1$ and $\gamma \geq \frac{1}{k^{O(1)}}$, there is a randomized one-pass streaming algorithm which, on inputs $a, b \in \{0,1\}^{16k^2/\gamma}$ accepts iff $\Delta(a, b) \leq k$ with error probability at most $\frac{\gamma}{4}$. The algorithm uses $O(\log k)$ bits of randomness, $O(k^{1+\epsilon} + \log n)$ space, and time per element is $k^{O(1)}$. The algorithm expects its inputs a, b to be given in terms of XOR representations of length n.*

Proof Sketch. Fix a positive constant $\delta < \epsilon$. Let $h : [16k^2/\gamma] \to [k^{1+\delta}]$ be a function picked at random. Let $j \in [k^{1+\delta}]$ be a fixed bucket. We have $\Pr[h(i) = j] = \frac{1}{k^{1+\delta}}$.

Define a set I of indices as follows: if $\Delta(a, b) \leq k$, then let I be the indices where a and b differ; otherwise, let I be any set of $k + 1$ indices where a and b differ. Let u be the size of a subset U of I. We have $\Pr[h(U) = j] \leq \frac{1}{(k^{1+\delta})^u}$. By a union bound over U of size u, $\Pr[\exists U : h(U) = j] \leq \frac{\binom{k+1}{u}}{(k^{1+\delta})^u} \leq \frac{(k+1)^u}{(k^{1+\delta})^u} \leq \frac{1}{k^{\delta u/2}}$. Therefore, since there are at most k^2 buckets, $\Pr[\exists U \; \exists \text{ a bucket } j : \; h(U) = j] \leq 1/k^{\delta u/2-2}$.

We want this probability to be less than $\gamma/8$. Therefore we select $u = O(1/\delta + \frac{\log(1/\gamma)}{\log k}) = O(1)$, where the last equality uses $\gamma \geq \frac{1}{k^{O(1)}}$ and δ is a constant.

Note that the above argument works if we used a function from a u-wise independent family of functions rather than a random function. This requires only $O(u \log(k/\gamma)) = O(\log k)$ bits of randomness and space $O(\log(k/\gamma))$ by Lemma 1. Hereafter, we assume that we have picked a hash function h from this family so that each bucket $j \in [k^{1+\delta}]$ contains at most u indices from I. Let B_j^a and B_j^b be the buckets formed by hashing a and b respectively, where $1 \leq j \leq k^{1+\delta}$.

Given boolean strings a', b' of the same length, define $\Delta_u(a', b')$ to be $\min\{\Delta(a', b'), u\}$. We will compute the function $F(a, b) = \sum_{j \in [k^{1+\delta}]} \Delta_u(B_j^a, B_j^b)$ and accept if the value computed is at most k. It can easily be seen, using the properties of h, that this computes $\mathsf{Ham}_{16k^2/\gamma, k}(a, b)$.

Using the ideas of the algorithm of Theorem 2, it is easily seen that for each j, there is a streaming algorithm that uses space and randomness $O(\log k)$ and computes $\Delta_u(B_j^a, B_j^b)$ with error at most $\gamma/8k^{1+\delta}$. Due to lack of space, we omit the description of the algorithm and its proof.

Taking a union bound over buckets, the error of this phase of the algorithm is at most $\gamma/8$. The overall error of the algorithm is at most $\gamma/4$. The space used per bucket and the number of random bits used is at most $O(\log k)$. Adding up over all buckets, the space used is bounded by $O(k^{1+\delta} \log k)$, which is at most $O(k^{1+\epsilon})$. The time taken by the algorithm to compute the values $\{q_j(\alpha) \mid j \in [k^{1+\delta}]\}$ is $k^{O(1)}$. Finally, checking if each $q_j(\alpha)$ evaluates to the same value as a polynomial of weight at most u takes time $k^{O(u)} = k^{O(1)}$. □

Setting γ to be a small constant in the Lemmas 5 and 6, we see that the space taken by the Inner algorithm overall is $O(k^{1+\epsilon}+\log n)$, the amount of randomness used is $O(\log n)$ and the time taken per item is $O((\log n)^{O(1)} + k^{O(1)})$.

Outer algorithm. The outer algorithm solves $\mathsf{PHam}_{n,k,2k}$, and this completes the algorithm for the $\mathsf{Ham}_{n,k}$ problem. We omit the description of the algorithm and proof of the below lemma from this extended abstract.

Lemma 7. *There is a randomized one-pass streaming algorithm that computes* $\mathsf{PHam}_{n,k,2k}$ *correctly with probability* $1-\gamma/2$ *using* $O(\log n \log(1/\gamma))$ *bits of space and randomness and time per item* $(\log n)^{O(1)} \log(1/\gamma)$.

4 Accepting Dyck_2 with Errors

In this section we consider the membership problem of $\Delta^{\leq k}(\mathsf{Dyck}_2)$. We assume that disregarding the type of the brackets the string is well-matched. We only consider the kind of errors where an opening(closing) parenthesis of one type is replaced by an opening(closing, respectively) parenthesis of the other type. We prove the following theorem:

Theorem 4. *For any k there exists a constant $c > 0$, there is a randomized one-pass algorithm such that, given a string $w \in \Sigma^n$, if $w \in \Delta^{\leq k}(\mathsf{Dyck}_2)$ then with probability at least $1 - 1/n^c$ it accepts w and if $w \notin \Delta^{\leq k}(\mathsf{Dyck}_2)$, then with probability $1 - 1/n^c$ it rejects w. The algorithm uses $O(k \log n + \sqrt{n \log n})$ space and takes* $\mathrm{poly}(k \log n)$ *time per item and $n^{k+O(1)}$ time for post-processing.*

It is easy to see that combining the ideas from [12] and from the previous section, we can accept $\Delta^{\leq k}(\mathsf{Dyck}_2)$ using $O(k\sqrt{n \log n})$ space. But for the bound stated in Theorem 4, we need more work.

In [12] a one-pass randomized streaming algorithm was given for the membership testing of Dyck_2. We refer to that as the MMN algorithm. We make one change to the MMN algorithm. We use the stack only to store indices from the set $[n]$, and do not store the partial evaluations of a polynomial on the stack.

Divide the input into $\sqrt{n/\log n}$ blocks of length $\sqrt{n \log n}$ each. In each block, check for balanced parentheses and if there are less than or equal to $k - Err$ mis-matches, then reduce the remaining string to a string (possibly empty) of closing parentheses followed by a string (possibly empty) of opening parentheses. Here, Err is a counter that maintains the number of mismatches found so far. If Err exceeds k, then halt the algorithm and reject.

Let x denote the reduced string obtained by matching parentheses within each block. (Note that this can be done in a streaming fashion.)

Observation 1. *Note that no two opening (or two closing) parentheses have the same index. Also, an opening parenthesis has the same index as another closing parenthesis if and only if they form a matching pair in the string obtained from the input string by disregarding the type of the parentheses.*

For example in the input $((][])[]$ *the indices of the opening parentheses would be* $(^1(^2[^3])[^4])$ *and the indices of the opening and closing parentheses would be* $(^1(^2[^3]^3)^2[^4]^4)^1$*. If we reorder the input such that all opening parentheses are in the first half with ascending index and the closing parentheses are in the second half with descending input we get* $(^1(^2[^3[^4]^4]^3)^2)^1$*.*

For every opening parenthesis the index is the number of open parentheses before. For the closing parentheses we keep a stack of intervals of open parentheses that are still unmatched. Since we process the input in at most $\sqrt{n/\log n}$ blocks we need at most $\sqrt{n/\log n}$ stack depth.

So we can compute the index of the parentheses, and now we show how to use this to compute $\Delta^{\leq k}(\mathsf{Dyck}_2)$. Now assume that, at any stage i, we have the index of the input symbol x_i. Let the sign of the opening parentheses be $+1$ and that of closing parentheses be -1. We think of the reduced string $x \in \{(,[,),]\}^*$ as a string over $\{0,1\}^*$ by replacing every occurrence of '(' and ')' by a 0 and every occurrence of '[' and ']' by a 1. We think of this string defining a Boolean polynomial $p_x(z) = \sum_i sign(x_i) x_i z^{\text{index of } x_i}$. Due to Observation 1, it is easy to see that the weight of the polynomial p_x is at most $k - Err$ if and only if $w \in \Delta^{\leq k}(\mathsf{Dyck}_2)$. We now check whether $w \in \Delta^{\leq k}(\mathsf{Dyck}_2)$ by evaluating p_x at a random $\alpha \in \mathbb{F}_{2^l}$. Assuming that we know how to compute index of x_i at step i, we can evaluate p_x as in the proof of Theorem 2.

Given below is the algorithm that uses the index finding procedure as described above, and evaluates polynomial p_x at a random location to test membership of w in $\Delta^{\leq k}(\mathsf{Dyck}_2)$. In addition to th space required for computing the indices, $O(l)$ bits are required to store evaluation of p_x. But this does not need to be stored multiple times on the stack. Therefore, the algorithm uses $O(l + \sqrt{n \log n}) = O(k \log n + \sqrt{n \log n})$ space.

The proof of correctness and the error analysis are similar to the proof of Theorem 2. Thus we get Theorem 4.

Reducing the randomness. The ideas used in reducing randomness for $\Delta^{\leq k}(1\text{-turn-}\mathsf{Dyck}_2)$ also work for reducing randomness in the membership testing of $\Delta^{\leq k}(\mathsf{Dyck}_2)$. Here, instead of hashing the input positions, we hash the indices using the random hash functions. For computing the indices, we use the procedure described above. Instead of computing polynomials, we compute hashes and follow the steps as in Sections 3.1, 3.2, and 3.3.

Theorem 5. *For every constant* $0 \leq \epsilon \leq 1$*, there is randomized one-pass algorithm that tests membership in* $\Delta^{\leq k}(\mathsf{Dyck}_2)$ *using* $O(k^{1+\epsilon} + \sqrt{n \log n})$ *space,* $O(\log n)$ *randomness,* $O(\log^{O(1)} n + k^{O(1)})$ *time and errs with probability* $1/8$*.*

5 Lower Bound

We will show a lower bound for $\mathsf{PHam}_{n,k,k+2}$ by reducing the augmented indexing problem $\mathsf{IND}^a_{\mathcal{U}}$ (see [17]) to it.

Let $\mathcal{U} \cup \{\perp\}$ denote a large input domain, where $\perp \notin \mathcal{U}$. Define the problem $\mathsf{IND}^a_{\mathcal{U}}$ as follows: Alice and Bob receive inputs $x = (x_1, x_2, \ldots, x_N) \in \mathcal{U}^N$ and $y = (y_1, y_2, \ldots, y_N) \in (U \cup \{\perp\})^N$, respectively. The inputs have the following promise: There is some unique $i \in [N]$, such that $y_i \in \mathcal{U}$, and for $j < i$: $x_j = y_j$, and for $j > i$: $y_j = \perp$. The problem $\mathsf{IND}^a_{\mathcal{U}}$ is defined only over such promise inputs and $\mathsf{IND}^a_{\mathcal{U}}(x, y) = 1$ if and only if $x_i = y_i$.

In [17, Corollary 3.1] they proved the following result:

Theorem 6 ([17]). *Any randomized one-way communication protocol that makes $\delta = 1/4|\mathcal{U}|$ error requires $\Omega(N \log 1/\delta)$ bits of communication.*

We use this result and prove a lower bound for $\mathsf{PHam}_{n,k,k+2}$.

Let $|\mathcal{U}| = n/k$. Let $f_A : \mathcal{U} \to \{0,1\}^{3n/k}$, and $f_B : (\mathcal{U} \cup \{\perp\}) \to \{0,1\}^{\frac{3n}{k}}$ be encoding functions defined as follows: $f_A : u_l \mapsto A_1 A_2 \ldots A_{n/k}$, where $A_j = 110$ if $j = l$ and 000 otherwise. $f_B : u_l \mapsto B_1 B_2 \ldots B_{n/k}$, where $B_j = 011$ if $j = l$ and 000 otherwise. $f_B(\perp) = 0^{3n/k}$.

On promise inputs $x, y \in \mathcal{U}^k$, let $F_A(x)$ and $F_B(y)$ be defined as $f_A(x_1) f_A(x_2) \ldots f_A(x_k)$ and $f_B(y_1) f_B(y_2) \ldots f_B(y_k)$, respectively.

Under this encoding, it is easy to see that $\mathsf{PHam}_{3n,2k,2k+2}(F_A(x), F_B(y)) = 1$ if and only if $\mathsf{IND}^a_{\mathcal{U}}(x, y) = 1$. Suppose $i+1$ is the first position where $\perp$ appears in y. For each $j < i$ we have $x_j = y_j$ so the Hamming distance of $f_A(x_j)$ and $f_B(y_j)$ is 2. Also for every position $j > i$ we have $y_j = \perp$ and hence $f_B(y_j) = 0^{3n/k}$, which results in a Hamming distance of 2 between $f_A(x_j)$ and $f_B(y_j)$. So the Hamming distance between $F_A(x)$ and $F_B(y)$ is $2(k-1)$ plus the Hamming distance between $f_A(x_i)$ and $f_B(y_i)$, which is 2 iff $x_i = y_i$ and 4 iff $x_i \neq y_i$ (since $x_i, y_i \in \mathcal{U}$). Therefore we get the following lower bound:

Theorem 7 (Theorem 1 restated). *Any randomized one-way protocol that makes at most k/n error and computes $\mathsf{PHam}_{3n,2k,2k+2}$, requires $\Omega(k \log(n/k))$ bits. In fact the lower bound holds for distinguishing between the case $\Delta(x, y) = 2k$ and $\Delta(x, y) = 2k + 2$.*

By Theorem 2 this bound is optimal when $n \geq k^2$ (and in fact when $n \geq k^{1+\epsilon}$, for constant $\epsilon > 0$).

References

1. Alon, N., Krivelevich, M., Newman, I., Szegedy, M.: Regular languages are testable with a constant number of queries. SIAM Journal on Computing, 645–655 (1999)
2. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC 1996, pp. 20–29. ACM, New York (1996), http://doi.acm.org/10.1145/237814.237823

3. Babu, A., Limaye, N., Radhakrishnan, J., Varma, G.: Streaming algorithms for language recognition problems. Tech. Rep. arXiv:1104.0848, Arxiv (2011)
4. Babu, A., Limaye, N., Varma, G.: Streaming algorithms for some problems in log-space. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 94–104. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-13562-0_10, doi:10.1007/978-3-642-13562-0-10
5. Barrington, D.A.M., Corbett, J.: On the relative complexity of some languages in NC^1. Information Processing Letters 32(5), 251 (1989)
6. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: 2006 SIAM Conference on Data Mining, pp. 328–339 (2006)
7. Chakrabarti, A., Cormode, G., Kondapally, R., McGregor, A.: Information cost tradeoffs for augmented index and streaming language recognition. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, FOCS 2010, pp. 387–396 (2010)
8. Ganguly, S.: Data stream algorithms via expander graphs. In: Proceedings of the 19th International Symposium on Algorithms and Computation, pp. 52–63 (2008)
9. Huang, W., Shi, Y., Zhang, S., Zhu, Y.: The communication complexity of the hamming distance problem. Information Processing Letters 99(4), 149–153 (2006)
10. Jain, R., Nayak, A.: The space complexity of recognizing well-parenthesized expressions. Tech. Rep. TR10-071, Electronic Colloquium on Computational Complexity (April 19, 2010) (revised July 5, 2010), http://eccc.hpi-web.de/
11. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, New York (2006)
12. Magniez, F., Mathieu, C., Nayak, A.: Recognizing well-parenthesized expressions in the streaming model. In: STOC 2009 (2009)
13. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science 1(2) (2005)
14. Nisan, N.: Pseudorandom generators for space-bounded computations. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC 1990, pp. 204–212. ACM, New York (1990), http://doi.acm.org/10.1145/100216.100242
15. Parnas, M., Ron, D., Rubinfeld, R.: Testing parenthesis languages. In: Proceedings of the 5th International Workshop on Randomization and Approximation Techniques in Computer Science, pp. 261–272. Springer, Heidelberg (2001)
16. Rudra, A., Uurtamo, S.: Data stream algorithms for codeword testing. In: ICALP (1), pp. 629–640 (2010)
17. Jayram, T.S, David, W.: Optimal bounds for johnson-lindenstrauss transforms and streaming problems with sub-constant error. In: SIAM: ACM-SIAM Symposium on Discrete Algorithms, SODA 2011 (2011)
18. Shaltiel, R.: Weak derandomization of weak algorithms: Explicit versions of yao's lemma. In: Annual IEEE Conference on Computational Complexity, pp. 114–125 (2009)
19. Vadhan, S.: Pseudorandomness, monograph in preparation for FnTTCS (2010), http://people.seas.harvard.edu/~salil/pseudorandomness/
20. Yao, A.C.C.: On the power of quantum fingerprinting. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC 2003, pp. 77–81 (2003)

Size and Computation of Injective Tree Automatic Presentations

Dietrich Kuske[1] and Thomas Weidner[2]

[1] Institut für Theoretische Informatik, TU Ilmenau, Germany
[2] Universität Leipzig, Institut für Informatik, Germany

Abstract. It has been shown that every tree automatic structure admits an injective tree automatic presentation, but no good size or time bounds are known. From an arbitrary tree automatic presentation, we construct an equivalent injective one in polynomial space that consequently has exponential size. Furthermore we also prove an exponential lower bound for the size of injective tree automatic presentations.

1 Introduction

Automatic structures allow us to represent relational structures using automata. This approach was introduced in [5,7] for word automata and later generalized to tree automata in [1,2]. It gained increasing interest over the last years [9,10,8,16,14,15,6,12,13].

Roughly speaking, a structure is tree automatic if it is the quotient structure of a structure, where the carrier set and the relations are given by tree automata, and a tree regular congruence. If this congruence is the identity relation, the structure is called injectively tree automatic. A presentation of a tree automatic structure is a tuple of tree automata that describe the carrier set, the relations, and the congruence; it is injective if the congruence is the identity.

It is known that every tree automatic structure admits an injective presentation [3]. That proof also shows that such an injective presentation can be computed effectively, but no good complexity bounds can be derived. This paper presents a construction of an (exponentially larger) injective presentation from an arbitrary presentation which can be carried out in polynomial space. Furthermore we show that this exponential blowup is unavoidable for some automatic structures.

The first order theory of tree automatic structures is decidable, but the complexity is in general non-elementary. Better complexity bounds are known for tree automatic structures of bounded degree [11]. Since these techniques only work for injective tree automatic structures, our upper bound provides the first upper bound for the uniform first-order theory of tree automatic structures of bounded degree.

2 Preliminaries

In this section we will collect some fundamental definitions and facts. Here the smallest natural number is one. Let Σ be an alphabet, i.e., a finite and nonempty set. We denote all finite words over Σ by Σ^* and for some non-negative number n let $\Sigma^{\leq n}$ be the set of all words of length less than or equal to n.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 424–435, 2011.

2.1 Trees

We will only consider binary trees in this article, the generalization to trees of bounded rank is easily possible.

Let $N \subseteq \{1,2\}^*$ be some set of positions. We call N *prefix-closed* if for all $x, y \in \{1,2\}^*$ whenever $x \,.\, y \in N$ holds also $x \in N$ is true. Furthermore, the *frontier of* N is the set $\mathrm{Fr}(N) \stackrel{\text{def}}{=} (N \,.\, \{1,2\}) \setminus N$. This set contains all positions directly below N.

A *tree over* Σ is a partial mapping $t\colon \{1,2\}^* \dashrightarrow \Sigma$ such that $\mathrm{dom}(t)$ is non-empty, finite, prefix-closed, and $x \,.\, 2 \in \mathrm{dom}(t)$ implies $x \,.\, 1 \in \mathrm{dom}(t)$ for all $x \in \{1,2\}^*$. Let T_Σ denote the set of all trees over Σ. The height of a tree is defined by $\mathrm{height}(t) \stackrel{\text{def}}{=} \max\{|x| : x \in \mathrm{dom}(t)\} + 1$. A position $x \in \mathrm{dom}(t)$ is called a leaf position if $x \,.\, \{1,2\} \cap \mathrm{dom}(t) = \emptyset$. Let $s, t \in \mathrm{T}_\Sigma$ and $x \in \mathrm{dom}(t)$. Then the subtree $t \restriction x$ of t rooted at x and the tree $t[x \mapsto s]$ obtained from t by replacing the subtree $t \restriction x$ rooted at x by s are defined as usual. Furthermore $f(t_1, \ldots, t_d)$ is the tree t with root-symbol f and $t \restriction i = t_i$ for $1 \leq i \leq d$.

The convolution $t = t_1 \otimes \cdots \otimes t_d$ of the trees $t_1, \ldots, t_d \in \mathrm{T}_\Sigma$ is defined by

$$\mathrm{dom}(t) \stackrel{\text{def}}{=} \bigcup_{i=1,\ldots,d} \mathrm{dom}(t_i) \text{ and}$$

$$t(x) \stackrel{\text{def}}{=} (f_1, \ldots, f_d) \quad \text{where} \quad f_i \stackrel{\text{def}}{=} \begin{cases} t_i(x) & \text{if } x \in \mathrm{dom}(t_i) \\ \square & \text{otherwise.} \end{cases}$$

Here $\square \notin \Sigma$ is a new symbol. We write $\Sigma_\square$ for $\Sigma \cup \{\square\}$ such that the convolution t becomes a tree over $\Sigma_\square^d$.

2.2 Tree Automata

A (bottom-up) *tree automaton over* Σ is a tuple $A = (Q, \delta, F)$ where Q is a finite set of states and $F \subseteq Q$ a set of final states. The relation $\delta \subseteq Q \times \Sigma \times Q^{\leq 2}$ is called the transition relation. We also write $\mathrm{Q}(A)$, $\mathrm{F}(A)$ and $\delta(A)$ for Q, F and, respectively, δ.

A *run* of A on some tree $t \in \mathrm{T}_\Sigma$ is a tree $\rho \in \mathrm{T}_Q$ such that $\mathrm{dom}(\rho) = \mathrm{dom}(t)$ and for all position $x \in \mathrm{dom}(t)$, we have

$$(\rho(x), t(x), \rho(x \,.\, 1), \ldots, \rho(x \,.\, k)) \in \delta\,,$$

where $0 \leq k \leq 2$ is the number of children of x. We write $\xrightarrow{t} q$ if there is a run ρ on the tree t with $\rho(\varepsilon) = q$. We further write $\xrightarrow{t} X$ for a set $X \subseteq Q$ if there is a $q \in X$ which satisfies $\xrightarrow{t} q$. This notation allows us to define the language of the tree automaton A as

$$\mathrm{L}(A) = \{t \in \mathrm{T}_\Sigma : \xrightarrow{t} F\}\,.$$

A tree automaton $A = (Q, \delta, F)$ is stored by saving all tuples in δ and the final states F. This requires space $\mathrm{O}(\log|Q| + \log|\Sigma|)$ for a single transition. Furthermore we assume that every state occurs as the left side of a transition. Therefore $|F| \leq |Q| \leq |\delta|$

always holds. The space needed to store the automaton A is hence given by $\mathrm{O}\big(|\delta| \cdot (\log|Q| + \log|\Sigma|)\big)$. Following these observations we define the size of A as $|A| \stackrel{\text{def}}{=} |\delta| \cdot (\log|Q| + \log|\Sigma|)$. The number of states $|A|_{\mathrm{Q}} \stackrel{\text{def}}{=} |Q|$ is another size measure.

A tree automaton A over $\Sigma_{\square}^{d}$ is called *d-dimensional tree automaton over Σ*; it recognises the relation

$$\mathrm{R}(A) = \{(t_1, \dots, t_d) \in \mathrm{T}_{\Sigma}^{d} : t_1 \otimes \cdots \otimes t_d \in \mathrm{L}(A)\}\,.$$

Note that the size of A is given by $|\delta(A)| \cdot (\log|\mathrm{Q}(A)| + d \cdot \log|\Sigma|)$.

3 An Upper Bound

Consider an equivalence relation $\sim$ over T_{Σ}. A *complete system of representatives* for ${}^{\mathrm{T}_{\Sigma}}\!/_{\sim}$ is a set $L \subseteq \mathrm{T}_{\Sigma}$ such that $[t]_{\sim} \cap L$ is a singleton set for all trees $t \in \mathrm{T}_{\Sigma}$. Here $[t]_{\sim} \subseteq \mathrm{T}_{\Sigma}$ is the equivalence class of t with respect to $\sim$.

The aim of this section is to construct in polynomial space from a 2-dimensional tree automaton recognizing an equivalence relation, a tree automaton which recognizes a complete system of representatives for this equivalence relation:

Theorem 3.1. *From a 2-dimensional tree automaton $A_{\sim}$ with $R(A_{\sim})$ an equivalence relation, one can compute in polynomial space a tree automaton B with $|B|_{\mathrm{Q}} = 2^{\mathrm{O}(|A_{\sim}|_{\mathrm{Q}})}$ and $|B| = |\Sigma|^{\mathrm{O}(1)} \cdot 2^{\mathrm{O}(|A_{\sim}|_{\mathrm{Q}})}$ which recognizes a complete system of representatives for ${}^{\mathrm{T}_{\Sigma}}\!/_{\mathrm{R}(A_{\sim})}$.*

The existence of the tree automaton B from the above theorem was shown in [3] and an analysis of Colcombet and Löding's proof also shows that it is effective, although of multi-exponential complexity. The new singly-exponential blowup is obtained by a refinement of the proof by Colcombet and Löding.

3.1 Shadow

For the rest of this section we fix a 2-dimensional tree automaton $A_{\sim}$ over some alphabet Σ such that $\sim \stackrel{\text{def}}{=} \mathrm{R}(A_{\sim})$ is an equivalence relation on T_{Σ}. We first introduce our notion of the shadow of an equivalence class. For a class $c \in {}^{\mathrm{T}_{\Sigma}}\!/_{\sim}$ we define the *shadow of c* as

$$\mathrm{S}(c) \stackrel{\text{def}}{=} \bigcap_{t \in c} \mathrm{dom}(t)\,.$$

Using this definition, we can define descriptions of an equivalence class.

Definition 3.2. *Let $c \in {}^{\mathrm{T}_{\Sigma}}\!/_{\sim}$. A tree $t \in c$ is a* description of c *if* $\mathrm{height}(t \restriction x) \leq |A_{\sim}|_{\mathrm{Q}}$ *holds for all $x \in \mathrm{dom}(t) \cap \mathrm{Fr}(\mathrm{S}(c))$. The set of all descriptions of c is denoted by* $\mathrm{Desc}(c)$.

Lemma 3.3. *Let $c \in {}^{\mathrm{T}_{\Sigma}}\!/_{\sim}$. Then $0 < |\mathrm{Desc}(c)| < \infty$.*

Proof. From the very definition, we learn that there are only finitely many descriptions of c. Let $t \in c$ be some tree. We prove the existence of some description by induction over

$$n_t \stackrel{\text{def}}{=} \left|\{x \in \mathrm{dom}(t) \cap \mathrm{Fr}(\mathrm{S}(c)) : \mathrm{height}(t \restriction x) > |A_\sim|_{\mathrm{Q}}\}\right| .$$

If $n_t = 0$ then t is already a description of c. Now assume $n_t > 0$. Hence there is $x \in \mathrm{dom}(t) \cap \mathrm{Fr}(\mathrm{S}(c))$ such that $\mathrm{height}(t \restriction x) > |A_\sim|_{\mathrm{Q}}$. As x is not in the shadow of c there exists a tree $t_0 \in c$ with $x \notin \mathrm{dom}(t_0)$. Let τ be a successful run of $A_\sim$ on $t \otimes t_0$. We consider the tree automaton $B = (\mathrm{Q}(A_\sim), \delta, \{\tau(x)\})$ with

$$\delta \stackrel{\text{def}}{=} \{(q, f, q_1, \ldots, q_k) : (q, (f, \Box), q_1, \ldots, q_k) \in \delta(A_\sim)\} .$$

Since $x \notin \mathrm{dom}(t_0)$ the tree $t \restriction x$ is in the language of B and $\tau \restriction x$ is an accepting run on it. From the pumping lemma [4, Corollary 1.2.3] it follows that there exists a tree $s \in \mathrm{L}(B)$ such that $\mathrm{height}(s) \leq |B|_{\mathrm{Q}} = |A_\sim|_{\mathrm{Q}}$. Let σ be an accepting run of B on s. Then $\sigma(\varepsilon) = \tau(x)$. We define trees $t_1 \stackrel{\text{def}}{=} t[x \mapsto s]$ and $\tau_1 \stackrel{\text{def}}{=} \tau[x \mapsto \sigma]$. One can show that τ_1 is an accepting run of $A_\sim$ on $t_1 \otimes t_0$. Therefore we obtain $t_1 \sim t_0 \sim t$ and from the definition of t_1 we get $n_{t_1} = n_t - 1$. This concludes the inductive proof. □

In the next step, we construct from $A_\sim$ in polynomial space a tree automaton that accepts the set of all descriptions. As an intermediate result we associate the shadow $\mathrm{S}([t])$ with a tree $t \in \mathrm{T}_\Sigma$. Thereto we encode a pair (t, N), where $N \subseteq \mathrm{dom}(t)$ is a set of positions, as a tree over $\Sigma_0 \stackrel{\text{def}}{=} \Sigma \times \{0, 1\}$. This tree is again denoted by (t, N). It is defined as

$$\mathrm{dom}\big((t, N)\big) \stackrel{\text{def}}{=} \mathrm{dom}(t) \qquad \text{and} \qquad (t, N)(x) \stackrel{\text{def}}{=} \big(t(x), \mathbb{1}_N(x)\big) ,$$

where $\mathbb{1}_N$ is the characteristic function of N. This definition allows us to specify the language L_{S} as

$$L_{\mathrm{S}} \stackrel{\text{def}}{=} \{(t, N) \in \mathrm{T}_{\Sigma_0} : N = \mathrm{S}([t])\} .$$

To show the regularity of L_{S}, we prove that the following two languages are regular:

$$L_{\mathrm{S}}^1 \stackrel{\text{def}}{=} \{(t, N) \in \mathrm{T}_{\Sigma_0} : N \subseteq \mathrm{S}([t])\} \text{ and } L_{\mathrm{S}}^2 \stackrel{\text{def}}{=} \{(t, N) \in \mathrm{T}_{\Sigma_0} : N \supseteq \mathrm{S}([t])\} .$$

Lemma 3.4. *From a tree automaton $A_\sim$ with $R(A_\sim)$ an equivalence relation, one can compute in polynomial space a tree automaton recognizing L_{S}^1 with $2^{\mathrm{O}(|A_\sim|_{\mathrm{Q}})}$ many states.*

Proof. Let $t \in \mathrm{T}_\Sigma$ be a tree and $N \subseteq \{1, 2\}^*$ be some set. Then we have $N \subseteq \mathrm{S}([t])$ iff $N \subseteq \mathrm{dom}(s)$ for all trees $s \in \mathrm{T}_\Sigma$ with $s \sim t$. Consider the relation

$$R \stackrel{\text{def}}{=} \{\big((t, N), s\big) \in \mathrm{T}_{\Sigma_0} \times \mathrm{T}_\Sigma : N \not\subseteq \mathrm{dom}(s) \text{ and } s \sim t\}$$

such that L_{S}^1 is the complement of the projection of R on the first component. A tree automaton B for the projection of R whose number of states is linear in $|A_\sim|_{\mathrm{Q}}$ can easily be computed. The complementation of this automaton can be carried out in space polynomial in $|B|$ and therefore polynomial in $|A_\sim|$. □

The other direction is more involved as the proof of the following lemma indicates.

Lemma 3.5. *From a tree automaton $A_\sim$ with $R(A_\sim)$ an equivalence relation, one can compute in polynomial space a tree automaton recognizing the tree language L_{S}^2 with $2^{\mathrm{O}(|A_\sim|_{\mathrm{Q}})}$ many states.*

Proof. Let $A_\sim = (Q, \delta, F)$. We will construct an automaton B with state set 2^Q such that for $t \in \mathrm{T}_\Sigma$, $N \subseteq \mathrm{dom}(t)$ and $\rho \in \mathrm{T}_{2^Q}$ with $\mathrm{dom}(t) = \mathrm{dom}(\rho)$, the following are equivalent:

1. ρ is an accepting run of B on (t, N)
2. – there are trees $s_1, \dots, s_n \in [t]$ and accepting runs $\rho_1, \dots \rho_n$ of $A_\sim$ on $s_i \otimes t$ such that $\rho(x) = \{\rho_1(x), \dots, \rho_n(x)\}$ for all $x \in \mathrm{dom}(t)$ and
   – $\mathrm{dom}(t) \setminus N \subseteq \bigcup_{1 \le i \le n} \mathrm{dom}(t) \setminus \mathrm{dom}(s_i)$.

First, let $\emptyset$ denote the "empty tree" and $Q_\emptyset = \{p \in Q \mid \exists s \in \mathrm{T}_\Sigma : \xrightarrow{\emptyset \otimes s} p\}$. A tuple $(M, (a, x), M_1, \dots, M_k)$ is a transition of B if and only if

- for all $p \in M$, there is $b_p \in \Sigma \cup \{\Box\}$ and $\bar{p} \in \left(\prod_{1 \le i \le k} M_i\right) \times Q_\emptyset^{\le 2-k}$ such that $(p, (a, b_p), \bar{p}) \in \delta$,
- for all $1 \le i \le k$ and $p_i \in M_i$, there are $b \in \Sigma \cup \{\Box\}$, $p_j \in M_j$ for $1 \le j \le k$ and $j \ne i$, and $p \in M$ such that $(p, (a, b), p_1, \dots, p_k) \in \delta$, and
- if $x = 0$, then there exists $p \in M$ so that one can choose $b_p = \Box$ in the first condition.

Finally a set of states $M \subseteq Q$ is accepting if and only if $M \subseteq F$. Now let $(t, N) \in \mathrm{T}_{\Sigma_0}$. Then

$$
\begin{aligned}
&B \text{ accepts the pair } (t, N) \in \mathrm{T}_{\Sigma_0} \\
\iff\ &\text{there exist trees } s_1, \dots, s_n \in [t] \text{ with } \mathrm{dom}(t) \setminus N \subseteq \textstyle\bigcup_{1 \le i \le n} \mathrm{dom}(t) \setminus \mathrm{dom}(s_i) \\
\iff\ &\mathrm{dom}(t) \setminus N \subseteq \textstyle\bigcup_{s \sim t} \mathrm{dom}(t) \setminus \mathrm{dom}(s) \\
\iff\ &N \subseteq \textstyle\bigcap_{s \sim t} \mathrm{dom}(s) = S([t]).
\end{aligned}
$$

□

Now the regularity of the tree language L_{S} is immediate from Lemmas 3.4 and 3.5:

Corollary 3.6. *From a tree automaton $A_\sim$ with $R(A_\sim)$ an equivalence relation, one can compute in polynomial space a tree automaton recognizing the tree language L_{S} with $2^{\mathrm{O}(|A_\sim|_{\mathrm{Q}})}$ many states.*

3.2 Final Steps

In this section we finish the construction of an automaton which recognizes a complete system of representatives for $\mathrm{T}_\Sigma/\!\sim$. For this, fix some linear ordering $\le_\Sigma$ on the finite alphabet Σ. Let s and t be two distinct trees and let p be the lexicographically minimal position where they differ (i.e., from the set $(\mathrm{dom}(s) \setminus \mathrm{dom}(t)) \cup (\mathrm{dom}(t) \setminus \mathrm{dom}(s)) \cup \{q \in \mathrm{dom}(s) \cap \mathrm{dom}(t) : s(q) \ne t(q)\}$). Then set $s <_{\mathrm{T}} t$ provided $p \notin \mathrm{dom}(t)$ or $p \in \mathrm{dom}(s) \cap \mathrm{dom}(t)$ and $s(p) <_\Sigma t(p)$. Then the reflexive closure $\le_{\mathrm{T}}$ of $<_{\mathrm{T}}$ is a linear order that can be accepted by a 2-dimensional tree automaton.

Lemma 3.7. *From a tree automaton $A_\sim$ with $R(A_\sim)$ an equivalence relation, one can compute in polynomial space a tree automaton B with $2^{O(|A_\sim|_Q)}$ many states that accepts $\{\min_{\leq_T} \mathrm{Desc}([t]) : t \in T_\Sigma\}$.*

Proof. First note that $s \in \mathrm{Desc}([t])$ iff $s \sim t$ and $\mathrm{dom}(s) \subseteq S([t]).\{1,2\}^{<|A_\sim|_Q}$. Hence $t = \min_{\leq_T} \mathrm{Desc}([t])$ iff there exists $N \subseteq \{1,2\}^*$ such that

(1) $(t, N) \in L_S$,
(2) $\mathrm{dom}(t) \subseteq N \cdot \{1,2\}^{<|A_\sim|_Q}$, and
(3) there is no tree $s \sim t$ with $\mathrm{dom}(s) \subseteq N \cdot \{1,2\}^{<|A_\sim|_Q}$ and $s <_T t$.

Classical constructions allow us to build the tree automaton B using the tree automaton from Corollary 3.6. □

Since, by Lemma 3.3, $\min_{\leq_T} \mathrm{Desc}([t])$ exists for all trees t, the language from the previous lemma is a complete system of representatives for ${}^{T_\Sigma}/_\sim$. Note that for a tree automaton A the condition $|A|_Q = 2^{O(n)}$ implies $|A| = |\Sigma|^{O(1)} \cdot 2^{O(n)}$. This is due to $|\delta(A)| \leq |\Sigma| \cdot |Q(A)|^{O(1)}$. Therefore Lemma 3.7 also proves Theorem 3.1.

3.3 Application to Tree Automatic Structures

In this part we introduce (injective) tree automatic structures and presentations. We show that every tree automatic presentation can be transformed into an equivalent injective tree automatic presentation of exponential size.

Before we can introduce tree automatic structures we need to define signatures and structures. We call a finite set $\mathcal{S}$ of relation symbols together with their arities $(m_r)_{r\in\mathcal{S}}$ a signature. A $\mathcal{S}$-structure $\mathcal{A}$ is a tuple $(U, (r^\mathcal{A})_{r\in\mathcal{S}})$ where U is an arbitrary set and $r^\mathcal{A} \subseteq U^{m_r}$ $(r \in \mathcal{S})$ are relations. A congruence on $\mathcal{A}$ is an equivalence relation $\sim$ such that

$$(u_1, \ldots, u_{m_r}) \in r^\mathcal{A} \iff (v_1, \ldots, v_{m_r}) \in r^\mathcal{A}$$

for every $r \in \mathcal{S}$ and $u_1 \sim v_1, \ldots, u_{m_r} \sim v_{m_r}$ in U.

A *tree automatic presentation* (over $\mathcal{S}$) is a tuple $\mathcal{P} = (\Sigma, A, A_\sim, (A_r)_{r\in\mathcal{S}})$ such that Σ is an alphabet, A is a tree automaton, A_r are m_r-dimensional tree automata with $R(A_r) \subseteq L(A)^{m_r}$ and $A_\sim$ is a 2-dimensional tree automaton such that $R(A_\sim) \subseteq L(A)^2$ is a congruence on $(L(A), (R(A_r))_{r\in\mathcal{S}})$. The $\mathcal{S}$-structure defined by $\mathcal{P}$ is given by

$$\mathcal{A}(\mathcal{P}) \stackrel{\mathrm{def}}{=} (L(A), (R(A_r))_{r\in\mathcal{S}})/R(A_\sim)\,.$$

We say $\mathcal{P}$ is injective if $R(A_\sim)$ is the identity relation on $L(A)$. A structure $\mathcal{A}$ is called *(injective) tree automatic structure* if there is an (injective) tree automatic presentation $\mathcal{P}$ such that $\mathcal{A}(\mathcal{P}) \cong \mathcal{A}$.

The size and number of states of $\mathcal{P}$ are the sums of the corresponding values for the automata, i.e.,

$$|\mathcal{P}| \stackrel{\mathrm{def}}{=} |A| + |A_\sim| + \sum_{r\in\mathcal{S}} |A_r| \quad \text{and} \quad |\mathcal{P}|_Q \stackrel{\mathrm{def}}{=} |A|_Q + |A_\sim|_Q + \sum_{r\in\mathcal{S}} |A_r|_Q\,.$$

We will apply Theorem 3.1 to automatic structures to obtain the next theorem.

Theorem 3.8. *From a tree automatic presentation $\mathcal{P}$ over some alphabet Σ and $M = \max\{m_r : r \in \mathcal{S}\}$, one can compute in polynomial space an injective tree automatic presentation $\mathcal{I}$ such that $\mathcal{A}(\mathcal{P}) \cong \mathcal{A}(\mathcal{I})$, $|\mathcal{I}| = |\Sigma|^{\mathrm{O}(M)} \cdot 2^{\mathrm{O}(M\cdot|\mathcal{P}|_{\mathrm{Q}})}$ and $|\mathcal{I}|_{\mathrm{Q}} = 2^{\mathrm{O}(M\cdot|\mathcal{P}|_{\mathrm{Q}})}$.*

Proof. Let $\mathcal{P} = (\Sigma, A, A_\sim, (A_r)_{r\in\mathcal{S}})$ and $\sim = \mathrm{R}(A_\sim) \subseteq \mathrm{L}(A)^2$. Let $\sim'$ be the least equivalence relation on the whole set T_Σ that contains $\sim$, i.e.,

$$s \sim' t \iff s \sim t \text{ or } s = t$$

for all $s, t \in \mathrm{T}_\Sigma$. Clearly $\sim'$ can be recognized by a tree automaton $A_{\sim'}$ with $\mathrm{O}(|A_\sim|_{\mathrm{Q}})$ many states. Now, we apply Theorem 3.1 to $A_{\sim'}$ and obtain a tree automaton B' such that $\mathrm{L}(B')$ is a complete system of representatives for $\mathrm{T}_\Sigma/\sim'$ and $|B'|_{\mathrm{Q}} = 2^{\mathrm{O}(|A_\sim|_{\mathrm{Q}})}$. By standard constructions we compute an automaton B such that $\mathrm{L}(B) = \mathrm{L}(B') \cap \mathrm{L}(A)$ and $|B|_{\mathrm{Q}} = 2^{\mathrm{O}(|\mathcal{P}|_{\mathrm{Q}})}$. By definition the set $\mathrm{L}(A)$ is $\sim$-closed, and therefore $\sim'$-closed. Hence $\mathrm{L}(B)$ is a complete system of representatives for $\mathrm{L}(A)/\sim$. Finally, we construct for every $r \in \mathcal{S}$ a tree automaton B_r by intersecting every component of $\mathrm{R}(A_r)$ with $\mathrm{L}(B)$. Then $|B_r|_{\mathrm{Q}} = \mathrm{O}(|A_r|_{\mathrm{Q}} \cdot |B|_{\mathrm{Q}}^{m_r})$. Let $\mathcal{I} = (B, (B_r)_{r\in\mathcal{S}})$. As $\sim$ is a congruence, $\mathcal{I}$ is the wanted injective tree automatic presentation. □

One may want an estimate for the size of $\mathcal{I}$ which only depends on $|\mathcal{P}|$. It is easy to check that for any tree automaton A, we have $|A|_{\mathrm{Q}} \cdot \log|A| = \mathrm{O}(|A|)$. We may also assume that a tree automatic presentation actually uses every letter of its alphabet. Therefore we can assume $|\Sigma| \le |A| \le |\mathcal{P}|$. In the proof of Theorem 3.8 we now have $\log|B| = \mathrm{O}(\log|\Sigma| + {}^{|A_\sim|}/_{\log|A_\sim|}) = \mathrm{O}({}^{|\mathcal{P}|}/_{\log|\mathcal{P}|})$. Hereby we get

$$|\mathcal{I}| = |B| + \sum_{r\in\mathcal{S}} |B_r| = \sum_{r\in\mathcal{S}} 2^{\mathrm{O}({}^{M\cdot|\mathcal{P}|}/_{\log|\mathcal{P}|})} \cdot |A_r| = 2^{\mathrm{O}({}^{M\cdot|\mathcal{P}|}/_{\log|\mathcal{P}|})}.$$

This result allows us to answer an open question from [11] that asks for the complexity of the uniform model checking of tree automatic structures of bounded degree.[1] A structure $\mathcal{A} = (U, (r^{\mathcal{A}})_{r\in\mathcal{S}})$ has *bounded degree* if there exists a natural number d such that any $x \in U$ belongs to at most d tuples from $\bigcup_{r\in\mathcal{S}} r^{\mathcal{A}}$. Then one obtains

Corollary 3.9. *The set of pairs $(\mathcal{P}, \varphi)$ with $\mathcal{P}$ a tree automatic presentation of a structure of bounded degree and φ a first-order sentence such that $\mathcal{A}(\mathcal{P}) \models \varphi$ is decidable in 5-fold exponential time.*

Proof. In the given time bound, $\mathcal{P}$ can be transformed into an equivalent injective presentation $\mathcal{P}'$ of exponential size. Then the result follows from [11, Cor. 3.8]. □

4 A Lower Bound

From Section 3 we know that we can construct both, a tree automaton recognizing a complete system of representatives and an injective tree automatic presentation of

[1] This answer is incorporated in the journal version of [11] that will appear in the Journal of Symbolic Logic.

exponential size. In this section we show that there is also an exponential lower bound in both cases. To show this, we construct a tree automatic structure and show that every injective tree automatic presentation of this structure or tree automaton recognizing a complete system of representatives for the carrier set is at least of exponential size.

4.1 State Complexity of Complementation Revisited

In this section, we will construct "small" automata A_n such that any automaton B accepting a language of size $m = |\Sigma^* \setminus \mathrm{L}(A_n)|$ is "large". Choosing an appropriate alphabet of size m, two states would suffice for the automaton B. Therefore, we will not consider the number of states $|B|_{\mathrm{Q}}$, but the value $|B|_{\mathrm{Q}} + \log|\Gamma|$ where Γ is the alphabet of B.

We now fix the alphabet $\Sigma \stackrel{\text{def}}{=} \{0,1\}$. For some non-empty word $w = x_0 \dots x_{k-1}$ in Σ^+ let $\mathrm{val}(w) = \sum_{0 \le i < k} x_i 2^i$ be the value of w as binary number, where the lowest bit is at the first position. Vice versa let $\mathrm{bin}_k(n)$ be the unique word $w \in \Sigma^*$ of length k such that $\mathrm{val}(w) = n$ (if $n \ge 2^k$, then $\mathrm{bin}_k(n)$ is undefined). We now consider word languages over Σ^3 and we will view words in $(\Sigma^3)^*$ as convolutions of three words over Σ of equal length.[2]

Definition 4.1. *Let $n \in \mathbb{N}$. The language L_n^{W} consists of all words $w \in (\Sigma^3)^*$ that satisfy*

(i) $|w| = k \cdot n$ *for some* $k > 0$. *Therefore* $w = (u_1 \otimes v_1 \otimes x_1) \cdots (u_k \otimes v_k \otimes x_k)$ *for some* $u_i, v_i, x_i \in \Sigma^n$.
(ii) $\mathrm{val}(u_1) = \mathrm{val}(v_1) = 0$ *and* $\mathrm{val}(u_k) = 2^n - 1$, *i.e.,* $u_1 = v_1 = 0^n$ *and* $u_k = 1^n$.
(iii) $\mathrm{val}(u_i) = \mathrm{val}(v_i) + 1$ *for all* $i \in \{2, \dots, k\}$.
(iv) $v_{i+1} = u_i$ *for all* $i \in \{1, \dots, k-1\}$.

Furthermore, the tree language L_n^{T} consists of all trees $t \in \mathrm{T}_{\Sigma^3}$ satisfying

(1) $u \,.\, 1 \in \mathrm{dom}(t)$ *implies* $u \,.\, 2 \in \mathrm{dom}(t)$, *i.e., t is a complete binary tree and*
(2) $t(\varepsilon)t(x_1)t(x_1x_2)\cdots t(x_1\cdots x_\ell) \in L_n^{\mathrm{W}}$ *for all leaf positions* $x_1 \cdots x_\ell \in \mathrm{dom}(t)$.

The languages we are really interested in are the complements of the sets L_n^{W} and L_n^{T}. Therefore, we next show that these complements can be recognized by "small" automata.

Lemma 4.2. *Let $n \in \mathbb{N}$. There is a word automaton A and a tree automaton B such that the following hold:*

$$|A| = \mathrm{O}(n \cdot \log n) \qquad |A|_{\mathrm{Q}} = \mathrm{O}(n) \qquad \mathrm{L}(A) = (\Sigma^3)^* \setminus L_n^{\mathrm{W}}$$
$$|B| = \mathrm{O}(n \cdot \log n) \qquad |B|_{\mathrm{Q}} = \mathrm{O}(n) \qquad \mathrm{L}(B) = \mathrm{T}_{\Sigma^3} \setminus L_n^{\mathrm{T}}$$

Proof. The negations of the conditions *(i)* to *(iii)* can be checked by deterministic word automata with $\mathrm{O}(n)$ many states and of size $\mathrm{O}(n \cdot \log n)$. The negation of condition *(iv)*

[2] Convolutions of words are analogously defined to convolutions of trees.

is "there is a position $j \in \{1, \dots, |w| - n\}$ such that the first component of w_j does not equal the second one of w_{j+n}". Certainly, this property can be checked by a non-deterministic word automaton with the same size and number of states. Therefore $\left(\Sigma^3\right)^* \setminus L_n^{\mathrm{W}}$ can be recognized by a word automaton with $\mathrm{O}(n)$ many states and of size $\mathrm{O}(n \cdot \log n)$.

The complement of L_n^{T} consists of all trees which contain a leaf position x such that the word of the labels from the root to x is in $(\Sigma^3)^* \setminus L_n^{\mathrm{W}}$. Again, this can be checked by a non-deterministic tree automaton with $\mathrm{O}(n)$ many states and of size $\mathrm{O}(n \cdot \log n)$. □

To show that no tree language of size $\left|L_n^{\mathrm{T}}\right|$ can be accepted by a "small" automaton, we next estimate the size of the languages L_n^{W} and L_n^{T}. For these estimations, define $\exp(0, n) \stackrel{\text{def}}{=} n$ and $\exp(k+1, n) \stackrel{\text{def}}{=} 2^{\exp(k,n)}$ for $n, k \in \mathbb{N}_0$.

Lemma 4.3. *Let $n \in \mathbb{N}$ be at least 2. Then $\exp(2, n) \leq \left|L_n^{\mathrm{W}}\right| < \infty$ and $\exp(3, n) \leq \left|L_n^{\mathrm{T}}\right| < \infty$.*

Proof. Let $w = (u_1 \otimes v_1 \otimes x_1) \cdots (u_k \otimes v_k \otimes x_k) \in L_n^{\mathrm{W}}$ with $u_i, v_i, x_i \in \Sigma^n$. By induction it follows that $\mathrm{val}(u_i) = i - 1$ and $\mathrm{val}(v_i) = \max\{i - 2, 0\}$ for all $i \in \{1, \dots, k\}$. By definition we have $\mathrm{val}(u_k) = 2^n - 1$ and therefore $k = 2^n$. Hence the word w is of length $n \cdot 2^n$ and L_n^{W} is finite. Vice versa every word $w \in (\Sigma^3)^*$ which is defined as above is in the language L_n^{W}. As we can still choose the x_i arbitrary, there are at least $|\Sigma|^{n \cdot 2^n} \geq \exp(2, n)$ many words in L_n^{W}.

From the definition of L_n^{T} and the observations for L_n^{W}, we obtain that L_n^{T} consists of full binary trees of height $n \cdot 2^n$. Since also here, the third components of the labelings can be chosen freely, L_n^{T} contains the set of Σ-labeled full binary trees of height $n \cdot 2^n$. But this set contains (for $n \geq 2$) $\exp(1, \exp(1, n \cdot 2^n) - 1) \geq \exp(3, n)$ elements. □

Proposition 4.4. *Let $n \in \mathbb{N}$ and B be a tree automaton over some alphabet Γ recognizing a finite language of size at least $\exp(3, n)$. Then $|B|_{\mathrm{Q}} + \log(\Gamma) \geq 2^n$.*

Proof. We first verify $|\mathrm{L}(B)| \leq \exp(2, |B|_{\mathrm{Q}} + \log|\Gamma|)$ by contradiction: Assume $\mathrm{L}(B)$ contains more than $\exp(2, |B|_{\mathrm{Q}} + \log|\Gamma|)$ elements. Then $\mathrm{L}(B)$ contains at least one tree of height greater than $|B|_{\mathrm{Q}}$. Hence, by the pumping lemma for regular tree languages [4, Corollary 1.2.3], $\mathrm{L}(B)$ is infinite, a contradiction.

Hence we have $\exp(3, n) \leq |\mathrm{L}(B)| \leq \exp(2, |B|_{\mathrm{Q}} + \log|\Gamma|)$ and therefore, as claimed, $|B|_{\mathrm{Q}} + \log|\Gamma| \geq 2^n$. □

4.2 A Tree Automatic Structure

Let $L \subseteq \mathrm{T}_\Sigma$ be some tree language. Our idea is to express the set complement $\mathrm{T}_\Sigma \setminus L$, or at least a set with the same cardinality, in terms of some injective tree automatic presentation.

Definition 4.5. *Let $n \in \mathbb{N}$ and let X and Y be new symbols not in $\Sigma = \{0, 1\}^3$. The tree automatic structure $\mathcal{A}_n \stackrel{\text{def}}{=} {}^{(U,R)}\!/_{\sim_n}$ is then given by*

$$U \stackrel{\text{def}}{=} \{\mathrm{X}(t), \mathrm{Y}(t) \in \mathrm{T}_{\Sigma \cup \{\mathrm{X},\mathrm{Y}\}} : t \in \mathrm{T}_\Sigma\}$$

$$R \stackrel{\text{def}}{=} \{\big(x(t), y(t)\big) \in U^2 : t \in \mathrm{T}_\Sigma,\ \{x, y\} = \{\mathrm{X}, \mathrm{Y}\}\}, \text{ and}$$

$$\sim_n \stackrel{\text{def}}{=} \{\big(x(t), y(t)\big) \in U^2 : t \in \mathrm{T}_\Sigma,\ x, y \in \{\mathrm{X}, \mathrm{Y}\} \text{ and } (t \in L_n^{\mathrm{T}} \Rightarrow x = y)\}.$$

Since the relation R is symmetric, we can think of the structures (U, R) and therefore $\mathcal{A}_n$ as an undirected graph. Then (U, R) is the disjoint union of infinitely many disjoint edges and $\mathcal{A}_n$ that of $|L_n^{\mathrm{T}}|$ disjoint edges and infinitely many isolated nodes (with a self-loop). Note that U and R can be accepted by a (2-dimensional) tree automaton and do not depend on n. To also accept the relation $\sim_n$, we modify the automaton B from Lemma 4.2 in the obvious way. Hence $\sim_n$ can be accepted by a two-dimensional automaton $A_{\sim_n}$ with $|A_{\sim_n}|_{\mathrm{Q}} = \mathrm{O}(n)$ and $|A_{\sim_n}| = \mathrm{O}(n \cdot \log n)$. This proves the next lemma.

Lemma 4.6. *There is $C_1 > 0$ such that for all $n \in \mathbb{N}$ there is a tree automatic presentation $\mathcal{P}_n$ of $\mathcal{A}_n$ with $|\mathcal{P}_n| \leq C_1 \cdot n \cdot \log n$ and $|\mathcal{P}_n|_{\mathrm{Q}} \leq C_1 \cdot n$.*

We now come to the lower bound for injective tree automatic presentations of $\mathcal{A}_n$:

Lemma 4.7. *There exist constants $c, d \geq 1$ such that for any $n \in \mathbb{N}$ with $n \geq 2$ and any injective tree automatic presentation $\mathcal{I}$ of $\mathcal{A}_n$, we have $2^n \leq c \cdot |\mathcal{I}|_{\mathrm{Q}} + \log |\Gamma|$ and $2^n \leq d \cdot |\mathcal{I}|$.*

Proof. Let $\mathcal{I} = (\Gamma, B, B_R)$ and consider the set

$$L = \{t \in \mathrm{L}(B) : (s, t) \in \mathrm{R}(B_R) \text{ for some } s \in \mathrm{L}(B) \setminus \{t\}\}.$$

The relation $\{(s, t) \in \mathrm{T}_\Gamma^2 : s \neq t\}$ can be accepted by a tree automaton with two states. Running this automaton in parallel with B_R, we get an automaton with $2 \cdot |B_R|_{\mathrm{Q}}$ states that accepts the relation

$$\{(s, t) \in \mathrm{R}(B_R) : s \neq t\}.$$

Then L is the projection of this relation to the first component. Hence L can be accepted by an automaton C with $|C|_{\mathrm{Q}} \leq c \cdot |B_R|_{\mathrm{Q}}$. Note that L is the set of nodes of $\mathcal{A}_n$ that are connected to some other node, hence $|L| = 2 \cdot |L_n^{\mathrm{T}}|$. From Lemma 4.3 and Proposition 4.4, we therefore get $|C|_{\mathrm{Q}} + \log |\Gamma| \geq 2^n$. Hence $2^n \leq |C|_{\mathrm{Q}} + \log |\Gamma| \leq c \cdot |B_R|_{\mathrm{Q}} + \log |\Gamma| \leq c \cdot |\mathcal{I}|_{\mathrm{Q}} + \log |\Gamma|$.

We can assume that every symbol from Γ actually appears in a tree in $\mathrm{L}(A)$ and hence in some transition in $\delta(A)$ such that $\log |\Gamma| \leq |\Gamma| \leq |\delta(A)| \leq |A| \leq |\mathcal{I}|$. Similarly, we can assume $|Q| \leq |\delta|$ for any automaton implying $|\mathcal{I}|_{\mathrm{Q}} \leq |\mathcal{I}|$. Therefore for the size of $\mathcal{I}$ we get $2^n \leq c \cdot |\mathcal{I}|_{\mathrm{Q}} + \log |\Gamma| \leq (c + 1) \cdot |\mathcal{I}|$. □

Now we are able to prove the lower bounds for injective tree automatic presentations.

Theorem 4.8. *There are $C_1, C_2, C_3 > 0$ such that the following hold:*

1. *for every $n \in \mathbb{N}$ there exists a tree automatic presentation $\mathcal{P}$ such that $|\mathcal{P}|_{\mathrm{Q}} \leq C_1 \cdot n$, $|\mathcal{P}| \leq C_1 \cdot n \cdot \log n$, and for every injective tree automatic presentation $\mathcal{I}$ of $\mathcal{A}(\mathcal{P})$ over some alphabet Γ it holds that $|\mathcal{I}|_{\mathrm{Q}} \geq C_2 \cdot 2^n - \log |\Gamma|$.*
2. *for every $m \in \mathbb{N}$ there exists a tree automatic presentation $\mathcal{P}$ such that $|\mathcal{P}| \leq C_1 \cdot m$, and for every injective tree automatic presentation $\mathcal{I}$ of $\mathcal{A}(\mathcal{P})$ it holds that $|\mathcal{I}| \geq C_2 \cdot 2^{\frac{m}{\log m}}$.*

Proof. Let c and d be the constants from Lemma 4.7. Then the first claim follows immediately from Lemmas 4.6 and 4.7 with $C_2 = 1/c$.

Choose $n = \lfloor m/\log m \rfloor$. Then $|\mathcal{P}_n| \leq C_1 \cdot \frac{m}{\log m} \cdot \log \frac{m}{\log m} \leq C_1 \cdot m$. Let $\mathcal{I}$ be some injective tree automatic presentation of $\mathcal{A}(\mathcal{P}_n)$. Then $|\mathcal{I}| \geq \frac{1}{d} \cdot 2^n$. Since $n > \frac{m}{\log m} - 1$, we get $|\mathcal{I}| \geq C_3 \cdot 2^{\frac{m}{\log m}}$ with $C_3 = 1/d$. □

4.3 Transfer to Word Automatic Structures

Word automatic structures have been introduced in [5,7]. Essentially they are defined as tree automatic structures, but with word automata instead of tree automata. Every word automatic presentation admits an injective word automatic presentation of exponential size. This has already been shown in [7]. In this section we will transfer our lower bound result for tree automatic structures to word automatic ones.

We use the languages L_n^{W} from Definition 4.1 to replace the languages L_n^{T} from the tree automatic case. Let $n \in \mathbb{N}$ and again $\Sigma = \{0,1\}^3$, and let $\mathrm{X}, \mathrm{Y} \notin \Sigma$ be two symbols. We define the structure $\mathcal{A}_n^{\mathrm{W}}$ by $\mathcal{A}_n^{\mathrm{W}} \stackrel{\mathrm{def}}{=} (U^{\mathrm{W}}, R^{\mathrm{W}})/\sim_n^{\mathrm{W}}$ with

$$U^{\mathrm{W}} \stackrel{\mathrm{def}}{=} \{w \,.\, X, w \,.\, Y : w \in \Sigma^*\},$$
$$R^{\mathrm{W}} \stackrel{\mathrm{def}}{=} \{(w \,.\, x, w \,.\, y) : w \in \Sigma^*,\ \{x,y\} = \{\mathrm{X},\mathrm{Y}\}\}, \text{ and}$$
$$\sim_n^{\mathrm{W}} \stackrel{\mathrm{def}}{=} \{(w \,.\, x, w \,.\, y) : w \in \Sigma^*,\ x,y \in \{\mathrm{X},\mathrm{Y}\} \text{ and } (w \in L_n^{\mathrm{W}} \Rightarrow x = y)\}.$$

Now fix an injective word automatic presentation $\mathcal{I} = (\Gamma, B, B_R)$ with word automata B and B_R over Γ. By Lemma 4.3 and similar arguments as in Proposition 4.4, and Lemma 4.7 we obtain $c \cdot |\mathcal{I}|_{\mathrm{Q}} + \log|\Gamma| \geq 2^n$ and $d \cdot |\mathcal{I}| \geq 2^n$ for constants $c, d > 0$. Together with corresponding arguments as before Lemma 4.6 and as in Theorem 4.8 we obtain the following theorem.

Theorem 4.9. *There are constants $C_1, C_2, C_3 > 0$ such that for every $n \in \mathbb{N}$ with $n \geq 2$ there exists a word automatic structure $\mathcal{P}$ such that $|\mathcal{P}| \leq C_1 \cdot n \cdot \log n$ and for every injective word automatic presentation $\mathcal{I}$ of $\mathcal{A}(\mathcal{P})$ it holds that $|\mathcal{I}| \geq C_2 \cdot 2^{\frac{n}{\log n}}$, and $|\mathcal{I}|_{\mathrm{Q}} \geq C_3 \cdot 2^n$.*

Remark 4.10. We define an injective tree automatic presentation of the word automatic structure $\mathcal{A}_n^{\mathrm{W}}$ as follows. Let $\Sigma = \{0,1\}$ and let L comprise all trees $t \in \mathrm{T}_\Sigma$ with $\mathrm{dom}(t) = 0^{<n} \,.\, 1 \,.\, \{0,1\}^{<n}$ implying $|\,\mathrm{dom}(t)| = n + n \cdot (2^n - 1) = n \cdot 2^n$ for $t \in L$. Furthermore, define $U \stackrel{\mathrm{def}}{=} \{x(t) : x \in \{X,Y\}, t \in \mathrm{T}_\Sigma, (t \in L \Rightarrow x = X)\}$. Finally, let $R = \{(x(t), y(t)) \in U^2 : x,y \in \{X,Y\}, (x = y \Rightarrow t \in L)\}$. Then $\mathcal{A}_n^{\mathrm{W}} \cong (U,R)$ and U and R can be accepted by tree automata with $\mathrm{O}(n)$ many states.

So the following open question remains: Does the exponential blowup also occur when we move from an arbitrary *word* automatic presentation to an equivalent injective *tree* automatic presentation?

References

1. Blumensath, A.: Automatic structures. Diplomarbeit, RWTH Aachen (1999)
2. Blumensath, A., Grädel, E.: Automatic Structures. In: LICS 2000, pp. 51–62. IEEE Computer Society Press, Los Alamitos (2000)
3. Colcombet, T., Löding, C.: Transforming structures by set interpretations. Logical Methods in Computer Science 3(2) (2007)
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications, http://www.grappa.univ-lille3.fr/tata (release October 12, 2007),
5. Hodgson, B.R.: On direct products of automaton decidable theories. Theor. Comput. Sci. 19, 331–335 (1982)
6. Kartzow, A.: Collapsible pushdown graphs of level 2 are tree-automatic. In: STACS 2010, Leibniz International Proceedings in Informatics (LIPIcs), vol. 5, pp. 501–512. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2010)
7. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
8. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: Richness and limitations. Logical Methods in Computer Science 3(2) (2007)
9. Khoussainov, B., Rubin, S., Stephan, F.: Definability and Regularity in Automatic Structures. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 440–451. Springer, Heidelberg (2004)
10. Khoussainov, B., Rubin, S., Stephan, F.: Automatic linear orders and trees. ACM Trans. Comput. Log. 6(4), 675–700 (2005)
11. Kuske, D., Lohrey, M.: Automatic structures of bounded degree revisited. To appear in Journal of Symbolic Logic in slightly extended form (2008), http://arxiv.org/abs/0810.4998
12. Kuske, D., Lohrey, M.: Some natural problems in automatic graphs. J. Symbolic Logic 75(2), 678–710 (2010)
13. Kuske, D., Liu, J., Lohrey, M.: The isomorphism problem on classes of automatic structures. In: LICS 2010, pp. 160–169. IEEE Computer Society, Los Alamitos (2010)
14. To, A.W., Libkin, L.: Recurrent Reachability Analysis in Regular Model Checking. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 198–213. Springer, Heidelberg (2008)
15. To, A.W., Libkin, L.: Algorithmic Metatheorems for Decidable LTL Model Checking over Infinite Systems. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 221–236. Springer, Heidelberg (2010)
16. Rubin, S.: Automata presenting structures: A survey of the finite string case. Bulletin of Symbolic Logic 14, 169–209 (2008)

Symmetric Functions Capture General Functions (Extended Abstract)

Richard J. Lipton[1], Kenneth W. Regan[2], and Atri Rudra[2,⋆]

[1] College of Computing, Georgia Tech, Atlanta, GA, USA
[2] Department of CSE, University at Buffalo (SUNY), Buffalo, NY, USA

Abstract. We show that the set of all functions is equivalent to the set of all symmetric functions (possibly over a larger domain) up to deterministic time complexity. In particular, for any function f, there is an equivalent symmetric function f_{sym} such that f can be computed from f_{sym} and vice-versa (modulo an extra deterministic linear time computation). For f over finite fields, f_{sym} is (necessarily) over an extension field. This reduction is optimal in size of the extension field. For polynomial functions, the degree of f_{sym} is not optimal. We present another reduction that has optimal degree "blowup" but is worse in the other parameters.

1 Introduction

Symmetric polynomials have been central in both arithmetic complexity and Boolean circuit complexity. All function families in ACC^0 are known to reduce to symmetric polynomials via small low-depth circuits. The Boolean majority function and related symmetric functions on $\{0,1\}^n$ are hard for low-depth circuit classes, but analogous functions over infinite fields have small constant-depth arithmetic formulas [8]. The best known lower bounds of $\Omega(n \log n)$ on arithmetic circuit size apply to some simple symmetric functions such as $x_1^n + \cdots + x_n^n$ [5]. Symmetric polynomials have a rich algebraic structure. It is therefore interesting to ask whether they are easier to compute than general polynomials.

Our main results are reductions from a general function $f \in \mathbb{F}[x_1, \ldots, x_n]$ of degree d to symmetric function(s) $f_{\text{sym}} \in \mathbb{F}'[y_1, \ldots, y_N]$ of some degree D. To be more precise, we say f reduces to f_{sym} if there exists an algorithm that on any input $a \in \mathbb{F}^n$ computes $f(a)$ by making some oracle calls to the function f_{sym}. (To make the reduction interesting, we would like the reduction algorithm to have low "complexity" modulo the oracle calls.) The notion of f_{sym} reducing to f is defined similarly. In this work, we will aim to show that f and f_{sym} are *equivalent*, i.e. f reduces to f_{sym} and vice-versa. Upper and lower bounds are classified according to whether $\mathbb{F}$ is a finite or infinite field, whether one or more queried functions f_{sym} are involved, by the degree s of $\mathbb{F}'$ as an extension of $\mathbb{F}$, and by D and N in relation to the given d and n. Once the best achievable combinations of those parameters are determined, the issues become the running

⋆ Supported by NSF CAREER grant CCF-0844796.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 436–447, 2011.

time of the reduction, whether it is computable by low-depth circuits or formulas (with oracle gate(s) for the call(s) to f_{sym}), and whether the entire computation is randomized and/or correct only for a large portion of the input space $\mathbb{F}^n$.

1.1 Symmetric Functions—Hard or Easy?

Here are some contexts in which symmetric functions are hard or powerful. Lu [17] remarked that all of the early circuit lower bounds were proved for symmetric functions such as parity and congruence of Hamming weight modulo m [10,21,18,19]. Grigoriev and Razborov [12] (see also [11]) proved exponential lower bounds for depth-3 circuits computing certain symmetric functions over finite fields. Beigel and Tarui [7] showed that every language in ACC^0 can be represented as a Boolean symmetric function in quasi-polynomially many arguments, with each argument a conjunction of inputs x_i or their negations.

In other contexts, symmetric functions are easy. Over GF(2) they depend on only $O(\log n)$ bits of input information, and hence have linear-size, $O(\log n)$-depth Boolean circuits. Beame et al. [6] showed that every n-input Boolean symmetric function can be computed by threshold circuits of linear size and $O(\log \log n)$ depth. The *elementary symmetric polynomials*, which form an algebra basis for all symmetric polynomials, are easy over all fields, and surprisingly easy in the rings of integers modulo certain composite numbers [13].

So are symmetric functions hard or easy? Or are there contexts in which they have all levels of complexity? In this paper, we prove some results of this last type. Thus we aim to transfer general issues of complexity entirely into the domain of symmetric functions, where we can use their special algebraic properties.

1.2 Our Results and Techniques

Given a polynomial $f(x_1, \ldots, x_n)$ over a field (or ring) $\mathbb{F}$, the objective is to find a symmetric polynomial f_{sym}, possibly over a larger field $\mathbb{F}'$ and/or in a different number m of variables, such that f reduces to f_{sym}, and importantly, vice-versa. The reductions from f to f_{sym} are substitutions, meaning we have functions $\gamma_1, \ldots, \gamma_n$ of possibly-new variables $y = y_1, \ldots, y_m$ such that

$$f_{\text{sym}}(y) = f(\gamma_1(y), \ldots, \gamma_n(y)).$$

Because symmetric functions form an algebra, if $\gamma_1, \ldots, \gamma_n$ are symmetric then so is f_{sym}. We presume that the γ_i are easy to compute; whereupon if f is easy then so is f_{sym}. Thus f_{sym} reduces to f. The reverse reduction, to recover values of $f(x)$ from calls to $f_{\text{sym}}(y)$ for suitable y, is the more-problematic one. Next we present a generic framework for such a reduction ($E_f(\cdots)$ is some function that computes some vectors based on the input and past history and D_f is an aggregating function):

1. On input x, compute $y^{(1)} = E_f(x)$.
2. Compute $z_1 = f_{\text{sym}}(y^{(1)})$.

3. For $i = 2, \ldots, m$, compute $y^{(j)} = E_f(x, z_1, z_2, \ldots, z_{j-1})$ and $z_j = f_{\mathrm{sym}}(y^{(j)})$.
4. Output $f(x) = D_f(z_1, \ldots, z_m)$.

We first make the observation that the reduction above becomes trivial if we do not care about the complexity of the function D_f. In particular, consider the following instantiation of the general procedure above. Define $f_{\mathrm{sym}}(u_1, \ldots, u_n) = \sum_{i=1}^{n} u_i$, and for $1 \leqslant j \leqslant n$ define $y^{(j)} \in \mathbb{F}^n$ to be the vector that is zero in every position except position j, where it has the value x_j. Note that by our choices $z_j = x_j$ for every $1 \leqslant j \leqslant n$. Thus, if we pick D_f to be f, then trivially $D_f(z_1, \ldots, z_n) = f(x)$. Of course this reduction is not interesting as the complexity of the reduction is exactly the same as that of computing f, which is undesirable.

Our first symmetrization makes $z_1 = f(x)$, so D_f is the identity function (and $m = 1$), but generally needs the encoding function $E_f(x)$ to map into an extension field. For $1 \leqslant j \leqslant m = n$ it takes γ_j to be the j-th elementary symmetric polynomial

$$e_j(x) = \sum_{S \subseteq [n], |S| = j} \prod_{i \in S} x_i, \quad \text{and defines} \quad f_{\mathrm{sym}}(x) = f(e_1(x), \ldots, e_n(x)).$$

Since the e_j are easily computed [13], so is f_{sym}.

To compute $f(b) = f(b_1, \ldots, b_n)$ by reduction to f_{sym}, we find $a = a_1, \ldots, a_n$ such that for each j,

$$b_j = e_j(a_1, \ldots, a_n), \qquad \text{so that} \qquad f(b) = f_{\mathrm{sym}}(a).$$

It turns out that the values a_j are found by splitting the univariate polynomial

$$\phi_b(X) = X^n + \sum_{i=1}^{n} b_i \cdot X^{i-1}$$

into linear factors. This is guaranteed to be possible only in the splitting field $\mathbb{F}'$ of ϕ_b. In other words, the complexity of computing f from f_{sym} via the reduction above is the same as finding roots of a degree n polynomial over $\mathbb{F}'$. Using known randomized algorithms for computing roots of a univariate polynomial (cf. [20]), the (time) complexity of the reduction is determined by the degree of the extension field $\mathbb{F}'$.

Now, the degree of $\mathbb{F}'$ over $\mathbb{F}$ is known to equal the least common multiple of the degrees of the irreducible factors of ϕ_b (see [9]). An easy calculation shows that this degree cannot be more than $n^{O(\sqrt{n})}$. However, this would be too expensive for lower bounds against low-level complexity classes.

Next, we sketch how we get around this predicament. We use the fact proved in [9] that the degree of the splitting field of a *random* monic polynomial is $n^{O(\log n)}$ with high probability. We employ this under two relaxations: (i) If we consider the average-case complexity of f, then it follows immediately that for *most* inputs, f and f_{sym} are equivalent under pseudo-polynomial time reductions. (ii) Under certain restrictions on the degree of f and $|\mathbb{F}|$, we can talk about

worst-case complexity of f and f_{sym}. In particular, we use well-known properties of Reed-Muller codes that have been used numerous times in the local testing algorithms and PCP constructions [3,2,1,15,14]. However, unlike the local testing results which need to handle errors, in our application we only need to handle erasures—roughly because we can efficiently determine the degree of $\mathbb{F}'$ without computing the vector a, which leads to slightly better bounds in the parameters of the reduction.

The drawback of this reduction is that the degree of $\mathbb{F}'$ is very large. For comparison, we show by a counting argument that the degree of $\mathbb{F}'$ need only be $O(\log_q n)$, which is super-exponentially better than the bound obtained above. However, note that by construction $\deg(f_{\text{sym}}) \leqslant n \cdot \deg(f)$– we show this to be tight with an *additive* constant factor from the optimal (again by a counting argument).

Our second symmetrization is superior in that it gives linear-time equivalence over any finite field $\mathbb{F}_q$. It is inferior only in giving somewhat higher degree of the resulting symmetric polynomial.

The intuition behind the second reduction is somewhat similar to the earlier reduction that defined $D_f = f$. In particular for every input value x_i, we will associate it with the pair (i, x_i). The main idea in defining f_{sym} from f is to convert every input (i, x_i) back to x_i. Doing this while ensuring that f_{sym} is symmetric requires a bit of care.

We compare the two methods in the table below, giving the deterministic reductions only. (We've shortened DTIME to DT to fit the table.)

	f from f_{sym}	f_{sym} from f	s	$\deg(f_{\text{sym}})$
Elem. Sym.	$\mathsf{DT}(n^{O(\sqrt{n})}q^{O(1)})$	$\mathsf{DT}(n \log^2 n)$	$n^{O(\sqrt{n})}$	$n \cdot \deg(f)$
Direct	$\mathsf{DT}(n)$	$\mathsf{DT}(n)$	$\lceil \log_q n \rceil + 1$	$snq^2 \cdot \deg(f)$
Lower Bds	?	?	$\lceil \log_q n \rceil - 3$	$\frac{n \deg(f)}{e^2}(1\text{-}o(1)) - 2n^2/5$

2 Preliminaries

We denote a field by $\mathbb{F}$. For a prime power q, we denote the finite field with q elements as $\mathbb{F}_q$. A function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ is equivalent to a polynomial over $\mathbb{F}_q$ and we will use $\deg(f)$ to denote the degree of the corresponding polynomial. To express the complexity of our reductions, we set up more notation. For any function $f : \mathbb{F}^n \to \mathbb{F}$: (i) $\mathrm{C}(f)$ denotes its time complexity (in the RAM model); (ii) $\mathrm{C}_\varepsilon(f)$ denotes the time complexity of computing f correctly on all but an ε fraction of the inputs, where $0 < \varepsilon < 1$; (iii) Over a field $\mathbb{F}$, $\mathsf{DTIME}(t(n))$ denotes $O(t(n))$ deterministic operations over $\mathbb{F}$;[1] (iv) $\mathsf{RTIME}(t(n))$ likewise denotes $O(t(n))$ (Las Vegas) randomized operations over $\mathbb{F}$; while (v) for any $0 \leqslant \delta < 1/2$, $\mathsf{RTIME}_\delta(t(n))$ denotes randomized $\mathbb{F}_r$-operations when we allow for an error probability of δ.

[1] We use parentheses $(\cdots)$ to emphasize that these time measures can be added in time expressions, whereas $\mathsf{DTIME}[t(n)]$ with $[\cdots]$ strictly denotes a class of functions.

When moving from a general function (over a finite field $\mathbb{F}$) to an equivalent symmetric function, the latter must be over an extension field of $\mathbb{F}$. We start with two elementary counting lower bounds on the degree of the extension field and on other parameters.

Theorem 1. *Let q be a prime power and $n \geqslant 1$ be an integer. If every $f : \mathbb{F}_q^n \to \mathbb{F}_q$ is equivalent to a symmetric function $f_{\text{sym}} : (\mathbb{F}_{q^s})^n \to \mathbb{F}_{q^s}$, then*

$$s \geqslant \lceil \log_q n \rceil - 3.$$

Recall that every function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ is equivalent to a polynomial (over $\mathbb{F}_q$) in n variables and degree at most qn. Next, we will lower bound the degree blow-up necessary in assigning every function to an equivalent symmetric function.

Theorem 2. *If every function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ of degree d is equivalent to some symmetric function $f_{\text{sym}} : \mathbb{F}_{q^s}^n \to \mathbb{F}_{q^s}$ of degree at most d_{sym} such that $s(d+n-1) \leqslant 2^{o(n)}$, then we must have*

$$d_{\text{sym}} \geqslant \frac{dn}{e^2} \cdot (1 - o(1)) - \frac{2}{5} \cdot n(n+2).$$

3 Results for the Elementary Symmetrization

We study the following particular substitution by the elementary symmetric polynomials:

$$f_{\text{sym}}(x) = f(e_1(x), e_2(x), \ldots, e_m(x)).$$

We first note that f_{sym} is almost as easy as f itself.

Proposition 3 *For any function $f : \mathbb{F}_q^n \to \mathbb{F}_q$,*

$$\mathrm{C}(f_{\text{sym}}) \leqslant \mathrm{C}(f) + \mathsf{DTIME}\left(n(\log n)^2\right).$$

Proof. Note that the result will follow if we can show how to compute $e_1(\mathbf{a}), \ldots, e_n(\mathbf{a})$ in $\mathsf{DTIME}(n(\log n)^2)$. This follows from the well-known identity that

$$X^n + \sum_{i=1}^{n} e_i(\mathbf{a}) \cdot X^{i-1} = \prod_{i=1}^{n} (X + a_i),$$

since the polynomial on the RHS can be multiplied out via divide-and-conquer and FFT-based polynomial multiplication. ■

We now consider the problem of showing the converse of Proposition 3, i.e. we would like to bound the complexity of f in terms of the complexity of f_{sym}. We can show the following converses:

Theorem 4. *Let $f : \mathbb{F}_q^n \to \mathbb{F}_q$ be a function, then (for any $0 < \delta < 1/2$) the following are true:*

$$\mathrm{C}(f) \leqslant \mathrm{C}(f_{\mathrm{sym}}) + \mathsf{DTIME}(n^{O(\sqrt{n})} \cdot q^{O(1)}). \tag{1}$$

$$\mathrm{C}_{\exp(-\Omega(\sqrt{\log n}))}(f) \leqslant \mathrm{C}(f_{\mathrm{sym}}) + \mathsf{RTIME}(n^{O(\log n)} \cdot \log^{O(1)} q), \tag{2}$$

$$\mathrm{C}_{\exp(-\Omega(\sqrt{\log n}))}(f) \leqslant \mathrm{C}(f_{\mathrm{sym}}) + \mathsf{DTIME}(n^{O(\log n)} \cdot q^{O(1)}). \tag{3}$$

Further, if $q \geqslant \Omega(\deg(f))$, then

$$\mathrm{C}(f) \leqslant O(\deg(f) \log(\frac{1}{\delta}) \cdot \mathrm{C}(f_{\mathrm{sym}})) + \mathsf{RTIME}_\delta(n^{O(\log n)} \cdot \log^{O(1)} q + q^{O(1)}). \tag{4}$$

Finally, if $1 \leqslant m \leqslant n$ and $\deg(f) \leqslant \min(m(q-1), O(q\sqrt{\log n}/\log q))$, then

$$\mathrm{C}(f) \leqslant O(q^m \cdot \log(\frac{1}{\delta}) \cdot \mathrm{C}(f_{\mathrm{sym}})) + \mathsf{RTIME}_\delta(n^{O(\log n)} \cdot \log^{O(1)} q + q^{O(1)}), \tag{5}$$

All of the results above start with the basic idea presented below.

3.1 The Basic Idea

Note that we can prove the converse of Proposition 3 if for every $\mathbf{b} = (b_1, \ldots, b_n)$ for which we want to compute $f(\mathbf{b})$, we could compute $\mathbf{a} = (a_1, \ldots, a_n)$ such that for every $1 \leqslant i \leqslant n$, $b_i = e_i(\mathbf{a})$ and evaluate $f_{\mathrm{sym}}(\mathbf{a})$ (which by definition would be $f(\mathbf{b})$). In other words, given the polynomial

$$\phi_{\mathbf{b}}(X) = X^n + \sum_{i=1}^{n} b_i \cdot X^{i-1},$$

we want to completely factorize it into linear factors, i.e. compute $\mathbf{a} = (a_1, \ldots, a_n)$ such that

$$\prod_{i=1}^{n} (X + a_i) = \phi_{\mathbf{b}}(X).$$

It is not very hard to see that such an $\mathbf{a}$ might not exist in $\mathbb{F}_q^n$. Thus, we will have to look into extension fields of $\mathbb{F}_q$. In particular, the *splitting field* of any polynomial over $\mathbb{F}_q$ is the smallest extension field over which the polynomial factorizes completely into linear factors. The following result is well-known:

Proposition 5 (cf. [9]) *Let $h(X)$ be a univariate polynomial over $\mathbb{F}_q$ of degree k. Let s denote the least common multiple of all the distinct degrees of irreducible factors of $h(X)$ (over $\mathbb{F}_q$). Then the splitting field of $h(X)$ is $\mathbb{F}_{q^s}$.*

Given the above, the algorithm for computing $\mathbf{a}$ is direct:

Invert($\mathbf{b}$)

1. Compute the polynomial $\phi_{\mathbf{b}}(X)$.
2. Compute the smallest $s = s(\mathbf{b})$ such that $\mathbb{F}_{q^s}$splits $\phi_{\mathbf{b}}(X)$.
3. Compute an irreducible polynomial of degree s over $\mathbb{F}_q$.
4. Compute the roots of $\phi_{\mathbf{b}}(X)$ over $\mathbb{F}_{q^s}$.

The correctness of the algorithm follows from the discussion above. (The computation of the irreducible polynomial in Step 3 is to form a representation of the finite field $\mathbb{F}_{q^s}$.) To analyze its running time, we define some notation: $\text{split}(\mathbf{b}, q)$ denotes the time required to compute the smallest $s = s(\mathbf{b}, q)$ such that $\mathbb{F}_{q^s}$ is the splitting field of $\phi_{\mathbf{b}}(X)$, $\text{irr}(k, q)$ is the time required to generate a degree k irreducible polynomial over $\mathbb{F}_q$ and $\text{root}(k, q)$ is the time required compute all the roots of a degree k polynomial over $\mathbb{F}_q$.

The above implies the following:

Lemma 1. *For any* $\mathbf{b} \in \mathbb{F}_q^n$, $\text{Invert}(\mathbf{b})$ *can be computed in time* $\mathsf{DTIME}(n) + \text{split}(\mathbf{b}, q) + \text{irr}(s, q) + \text{root}(n, q^s)$.

This would imply that if $\text{Invert}(\cdot)$ can be computed in time $T(n)$, then one has $\text{C}(f) \leqslant \text{C}(f_{\text{sym}}) + T(n)$. Unfortunately, the quantity s in Step 2 above can be as large as $n^{O(\sqrt{n})}$. This implies that $T(n) = n^{O(\sqrt{n})}$, which leads to (1). To get a better time complexity, we will use the fact that for *random* $\mathbf{b} \in \mathbb{F}_q^n$, $s(\mathbf{b})$ is quasi-polynomial with high probability. This almost immediately implies (2) and (3). The rest of the claims follow from the usual testing algorithms for Reed-Muller codes (though in our case we only need to handle *erasures*).

3.2 Proof of Theorem 4

We first bound the time complexity of the $\text{Invert}(\cdot)$ algorithm. In particular, Lemma 1 and known deterministic bounds on $\text{irr}(k, q)$ and $\text{root}(k, q)$ and the simple fact that $\text{split}(\mathbf{b}, q) \in \mathsf{DTIME}(n^3 \log q)$ show that $\text{Invert}(\mathbf{b})$ can be computed in $\mathsf{DTIME}\left(n^{O(1)} \cdot s(\mathbf{b}, q)^{O(1)} \cdot q^{O(1)}\right)$. Also by using randomized bounds on $\text{irr}(k, q)$ and $\text{root}(k, q)$ one obtains that $\text{Invert}(\mathbf{b})$ can be computed in $\mathsf{RTIME}\left(n^{O(1)} \cdot s(\mathbf{b}, q)^{O(1)} \cdot \log^{O(1)} q\right)$.

The discussion above proves statement along with the fact that the degree of the splitting field is at most $n^{O(\sqrt{n})}$ implies (1).

We will need the following result that follows from [9]:

Corollary 1 (cf. [9]).

$$\Pr_{\mathbf{b} \in \mathbb{F}_q^n} \left[\log(s(\mathbf{b}, q)) > \log^2 n\right] \leqslant \exp\left(-\Omega\left(\sqrt{\log n}\right)\right).$$

Corollary 1 says that for all but an $\exp(-\Omega(\sqrt{n}))$ fraction of $\mathbf{b} \in \mathbb{F}_q^n$, we have $s(\mathbf{b}, q) \leqslant 2^{\log^2 n} = n^{\log n}$. This along with the discussion above proves (2) and (3). (After Step 3 in $\text{Invert}(\cdot)$ if $\log s > \log^2 n$ then we halt and output "fail.")

We now move to the proof of (4). Call $\mathbf{c} \in \mathbb{F}_q^n$ *good* if $s(\mathbf{c}, q) \leqslant n^{\log n}$. (Otherwise call it *bad*.) Note that by Step 2 of $\text{Invert}(\mathbf{c})$, we will know if $\mathbf{c}$ is good or bad. If it is good then we continue with the algorithm else we halt and output "fail." Recall that we want to compute $f(\mathbf{b})$. Note that if $\mathbf{b}$ is good then we can run $\text{Invert}(\mathbf{b})$ in $\mathsf{RTIME}(n^{O(\log n)} \log^{O(1)} q)$ (and hence compute $f_{\text{sym}}(\text{Invert}(\mathbf{b})) = f(\mathbf{b})$). However, in the worst-case $\mathbf{b}$ might be bad. Thus, we do the following: we pick a random line through $\mathbf{b}$ and evaluate the function f restricted to the line on points other than $\mathbf{b}$.

In particular, consider the univariate polynomial $P_{\mathbf{b}}(X) = f(\mathbf{b} + \mathbf{m} \cdot X)$ for a uniformly random $\mathbf{m} \in \mathbb{F}_q^n$. Consider any subset $S \subseteq \mathbb{F}_q^*$ with $|S| = 3(\deg(f)+1)$. (Note that we will need $q - 1 \geqslant 3(\deg(f) + 1)$, which will be satisfied by the condition on $q \geqslant \Omega(\deg(f))$. Now by Corollary 1 in expectation (over the choice of $\mathbf{m}$), at most $\exp(-\Omega(\sqrt{\log n}) \cdot 3(\deg(f) + 1) \leqslant (\deg(f) + 1)$ points in the set $\{\mathbf{b} + \alpha \cdot \mathbf{m} | \alpha \in S\}$ are bad. (The inequality follows for large enough n.) Thus, by Markov's inequality, with probability at least $1/2$ (over the choice of $\mathbf{m}$) there are at most $2(\deg(f)+1)$ bad points in $\{\mathbf{b}+\alpha\mathbf{m} | \alpha \in S\}$. (Recall that we know when a point is bad, and thus we can recognize when we have at most $2(\deg(f) + 1)$ bad points.) In other words, our algorithm will compute $P_{\mathbf{b}}(\mathbf{b} + \alpha\mathbf{m})$ correctly for at least $\deg(f) + 1$ points $\alpha \in S$. It is well-known (e.g. via polynomial interpolation) that we can compute $P_{\mathbf{b}}(X)$ in $\mathsf{DTIME}(\deg(f)^3) \in \mathsf{DTIME}(q^3)$ by our assumption on q. Note that we can now read off $f(\mathbf{b}) = P_{\mathbf{b}}(0)$. Note that the procedure above has an error probability of at most $1/2$. We can reduce this to δ by running $O(\log(1/\delta))$ independent runs of this procedure and stop whenever we compute $f(\mathbf{b})$.

The proof of (5) is a generalization of the proof for (4) to the multivariate case. In particular, given an integer $1 \leqslant x \leqslant n$, we pick a random subspace of $\mathbb{F}_q^n$ of dimension m by picking m basis vectors (say $\mathbf{e}_1, \ldots \mathbf{e}_m$) at random. Now consider the set $S = \{\mathbf{b} + \sum_{j=1}^{m} \alpha_j \cdot \mathbf{e}_j | (\alpha_1, \ldots, \alpha_m) \in \mathbb{F}_q^m\}$. It is easy to see that the function f restricted to S is an m-variate polynomial with the degree at most $\deg(f)$ (here the scalars $\alpha_1, \ldots, \alpha_m$ are thought of as the variables). It follows from the known distance properties of Reed-Muller codes (cf. [4]), we can recover $f(\mathbf{b})$ if at most $q^{-\deg(f)/q}$ fraction of the points in S are bad. This happens with probability at least a $1/2$ (by Markov's argument and Corollary 1) if $\exp(-\Omega(\sqrt{\log n}))$ is at most $q^{-\deg(f)/q}/2$. This inequality is implied by the assumption that $\deg(f) \leqslant O(q\sqrt{\log n}/\log q)$. Also we need $\deg(f) \leqslant (q-1)m$ to ensure that f projected down to S is still a well defined Reed-Muller code (and in particular, we can apply the distance properties of Reed-Muller codes). Finally, we run the procedure above independently $O(\log(1/\delta))$ times to bring the error probability down to δ.

4 Results for the Second Symmetrization

4.1 Functions over Finite Fields

We state our main result for functions defined over finite fields:

Theorem 6. *Let $n \geqslant 1$ be an integer and q be a prime power. Define $s = 1 + \lceil \log n \rceil$. Then for every function $f : \mathbb{F}_q^n \to \mathbb{F}_q$, there exists a symmetric function $f_{\mathrm{sym}} : \mathbb{F}_{q^s}^n \to \mathbb{F}_q$ such that*

$$C(f_{\mathrm{sym}}) \leqslant C(f) + \mathsf{DTIME}(n), \tag{6}$$

$$C(f) \leqslant C(f_{\mathrm{sym}}) + \mathsf{DTIME}(n). \tag{7}$$

Further, $\deg(f_{\mathrm{sym}}) \leqslant snq^2 \cdot \deg(f)$.

In the rest of the section, we will prove the theorem above. Before we describe f_{sym}, we first set up some simple notation (and assumptions). We will assume that we have access to an irreducible polynomial of degree s over $\mathbb{F}_q$ that defines $\mathbb{F}_{q^s}$.[2] In particular, we will assume that every element $\alpha \in \mathbb{F}_{q^s}$ is represented as $\sum_{\ell=0}^{s-1} \alpha_\ell \cdot \gamma^\ell$ for some root $\gamma \in \mathbb{F}_{q^s}$ of the irreducible polynomial. Further, we will assume that $[n]$ is embedded into $\mathbb{F}_q^{s-1}$. (Note that by definition of s, $q^{s-1} \geqslant n$.) From now on we will think of $i \in [n]$ interchangeably as an integer in $[n]$ and an element in $\mathbb{F}_q^{s-1}$. We first claim the existence of certain polynomials.

Lemma 2. *There exist s explicit univariate polynomials $\pi_k : \mathbb{F}_{q^s} \to \mathbb{F}_q$ ($0 \leqslant k \leqslant s-1$) such that for any $\alpha = \alpha_{s-1} \cdot \gamma^{s-1} + \cdots + \alpha_0 \in \mathbb{F}_{q^s}$, $\pi_k(\alpha) = \alpha_k$. Further, $\deg(\pi_k) = q^{s-1}$.*

Proof. For any $\alpha = \sum_{i=0}^{s-1} \alpha_i \gamma^i \in \mathbb{F}_{q^s}$, $\alpha^{q^j} = \sum_{i=0}^{s-1} \alpha_i (\gamma^i)^{q^j}$ for every $0 \leqslant j \leqslant s-1$. Thus, we have

$$(\alpha \;\; \alpha^q \;\; \alpha^{q^2} \;\; \cdots \;\; \alpha^{q^{s-1}})^T = V \cdot (\alpha_0 \;\; \alpha_1 \;\; \alpha_2 \;\; \cdots \;\; \alpha_{s-1})^T,$$

where V is the Vandermonde matrix with the ℓth row being the first s powers of γ^{q^ℓ} (starting from the 0th power)—note that all the elements γ^{q^ℓ} are distinct. Thus, we have that α_k is the inner product of the kth row of the inverse of the Vandermonde matrix and $(\alpha \;\; \alpha^q \;\; \alpha^{q^2} \;\; \cdots \;\; \alpha^{q^{s-1}})$. The definition for $\pi_k(X)$ then follows from the (known) expressions for entries of the inverse of the Vandermonde matrix (cf. [16]). ■

Lemma 3. *Fix $j \in [n]$. There exists an explicit n-variate* symmetric *polynomial $\phi_j(X_1, \ldots, X_n) : (\mathbb{F}_{q^s})^n \to \mathbb{F}_q$ of degree at most sq^s such that for any choice of $\alpha^i = \alpha^i_{s-1} \cdot \gamma^{s-1} + \cdots + \alpha^i_1 \cdot \gamma + \alpha^i_0 \in \mathbb{F}_{q^s}$ ($1 \leqslant i \leqslant n$),*

$$\phi_j(\alpha^1, \alpha^2, \ldots, \alpha^n) = \sum_{i \in [n], (\alpha^i_{s-1}, \alpha^i_{s-2}, \cdots, \alpha^i_1) = j} \alpha^i_0, \tag{8}$$

where we consider $(\alpha^i_{s-1}, \alpha^i_{s-2}, \cdots, \alpha^i_1) \in \mathbb{F}_q^{s-1}$.

Proof. For any $j = (j_1, \ldots, j_{s-1}) \in \mathbb{F}_q^{s-1}$, consider the degree $(s-1)(q-1)$ polynomial $A_j(Y_1, \ldots, Y_{s-1})$ over $\mathbb{F}_q$:

$$A_j(Y_1, \ldots, Y_{s-1}) = \prod_{\ell=1}^{s-1} \prod_{\beta \in \mathbb{F}_q, \beta \neq j_\ell} \left(\frac{Y_\ell - \beta}{j_\ell - \beta} \right).$$

Note that $A_j(i) = 1$ if $i = j$ else $A_j(i) = 0$ for every $i \in \mathbb{F}_q^{s-1}$. Now consider the polynomial

$$\phi_j(X_1, \ldots, X_n) = \sum_{i=1}^{n} A_j(\pi_1(X_i), \ldots, \pi_{s-1}(X_i)) \cdot \pi_0(X_i).$$

[2] Otherwise in the reduction we can compute one in time $\mathsf{DTIME}(s^{O(1)} \cdot q^{O(1)})$.

Now by the properties of $A_j(\cdot)$ mentioned above, in the RHS of the equation above for $\phi_j(\alpha^1, \ldots, \alpha^n)$, the only summands that will contribute are those i for which $(\alpha^i_{s-1}, \alpha^i_{s-2}, \cdots, \alpha^i_1) = j$ (this follows from Lemma 2). Further each such summand contributes α^i_0 to the sum (by Lemma 2). This proves (8). Further, it follows from definition that ϕ_j is a symmetric polynomial. Finally, note that for every $i \in [n]$, $A_j(\pi_1(X_i), \ldots, \pi_{s-1}(X_i))$ has degree $(s-1)(q-1) \cdot q^{s-1}$. Further, as π_0 has degree q^{s-1}, $\deg(\phi_j) = q^{s-1}(s-1)(q-1) + q^{s-1} \leqslant sq^s$. ■

We are now ready to define $f_{\mathrm{sym}} : \mathbb{F}_{q^s} \to \mathbb{F}_q$. For any $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n_{q^s}$, define

$$f_{\mathrm{sym}}(\mathbf{a}) = f\left(\phi_1(\mathbf{a}), \phi_2(\mathbf{a}), \ldots, \phi_n(\mathbf{a})\right).$$

Since each of $\phi_1, \ldots, \phi_n$ are symmetric, so is f_{sym}. Further, as each of $\phi_1, \ldots, \phi_n$ has degree at most sq^s, $\deg(f_{\mathrm{sym}}) \leqslant sq^s \cdot \deg(f) \leqslant snq^2 \cdot \deg(f)$, where the last inequality follows from the fact that our choice of s implies $q^{s-1} \leqslant nq$.

In what follows, we will assume that any $\alpha \in \mathbb{F}_{q^s}$ is presented to us as $(\alpha_{s-1}, \ldots, \alpha_1, \alpha_0)$, where $\alpha = \alpha_{s-1}\gamma^{s-1} + \cdots + \alpha_1\gamma + \alpha_0$. Note that this implies that $\pi_k(\alpha)$ (for $0 \leqslant k \leqslant s-1$) can be "computed" in constant time.[3]

We first prove (6). Given $\mathbf{b} \in \mathbb{F}^n_{q^s}$, we will compute $\mathbf{a} \in \mathbb{F}^n_q$ in $\mathsf{DTIME}(n)$ such that $f_{\mathrm{sym}}(\mathbf{b}) = f(\mathbf{a})$. Note that this suffices to prove (6). Notice that by definition of f_{sym}, this is satisfied if $a_j = \phi_j(\mathbf{b})$ for $j \in [n]$. Further note that given $\beta_1, \ldots, \beta_{s-1} \in \mathbb{F}_q$, one can compute $A_j(\beta_1, \ldots, \beta_{s-1})$ in $\mathsf{DTIME}(s)$.[4] This along with the assumption that $\pi_k(\alpha)$ can be computed in constant time for any $\alpha \in \mathbb{F}_{q^s}$, implies that $\phi_j(\mathbf{b})$ can be computed in $\mathsf{DTIME}(ns)$. This would immediately imply a total of $\mathsf{DTIME}(n^2 s)$ for computing $\mathbf{a}$. However, note that as all the a_j values are sums, we can compute $\mathbf{a}$ in one pass over $\mathbf{b}$ with space $O(n)$. Thus, we can compute $\mathbf{a}$ from $\mathbf{b}$ in $\mathsf{DTIME}(n)$, as desired.

Finally, we prove (7). We will show that given $\mathbf{a} \in \mathbb{F}^n_q$ we can compute $\mathbf{b} \in \mathbb{F}^n_{q^s}$ in $\mathsf{DTIME}(n)$ such that $f(\mathbf{a}) = f_{\mathrm{sym}}(\mathbf{b})$. Note that this will prove (7). Further, notice that we will be done if we can show that for $j \in [n]$, $a_j = \phi_j(\mathbf{b})$. The definition actually is pretty easy: for every $i = (i_{s-1}, \ldots, i_1) \in [n]$, define $b_i = i_{s-1}\gamma^{s-1} + \cdots + i_1\gamma + a_i$. Now it can be checked that $\phi_j(\mathbf{b}) = \pi_0(b_j) = a_j$, as desired. Further, it is easy to check that one can compute $\mathbf{b}$ in $\mathsf{DTIME}(n)$.

4.2 Functions over Reals

We state our main result for $\mathbb{F} = \mathbb{R}$:

Theorem 7. *Let $n \geqslant 1$ be an integer. Then for every function $f : \mathbb{R}^n \to \mathbb{R}$, there exists a symmetric function $f_{\mathrm{sym}} : \mathbb{R}^n \to \mathbb{R}$ such that*

$$C(f_{\mathrm{sym}}) \leqslant C(f) + \mathsf{DTIME}(n), \tag{9}$$

[3] If not, by the proof of Lemma 2, one can compute both values in $O(s^2 \log q)$ operations over $\mathbb{F}_{q^s}$.

[4] Here we are also assuming that the map from $[n]$ to the corresponding element in $\mathbb{F}^{s-1}_q$ can be computed in constant time.

$$C(f) \leqslant C(f_{\text{sym}}) + \mathsf{DTIME}(n). \tag{10}$$

The proof is similar to the one for finite fields, so we only sketch the differences here. Towards this end, we think of every element $x \in \mathbb{R}$ as a triple $(\lfloor \lfloor x \rfloor / n \rfloor, \lfloor x \rfloor \bmod n, x - \lfloor x \rfloor) \in \mathbb{Z} \times \mathbb{Z} \times [0, 1)$. In particular, for any $1 \leqslant j \leqslant n$, we define

$$\phi_j((u_1, v_1, w_1), (u_2, v_2, w_2), \ldots, (u_n, v_n, w_n)) = \sum_{i=1}^{n} \delta_{v_i,j} \cdot (u_i + w_i),$$

where $\delta_{\ell,k} = 1$ if $\ell = k$ and is zero otherwise. f_{sym} is defined as before and the reduction from f_{sym} to f is also as before. The reduction from f to f_{sym} is also pretty much the same as before except we define $b_i = i + n \cdot \lfloor a_i \rfloor + a_i - \lfloor a_i \rfloor$.

Acknowledgments. We thanks an anonymous reviewer for comments that improved the presentation of the paper.

References

1. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and hardness of approximation problems. J. Assn. Comp. Mach. 45, 501–555 (1998)
2. Arora, S., Safra, S.: Probabilistic checking of proofs: a new characterization of NP. J. Assn. Comp. Mach. 45, 70–122 (1998)
3. Arora, S., Sudan, M.: Improved low-degree testing and its applications. Combinatorica 23(3), 365–426 (2003)
4. Assumus Jr., E.F., Key, J.D.: Polynomial codes and Finite Geometries in Handbook of Coding Theory. In: Pless Jr., V.S., Huffman, W.C. (eds.), vol. II, ch.16. Elsevier, Amsterdam (1998)
5. Baur, W., Strassen, V.: The complexity of partial derivatives. Theor. Comp. Sci. 22, 317–330 (1982)
6. Beame, P., Brisson, E., Ladner, R.: The complexity of computing symmetric functions using threshold circuits. Theor. Comp. Sci. 100, 253–265 (1992)
7. Beigel, R., Tarui, J.: On ACC. In: Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science, pp. 783–792 (1991)
8. Ben-Or, M.: Lower bounds for algebraic computation trees. In: Proc. 15th Annual ACM Symposium on the Theory of Computing, pp. 80–86 (1983)
9. Dixon, J.D., Panario, D.: The degree of the splitting field of a random polynomial over a finite field. The Electronic Journal of Combinatorics 11(1) (2004), http://www.combinatorics.org/Volume_11/Abstracts/v11i1r70.html
10. Furst, M., Saxe, J., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. Math. Sys. Thy. 17, 13–27 (1984)
11. Grigoriev, D., Karpinski, M.: An exponential lower bound for depth 3 arithmetic circuits. In: Proc. 30th Annual ACM Symposium on the Theory of Computing, pp. 577–582 (1998)
12. Grigoriev, D., Razborov, A.: Exponential lower bounds for depth 3 algebraic circuits in algebras of functions over finite fields. Applicable Algebra in Engineering, Communication, and Computing 10, 465–487 (2000) (preliminary version FOCS 1998)

13. Grolmusz, V.: Computing elementary symmetric polynomials with a subpolynomial number of multiplications. SIAM Journal on Computing 32, 2002–02 (2002)
14. Jutla, C.S., Patthak, A.C., Rudra, A., Zuckerman, D.: Testing low-degree polynomials over prime fields. Random Struct. Algorithms 35(2), 163–193 (2009)
15. Kaufman, T., Ron, D.: Testing polynomials over general fields. SIAM Journal on Computing 36(3), 779–802 (2006)
16. Klinger, A.: The Vandermonde matrix. The American Mathematical Monthly 74(5), 571–574 (1967)
17. Lu, C.J.: An exact characterization of symmetric functions in $q\mathrm{AC}^0[2]$. In: Proc. 4th International Combinatorics and Computing Conference, pp. 167–173 (1998)
18. Razborov, A.: Lower bounds for the size of circuits of bounded depth with basis $\{\wedge, \oplus\}$. Mathematical Notes (formerly of the Academy of Natural Sciences of the USSR) 41, 333–338 (1987)
19. Razborov, A.: On the method of approximations. In: Proc. 21st Annual ACM Symposium on the Theory of Computing, pp. 167–176 (1989)
20. Shoup, V.: A computational introduction to number theory and algebra. Cambridge University Press, New York (2008)
21. Smolensky, R.: Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In: Proc. 19th Annual ACM Symposium on the Theory of Computing, pp. 77–82 (1987)

Compressed Word Problems for Inverse Monoids

Markus Lohrey

Universität Leipzig, Institut für Informatik, Germany
lohrey@informatik.uni-leipzig.de

Abstract. The compressed word problem for a finitely generated monoid M asks whether two given compressed words over the generators of M represent the same element of M. For string compression, straight-line programs, i.e., context-free grammars that generate a single string, are used in this paper. It is shown that the compressed word problem for a free inverse monoid of finite rank at least two is complete for Π_2^p (second universal level of the polynomial time hierarchy). Moreover, it is shown that there exists a fixed finite idempotent presentation (i.e., a finite set of relations involving idempotents of a free inverse monoid), for which the corresponding quotient monoid has a PSPACE-complete compressed word problem. The ordinary uncompressed word problem for such a quotient can be solved in logspace [10]. Finally, a PSPACE-algorithm that checks whether a given element of a free inverse monoid belongs to a given rational subset is presented. This problem is also shown to be PSPACE-complete (even for a fixed finitely generated submonoid instead of a variable rational subset).

1 Introduction

The decidability and complexity of algorithmic problems in finitely generated monoids and groups is a classical topic at the borderline of computer science and mathematics. The most basic question of this kind is the *word problem*, which asks whether two words over the generators represent the same element. Markov and Post proved independently that the word problem for finitely presented monoids is undecidable in general. Later, Novikov and Boone extended the result of Markov and Post to finitely presented groups, see the the survey [15] for references.

In this paper, we are interested in *inverse monoids*. A monoid is inverse, if for each element x there exists a unique "inverse" x^{-1} such that $x = xx^{-1}x$ and $x^{-1} = x^{-1}xx^{-1}$ [3]. In the same way as groups can be represented by sets of permutations, inverse monoids can be represented by sets of partial injections [3]. Algorithmic questions for inverse monoids received increasing attention in the past and inverse monoid theory found several applications in combinatorial group theory, see e.g. [10] and the survey [15] for further references.

Since the class of inverse monoids forms a variety of algebras (with respect to the operations of multiplication, inversion, and the identity element), the free inverse monoid $\mathsf{FIM}(\Gamma)$ generated by a set Γ exists. Munn gave in [16] an explicit representation of the free inverse monoid $\mathsf{FIM}(\Gamma)$. Elements can be represented by finite subtrees of the Cayley-graph of the free group generated by Γ (so called *Munn trees*). Moreover, there are two distinguished nodes (an initial node and a final node). Multiplication of two elements of $\mathsf{FIM}(\Gamma)$ amounts of gluing the two Munn trees together, where the final node

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 448–459, 2011.

of the first Munn tree is identified with the initial node of the second Munn tree. This gives rise to a very simple algorithm for the word problem of $\mathsf{FIM}(\Gamma)$, which can moreover implemented in linear time. In [10], it was also shown (using Munn trees together with a result of Lipton and Zalcstein [5] saying that the word problem for a finitely generated free group can be solved in logspace) that the word problem for $\mathsf{FIM}(\Gamma)$ can be solved in logspace.

Although the word problem for a free inverse monoid can be solved very efficiently, there are several subtle differences between the algorithmic properties of free inverse monoids on the one hand and free monoids and free groups on the other hand. Let us give two examples:

- Solvability of equations: By the seminal results of Makanin, this problem is decidable for free monoids and free groups. On the other hand, solvability of equations in a finitely generated free inverse monoid of rank at least 2 (the rank is the minimal number of generators) is undecidable [19].
- Rational subset membership problem: Membership in a given rational subset of a free monoid or free group can be decided in polynomial time. The same problem is NP-complete for finitely generated free inverse monoids of rank at least two [2].

In this paper, we show that in a certain sense also the word problem is harder for free inverse monoids than free monoids (groups). For this we consider the *compressed word problem*, where the input words are given succinctly by so called *straight-line programs* (SLPs) [18]. An SLP is a context free grammar that generates only one word, see Section 4. Since the length of this word may grow exponentially with the size (number of productions) of the SLP, SLPs can be seen as a compact string representation. SLPs turned out to be a very flexible compressed representation of strings, which are well suited for studying algorithms for compressed strings; see [8] for references. In the compressed word problem for a finitely generated monoid M the input consists of two SLPs that generate words over the generators of M, and it is asked whether these two words represent the same element of M. Hence, the compressed word problem for a free monoid simply asks, whether two SLPs generate the same word. Plandowski proved in [17] that this problem can be solved in polynomial time; the best algorithm is due to Lifshits [4] and has a cubic running time. Based on Plandowski's result, it was shown in [7] that the compressed word problem for a free group can be solved in polynomial time. This result has algorithmic implications for the ordinary (uncompressed) word problem: In [11,20] it was shown that the word problem for the automorphism group of a group G can be reduced in polynomial time to the *compressed* word problem for G (more general: the word problem for the endomorphism monoid of a monoid M can be reduced in polynomial time to the *compressed* word problem for M). Hence, the word problem for the automorphism group of a free group turned out to be solvable in polynomial time [20], which solved an open problem from combinatorial group theory. Generalizations of this result for larger classes of groups can be found in [11,13].

Our first main result states that the compressed word problem for every finitely generated free inverse monoid of rank at least two is complete for Π_2^p, the second universal level of the polynomial time hierarchy (Thm. 4). The upper bound follows easily using Munn's solution for the word problem together with the above mentioned result of Lipton and Zalcstein for free groups. The lower bound is based on a reduction from

a variant of the SUBSETSUM problem together with an encoding of a SUBSETSUM instance by an SLP [7]. Hence, the compressed word problem for free inverse monoids is indeed computationally harder than the compressed word problem for free monoids (groups) (unless $\mathsf{P} = \Pi_2^p$). It is not difficult to see that the compressed word problem for a free inverse monoid of rank 1 can be solved in polynomial time (Prop. 1).

In [14], Margolis and Meakin presented a large class of finitely presented inverse monoids with decidable word problems. An inverse monoid from that class is of the form $\mathsf{FIM}(\Gamma)/P$, where P is a presentation consisting of a finite number of relations $e = f$, where e and f are idempotents of $\mathsf{FIM}(\Gamma)$; we call such a presentation idempotent. An alternative proof for the decidability result of Margolis and Meakin was given in [21]. In [10] it was shown that the word problem for every inverse monoid $\mathsf{FIM}(\Gamma)/P$, where P is an idempotent presentation, can be solved in logspace. This implies that the compressed word problem for each of these inverse monoids belongs to the class PSPACE. Our second main result states that the are specific idempotent presentations P such that the compressed word problem for $\mathsf{FIM}(\Gamma)/P$ is PSPACE-complete (Thm. 5).

In the last part of the paper we consider the compressed variant of the rational subset membership problem. The class of rational subsets of a monoid M is the smallest class of subsets, which contains all finite subsets, and which is closed under union, product and Kleene star (A^* is the submonoid generated by the subset $A \subseteq M$). If M is finitely generated by Γ, then a rational subset of M can be represented by a finite automaton over the alphabet Γ. In this case, the rational subset membership problem asks, whether a given element of M (given by a finite word over Γ) belongs to a given rational subset (given by a finite automaton over Γ). Especially for groups, this problem is intensively studied, see e.g. [12]. In [2], it was shown that the rational subset membership problem for a free inverse monoid of finite rank at least two is NP-complete. Here, we consider the *compressed rational subset membership problem*, where the input consists of an SLP-compressed word over the generators and a finite automaton over the generators. We show that the compressed rational subset membership problem for a free inverse monoid of finite rank at least two is PSPACE-complete. The difficult part of the proof is to show membership in PSPACE. PSPACE-hardness holds already for the case that the rational subset is a fixed finitely generated submonoid (Thm. 6).

Proofs that are omitted in this paper can be found in the long version [9].

2 Preliminaries

Let Γ be a finite alphabet. The *empty word* over Γ is denoted by ε. Let $s = a_1 \cdots a_n \in \Gamma^*$ be a word over Γ, where $n \geq 0$ and $a_1, \ldots, a_n \in \Gamma$ for $1 \leq i \leq n$. The *length* of s is $|s| = n$. For $1 \leq i \leq n$ let $s[i] = a_i$ and for $1 \leq i \leq j \leq n$ let $s[i,j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $s[i,j] = \varepsilon$. For $n \in \mathbb{N}$ let $\Gamma^{\leq n} = \{w \in \Gamma^* \mid |w| \leq n\}$. We write $s \preceq t$ for $s, t \in \Gamma^*$, if s is a prefix of t. A set $A \subseteq \Gamma^*$ is *prefix-closed*, if $u \preceq v \in A$ implies $u \in A$. We denote with $\Gamma^{-1} = \{a^{-1} \mid a \in \Gamma\}$ a disjoint copy of the finite alphabet Γ. For $a^{-1} \in \Gamma^{-1}$ we define $(a^{-1})^{-1} = a$; thus, $^{-1}$ becomes an involution on the alphabet $\Gamma \cup \Gamma^{-1}$. We extend this involution to words from $(\Gamma \cup \Gamma^{-1})^*$ by setting $(a_1 \cdots a_n)^{-1} = a_n^{-1} \cdots a_1^{-1}$, where $a_i \in \Gamma \cup \Gamma^{-1}$. For $a \in \Gamma \cup \Gamma^{-1}$ and $n \geq 0$ we

use a^{-n} as an abbreviation for the word $(a^{-1})^n$. We use standard terminology from automata theory. A *nondeterministic finite automaton* (NFA) over an input alphabet Γ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. For a *deterministic finite automaton*, $\delta : Q \times \Sigma \to_p Q$ is a partial mapping from $Q \times \Sigma$ to Q.

Complexity theory: We assume some basic background in complexity theory. Recall that Π_2^p (the second universal level of the polynomial time hierarchy) is the class of all languages L for which there exists a polynomial time predicate $P(x, y, z)$ and a polynomial $p(n)$ such that $L = \{x \in \Sigma^* \mid \forall y \in \Sigma^{\leq p(|x|)} \exists z \in \Sigma^{\leq p(|x|)} : P(x, y, z)\}$. POLYLOGSPACE denotes the class $\mathsf{NSPACE}(\log(n)^{O(1)}) = \mathsf{DSPACE}(\log(n)^{O(1)})$. A PSPACE-transducer is a deterministic Turing machine with a read-only input tape, a write-only output tape and a working tape, whose length is bounded by $n^{O(1)}$, where n is the input length. The output is written from left to right on the output tape, i.e., in each step the transducer either outputs a new symbol on the output tape, in which case the output head moves one cell to the right, or the transducer does not output a new symbol in which case the output head does not move. Moreover, we assume that the transducer terminates for every input. This implies that a PSPACE-transducer computes a mapping $f : \Sigma^* \to \Theta^*$, where $|f(w)|$ is bounded by $2^{|w|^{O(1)}}$. A POLYLOGSPACE-transducer is defined in the same way as a PSPACE-transducer, except that the length of the working tape is bounded by $\log(n)^{O(1)}$. The proof of the following lemma uses the same idea that shows that logspace reducibility is transitive.

Lemma 1. *Assume that $f : \Sigma^* \to \Theta^*$ can be computed by a* PSPACE*-transducer and that $g : \Theta^* \to \Delta^*$ can be computed by a* POLYLOGSPACE*-transducer. Then the mapping $f \circ g$ can be computed by a* PSPACE*-transducer. In particular, if the language $L \subseteq \Theta^*$ belongs to* POLYLOGSPACE*, then $f^{-1}(L)$ belongs to* PSPACE*.*

Free groups: It is common to identify a congruence α on a monoid M with the surjective homomorphism from M to the quotient M/α that maps an element $m \in M$ to the congruence class of m with respect to α. The *free group* $\mathsf{FG}(\Gamma)$ generated by the set Γ is the quotient monoid

$$\mathsf{FG}(\Gamma) = (\Gamma \cup \Gamma^{-1})^*/\delta, \tag{1}$$

where δ is the smallest congruence on $(\Gamma \cup \Gamma^{-1})^*$ that contains all pairs (bb^{-1}, ε) for $b \in \Gamma \cup \Gamma^{-1}$. It is well known that for every $u \in (\Gamma \cup \Gamma^{-1})^*$ there exists a unique word $r(u) \in (\Gamma \cup \Gamma^{-1})^*$ (the *reduced normal form of* u) such that $\delta(u) = \delta(r(u))$ and $r(u)$ does not contain a factor of the form bb^{-1} for $b \in \Gamma \cup \Gamma^{-1}$. It holds $\delta(u) = \delta(v)$ if and only if $r(u) = r(v)$. Since the word $r(u)$ can be calculated from u in linear time, the word problem for $\mathsf{FG}(\Gamma)$ can be solved in linear time. Let $\mathsf{IRR}(\Gamma) = \{r(u) \mid u \in (\Gamma \cup \Gamma^{-1})^*\}$ be the set of all *irreducible* words. The epimorphism $\delta : (\Gamma \cup \Gamma^{-1})^* \to \mathsf{FG}(\Gamma)$ restricted to $\mathsf{IRR}(\Gamma)$ is a bijection.

The Cayley-graph of $\mathsf{FG}(\Gamma)$ with respect to the standard generating set $\Gamma \cup \Gamma^{-1}$ will be denoted by $\mathcal{C}(\Gamma)$. Its vertex set is $\mathsf{FG}(\Gamma)$ and there is an a-labeled edge ($a \in \Gamma \cup \Gamma^{-1}$) from $x \in \mathsf{FG}(\Gamma)$ to $y \in \mathsf{FG}(\Gamma)$ if $y = xa$ in $\mathsf{FG}(\Gamma)$. Note that $\mathsf{FG}(\Gamma)$ is a

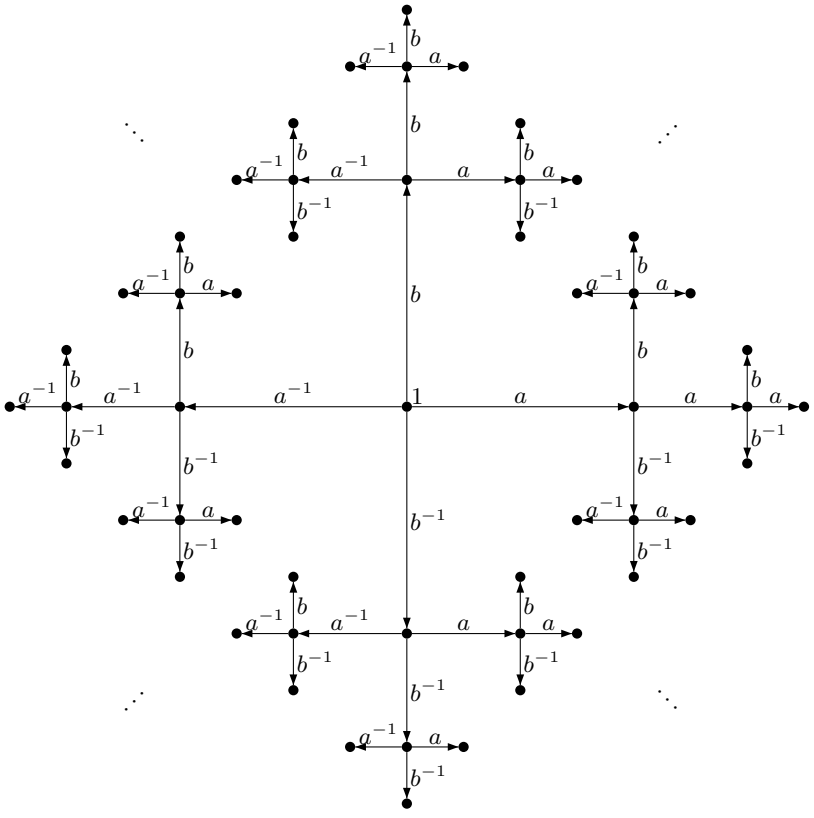


Fig. 1. The Cayley-graph $\mathcal{C}(\{a, b\})$ of the free group $\mathsf{FG}(\{a, b\})$

finitely-branching tree. Figure 1 shows a finite portion of $\mathcal{C}(\{a, b\})$. Here, and in the following, we only draw one directed edge between two points. Thus, for every drawn a-labeled edge we omit the a^{-1}-labeled reversed edge.

3 Inverse Monoids

A monoid M is called an *inverse monoid* if for every $m \in M$ there is a *unique* $m^{-1} \in M$ such that $m = mm^{-1}m$ and $m^{-1} = m^{-1}mm^{-1}$. For detailed reference on inverse monoids see [3]; here we only recall the basic notions. Since the class of inverse monoids forms a variety of algebras (with respect to the operations of multiplication, inversion, and the identity element), the free inverse monoid $\mathsf{FIM}(\Gamma)$ generated by a set Γ exists. Vagner gave an explicit presentation of $\mathsf{FIM}(\Gamma)$: Let ρ be the smallest congruence on the free monoid $(\Gamma \cup \Gamma^{-1})^*$ which contains for all words $v, w \in (\Gamma \cup \Gamma^{-1})^*$ the pairs $(w, ww^{-1}w)$ and $(ww^{-1}vv^{-1}, vv^{-1}ww^{-1})$; these identities are also called Vagner equations. Then $\mathsf{FIM}(\Gamma) \simeq (\Gamma \cup \Gamma^{-1})^*/\rho$. An element x of an inverse monoid M is idempotent (i.e., $x^2 = x$) if and only if x is of the form mm^{-1} for some $m \in M$.

Hence, Vagner's presentation of $\mathsf{FIM}(\Gamma)$ implies that idempotent elements in an inverse monoid commute. Since the Vagner equations are true in the free group $\mathsf{FG}(\Gamma)$, there exists a congruence γ on $\mathsf{FIM}(\Gamma)$ such that $\mathsf{FG}(\Gamma) = \mathsf{FIM}(\Gamma)/\gamma$. When viewing congruences as homomorphisms, we have $\delta = \rho \circ \gamma$, where δ is the congruence on $(\Gamma \cup \Gamma^{-1})^*$ from (1). Elements of $\mathsf{FIM}(\Gamma)$ can be also represented via *Munn trees*: The Munn tree $\mathsf{MT}(u)$ of $u \in (\Gamma \cup \Gamma^{-1})^*$ is a finite and prefix-closed subset of $\mathsf{IRR}(\Gamma)$; it is defined by

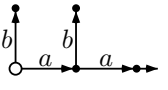
$$\mathsf{MT}(u) = \{r(v) \mid v \preceq u\}.$$

By identifying an irreducible word $v \in \mathsf{IRR}(\Gamma)$ with the group element $\delta(v)$, $\mathsf{MT}(u)$ becomes the set of all nodes along the unique path in $C(\Gamma)$ that starts in 1 and that is labeled with the word u. The subgraph of the Cayley-graph $C(\Gamma)$, which is induced by $\mathsf{MT}(u)$ is connected. Hence it is a finite tree and we can identify $\mathsf{MT}(u)$ with this tree. The following result is known as Munn's Theorem:

Theorem 1 ([16]). *For all $u, v \in (\Gamma \cup \Gamma^{-1})^*$, we have: $\rho(u) = \rho(v)$ if and only if ($r(u) = r(v)$ and $\mathsf{MT}(u) = \mathsf{MT}(v)$).*

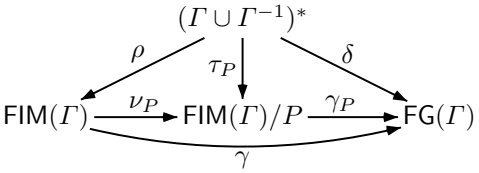
Thus, $\rho(u) \in \mathsf{FIM}(\Gamma)$ can be uniquely represented by the pair $(\mathsf{MT}(u), r(u))$. In fact, if we define on the set of all pairs $(U, v) \in 2^{\mathsf{IRR}(\Gamma)} \times \mathsf{IRR}(\Gamma)$ (with $v \in U$ and U finite and prefix-closed) a multiplication by $(U, v)(V, w) = (r(U \cup vV), r(vw))$, then the resulting monoid is isomorphic to $\mathsf{FIM}(\Gamma)$. Quite often, we represent an element $\rho(u) \in \mathsf{FIM}(\Gamma)$ by a diagram for its Munn tree, where in addition the node ε is represented by a bigger circle and the node $r(u)$ is marked by an outgoing arrow. If $r(u) = \varepsilon$, then we omit this arrow. By Thm. 1 such a diagram uniquely specifies an element of $\mathsf{FIM}(\Gamma)$.

Example 1. The diagram for $\rho(bb^{-1}abb^{-1}a) \in \mathsf{FIM}(\{a, b\})$ looks as follows:



Thm. 1 leads to a polynomial time algorithm for the word problem for $\mathsf{FIM}(\Gamma)$. For instance, the reader can easily check that $bb^{-1}abb^{-1}a = aaa^{-1}bb^{-1}a^{-1}bb^{-1}aa$ in $\mathsf{FIM}(\{a, b\})$ by using Munn's Theorem. In fact, every word that labels a path from ε to aa (the node with the outgoing arrow) and that visits all nodes of the above diagram represents the same element of $\mathsf{FIM}(\{a, b\})$ as $bb^{-1}abb^{-1}a$. Munn's theorem also implies that an element $\rho(u) \in \mathsf{FIM}(\Gamma)$ (where $u \in (\Gamma \cup \Gamma^{-1})^*$) is idempotent (i.e., $\rho(uu) = \rho(u)$) if and only if $r(u) = \varepsilon$.

For a finite set $P \subseteq (\Gamma \cup \Gamma^{-1})^* \times (\Gamma \cup \Gamma^{-1})^*$ define $\mathsf{FIM}(\Gamma)/P = (\Gamma \cup \Gamma^{-1})^*/\tau_P$ to be the inverse monoid with the set Γ of generators and the set P of relations, where τ_P is the smallest congruence on $(\Gamma \cup \Gamma^{-1})^*$ generated by $\rho \cup P$. Viewed as a morphism, this congruence factors as $\tau_P = \rho \circ \nu_P$ with $\mathsf{FIM}(\Gamma)/\nu_P = \mathsf{FIM}(\Gamma)/P$. We say that $P \subseteq (\Gamma \cup \Gamma^{-1})^* \times (\Gamma \cup \Gamma^{-1})^*$ is an *idempotent presentation* if for all $(e, f) \in P$, $\rho(e)$ and $\rho(f)$ are both idempotents of $\mathsf{FIM}(\Gamma)$, i.e., $r(e) = r(f) = \varepsilon$ by the remark above. In this paper, we are concerned with inverse monoids of the form $\mathsf{FIM}(\Gamma)/P$ for a finite idempotent presentation P. In this case, since every identity $(e, f) \in P$ is true in $\mathsf{FG}(\Gamma)$ (we have $\delta(e) = \delta(f) = 1$), there also exists a congruence γ_P on $\mathsf{FIM}(\Gamma)/P$ with $(\mathsf{FIM}(\Gamma)/P)/\gamma_P = \mathsf{FG}(\Gamma)$. The following commutative diagram summarizes all morphisms introduced so far.



In the sequel, the meaning of the congruences $\rho, \delta, \gamma_P, \gamma, \tau_P$, and ν_P will be fixed.

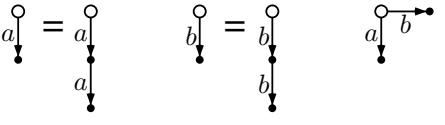
To solve the word problem for $\mathsf{FIM}(\Gamma)/P$, Margolis and Meakin [14] used a closure operation for Munn trees, which is based on work of Stephen [22]. We shortly review the ideas here. As remarked in [14], every idempotent presentation P can be replaced by the idempotent presentation $P' = \{(e, ef), (f, ef) \mid (e, f) \in P\}$, i.e., $\mathsf{FIM}(\Gamma)/P = \mathsf{FIM}(\Gamma)/P'$. Since $\mathsf{MT}(e) \subseteq \mathsf{MT}(ef) \supseteq \mathsf{MT}(f)$ if $r(e) = r(f) = \varepsilon$, we can restrict in the following to idempotent presentations P such that $\mathsf{MT}(e) \subseteq \mathsf{MT}(f)$ for all $(e, f) \in P$. Define a rewriting relation $\Rightarrow_P$ on prefix-closed subsets of $\mathsf{IRR}(\Gamma)$ as follows, where $U, V \subseteq \mathsf{IRR}(\Gamma)$: $U \Rightarrow_P V$ if and only if

$$\exists (e, f) \in P \; \exists u \in U \big(r(u\,\mathsf{MT}(e)) \subseteq U \text{ and } V = U \cup r(u\,\mathsf{MT}(f)) \big).$$

Finally, define the closure of $U \subseteq \mathsf{IRR}(\Gamma)$ with respect to the presentation P as

$$\mathsf{cl}_P(U) = \bigcup \{V \mid U \stackrel{*}{\Rightarrow}_P V\}.$$

Example 2. Assume that $\Gamma = \{a, b\}$, $P = \{(aa^{-1}, a^2a^{-2}), (bb^{-1}, b^2b^{-2})\}$ and $u = aa^{-1}bb^{-1}$. The graphical representations for these elements look as follows:



Then the closure $\mathsf{cl}_P(\mathsf{MT}(u))$ is $\{a^n \mid n \geq 0\} \cup \{b^n \mid n \geq 0\} \subseteq \mathsf{IRR}(\Gamma)$.

Margolis and Meakin proved the following result:

Theorem 2 ([14]). *Let P be an idempotent presentation and let $u, v \in (\Gamma \cup \Gamma^{-1})^*$. Then $\tau_P(u) = \tau_P(v)$ if and only if $(r(u) = r(v)$ and $\mathsf{cl}_P(\mathsf{MT}(u)) = \mathsf{cl}_P(\mathsf{MT}(v)))$.*

The result of Munn for $\mathsf{FIM}(\Gamma)$ (Thm. 1) is a special case of this result for $P = \emptyset$. Note also that $\mathsf{cl}_P(\mathsf{MT}(u)) = \mathsf{cl}_P(\mathsf{MT}(v))$ if and only if $\mathsf{MT}(u) \subseteq \mathsf{cl}_P(\mathsf{MT}(v))$ and $\mathsf{MT}(v) \subseteq \mathsf{cl}_P(\mathsf{MT}(u))$. Margolis and Meakin used Thm. 2 in connection with Rabin's tree theorem in order to give a solution for the word problem for the monoid $\mathsf{FIM}(\Gamma)/P$. Using tree automata techniques, a logspace algorithm for the word problem for $\mathsf{FIM}(\Gamma)/P$ was given in [10]. For this result, it is important that the idempotent presentation P is not part of the input. The uniform version of the word problem, where P is part of the input, is EXPTIME-complete [10].

4 Straight-Line Programs

We are using straight-line programs as a succinct representation of strings with reoccurring subpatterns [18]. A *straight-line program (SLP) over a finite alphabet Γ* is a

context free grammar $\mathbb{A} = (V, \Gamma, S, P)$, where V is the set of *nonterminals*, Γ is the set of *terminals*, $S \in V$ is the *initial nonterminal*, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of *productions* such that (i) for every $X \in V$ there is exactly one $\alpha \in (V \cup \Gamma)^*$ with $(X, \alpha) \in P$ and (ii) there is no cycle in the relation $\{(X, Y) \in V \times V \mid \exists \alpha \in (V \cup \Gamma)^* Y (V \cup \Gamma)^* : (X, \alpha) \in P\}$. These conditions ensure that the language generated by the straight-line program $\mathbb{A}$ contains exactly one word $\mathsf{val}(\mathbb{A})$.

Remark 1. The following problems can be solved in polynomial time:

(a) Given an SLP $\mathbb{A}$, calculate $|\mathsf{val}(\mathbb{A})|$ in binary representation.
(b) Given an SLP $\mathbb{A}$ and two binary coded numbers $1 \leq i \leq j \leq |\mathsf{val}(\mathbb{A})|$, compute an SLP $\mathbb{B}$ with $\mathsf{val}(\mathbb{B}) = \mathsf{val}(\mathbb{A})[i, j]$.

Also notice that $\mathsf{val}(\mathbb{A})$ can be computed from $\mathbb{A}$ by a PSPACE-transducer.

Plandowski [17] presented a polynomial time algorithm for testing whether $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ for two given SLPs $\mathbb{A}$ and $\mathbb{B}$. A cubic algorithm was presented by Lifshits [4].

Let M be a finitely generated monoid and let Γ be a finite generating set for M. The *compressed word problem* for M is the following computational problem:

INPUT: SLPs $\mathbb{A}$ and $\mathbb{B}$ over the alphabet Γ.
QUESTION: Does $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ hold in M?

The above mentioned result of Plandowski [17] means that the compressed word problem for a finitely generated free monoid can be solved in polynomial time. The following result was shown in [7].

Theorem 3 ([7]). *For every finite alphabet Γ, the compressed word problem for $\mathsf{FG}(\Gamma)$ can be solved in polynomial time (and is* P*-complete if $|\Gamma| \geq 2$).*

5 Compressed Word Problem for $\mathsf{FIM}(\Gamma)$

Recall that the word problem for $\mathsf{FIM}(\Gamma)$ can be solved in logspace [10]. In the compressed setting we have:

Theorem 4. *For every finite alphabet Γ with $|\Gamma| \geq 2$, the compressed word problem for $\mathsf{FIM}(\Gamma)$ is Π_2^p-complete.*

Proof. For the Π_2^p upper bound, let $\mathbb{A}$ and $\mathbb{B}$ be SLPs over some alphabet $\Gamma \cup \Gamma^{-1}$ and let $m = |\mathsf{val}(\mathbb{A})|$ and $n = |\mathsf{val}(\mathbb{B})|$. These numbers can be computed in polynomial time by Remark 1. By Thm. 1, we have $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ in $\mathsf{FIM}(\Gamma)$ if and only if:

$$\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B}) \text{ in } \mathsf{FG}(\Gamma) \tag{2}$$

$$\forall i \in \{0, \ldots, m\}\, \exists j \in \{0, \ldots, n\} : \mathsf{val}(\mathbb{A})[1, i] = \mathsf{val}(\mathbb{B})[1, j] \text{ in } \mathsf{FG}(\Gamma) \tag{3}$$

$$\forall i \in \{0, \ldots, n\}\, \exists j \in \{0, \ldots, m\} : \mathsf{val}(\mathbb{B})[1, i] = \mathsf{val}(\mathbb{A})[1, j] \text{ in } \mathsf{FG}(\Gamma) \tag{4}$$

Thm. 3 implies that (2) can be checked in polynomial time, whereas (3) and (4) are Π_2^p-properties.

It suffices to prove the lower bound for $\Gamma = \{a, b\}$. We make a logspace reduction from the following Π_2^p-complete problem [1], where $\overline{u} \cdot \overline{v} = u_1v_1 + \cdots + u_nv_n$ denotes the scalar product of two integer vectors $\overline{u} = (u_1, \ldots, u_n)$, $\overline{v} = (v_1, \ldots, v_n)$:

INPUT: vectors $\overline{u} = (u_1, \ldots, u_m) \in \mathbb{N}^m$, $\overline{v} = (v_1, \ldots, v_n) \in \mathbb{N}^n$, and $t \in \mathbb{N}$ (all coded binary)
QUESTION: Does $\forall \overline{x} \in \{0,1\}^m \exists \overline{y} \in \{0,1\}^n : \overline{u} \cdot \overline{x} + \overline{v} \cdot \overline{y} = t$ hold?

Let $s = u_1 + \cdots + u_m + v_1 + \cdots + v_n$, $s_u = u_1 + \cdots + u_m$, and $s_v = v_1 + \cdots + v_n$. W.l.o.g. we can assume $t < s$. Using the construction from [7] (proof of Theorem 5.2) we can construct in logspace an SLP $\mathbb{A}_1$ such that $\mathsf{val}(\mathbb{A}_1) = \prod_{\overline{x} \in \{0,1\}^m} a^{\overline{u} \cdot \overline{x}} A_1 a^{s_u - \overline{u} \cdot \overline{x}}$. Here the product is taken over all tuples from $\{0,1\}^m$ in lexicographic order. By replacing A_1 by $A_2 a^{s_v}$ (which can be easily generated by a small SLP), we obtain an SLP $\mathbb{A}_2$ with $\mathsf{val}(\mathbb{A}_2) = \prod_{\overline{x} \in \{0,1\}^m} a^{\overline{u} \cdot \overline{x}} A_2 a^{s - \overline{u} \cdot \overline{x}}$. Similarly, we obtain an SLP $\mathbb{A}_3$ with $\mathsf{val}(\mathbb{A}_3) = \prod_{\overline{y} \in \{0,1\}^n} a^{\overline{v} \cdot \overline{y}} (bb^{-1} a^{-s_v}) a^{s_v - \overline{v} \cdot \overline{y}}$. Finally, be replacing A_2 in $\mathbb{A}_2$ by the start nonterminal of $\mathbb{A}_3$ we obtain an SLP $\mathbb{A}$ with

$$\mathsf{val}(\mathbb{A}) = \prod_{\overline{x} \in \{0,1\}^m} \left[a^{\overline{u} \cdot \overline{x}} \prod_{\overline{y} \in \{0,1\}^n} \left(a^{\overline{v} \cdot \overline{y}} bb^{-1} a^{-s_v} a^{s_v - \overline{v} \cdot \overline{y}} \right) a^{s - \overline{u} \cdot \overline{x}} \right].$$

Moreover, it is easy to construct a second SLP $\mathbb{B}$ such that

$$\mathsf{val}(\mathbb{B}) = \mathsf{val}(\mathbb{A}) a^{-s \cdot 2^m} \left(a^t bb^{-1} a^{s-t} \right)^{2^m}.$$

We claim that $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ in $\mathsf{FIM}(\{a, b\})$ if and only if

$$\forall \overline{x} \in \mathbb{N}^m \exists \overline{y} \in \mathbb{N}^n : \overline{u} \cdot \overline{x} + \overline{v} \cdot \overline{y} = t. \tag{5}$$

We have $r(\mathsf{val}(\mathbb{A})) = r(\mathsf{val}(\mathbb{B})) = a^{s \cdot 2^m}$. Thus, $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ holds in $\mathsf{FIM}(\{a, b\})$ if and only if $\mathsf{MT}(\mathsf{val}(\mathbb{A})) = \mathsf{MT}(\mathsf{val}(\mathbb{B}))$. Since $\mathsf{val}(\mathbb{A})$ is a prefix of $\mathsf{val}(\mathbb{B})$, we obtain $\mathsf{MT}(\mathsf{val}(\mathbb{A})) \subseteq \mathsf{MT}(\mathsf{val}(\mathbb{B}))$. Moreover, for the prefix $\mathsf{val}(\mathbb{A}) a^{-s \cdot 2^m}$ of $\mathsf{val}(\mathbb{B})$ we have $r(\mathsf{val}(\mathbb{A}) a^{-s \cdot 2^m}) = \varepsilon$ and $\mathsf{MT}(\mathsf{val}(\mathbb{A}) a^{-s \cdot 2^m}) = \mathsf{MT}(\mathsf{val}(\mathbb{A}))$. This and the fact that $\mathsf{MT}(\mathsf{val}(\mathbb{A})) \subseteq \mathsf{MT}(\mathsf{val}(\mathbb{B}))$ implies that $\mathsf{MT}(\mathsf{val}(\mathbb{A})) = \mathsf{MT}(\mathsf{val}(\mathbb{B}))$ if and only if

$$\mathsf{MT}((a^t bb^{-1} a^{s-t})^{2^m}) \subseteq \mathsf{MT}(\mathsf{val}(\mathbb{A})). \tag{6}$$

We show that (6) is equivalent to (5). We have

$$\mathsf{MT}((a^t bb^{-1} a^{s-t})^{2^m}) = \{a^i \mid 0 \leq i \leq s \cdot 2^m\} \cup \{a^{t+k \cdot s} b \mid 0 \leq k < 2^m\}.$$

Since $r(\mathsf{val}(\mathbb{A})) = a^{s \cdot 2^m}$, we have $a^i \in \mathsf{MT}(\mathsf{val}(\mathbb{A}))$ for all $0 \leq i \leq s \cdot 2^m$. Hence, (6) is equivalent to $a^{t+k \cdot s} b \in \mathsf{MT}(\mathsf{val}(\mathbb{A}))$ for every $0 \leq k < 2^m$, i.e. (for a bit vector $\overline{u} = (u_1, \ldots, u_n) \in \{0,1\}^n$ let $n(\overline{u}) = \sum_{i=1}^n u_i 2^{i-1}$ be the number represented by $\overline{u}$)

$$\forall \overline{x} \in \{0,1\}^m : a^{n(\overline{x}) \cdot s + t} b \in \mathsf{MT}(\mathsf{val}(\mathbb{A})). \tag{7}$$

Now, $\mathsf{MT}(\mathsf{val}(\mathbb{A})) \cap a^* b = \{a^{n(\overline{x}) \cdot s + \overline{u} \cdot \overline{x} + \overline{v} \cdot \overline{y}} b \mid \overline{x} \in \{0,1\}^m, \overline{y} \in \{0,1\}^n\}$. Hence, (7) if and only if $\forall \overline{x} \in \{0,1\}^m \exists \overline{y} \in \{0,1\}^n : \overline{u} \cdot \overline{x} + \overline{v} \cdot \overline{y} = t$. □

For a free inverse monoid of rank one, the compressed word problem is simpler:

Proposition 1. *The compressed word problem for* $\mathsf{FIM}(\{a\})$ *can be solved in polynomial time.*

6 Compressed Word Problems for $\mathsf{FIM}(\Gamma)/P$

For an inverse monoid of the form $\mathsf{FIM}(\Gamma)/P$, where Γ is finite and P is a finite idempotent presentation, the word problem can be still solved in logspace [10]. In this case, the complexity of the compressed word problem reaches even PSPACE:

Theorem 5. *The following holds:*

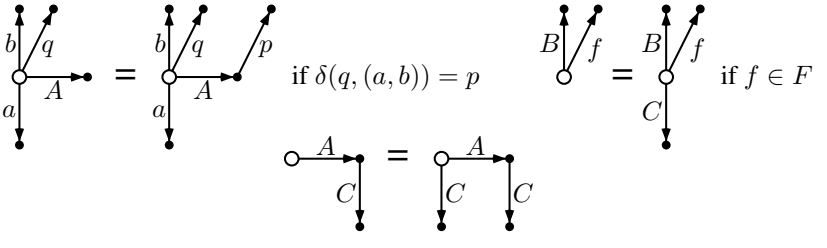
(a) For every finite idempotent presentation $P \subseteq (\Gamma \cup \Gamma^{-1})^* \times (\Gamma \cup \Gamma^{-1})^*$, the compressed word problem for $\mathsf{FIM}(\Gamma)/P$ belongs to PSPACE.
(b) There exists a fixed finite idempotent presentation $P \subseteq (\Gamma \cup \Gamma^{-1})^* \times (\Gamma \cup \Gamma^{-1})^*$ such that the compressed word problem for $\mathsf{FIM}(\Gamma)/P$ is PSPACE*-complete.*

Proof. Let us first show (a). In [10], it was shown that the ordinary word problem for $\mathsf{FIM}(\Gamma)/P$ can be solved in logarithmic space. Since $\mathsf{val}(\mathbb{A})$ can be computed from $\mathbb{A}$ by a PSPACE-transducer (Remark 1), statement (a) follows from Lemma 1.

For the lower bound in (b), we use the following recent result from [8]: There exists a fixed regular language L over some paired alphabet $\Sigma \times \Theta$ such that the following problem is PSPACE-complete (for strings $u \in \Sigma^*, v \in \Theta^*$ with $|u| = |v| = n$ let $u \otimes v = (u[1], v[1]) \cdots (u[n], v[n]) \in (\Sigma \times \Theta)^*$):

INPUT: SLPs $\mathbb{A}$ (over Σ) and $\mathbb{B}$ (over Θ) with $|\mathsf{val}(\mathbb{A})| = |\mathsf{val}(\mathbb{B})|$
QUESTION: Does $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B}) \in L$ hold?

W.l.o.g. assume that $\Sigma \cap \Theta = \emptyset$. Let $\mathcal{A} = (Q, \Sigma \times \Theta, \delta, q_0, F)$ be a deterministic finite automaton with $L(\mathcal{A}) = L$. Let $\Gamma = \Sigma \cup \Theta \cup Q \cup \{A, B, C\}$ (all unions are assumed to be disjoint). Consider the fixed idempotent presentation over the alphabet Γ with the following relations:

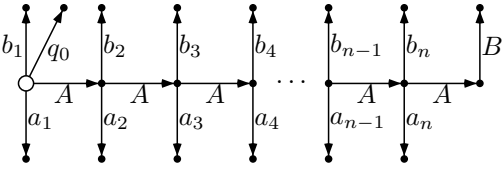


With the upper left relation, we simulate the automaton $\mathcal{A}$. The upper right relation allows to add a C-labeled edge as soon as a final state is reached; the B-labeled edge acts as a kind of end marker for the input word. Finally, the last relation allows to propagate the C-labeled edge back to the origin (node 1).

Assume that $\mathsf{val}(\mathbb{A}) = a_1 \cdots a_n$ and $\mathsf{val}(\mathbb{B}) = b_1 \cdots b_n$. Consider the string

$$w = q_0 q_0^{-1} \prod_{i=1}^{n} (a_i a_i^{-1} A) B B^{-1} \prod_{i=0}^{n-1} (A^{-1} b_{n-i} b_{n-i}^{-1}).$$

It is easy to compute from $\mathbb{A}$ and $\mathbb{B}$ in polynomial time an SLP $\mathbb{C}$ with $\mathsf{val}(\mathbb{C}) = w$. The Munn tree $\mathsf{MT}(w)$ looks as follows:



We claim that $w = CC^{-1}w$ in $\mathsf{FIM}(\Gamma)/P$ if and only if $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$. Clearly, $w = CC^{-1}w = 1$ in $\mathsf{FG}(\Gamma)$. Moreover, $\mathsf{cl}_P(\mathsf{MT}(w)) = \mathsf{cl}_P(\mathsf{MT}(CC^{-1}w))$ if and only if $C \in \mathsf{cl}_P(\mathsf{MT}(w))$. Thus, it suffices to show that $C \in \mathsf{cl}_P(\mathsf{MT}(w))$ if and only if $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$. First, assume that $\mathsf{val}(\mathbb{A}) \otimes \mathsf{val}(\mathbb{B}) \notin L(\mathcal{A})$. Let q_i be the state of $\mathcal{A}$ after reading $(a_1, b_1) \cdots (a_i, b_i)$ $(0 \leq i \leq n)$. Thus, $q_n \notin F$. This implies that $\mathsf{cl}_P(\mathsf{MT}(w)) = \mathsf{MT}(w) \cup \{A^i q_i \mid 0 \leq i \leq n\}$. Hence, $C \notin \mathsf{cl}_P(\mathsf{MT}(w))$. On the other hand, if $q_n \in F$, then $\mathsf{cl}_P(\mathsf{MT}(w)) = \mathsf{MT}(w) \cup \{A^i q_i, A^i C \mid 0 \leq i \leq n\}$ and therefore $C \in \mathsf{cl}_P(\mathsf{MT}(w))$. □

7 Rational Subset Membership Problems

In this section we briefly outline our results on the compressed variant of the rational subset membership problem for free inverse monoids. We start with a lower bound.

Theorem 6. *There exists a fixed alphabet Γ and a fixed finite subset $K \subseteq (\Gamma \cup \Gamma^{-1})^*$ such that the following problem is* PSPACE*-hard:*
INPUT: An SLP $\mathbb{A}$ over the alphabet $\Gamma \cup \Gamma^{-1}$
QUESTION: Does $\rho(\mathsf{val}(\mathbb{A})) \in \rho(K^*)$ hold?

Note that $\rho(K^*)$ is the submonoid of $\mathsf{FIM}(\Gamma)$ generated by $\rho(K)$. Let us now turn to an upper bound.

Theorem 7. *The following problem belongs to* PSPACE*:*
INPUT: An SLP $\mathbb{A}$ over an alphabet $\Gamma \cup \Gamma^{-1}$ and an NFA $\mathcal{A}$ over the alphabet $\Gamma \cup \Gamma^{-1}$.
QUESTION: Does $\rho(\mathbb{A}) \in \rho(L(\mathcal{A}))$ hold?

The proof of Thm. 7 is based on tree automata techniques. Recall that a Munn tree $\mathsf{MT}(u)$ can be viewed as an edge labeled tree. The node ε can be made the root of the tree. Such a rooted edge-labeled tree can be evaluated by a tree automaton. Usually, tree automata work on node labeled trees, but this is only a technicality. The proof of Thm. 7 is based on the following two lemmas.

Lemma 2. *There is a* PSPACE*-transducer, which computes $\mathsf{MT}(\mathsf{val}(\mathbb{A}))$ for a given input SLP $\mathbb{A}$.*

Lemma 3. *There is a* PSPACE*-transducer, which computes from a given nondeterministic finite automaton $\mathcal{A}$ over the alphabet $\Gamma \cup \Gamma^{-1}$ and a given SLP $\mathbb{A}$ over the alphabet $\Gamma \cup \Gamma^{-1}$ a nondeterministic tree automaton $\mathcal{B} = \mathcal{B}(\mathcal{A}, \mathbb{A})$ such that: $\rho(\mathsf{val}(\mathbb{A})) \in \rho(L(\mathcal{A}))$ if and only if $\mathsf{MT}(\mathsf{val}(\mathbb{A}))$ is accepted by $\mathcal{B}$.*

Proof of Thm. 7. We apply Lemma 1, where $f : (\mathbb{A}, \mathcal{A}) \mapsto (\mathsf{MT}(\mathsf{val}(\mathbb{A})), \mathcal{B}(\mathcal{A}, \mathbb{A}))$ and L is the uniform membership problem for tree automata, i.e., the set of all pairs $(T, \mathcal{B})$, where T is a tree and $\mathcal{B}$ is a tree automaton that accepts T. By [6], L belongs to LOGCFL and hence to POLYLOGSPACE. Moreover, the mapping f can be computed by a PSPACE-transducer by Lemma 2 and 3. □

References

1. Berman, P., Karpinski, M., Larmore, L.L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two-dimensional texts. J. Comput. Syst. Sci. 65(2), 332–350 (2002)
2. Diekert, V., Lohrey, M., Miller, A.: Partially commutative inverse monoids. Semigroup Forum 77(2), 196–226 (2008)
3. Lawson, M.V.: Inverse Semigroups: The Theory of Partial Symmetries. World Scientific, Singapore (1999)
4. Lifshits, Y.: Processing compressed texts: A tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
5. Lipton, R.J., Zalcstein, Y.: Word problems solvable in logspace. J. Assoc. Comput. Mach. 24(3), 522–526 (1977)
6. Lohrey, M.: On the parallel complexity of tree automata. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 201–215. Springer, Heidelberg (2001)
7. Lohrey, M.: Word problems and membership problems on compressed words. SIAM J. Comput. 35(5), 1210–1240 (2006)
8. Lohrey, M.: Leaf languages and string compression. Inf. Comput. 209(6), 951–965 (2011)
9. Lohrey, M.: Compressed word problems for inverse monoids., http://arxiv.org/abs/1106.1000
10. Lohrey, M., Ondrusch, N.: Inverse monoids: decidability and complexity of algebraic questions. Inf. Comput. 205(8), 1212–1234 (2007)
11. Lohrey, M., Schleimer, S.: Efficient computation in groups via compression. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 249–258. Springer, Heidelberg (2007)
12. Lohrey, M., Steinberg, B.: Tilings and submonoids of metabelian groups. Theory Comput. Syst. 48(2), 411–427 (2011)
13. Macdonald, J.: Compressed words and automorphisms in fully residually free groups. Internat. J. Algebra Comput. 20(3), 343–355 (2010)
14. Margolis, S., Meakin, J.: Inverse monoids, trees, and context-free languages. Trans. Amer. Math. Soc. 335(1), 259–276 (1993)
15. Margolis, S., Meakin, J., Sapir, M.: Algorithmic problems in groups, semigroups and inverse semigroups. In: Fountain, J. (ed.) Semigroups, Formal Languages and Groups, pp. 147–214. Kluwer Academic Publishers, Dordrecht (1995)
16. Munn, W.: Free inverse semigroups. Proc.London Math. Soc. 30, 385–404 (1974)
17. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)
18. Plandowski, W., Rytter, W.: Complexity of language recognition problems for compressed words. In: Karhumäki, J., Maurer, H.A., Paun, G., Rozenberg, G. (eds.) Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 262–272. Springer, Heidelberg (1999)
19. Rozenblat, B.V.: Diophantine theories of free inverse semigroups. Sib. Math. J. 26, 860–865 (1985); English translation
20. Schleimer, S.: Polynomial-time word problems. Comment. Math. Helv. 83, 741–765 (2008)
21. Silva, P.V.: Rational languages and inverse monoid presentations. Internat. J. Algebra Comput. 2, 187–207 (1992)
22. Stephen, J.: Presentations of inverse monoids. J. Pure Appl. Algebra 63, 81–112 (1990)

Pushing for Weighted Tree Automata

Andreas Maletti* and Daniel Quernheim*

Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung
Azenbergstraße 12, 70174 Stuttgart, Germany
{Andreas.Maletti,Daniel.Quernheim}@ims.uni-stuttgart.de

Abstract. Explicit pushing for weighted tree automata over semifields is introduced. A careful selection of the pushing weights allows a normalization of bottom-up deterministic weighted tree automata. Automata in the obtained normal form can be minimized by a simple transformation into an unweighted automaton followed by unweighted minimization. This generalizes results of Mohri and Eisner for deterministic weighted string automata to the tree case. Moreover, the new strategy can also be used to test equivalence of two bottom-up deterministic weighted tree automata M_1 and M_2 in time $O(|M| \log |Q|)$, where $|M| = |M_1| + |M_2|$ and $|Q|$ is the sum of the number of states of M_1 and M_2. This improves the previously best running time $O(|M_1| \cdot |M_2|)$.

1 Introduction

Automata theory is a main branch of theoretical computer science with successful applications in many diverse fields such as natural language processing, system verification, and pattern recognition. Recently, renewed interest in tree automata was sparked by applications in natural language processing and XML processing. These applications require efficient algorithms for basic manipulations of tree automata such as determinization [7], inference [20], and minimization [17,16].

In natural language processing, weighted devices are often used to model probabilities, cost functions, or other features. In this contribution, we consider pushing [21,10] for weighted tree automata [1,11] over commutative semifields [15,14]. Roughly speaking, pushing moves transition weights along a path. If the weights are properly selected, then pushing can be used to canonicalize a (bottom-up) deterministic weighted tree automaton [3]. The obtained canonical representation has the benefit that it can be minimized using unweighted minimization, in which the weight is treated as a transition label. This strategy has successfully been employed in [21,10] for deterministic weighted (finite-state) string automata, and we adapt it here for tree automata. In particular, we improve the currently best minimization algorithm [19] for deterministic weighted tree automata from $O(|M| \cdot |Q|)$ to $O(|M| \log |Q|)$, which coincides with the complexity of minimization in the unweighted case [17].

* Both authors were supported by the German Research Foundation (DFG) grant MA/4959/1-1.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 460–471, 2011.

The improvement is achieved by a careful selection of signs of life [19]. Intuitively, a sign of life for a state q is a context which takes q into a final state. In particular, equivalent states will receive the same sign of life, which ensures that their pushing weights are determined using the same evaluation context. This property sets our algorithm apart from the similar algorithm in [19, Algorithm 1] and allows a proper canonicalization. After the pushing weights are determined we perform pushing, which we define for general (potentially nondeterministic) weighted tree automata. We prove that the semantics is preserved and that equivalent states have equally weighted corresponding transitions after pushing, which allows us to reduce minimization to the unweighted case [17].

Secondly, we apply pushing to equivalence testing. The currently fastest algorithm [9] for checking equivalence of two deterministic weighted tree automata M_1 and M_2 runs in time $O(|M_1| \cdot |M_2|)$. Our algorithm that computes signs of life can also handle states in different automata with the help of a particular sum construction. The pushing weight (and the evaluation context) is determined carefully, so that equivalent states in different automata receive the same sign of life. This allows us to minimize both input automata and then only test the corresponding unweighted automata for isomorphism. This approach reduces the run-time complexity to $O(|M| \log |Q|)$, where $|M| = |M_1| + |M_2|$ and $|Q| = |Q_1| + |Q_2|$ is the number of total states.

2 Preliminaries

The set of nonnegative integers is $\mathbb{N}$. Given $l, u \in \mathbb{N}$ we denote $\{i \in \mathbb{N} \mid l \leq i \leq u\}$ simply by $[l, u]$. Let $k \in \mathbb{N}$ and Q a set. We write Q^k for the k-fold Cartesian product of Q, and the empty tuple $() \in Q^0$ is displayed as ε. An alphabet is a finite, nonempty set of symbols. A ranked alphabet (Σ, rk) consists of an alphabet Σ and a mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$. Whenever 'rk' is clear from the context, we simply drop it. The subset Σ_k of k-ary symbols of Σ is $\Sigma_k = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\}$. We let $\Sigma(Q) = \{\sigma(w) \mid \sigma \in \Sigma_k, w \in Q^k\}$. To improve the readability, we often write $\sigma(q_1, \ldots, q_k)$ instead of $\sigma(q_1 \cdots q_k)$, where $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$. The set $T_\Sigma(Q)$ of Σ-trees indexed by Q is inductively defined to be the smallest set such that $Q \subseteq T_\Sigma(Q)$ and $\Sigma(T_\Sigma(Q)) \subseteq T_\Sigma(Q)$. We write T_Σ for $T_\Sigma(\emptyset)$. The size $|t|$ of a tree t is inductively defined by $|q| = 1$ for every $q \in Q$ and $|\sigma(t_1, \ldots, t_k)| = 1 + \sum_{i=1}^{k} |t_i|$ for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(Q)$.

We reserve the use of a special symbol $\square \notin Q$ that is not an element in any considered alphabet. The set $C_\Sigma(Q)$ of Σ-contexts indexed by Q is defined as the smallest set such that $\square \in C_\Sigma(Q)$ and $\sigma(t_1, \ldots, t_{i-1}, C, t_{i+1}, \ldots, t_k) \in C_\Sigma(Q)$ for every $\sigma \in \Sigma_k$ with $k \geq 1$, index $i \in [1, k]$, $t_1, \ldots, t_k \in T_\Sigma(Q)$, and $C \in C_\Sigma(Q)$. We write C_Σ for $C_\Sigma(\emptyset)$. Note that $C_\Sigma(Q) \subseteq T_\Sigma(Q \cup \{\square\})$. Let $C \in C_\Sigma(Q)$ and $t \in T_\Sigma(Q)$. Then $C[t]$ is the tree obtained from C by replacing $\square$ by t.

A (commutative) semifield [15,14] is a tuple $(A, +, \cdot, 0, 1)$ such that $(A, +, 0)$ and $(A, \cdot, 1)$ are commutative monoids, of which $(A \setminus \{0\}, \cdot, 1)$ is a group; $a \cdot 0 = 0$ for every $a \in A$; and $\cdot$ distributes over $+$. The multiplicative inverse of $a \in A \setminus \{0\}$ is denoted by a^{-1}; i.e., $a \cdot a^{-1} = 1$ for every $a \in A \setminus \{0\}$.

Let $(A, +, \cdot, 0, 1)$ be an arbitrary commutative semifield.

A weighted (finite-state) tree automaton [6,5,18,4,3] (for short: wta) is a tuple $M = (Q, \Sigma, \mu, F)$ such that (i) Q is an alphabet of states; (ii) Σ is a ranked alphabet; (iii) $\mu\colon \Sigma(Q) \times Q \to A$ is a transition weight mapping; and (iv) $F \subseteq Q$ is a set of final states. Note that the restriction to final states instead of final weights does not restrict the expressive power [3, Lemma 6.1.4]. We often write $t \to q$ for (t, q). The size $|M|$ of M is $|M| = \sum_{(t \to q) \in \mu^{-1}(A \setminus \{0\})} (|t| + 1)$. We extend μ to $h_\mu\colon T_\Sigma(Q) \times Q \to A$ by $h_\mu(q \to q) = 1$, $h_\mu(p \to q) = 0$, and

$$h_\mu(\sigma(t_1, \dots, t_k) \to q) = \sum_{q_1, \dots, q_k \in Q} \mu(\sigma(q_1, \dots, q_k) \to q) \cdot \prod_{i=1}^{k} h_\mu(t_i \to q_i)$$

for all $p, q \in Q$ with $p \neq q$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_\Sigma(Q)$. The wta M recognizes the weighted language $M\colon T_\Sigma \to A$ such that $M(t) = \sum_{q \in F} h_\mu(t \to q)$ for every $t \in T_\Sigma$. Two wta M and M' are equivalent if $M = M'$.

The wta $M = (Q, \Sigma, \mu, F)$ is *(bottom-up total) deterministic* (or a dwta) if for every $t \in \Sigma(Q)$ there exists exactly one $q \in Q$ such that $\mu(t \to q) \neq 0$. For dwta we prefer the presentation $(Q, \Sigma, \delta, c, F)$ with $\delta\colon \Sigma(Q) \to Q$ and $c\colon \Sigma(Q) \to A \setminus \{0\}$, which are defined such that $\delta(t) = q$ and $c(t) = \mu(t \to q)$ for every $t \in \Sigma(Q)$, where q is the unique state such that $\mu(t \to q) \neq 0$. The mappings δ and c are extended to $\underline{\delta}\colon T_\Sigma(Q) \to Q$ and $\underline{c}\colon T_\Sigma(Q) \to A \setminus \{0\}$ by

$$\underline{\delta}(q) = q \qquad \underline{\delta}(\sigma(t_1, \dots, t_k)) = \delta(\sigma(\underline{\delta}(t_1), \dots, \underline{\delta}(t_k)))$$

$$\underline{c}(q) = 1 \qquad \underline{c}(\sigma(t_1, \dots, t_k)) = c(\sigma(\underline{\delta}(t_1), \dots, \underline{\delta}(t_k))) \cdot \prod_{i=1}^{k} \underline{c}(t_i)$$

for every $q \in Q$, $\sigma \in \Sigma_k$, and $t_1, \dots, t_k \in T_\Sigma(Q)$. For every $t \in T_\Sigma$ we can observe that $M(t) = \underline{c}(t)$ if $\underline{\delta}(t) \in F$ and $M(t) = 0$ otherwise.

An equivalence relation $\equiv$ on Q is a reflexive, symmetric, and transitive subset $\equiv \subseteq Q^2$. The equivalence class (or block) of $q \in Q$ is $[q]_\equiv = \{p \in Q \mid p \equiv q\}$, and $(P/\equiv) = \{[p]_\equiv \mid p \in P\}$ for every $P \subseteq Q$. Whenever $\equiv$ is obvious from the context, we simply omit it. The equivalence $\equiv$ respects finality if $[q] \subseteq F$ or $[q] \subseteq Q \setminus F$ for all $q \in Q$ (i.e., all states of a block are either final or nonfinal).

Suppose that M is deterministic. Let $\equiv_M \subseteq Q^2$ be the Myhill-Nerode equivalence relation [2]

$$\equiv_M = \{(q, p) \in Q^2 \mid \exists a \in A \setminus \{0\}, \forall C \in C_\Sigma(Q)\colon \underline{c}(C[q]) = a \cdot \underline{c}(C[p])\} \ .$$

If M is clear from the context, then we just write $\equiv$ instead of $\equiv_M$. The deterministic (unweighted) tree automaton (dta) [12,13,8] for M is $M' = (Q, \Sigma, \delta, F)$. It recognizes the tree language $L(M') \subseteq T_\Sigma$, which is $\{t \in T_\Sigma \mid \underline{\delta}(t) \in F\}$.

Let $M_1 = (Q_1, \Sigma, \delta_1, c_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, c_2, F_2)$ be two arbitrary dwta. A *congruential relation* $\sim \subseteq Q_1 \times Q_2$ (between M_1 and M_2) is such that $\delta_1(\sigma(q_1, \dots, q_k)) \sim \delta_2(\sigma(p_1, \dots, p_k))$ for every $\sigma \in \Sigma_k$, $q_1, \dots, q_k \in Q_1$, and $p_1, \dots, p_k \in Q_2$ such that $q_i \sim p_i$ for every $i \in [1, k]$. Note that this definition

of a congruential relation completely disregards the weights. It coincides with the classical notion of *congruence* if $M_1 = M_2$. The equivalence $\equiv_M$ is a congruence [2]. If $Q_1 \cap Q_2 = \emptyset$, then a congruential relation $\sim$ extends to a congruential equivalence $\cong \subseteq (Q_1 \cup Q_2)^2$ such that $\cong = (\sim \cup \sim^{-1})^*$.

Finally, let us introduce a particular sum of 2 dwta, which preserves determinism. Without loss of generality, let $Q_1 \cap Q_2 = \emptyset$. The dwta $M_1 \uplus M_2$ is $(Q_1 \cup Q_2, \Sigma, \delta, c, F_1 \cup F_2)$, where $\delta(t_1) = \delta_1(t_1)$, $\delta(t_2) = \delta_2(t_2)$, $c(t_1) = c_1(t_1)$, and $c(t_2) = c_2(t_2)$ for every $t_1 \in \Sigma'(Q_1)$ and $t_2 \in \Sigma'(Q_2)$ with $\Sigma' = \Sigma \setminus \Sigma_0$. As usual, we assume that unspecified transitions go to a nonfinal sink state. Clearly, $M_1 \uplus M_2$ does not compute the sum of M_1 and M_2 because it is missing all transitions for nullary symbols. Outside of nullary symbols (the initial steps) it behaves just like the standard sum [3] of M_1 and M_2.

3 Efficient Computation of Signs of Life

In this section, we show how to efficiently compute signs of life (see Def. 1), which are evidence that a final state can be reached, and weights for pushing (see Sect. 4). Our algorithm (see Alg. 1) is similar to [19, Alg. 1], but we guarantee that equivalent states receive the same sign of life. This last property will prove to be essential in Sects. 5 and 6. Since we also want to use the computed signs of life for equivalence testing (see Sect. 6), we potentially work with a sum $M_1 \uplus M_2$ of two dwta M_1 and M_2 here, where the transitions for nullary symbols have been removed in order to preserve determinism.

Let $M = (Q, \Sigma, \delta, c, F)$ be a dwta with $Q = Q_1 \cup Q_2$ and $Q_1 \cap Q_2 = \emptyset$, and let $g\colon Q_1 \to Q_2$.

We extend g to a mapping $\underline{g}\colon T_\Sigma(Q_1) \to T_\Sigma(Q_2)$ such that $\underline{g}(q_1) = g(q_1)$ for every $q_1 \in Q_1$ and $\underline{g}(\sigma(t_1, \ldots, t_k)) = \sigma(\underline{g}(t_1), \ldots, \underline{g}(t_k))$ for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(Q_1)$. Let us first recall the definition of a sign of life from [19], which we adjust here to handle the potential sum.

Definition 1 (see [19, Sect. 2]). *Let $C \in C_\Sigma(Q_1)$, $q_1 \in Q_1$, and $q_2 \in Q_2$. Then C is a* sign of life *for q_1 if $\underline{\delta}(C[q_1]) \in F$. It is a* sign of life *for q_2 if $\underline{\delta}(\underline{g}(C)[q_2]) \in F$. Any state that has a sign of life is* live*, and any state without a sign of life is* dead*.*

Next, we explain Alg. 1 briefly. Let M_1 be the dwta obtained by restricting M to Q_1. First we realize that every final state $q \in F$ is trivially live. We set the sign of life for its block $[q]$ to the trivial context $\Box$ (line 3) and its weight to 1 (line 4). Since the congruence $\cong$ respects finality, the block $[q]$ contains only final states. Overall, this initialization takes time $O(|Q|)$. Next, we add all transitions leading to a final state of $F \cap Q_1$ to the FIFO queue T, which takes time $O(|M_1|)$. Clearly each transition using Q_1 can be added at most once to T, so the 'while-loop' executes at most once per transition. In the loop, we inspect the transition $\tau = \sigma(q_1, \ldots, q_k)$ that we took from T. We check each source state $q_i \in Q_1$ (with $i \in [1, k]$) whether it has been explored before. If not,

Alg. 1. ComputeSoL: Compute signs of life and their weight

Require: dwta $M = (Q, \Sigma, \delta, c, F)$, $g\colon Q_1 \to Q_2$ with $Q = Q_1 \cup Q_2$ and $Q_1 \cap Q_2 = \emptyset$, and congruential equivalence $\cong$ such that $g \subseteq \cong$ and $\cong$ respects finality

Ensure: live states $L \subseteq Q$, $\mathrm{sol}\colon (L/\cong) \to C_\Sigma(Q_1)$, and $\lambda\colon L \to A \setminus \{0\}$ such that $\lambda(q_1) = \underline{c}(\mathrm{sol}(B)[q_1])$ for all $q_1 \in L \cap Q_1$ and $\lambda(q_2) = \underline{c}(\underline{g}(\mathrm{sol}(B))[q_2])$ for all $q_2 \in L \cap Q_2$, where $B = [q_1]_\cong$

$L \leftarrow F$ // all final states are live
2: **for all** $q \in F$ **do**
$\mathrm{sol}([q]_\cong) \leftarrow \square$ // sign of life is the trivial context for final states...
4: $\lambda(q) \leftarrow 1$ // ...with trivial weight
$T \leftarrow$ **new** FifoQueue
6: Append$(T, \{\tau \in \Sigma(Q_1) \mid \delta(\tau) \in F \cap Q_1\})$ // add all transitions leading to $F \cap Q_1$
while T is not empty **do**
8: $\tau \leftarrow$ RemoveHead(T) // get first transition
let $\tau = \sigma(q_1, \ldots, q_k)$ with $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q_1$
10: **for all** $i \in [1, k]$ such that $q_i \notin L$ **do**
let $C = \sigma(q_1, \ldots, q_{i-1}, \square, q_{i+1}, \ldots, q_k)$ // prepare context
12: $L \leftarrow L \cup [q_i]_\cong$ // all equivalent states are live; add to L
$\mathrm{sol}([q_i]_\cong) \leftarrow \mathrm{sol}([\delta(\tau)]_\cong)[C]$ // add transition to sign of life of target state
14: **for all** $q \in [q_i]_\cong$ **do**
if $q \in Q_1$ **then**
16: $\lambda(q) \leftarrow \lambda(\delta(C[q])) \cdot c(C[q])$ // set new weight
Append$(T, \{\tau' \in \Sigma(Q_1) \mid \delta(\tau') = q\})$ // add transitions leading to q
18: **else**
$\lambda(q) \leftarrow \lambda(\delta(\underline{g}(C)[q])) \cdot c(\underline{g}(C)[q])$ // set new weight
20: **return** $(L, \mathrm{sol}, \lambda)$

then its whole block $[q_i]$ is unexplored, since we explore the states by blocks. Overall, we thus perform at most $|M_1|$ checks. Once we discover an unexplored state q_i (i.e., a state not yet marked as live), we mark its whole block $[q_i]$ as explored (and live). In addition, we set the block's sign of life to the sign of life of the target state's block $[\delta(\tau)]$ extended by the context C created from the current transition τ (line 13). Finally, for each state q in the current block $[q_i]$ we compute the weight of the sign of life by plugging q into the current transition instead of q_i. If $q \in Q_2$, then we adjust the transition using $\underline{g}$. In this way, we obtain a transition weight a and a target state p. Since the weight of the sign of life for p has already been computed (because otherwise the transition τ would not have been in the queue T), we simply set $\lambda(q) = \lambda(p) \cdot a$ (line 16 and 19). Clearly, this is done at most once for each state, so we obtain a total complexity (counted over all loops) of $O(|Q|)$ for this part. We complete the iteration by adding all transitions to newly explored states of Q_1 to T. Overall, we obtain the complexity $O(|M_1| + |Q|)$.

Theorem 2. *Algorithm 1 is correct and runs in time* $O(|M_1| + |Q|)$.

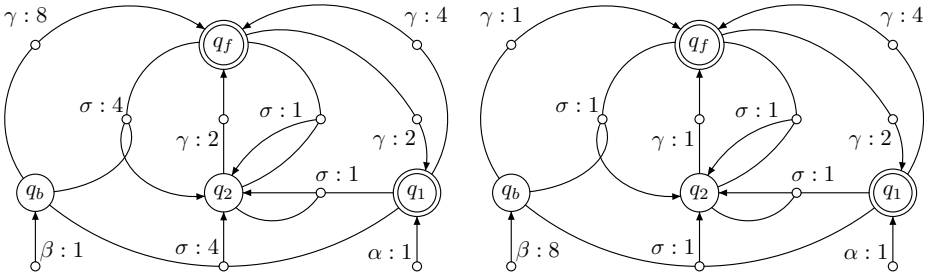


Fig. 1. Dwta over the Real numbers before (left) and after (right) pushing

Proof. We already argued the run-time complexity, so let us prove the post-condition. For $q \in F$, we have $\lambda(q) = 1 = \underline{c}(q) = \underline{c}(\Box[q])$ by lines 3–4, which proves the post-condition because $\mathrm{sol}([q]_{\cong}) = \Box$. Let $C' = C$ if $q \in Q_1$, and otherwise let $C' = \underline{g}(C)$. In the main loop, we set $\lambda(q) = \lambda(\delta(C'[q])) \cdot c(C'[q])$ in line 16 or 19. Since $q' = \delta(C'[q])$ has already been explored in a previous iteration, we have $\lambda(q') = \underline{c}(C''[q'])$ by the induction hypothesis, where $C'' = \mathrm{sol}([q']_{\cong})$ if $q' \in Q_1$ and $C'' = \underline{g}(\mathrm{sol}([q']_{\cong}))$. Consequently,

$$\lambda(q) = \underline{c}(C''[q']) \cdot c(C'[q]) = \underline{c}(C''[C'[q]]) = \underline{c}((C''[C'])[q]) \ ,$$

which proves the post-condition because $\mathrm{sol}([q]_{\cong}) = \mathrm{sol}([\delta(C[q])]_{\cong})[C]$ by line 13. Since $\lambda(q) \neq 0$, it also proves that $\mathrm{sol}([q]_{\cong})$ is a sign of life for q and that q is live. The proof that all states $q \notin L$ are indeed dead is simple and omitted here. □

Example 3. An example dwta is depicted in Fig. 1 (left). For any transition (small circles), the arrow leads to the target state and the source states have been arranged in a counter-clockwise fashion (starting from the target arrow). As usual, final states are indicated by double-circles. Let us note that the coarsest congruence $\cong$ that respects finality is $\{\{q_1, q_f\}, \{q_2, q_b\}\}$. We use this partition together with $g = \emptyset$ in Alg. 1. First, we mark all final states $\{q_1, q_f\}$ as live. Their block is assigned the trivial context $\Box$ and each final state is assigned the trivial weight 1. We then initialize the FIFO queue with the transitions $\{\gamma(q_b), \gamma(q_2), \gamma(q_1), \gamma(q_f)\}$ leading to q_1 or q_f. Let us pick the first transition $\gamma(q_b)$ from the queue. Since q_b has not yet been marked as live, we consider all transitions $\gamma(\Box)[q] = \gamma(q)$ where $q \in [q_b]_{\cong} = \{q_b, q_2\}$. The sign of life for $[q_b]_{\cong}$ is $\gamma(\Box)$, and the corresponding weights $\lambda(q_b)$ and $\lambda(q_2)$ are $\lambda(q_b) = \lambda(q_f) \cdot 2 = 2$ and $\lambda(q_2) = \lambda(q_f) \cdot 8 = 8$, respectively. For all remaining transitions, all source states are already live. Consequently, we have computed all signs of life and the pushing weights $\lambda(q_1) = \lambda(q_f) = 1$, $\lambda(q_2) = 2$, and $\lambda(q_b) = 8$.

4 Pushing

Recall that the Myhill-Nerode congruence states that there is a unique scaling factor for every pair (p, q) of equivalent states. Thus, any fixed sign of life

can be used to determine the scaling factor between p and q. In the previous section, we computed a sign of life $\mathrm{sol}(q)$ for each live state $q \in L$ as well as the weight $\lambda(q)$ of $\mathrm{sol}(q)$. Now, we will use these weights to normalize the wta by *pushing* [21,10,22]. Intuitively, pushing cancels the scaling factor for equivalent states. In weighted (finite-state) string automata, pushing is performed from the final states towards the initial states. Since we work with bottom-up wta [3] (i.e., our notion of determinism is bottom-up), this works analogously here by moving weights from the root towards the leaves.

In this section we work with an arbitrary wta $M = (Q, \Sigma, \mu, F)$ and an arbitrary mapping $\lambda\colon Q \to A \setminus \{0\}$ such that $\lambda(q) = 1$ for every $q \in F$.[1]

Definition 4 (see [21, p. 296]). *The* pushed *wta* $\mathrm{push}_\lambda(M)$ *is* (Q, Σ, μ', F) *such that* $\mu'(\sigma(q_1, \dots, q_k) \to q) = \lambda(q) \cdot \mu(\sigma(q_1, \dots, q_k) \to q) \cdot \prod_{i=1}^{k} \lambda(q_i)^{-1}$ *for every* $\sigma \in \Sigma_k$ *and* $q, q_1, \dots, q_k \in Q$.

The mapping λ records the pushed weights. In plain words, every transition leading to a state $q \in Q$ charges the additional weight $\lambda(q)$, and every transition leaving the state q compensates this by charging the weight $\lambda(q)^{-1}$. Next, let us show that M and $\mathrm{push}_\lambda(M)$ are equivalent.

Proposition 5 (see [21, Lm. 4]). *The wta M and $\mathrm{push}_\lambda(M)$ are equivalent. Moreover, if M is deterministic, then so is $\mathrm{push}_\lambda(M)$.*

Proof. Let $\mathrm{push}_\lambda(M) = (Q, \Sigma, \mu', F)$. The preservation of determinism is obvious. We prove that $h_{\mu'}(t \to q) = \lambda(q) \cdot h_\mu(t \to q)$ for every $t \in T_\Sigma$ and $q \in Q$ by induction. Let $t = \sigma(t_1, \dots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma$. We have $h_{\mu'}(t_i \to q_i) = \lambda(q_i) \cdot h_\mu(t_i \to q_i)$ for every $i \in [1, k]$ and $q_i \in Q$ by the induction hypothesis. Consequently,

$$
\begin{aligned}
& h_{\mu'}(t \to q) \\
&= \sum_{q_1, \dots, q_k \in Q} \mu'(\sigma(q_1, \dots, q_k) \to q) \cdot \prod_{i=1}^{k} h_{\mu'}(t_i \to q_i) \\
&= \sum_{q_1, \dots, q_k \in Q} \lambda(q) \cdot \mu(\sigma(q_1, \dots, q_k) \to q) \cdot \prod_{i=1}^{k} \lambda(q_i)^{-1} \cdot \prod_{i=1}^{k} \Big(\lambda(q_i) \cdot h_\mu(t_i \to q_i)\Big) \\
&= \lambda(q) \cdot h_\mu(t \to q) \ .
\end{aligned}
$$

We complete the proof as follows.

$$
\mathrm{push}_\lambda(M)(t) = \sum_{q \in F} h_{\mu'}(t \to q) = \sum_{q \in F} \lambda(q) \cdot h_\mu(t \to q) = \sum_{q \in F} h_\mu(t \to q) = M(t)
$$

because $\lambda(q) = 1$ for every $q \in F$. □

[1] This additional requirement is necessary because our wta have final states [3, Section 4.1.3]. A model with final weights [3, Section 4.1.3] would be equally powerful, but could lift this restriction.

Alg. 2. Overall structure of our minimization algorithm

Require: a dwta M
Ensure: return a minimal, equivalent dwta

$\cong \leftarrow \textsc{ComputeCoarsestCongruence}(M)$ // complexity: $O(|M| \log |Q|)$
$(L, \mathrm{sol}, \lambda) \leftarrow \textsc{ComputeSoL}(M, \cong, \emptyset)$ // complexity: $O(|M|)$
$M' \leftarrow \mathrm{push}_\lambda(M)$ // complexity: $O(|M|)$
$N \leftarrow \textsc{Minimize}(\mathrm{alph}(M'), \cong)$ // complexity: $O(|M| \log |Q|)$
return $\mathrm{alph}^{-1}(N)$

Example 6 (cont'd). Let us return to our example dwta M in Fig. 1 (left) and perform pushing. The pushing weights λ are given in Ex. 3. For the transition $\tau = \sigma(q_b, q_f)$ we have $\delta(\tau) = q_2$ and $c(\tau) = 4$. In $\mathrm{push}_\lambda(M)$ we have the new weight $c'(\tau) = \lambda(q_2) \cdot c(\tau) \cdot \lambda(q_b)^{-1} \cdot \lambda(q_f)^{-1} = 2 \cdot 4 \cdot 8^{-1} \cdot 1^{-1} = 1$. The dwta $\mathrm{push}_\lambda(M)$ is presented in Fig. 1 (right). It is evident in $\mathrm{push}_\lambda(M)$ that q_2 and q_b are equivalent, whereas q_1 and q_f are not.

5 Minimization

We will now turn to the main application of weight pushing for dwta: efficient minimization. The overall structure is presented in Alg. 2. Note that the coarsest congruence for a dwta $M = (Q, \Sigma, \delta, c, F)$ that respects finality can be obtained by minimization [17] of the underlying unweighted automaton (Q, Σ, δ, F).

> *Let $M = (Q, \Sigma, \delta, c, F)$ be a dwta, $\lambda \colon Q \to A$ be the pushing weights computed in Alg. 1, and $\mathrm{push}_\lambda(M) = (Q, \Sigma, \delta, c', F)$.*

The dwta $\mathrm{push}_\lambda(M)$ has the property that $c'(\sigma(q_1, \ldots, q_k)) = c'(\sigma(p_1, \ldots, p_k))$ for all $\sigma \in \Sigma_k$ and states $q_1, \ldots, q_k, p_1, \ldots, p_k \in Q$ such that $q_i \equiv p_i$ for every $i \in [1, k]$. In analogy to the string case [10], this property allows us to treat the transition weight as part of the input symbol. In this way we obtain a dta, which we can minimize using, for example, the algorithm of [17]. After the minimization, we can expand the input symbol again into a symbol from Σ and the transition weight.

Definition 7. *Let $W = \{c(\tau) \mid \tau \in \Sigma(Q)\}$ be the set of occurring weights. The* syntactic dta *for M is* $\mathrm{alph}(M) = (Q, \Sigma \times W, \delta', F)$*, where*

- $(\Sigma \times W)_k = \Sigma_k \times W$ *for every $k \in \mathbb{N}$, and*
- $\delta'(\langle \sigma, w \rangle(q_1, \ldots, q_k)) = q$ *if and only if*

$$\delta(\sigma(q_1, \ldots, q_k)) = q \quad \text{and} \quad c(\sigma(q_1, \ldots, q_k)) = w$$

for every $\sigma \in \Sigma_k$, $w \in W$, and $q_1, \ldots, q_k \in Q$. If no such $q \in Q$ exists, then $\delta'(\langle \sigma, w \rangle(q_1, \ldots, q_k))$ is undefined.

Note that a dta with the above structure can be turned back into a dwta. We will write alph^{-1} for this operation. Clearly, these constructions can be performed in time $O(|M|)$. To prove that the change to the unweighted setting is correct, we still have to prove that the involved congruences coincide. Let $\cong$ be the classical (state) equivalence for $\mathrm{alph}(\mathrm{push}(M))$, and let $\equiv$ be the (state) equivalence for M. To prove that they coincide, we show both inclusions.

Lemma 8. *The equivalence $\equiv$ is a congruence of* $\mathrm{alph}(\mathrm{push}(M))$ *that respects finality.*

Proof. Let $\mathrm{alph}(\mathrm{push}(M)) = (Q, \Sigma \times W, \delta', F)$ and $\mathrm{push}(M) = (Q, \Sigma, \delta, c', F)$. Since M and $\mathrm{alph}(\mathrm{push}(M))$ have the same final states, $\equiv$ respects finality. For the congruence property, let $\sigma \in \Sigma_k$ and $q_1, \dots, q_k, p_1, \dots, p_k \in Q$ be such that $q_i \equiv p_i$ for every $i \in [1, k]$. Then $\delta(\sigma(q_1, \dots, q_k)) \equiv \delta(\sigma(p_1, \dots, p_k))$ because $\equiv$ is a congruence for M. If $c'(\sigma(q_1, \dots, q_k)) = w = c'(\sigma(p_1, \dots, p_k))$, then

$$\begin{aligned}\delta'(\langle\sigma, w\rangle(q_1, \dots, q_k)) &= \delta(\sigma(q_1, \dots, q_k)) \\ &\equiv \delta(\sigma(p_1, \dots, p_k)) = \delta'(\langle\sigma, w\rangle(p_1, \dots, p_k)) \ .\end{aligned}$$

For the remaining combinations of $\langle\sigma, w'\rangle$ both transitions would be undefined (or go to the sink state), which would prove the congruence property. It remains to show that $c'(\sigma(q_1, \dots, q_k)) = c'(\sigma(p_1, \dots, p_k))$.

By Def. 4, we have

$$c'(\sigma(q_1, \dots, q_k)) = \lambda(\delta(\sigma(q_1, \dots, q_k))) \cdot c(\sigma(q_1, \dots, q_k)) \cdot \prod_{i=1}^{k} \lambda(q_i)^{-1}$$

$$c'(\sigma(p_1, \dots, p_k)) = \lambda(\delta(\sigma(p_1, \dots, p_k))) \cdot c(\sigma(p_1, \dots, p_k)) \cdot \prod_{i=1}^{k} \lambda(p_i)^{-1} \ .$$

Now we prove that

$$\begin{aligned}&\lambda(\delta(C_j[q_j])) \cdot c(C_j[q_j]) \cdot \prod_{i=1}^{j-1} \lambda(p_i)^{-1} \cdot \prod_{i=j}^{k} \lambda(q_i)^{-1} \\ &= \lambda(\delta(C_j[p_j])) \cdot c(C_j[p_j]) \cdot \prod_{i=1}^{j} \lambda(p_i)^{-1} \cdot \prod_{i=j+1}^{k} \lambda(q_i)^{-1}\end{aligned}$$

for every $j \in [1, k]$, where $C_j = \sigma(p_1, \dots, p_{j-1}, \Box, q_{j+1}, \dots, q_k)$. Let $q'_j = \delta(C_j[q_j])$ and $p'_j = \delta(C_j[p_j])$. Since $q_j \equiv p_j$, we obtain that $q'_j \equiv p'_j$ because $\equiv$ is a congruence. Consequently, $\mathrm{sol}(q'_j) = C = \mathrm{sol}(p'_j)$. Moreover, we have

$$\frac{\lambda(q_j)}{\lambda(p_j)} = \frac{c(C[C_j[q_j]])}{c(C[C_j[p_j]])} = \frac{c(C[q'_j]) \cdot c(C_j[q_j])}{c(C[p'_j]) \cdot c(C_j[p_j])} \quad \text{and} \quad \frac{\lambda(q'_j)}{\lambda(p'_j)} = \frac{c(C[q'_j])}{c(C[p'_j])} \ ,$$

where the former holds because $C[C_j]$ is a sign of life for both q_j and p_j and the latter holds by definition. With these equations, let us inspect the main equality.

$$\frac{\lambda(\delta(C_j[q_j])) \cdot c(C_j[q_j]) \cdot \prod_{i=1}^{j-1} \lambda(p_i)^{-1} \cdot \prod_{i=j}^{k} \lambda(q_i)^{-1}}{\lambda(\delta(C_j[p_j])) \cdot c(C_j[p_j]) \cdot \prod_{i=1}^{j} \lambda(p_i)^{-1} \cdot \prod_{i=j+1}^{k} \lambda(q_i)^{-1}}$$

$$= \frac{\lambda(q'_j) \cdot c(C_j[q_j]) \cdot \lambda(q_j)^{-1}}{\lambda(p'_j) \cdot c(C_j[p_j]) \cdot \lambda(p_j)^{-1}} = \frac{c(C[q'_j]) \cdot c(C_j[q_j]) \cdot \lambda(p_j)}{c(C[p'_j]) \cdot c(C_j[p_j]) \cdot \lambda(q_j)}$$
$$= \frac{c(C[q'_j]) \cdot c(C_j[q_j]) \cdot c(C[p'_j]) \cdot c(C_j[p_j])}{c(C[p'_j]) \cdot c(C_j[p_j]) \cdot c(C[q'_j]) \cdot c(C_j[q_j])} = 1$$

Repeated application (from $i = 1$ to k) yields the desired statement. □

Theorem 9. *We have* $\equiv = \cong$.

Proof. Lemma 8 shows that $\equiv$ is a congruence of $\mathrm{alph}(\mathrm{push}(M))$ that respects finality. Since $\cong$ is the coarsest congruence of $\mathrm{alph}(\mathrm{push}(M))$ that respects finality by [17], we obtain that $\equiv \subseteq \cong$. The converse is trivial to prove. □

The currently fastest dwta minimization algorithm is presented in [19]. It runs in time $O(|M| \cdot |Q|)$. With the help of pushing, we achieve the run-time of the fastest minimization algorithm in the unweighted case.

Corollary 10. *For every dwta* $M = (Q, \Sigma, \delta, c, F)$*, we can compute a minimal, equivalent dwta in time* $O(|M| \log |Q|)$.

6 Testing Equivalence

In this section, we want to decide whether two given dwta are equivalent. To this end, let $M_1 = (Q_1, \Sigma, \delta_1, c_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, c_2, F_2)$ be dwta. The overall approach is presented in Alg. 3. First, we need to compute a correspondence between states. For every $q_1 \in Q_1$, we compute a tree $t \in \underline{\delta_1}^{-1}(q_1)$. If $\underline{\delta_1}^{-1}(q_1) = \emptyset$, then q_1 is not reachable and can be deleted. To avoid these details, let us assume that all states of Q_1 are reachable. In this case, we can compute an access tree $h(q_1) \in T_\Sigma$ for every state $q_1 \in Q_1$ in time $|M_1|$ using standard breadth-first search, where we unfold each state (i.e., explore all transitions leading to it) at most once. To keep the representation efficient, we store the access trees in the format $\Sigma(Q_1)$, where the states refer to their respective access trees. To obtain the correspondence g, we compute the corresponding state of Q_2 that is reached when processing the access trees. Formally, $g(q_1) = \underline{\delta_2}(h(q_1))$ for every $q_1 \in Q_1$. Consequently, we have that $h(q_1) \in \underline{\delta_1}^{-1}(q_1) \cap \underline{\delta_2}^{-1}(g(q_1))$ for every $q_1 \in Q_1$.

Next, we compute the coarsest congruence for the (reduced) sum $M_1 \uplus M_2$, where we intend to compute a congruential equivalence. This can be achieved by a simple modification of the standard minimization algorithms (for example [17]), in which we replace states $q_1 \in Q_1$ by their corresponding state $g(q_1)$, whenever we test a state of Q_2. For example, when splitting a block using the context $C = \sigma(q, \Box, q')$ with $q, q' \in Q_1$, we use this context for all states of Q_1 and the context $\underline{g}(C) = \sigma(g(q), \Box, g(q'))$ for all states of Q_2. If we find corresponding states that are not related by the congruence, then the dwta are obviously not equivalent because for corresponding states there exists a common tree t that

Alg. 3. Overall structure of our equivalence test

Require: dwta $M_1 = (Q_1, \Sigma, \delta_1, c_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, c_2, F_2)$
Ensure: return yes if and only if M_1 and M_2 are equivalent

$g \leftarrow \textsc{ComputeCorrespondence}(M_1, M_2)$ // complexity: $O(|M_1|)$
$M \leftarrow M_1 \uplus M_2$ // (reduced) sum of M_1 and M_2
$\cong \leftarrow \textsc{ComputeCoarsestCongruence}'(M, g)$ // complexity: $O(|M| \log |Q|)$
if $g \not\subseteq \cong$ **then**
  return no // g is not compatible with the coarsest congruence
$(L, \mathrm{sol}, \lambda) \leftarrow \textsc{ComputeSoL}(M, \cong, g)$ // complexity: $O(|M|)$
$\lambda_1 = \lambda|_{Q_1}$; $\lambda_2 = \lambda|_{Q_2}$ // prepare pushing weights
$M_1 \leftarrow \mathrm{push}_{\lambda_1}(M_1)$; $M_2 \leftarrow \mathrm{push}_{\lambda_2}(M_2)$ // complexity: $O(|M_1| + |M_2|)$
$N_1 \leftarrow \textsc{Minimize}(\mathrm{alph}(M_1), \cong|_{Q_1 \times Q_1})$ // complexity: $O(|M_1| \log |Q_1|)$
$N_2 \leftarrow \textsc{Minimize}(\mathrm{alph}(M_2), \cong|_{Q_2 \times Q_2})$ // complexity: $O(|M_2| \log |Q_2|)$
return $\textsc{Isomorphic?}(N_1, N_2)$

leads M_1 and M_2 into the respective state. Since the states are related by the congruence, there exists a context C that is accepted in only one of the states. This yields a difference of acceptance on $C[t]$.

Next, we compute signs of life and pushing weights. It is again important that equivalent states (in $M_1 \uplus M_2$) receive the same sign of life. We minimize M_1 and M_2 using the method of Section 5 (i.e., we perform pushing followed by unweighted minimization). Finally, we test the obtained unweighted dta for isomorphism. An easy adaptation of the statement (and proof) of Lemma 8 can be used to show that $q_1 \in Q_1$ and $q_2 \in Q_2$ are equivalent in $\mathrm{alph}(\mathrm{push}_{\lambda_1}(M_1))$ and $\mathrm{alph}(\mathrm{push}_{\lambda_2}(M_2))$, respectively (see Algorithm 3), if and only if $q_1 \equiv_M q_2$. This proves the correctness of Algorithm 3, whose run-time should be compared to the previously (asymptotically) fastest equivalence test for dwta of [9], which runs in time $O(|M_1| \cdot |M_2|)$.

Theorem 11. *We can test equivalence of M_1 and M_2 in time $O(|M| \log |Q|)$, where $|M| = |M_1| + |M_2|$ and $|Q| = |Q_1| + |Q_2|$.*

Acknowledgements. The authors gratefully acknowledge the insight and suggestions provided by the reviewers.

References

1. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoret. Comput. Sci. 18(2), 115–148 (1982)
2. Borchardt, B.: The Myhill-Nerode theorem for recognizable tree series. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 146–158. Springer, Heidelberg (2003)
3. Borchardt, B.: The Theory of Recognizable Tree Series. Ph.D. thesis, TU Dresden (2005)
4. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. J. Autom. Lang. Comb. 8(3), 417–463 (2003)

5. Bozapalidis, S.: Equational elements in additive algebras. Theory Comput. Syst. 32(1), 1–33 (1999)
6. Bozapalidis, S., Louscou-Bozapalidou, O.: The rank of a formal tree power series. Theoret. Comput. Sci. 27(1–2), 211–215 (1983)
7. Büchse, M., May, J., Vogler, H.: Determinization of weighted tree automata using factorizations. J. Autom. Lang. Comb. 15(3–4) (2010)
8. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), http://www.grappa.univ-lille3.fr/tata
9. Drewes, F., Högberg, J., Maletti, A.: MAT learners for tree series — an abstract data type and two realizations. Acta Inform. 48(3), 165–189 (2011)
10. Eisner, J.: Simpler and more general minimization for weighted finite-state automata. In: Hearst, M., Ostendorf, M. (eds.) HLT-NAACL 2003, pp. 64–71. ACL (2003)
11. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata. EATCS Monographs in Theoretical Computer Science, ch. 9, pp. 313–403. Springer, Heidelberg (2009)
12. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
13. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 1, vol. 3, pp. 1–68. Springer, Heidelberg (1997)
14. Golan, J.S.: Semirings and their Applications. Kluwer Academic Publishers, Dordrecht (1999)
15. Hebisch, U., Weinert, H.J.: Semirings – Algebraic Theory and Applications in Computer Science. World Scientific, Singapore (1998)
16. Högberg, J., Maletti, A., May, J.: Bisimulation minimisation for weighted tree automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 229–241. Springer, Heidelberg (2007)
17. Högberg, J., Maletti, A., May, J.: Backward and forward bisimulation minimization of tree automata. Theoret. Comput. Sci. 410(37), 3539–3552 (2009)
18. Kuich, W.: Formal power series over trees. In: Bozapalidis, S. (ed.) DLT 1997, pp. 61–101. Aristotle University of Thessaloniki (1998)
19. Maletti, A.: Minimizing deterministic weighted tree automata. Inform. Comput. 207(11), 1284–1299 (2009)
20. May, J., Knight, K., Vogler, H.: Efficient inference through cascades of weighted tree transducers. In: Hajič, J., Carberry, S., Clark, S., Nivre, J. (eds.) ACL 2010, pp. 1058–1066. ACL (2010)
21. Mohri, M.: Finite-state transducers in language and speech processing. Comput. Linguist. 23(2), 269–311 (1997)
22. Post, M., Gildea, D.: Weight pushing and binarization for fixed-grammar parsing. In: de la Clergerie, E.V., Bunt, H. (eds.) IWPT 2009, pp. 89–98. ACL (2009)

Periodicity Algorithms for Partial Words*

Florin Manea[1,2], Robert Mercaş[1], and Cătălin Tiseanu[2]

[1] Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, PSF 4120, D-39016 Magdeburg, Germany
[2] Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, RO-010014 Bucharest, Romania
flmanea@fmi.unibuc.ro,
{robertmercas,ctiseanu}@gmail.com

Abstract. In this paper we investigate several periodicity-related algorithms for partial words. First, we show that all periods of a partial word of length n are determined in $\mathcal{O}(n \log n)$ time, and provide algorithms and data structures that help us answer in constant time queries regarding the periodicity of their factors. For this we need a $\mathcal{O}(n^2)$ preprocessing time and a $\mathcal{O}(n)$ updating time, whenever the words are extended by adding a letter. In the second part we show that substituting letters of a word w with holes, with the property that no two holes are too close to each other, to make it periodic can be done in optimal time $\mathcal{O}(|w|)$. Moreover, we show that inserting the minimum number of holes such that the word keeps the property can be done as fast.

Keywords: Combinatorics on Words, Periodicity, Partial Words.

1 Introduction

Periodicity is one of the most fundamental properties of words. Problems correlated to periodicity computation have applications in formal languages and automata theory, algorithmic combinatorics on words, data compression, string searching and pattern matching algorithms (see [1–3] and the references therein). The first idea of a fast algorithm identifying all periods of a word was given in [4], with a small flaw, together with the first time-space optimal string matching algorithm. In [5], Crochemore provides the first correct time-space optimal algorithm computing the periods of a word. This solution, and many other subsequent efficient solutions of the problem, rely heavily on the possibility of performing string matching in linear time and space.

For partial words, sequences that beside regular symbols contain some holes or "don't cares", the concept of periodicity was also deeply analyzed ([3] surveys most of the work in this area, and discusses the results obtained in comparison with the ones obtained for words). To start with, the problem of testing the primitivity of a partial word (i.e., partial words with no period that divides their

* The work of Florin Manea and Robert Mercaş is supported by the *Alexander von Humboldt Foundation*.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 472–484, 2011.

length) was discussed in [6, 7], where partial solutions were proposed; similarly to the classical case, these solutions were based on matching algorithms. To this end, we recall that a fast pattern matching algorithm is provided in [8], starting from the ideas initiated in [9]. More precisely, deterministic pattern matching algorithms solving the problem in $\mathcal{O}(n \log n)$ are known (see also [10]).

The study of repetitions in partial words was initiated in [11]. A string is said to contain a repetition if it has consecutive factors compatible with the same full word. In [11], it is proved that over a binary alphabet there exist infinite partial words that are cube-free, which, in other words, means that for all factors the periods are greater than one third of the factor's length. In [12] the authors solve a conjecture regarding the minimum size alphabet needed to construct an infinite partial word that remains overlap-free even after an arbitrarily insertion of holes; that is, all factors of the infinite word have periods greater than half their length. As an important step of their proof, the authors use a $\mathcal{O}(nd)$ algorithm that determines if, after hole insertions such that between each two holes there are at least $d-1$ non-hole symbols, a word has a certain period.

Algorithms regarding freeness of factors of partial words, factors free of some property, were firstly discussed in [11, 13]. In [13] the authors construct data structures which enable them, after a preprocessing phase done in $\mathcal{O}(n^2)$, to answer queries regarding the freeness of their factors in constant time. Moreover, the authors provide a method to update the data structures, in $\mathcal{O}(n \log n)$ time, whenever a symbol is concatenated to the right end of the existing string, and still answer the queries in constant time.

This paper proposes a series of algorithms for some basic problems related to periodicity in partial words, more efficient than the already existing ones, and discusses possible generalizations of these problems. After presenting some basics regarding partial words and periodicity, in the end of this section, our paper continues with two main parts. First, in Section 2 results from [6, 7, 13] are extended and improved. We investigate how all periods of a partial word of length n are determined in $\mathcal{O}(n \log n)$ time, and then we provide algorithms and data structures that help us answer in constant time queries regarding the periodicity of factors. Whenever the words are extended, we have a $\mathcal{O}(n^2)$ preprocessing time and a $\mathcal{O}(n)$ updating time. In Section 3, we give algorithms that identify ways of making a word periodic by substituting in optimal time some letters by holes, in a restricted way, and improve the already mentioned results from [12].

While the results of Section 2 regard some natural algorithmic questions on the periodicity of words, note that the results presented in Section 3 may become useful in the area of combinatorics on words. For instance, one may be interested in constructing words that do not contain any periodic partial words, that have at least several symbols between any of their holes ([11, 12]); that is, one looks for words in which we can randomly substitute symbols with holes, such that no two holes are too close, and they remain nonperiodic. Theorem 2, from the Section 3, enables us to test efficiently whether a given word verifies this property or not. Also, in [14, 15] the idea of producing from given full words, by substitution of

symbols with holes, partial words that verify some combinatorial properties is discussed, and possible connections with bio-informatics are established.

We continue with several basic definitions. Note that for most of the claims we only give sketches of proofs, and the algorithms are only intuitively specified.

Let V be a nonempty finite set of symbols called an *alphabet*. Each element $a \in V$ is called a *letter*. A *full word* over V is a finite sequence of letters from V, while a *partial word* over V is a finite sequence of letters from $V_\diamond = V \cup \{\diamond\}$, the alphabet V extended with the hole symbol $\diamond$.

The *length* of a partial word u is denoted by $|u|$ and represents the total number of symbols in u. The *empty word* is the sequence of length zero and is denoted by ε. A partial word u is a *factor* of a partial word v if $v = xuy$ for some x, y. We say that u is a *prefix* of v if $x = \varepsilon$ and a *suffix* of v if $y = \varepsilon$. We denote by $u[i]$ the symbol at position i in u, and by $u[i..j]$ the factor of u starting at position i and ending at position j, consisting of the concatenation of the symbols $u[i], \ldots, u[j]$, where $1 \leq i \leq j \leq |u|$.

The powers of a partial word u are defined recursively by $u^0 = \varepsilon$ and for $n \geq 1$, $u^n = uu^{n-1}$. The *period* of a partial word u over V is a positive integer p such that $u[i] = u[j]$ whenever $u[i], u[j] \in V$ and $i \equiv j \pmod{p}$. In such a case, we say u is *p-periodic*.

If u and v are two partial words of equal length, then u is said to be *contained* in v, denoted by $u \subset v$, if $u[i] = v[i]$, for all $u[i] \in V$. Partial words u and v are *compatible*, denoted by $u \uparrow v$, if there exists w such that $u \subset w$ and $v \subset w$.

A partial word u is said to be d-valid, for a positive integer d, if $u[i..i+d-1]$ contains at most one $\diamond$-symbol, for all $1 \leq i \leq |u| - d + 1$.

For a complete view on the basic definitions regarding combinatorics on words and partial words we refer the reader to [1–3]. The basic definitions needed to follow the algorithms presented here are found in [16]; we just stress out that all time bounds provided in this paper hold on the *unit-cost RAM model*.

2 Testing the Periodicity of Partial Words

In this section we present a series of algorithms that identify, efficiently, all the periods of a partial word, and then we follow an approach from [13] and discuss how one can efficiently build data structures that allow us to answer in constant time queries asking whether a factor of a partial word is periodic. We also present a method to update these data structures when a new symbol is added to the initial word. For the rest of this section, let w be a partial word of length n.

It is trivial to check in linear time whether there exists a symbol a_i that contains all symbols of the set $S_i = \{w[i + kp] \mid i + kp \leq n, k \in \mathbb{N}\}$, for all $i \leq n$, given w and p as input. This is equivalent to deciding whether the partial word w is p-periodic.

Let us provide an algorithm that finds all periods of a given partial word w, of length n. Note that w is p-periodic if and only if w is compatible with the prefix of length n of the word $w[ip + 1..n]\diamond^n$, for all $i \in \mathbb{N}$ such that $ip \leq n$.

First, for all values i with $1 \leq i \leq n$, we store an n position array Occ such that $Occ[i] = 1$, if $w \uparrow w[i..n]\diamond^{i-1}$, and $Occ[i] = 0$, otherwise. This step can

be completed in $\mathcal{O}(n \log n)$ time by running the pattern matching algorithm from [8] for w and $w \diamond^n$. Next, we identify the positive numbers p that fulfill $Occ[kp+1] = 1$, for all k such that $0 \leq kp \leq n$. As we have noted above, such a number p is a period of w.

In order to prove that this step is implemented in $\mathcal{O}(n \log n)$ time, we first note that, for all i with $Occ[i+1] = 0$, none of i's divisors can be a period of w. The total number of divisors we need to analyze is $\sum_{i=1}^{n} \sigma(i)$, where $\sigma(i)$ is the number of positive divisors of i. Since $(\sum_{i=1}^{n} \frac{1}{i})/(\log n)$ converges to a positive constant, $\sum_{i=1}^{n} \sigma(i) \in \mathcal{O}(n \log n)$. In $\mathcal{O}(n \log n)$ space and time, using a method based on the Sieve of Eratosthenes, we create an array L of lists $L[i]$ of divisors for every $i \in \{1, \ldots, n\}$. Next, we define Per, an array with $n-1$ elements, all initially set to 1. Going through all $i \in \{0, \ldots, n-1\}$ with $Occ[i+1] = 0$, we set $Per[j] = 0$ for all j's that divide i. Computing Per takes $\mathcal{O}(n \log n)$ time, and from $Per[i] = 1$ it follows that $Per[ki] = 1$, for all k with $1 \leq ki + 1 \leq n$.

According to the above remarks, a number p is a period of w if and only if $Per[p] = 1$. Consequently, all periods of w, and thus, the minimal one, are computed in $\mathcal{O}(n \log n)$ time. Moreover, to decide if w is primitive we only need to check whether there exists p such that $Per[p] = 1$ and p divides $|w|$.

These results are particularly useful in two applications: finding the minimal period of a partial word and deciding whether a word is primitive. We are aware of the claims and proofs that these problems can be solvable in linear time (see [7] and, respectively, [6]). However, the algorithms proposed in these papers are relying on the fact that one can find all factors of a partial word ww that are compatible with w in linear time, by extending to the case of partial words some string matching algorithms for full words, that work in linear time. The proof of this fact was not given formally, and we are not convinced that such results actually hold, especially since, for partial words the length does not always equal the sum between the period and the border of the word as in the case of regular words. We refer the reader, for instance, to the discussions in [9].

Next, we consider the problem of how to construct data structures that allow us to answer in constant time queries regarding the periodicity of the factors of a partial word, and can be easily updated when the word is extended.

Problem 1.

1. Given a partial word w of length n over an alphabet V, preprocess it in order to answer, for $i \leq j$ and $i, j, p \in \{1, \ldots, n\}$, the following types of queries:

"Is $w[i..j]$ p-periodic?", denoted $\mathbf{per}(i, j, p)$.

"Which is the minimum period of $w[i..j]$?", denoted $\mathbf{minper}(i, j)$.

2. For a partial word w, consider the update operation : add a symbol $a \in V_\diamond$ to the right end of w, to obtain wa. Preprocess w and define a method to update the data structures constructed during this process, in order to answer in constant time **per** queries, after several update operations were applied to w.

We describe a solution for the first part of the problem, as one can easily adapt it to solve the second part as well.

First define a matrix A, such that, for $i, l \in \{1, \ldots, n\}$, we have $A[i][l] = k$ with $0 < k \leq i$ and $i - k$ divisible by l, if $w[k + l] = w[k + 2l] = \ldots = w[i] = \diamond$

and $w[k] \neq \diamond$, or, if such a k does not exist, $A[i][l]$ equals the leftmost position t of the word, where $i - t$ is divisible by l. The matrix is computed in quadratic time by a dynamic programming approach.

Then, we define a matrix T with $T[i][l] = \min\{j \mid 0 < j \leq i \text{ and } w[j..i]$ l-periodic$\}$, where $i, l \in \{1, \ldots, n\}$. This matrix can be used to answer **per** queries in constant time: we answer **yes** to query $\mathbf{per}(i, j, p)$ if $T[j][p] \leq i$, and **no** otherwise. Moreover, the matrix T can be computed in quadratic time by dynamic programming, using the already computed matrix A.

To efficiently answer **minper** queries, we define a matrix P_m, where, for $i, j \in \{1, \ldots, n\}$ with $j \leq i$, we have $P_m[j][i] = \min\{p \mid w[j..i]$ is p-periodic$\}$.

Note that, whenever $j > i$, we have $P_m[j][i] = 0$. Moreover, if $w[j..i]$ is not p-periodic, for any $p < i-j+1$, then $P_m[j][i] = i-j+1$. Also, $P_m[j][i] \geq P_m[j+1][i]$. Consequently, the matrix P_m can be computed using the following approach. For a number i, with $1 \leq i \leq n$, we first identify the longest factor $w[j_1..i]$ of $w[1..i]$ that is 1-periodic, and conclude that the minimum period of $w[\ell..i]$ is 1 for $j_1 \leq \ell \leq i$; then we identify the longest factor $w[j_2..i]$ of $w[1..i]$ that is 2-periodic, and conclude that the minimum period of $w[\ell..i]$ is 2 for $j_2 \leq \ell \leq j_1 - 1$. The process continues in this way, for the periods p, $3 \leq p \leq i$; in general we identify the longest factor $w[j_p..i]$ of $w[1..i]$ that is p-periodic, and conclude that the minimum period of $w[\ell..i]$ is p for $j_p \leq \ell \leq j_{p-1} - 1$. Clearly, for a fixed i we determine all the values $P_m[j][i]$ in $\mathcal{O}(n)$ time. Therefore, we compute the matrix P_m in $\mathcal{O}(n^2)$ time.

The answer to a query $\mathbf{minper}(j, i)$ is $P_m[j][i]$, and can be returned in $\mathcal{O}(1)$ time.

To solve the second part of Problem 1, assume that for a partial word w, of length n, we compute the matrices A and T, as described. Let wa be the word obtained after adding $a \in V_\diamond$ to w, and let us update A and T by adding to each of them a new column and a new row. Note that, for all $i \in \{1, \ldots, n+1\}$, $A[i][n+1] = i$ and $T[i][n+1] = 1$; moreover, for all $l \in \{1, \ldots, n\}$, both $A[n+1][l]$ and $T[n+1][l]$ are computed, in linear time, using the same dynamic programming approach as in the static case. Thus, according to [13], one can add the new rows and columns to T and A in time $\mathcal{O}(n)$. Once all structures are updated, we answer **per** queries exactly as previously described.

To update P_m we set $P_m[n+1][n+1] = 1$ and $P_m[n+1][i] = 0$, for all $i \leq n$, and use the same algorithm that computes the matrix P_m to compute $P_m[j][n+1]$ with $j \leq n$. This takes $\mathcal{O}(n)$ time, while answering **minper** queries is done as before. When the word is updated several times, we iterate this method.

The next theorem summarizes the results obtained in this section so far.

Theorem 1. *Let w be a partial word of length n.*
1. All the periods of w can be computed in time $\mathcal{O}(n \log n)$.
2. The partial word w can be processed in time $\mathcal{O}(n^2)$ in order to answer in constant time **per** *and* **minper** *queries. After an update operation, the previously constructed data structures are updated in $\mathcal{O}(n)$ time, and both* **per** *and* **minper** *queries are answered in time $\mathcal{O}(1)$.*

Remark 1. Computing the matrix T for a partial word enables us to also identify all the periodic factors of that word.

3 From Full Words to Periodic Partial Words

In this section we change our focus to constructing, in linear time, a p-periodic partial word, starting from a full word, by replacing some of its symbols with holes such that no two consecutive holes are too close one to the other:

Problem 2. [12] Given a word $w \in V^*$, and positive integers d and p, $d, p \leq |w|$, decide whether there exists a p-periodic d-valid partial word contained in w.

The input of this problem consists of the word w and the numbers p and d. The alphabet V can be arbitrarily large, but, clearly, we can assume that $|V| \leq n$ (that is, we do not care about symbols that do not appear in w).

We first approach a related problem.

Problem 3. Let q and m be positive integers, let M be a sequence of m arrays with at most q elements from V, denoted $M_1, \ldots, M_m$, and let d be a positive integer, $d \leq m$. Furthermore, let $k \geq 1$ be a positive constant, and $p_0, \ldots, p_k \in \{1, \ldots, m\}$ such that $p_0 = 0$, $p_k = m$ and $p_i < p_{i+1}$, for $i \in \{0, \ldots, k-1\}$; assume that the arrays $M_{p_i+1}, \ldots, M_{p_{i+1}}$ have $q - i$ elements. Decide whether one can replace by holes several symbols of M_i in order to obtain arrays M'_i, where $i \in \{1, \ldots, m\}$, such that the following two conditions hold:

1. There exists a symbol s_i that contains all the symbols of M'_i with $i \in \{1, \ldots, m\}$. This condition is called **the periodicity condition.**
2. For all $j \leq q$ and $i \in \{1, \ldots, m - d + 1\}$, a multiset $\{M'_i[j], \ldots, M'_{i+d-1}[j]\}$ contains at most one hole. If one of the arrays M'_ℓ in the set has less than j elements, then the symbol $M'_{\ell[j]}$ is missing from the set. This condition is called **the d-validity condition**.

The input consists of the sequence M and the number d; again, the alphabet V can be arbitrarily large, but we can assume that $|V| \leq mq$.

Intuitively, the solution of this problem is based on the following idea: a possible way to substitute some of the symbols of an array with holes, in order to fulfill the periodicity condition, induces some restrictions on the way the substitutions can be applied on the $d-1$ arrays that follow it (due to the d-validity condition). Thus, we obtain a set of restrictions for each d consecutive arrays. We propose a formalization of the substitutions that can be made on each array as boolean variables and of the restrictions induced by these substitutions as formulas involving the variables. The fact that all the restrictions must be simultaneously fulfilled reflects in that the conjunction of all the formulas must be satisfiable. We show how the formulas can be constructed and how the satisfiability of their conjunction can be decided efficiently.

More formally, the solution of Problem 3 has two main steps. But before discussing them, note that there are two types of substitutions (also called replacements) that we can apply to the symbols of an array M_i:

Type 1*:* We replace all the symbols $M_i[j] \neq M_i[1]$, with holes.

Type 2*:* There exists a symbol $M_i[j_0] \neq M_i[1]$ such that we replace all the symbols $M_i[j]$, different from $M_i[j_0]$, with holes.

Moreover, an array necessitates no substitution if all its symbols are equal.

The first step of our solution consists in defining the boolean variables x_i, for $1 \leq i \leq m$, where $x_i = 1$ if and only if we need to apply a *Type 2* replacement to M_i to reach a valid solution. We derive, for each d consecutive arrays of M, say $M_i, \ldots, M_{i+d-1}$, a boolean formula ϕ_i, involving some of the variables defined above. Intuitively, the formula ϕ_i can be satisfied if and only if some of the symbols of the corresponding arrays can be replaced with holes and obtain a sequence that verifies both the periodicity condition and the d-validity condition.

As a final step, from all these formulas we construct a new formula ϕ_M, which is true for an assignment of the variables x_i, with $1 \leq i \leq m$, if and only if Problem 3 has a solution (that can be obtained from this assignment). The formula ϕ_M has a quite simple form, and a solution for it can be easily obtained.

We claim that all these steps can be implemented in $\mathcal{O}(mq)$ time. The time complexity does not depend on the cardinality of V and the value of d.

Coming back to the solution of Problem 2, we show how it can be transformed in linear time $\mathcal{O}(|w|)$ into an instance of Problem 3, which can be solved also in time $\mathcal{O}(|w|)$; one can also give, effectively, the substitutions that must be performed in the word, in the same time complexity. Note, once more, that the complexity of our solutions depends only on $|w|$, and not on the cardinality of the input alphabet, nor on d, nor on p. Summarizing, we show:

Theorem 2. *Problem 2 can be decided in linear time $\mathcal{O}(|w|)$. A solution for this Problem can be obtained in the same time complexity.*

3.1 Solution of Problem 3

Here, we give some more technical details on the solution of Problem 3, especially on how it can be reformulated as a boolean-formula satisfiability problem, and on how this problem can be solved efficiently.

For the sequence of arrays M, define the sequence of binary arrays M^b, with $M_i^b[j] = 1$ if $M_i[j] = M_i[1]$, and $M_i^b[j] = 0$ otherwise. The following remarks are due to the d-validity condition:

Remark 2. In d consecutive arrays, only one *Type 2* replacement is possible. Also, if there exits k with $M_{j_0}^b[k] = M_{j_1}^b[k] = 0$, where $1 < k \leq q$ and $0 < j_1 - j_0 \leq d$, then one cannot apply *Type 1* replacements to both M_{j_0} and M_{j_1}.

Let us consider d consecutive arrays, $M_i, \ldots, M_{i+d-1}$, all with q elements, and assume $q \geq 2$, as the case when $q = 1$ is trivial. From Remark 2 we have:

Remark 3. If three arrays j_1, j_2 and j_3, from the above, have a position k with $1 < k \leq q$, where $M_{j_1}^b[k] = M_{j_2}^b[k] = M_{j_3}^b[k] = 0$, then Problem 3 has no solution.

For k with $1 < k \leq q$, let $L_i(k) = \{j \mid j \in \{i, \ldots, i+d-1\}, M_j^b[k] = 0\}$ if the set is nonempty, and $L_i(k) = \{1, \ldots, m+1\}$, otherwise. Moreover, denote by

$L_i = \bigcap_{k\in\{2,\ldots,q\}} L_i(k)$. Following our remarks, the only cases one should analyze are $|L_i| \in \{0, 1, 2\}$, since $|L_i| = m + 1$ requires no substitutions in the arrays, while if $3 \leq |L_i(k)| \leq d$, for some k, then the problem has no solution.

If $L_i = \emptyset$ and any two arrays have 0 at the same position, the problem has no solution. Otherwise, *Type 1* substitutions are done to all arrays that have 0.

If $L_i = \{j_0\}$, then either only one of the arrays $M_i^b, \ldots M_{i+d-1}^b$ contains 0, so we apply only *Type 1* substitutions, and have $\phi_i = 1$, or more than one array M_j^b contains 0, and, thus, we apply a *Type 2* replacement to M_{j_0}, *Type 1* substitutions to all other arrays, if possible, and have $\phi_i = x_j$.

Finally, whenever $L_i = \{j_1, j_2\}$ with $j_1 < j_2$, if there exists an array that contains 0, other than $M_{j_1}^b$ and $M_{j_2}^b$, then the problem has no solution, following Remark 3. Otherwise, we apply a *Type 2* replacement to one of the two arrays, *Type 1* substitutions to all the others, and set $\phi_i = x_{j_1}$ **xor** x_{j_2}.

A similar discussion can be done when $M_i, \ldots, M_{i+d-1}$ have different lengths.

Making use of the previous remarks, one obtains $InitialFormula(M, i)$, an algorithm returning the formula ϕ_i.

Remark 4. Following the above, if x_i does not appear in any of the ϕ_j formulas, where $1 \leq j \leq m - d + 1$, then the type of substitution applied to that array is irrelevant for the construction of a solution for the problem. However, if x_i appears in a formula, then there exists j with $0 < |j - i| < d$ and a position k, such that $M_i^b[k] = 0$ and $M_j^b[k] = 0$.

Now, a new algorithm $Simplify(\phi_1, \ldots, \phi_{m-d+1})$ groups the formulas ϕ_i, where $1 \leq i \leq m - d + 1$, according to their first variable, and gets the sets $F_1, \ldots, F_m$ of formulas whose first variables are, respectively, $x_1, \ldots, x_m$. In the same algorithm, we ensure that each of these sets contains at most one formula.

From an intuitive point of view, algorithms $InitialFormula$ and $Simplify$ implement the first step of our solution. They reveal which substitutions can be applied simultaneously on d consecutive arrays of M, such that both the periodicity and the d-validity conditions are fulfilled. It remains to put together all these restrictions and produce a solution for the whole sequence M.

With the help of another algorithm $Filter(F_1, \ldots, F_m)$, we simplify even more the formulas by keeping in mind that they must all be simultaneously satisfied, and by eliminating the variables that clearly cannot be true. More precisely, we detect for which of the arrays i we should apply a *Type 2* replacement, but cannot do this because of the *Type 1* replacements on the $d-1$ arrays that precede and the $d-1$ arrays that succeed it (and set $x_i = 0$). In this way, the sets $F_1, \ldots, F_m$ are updated. Moreover, we compute the set of equalities U containing $x_i = x_i^\ell$, for $i \in \{1, \ldots, m\}$ and some $\ell \leq q$, that indicate which symbol $s = M_i[\ell]$ cannot be substituted by a hole whenever a *Type 2* replacement is applied to M_i. Note also that this algorithm identifies the case when one of the formulas ϕ_i cannot be satisfied, thus, the initial problem has no solution.

In the end, starting from the simplified list of formulas, we construct ϕ_M by running another algorithm, $Formula(\phi_1, \ldots, \phi_{m-d+1})$. In fact this algorithm detects some redundant restrictions that appear in the list of formulas and eliminates

them; during this processing one can also identify a case when the problem has no solution. More precisely, we obtain first the sets of formulas $F_1, \ldots, F_m$, and the set of equalities U, by running $Filter(Simplify(\phi_1, \ldots, \phi_{m-d+1}))$. Then we modify iteratively these sets in the following manner. If we have two formulas $\rho = x_{j_1}$ **xor** x_{j_2} and $\phi = x_{j_1'}$ with $j_1' - j_1 \geq d$ and $j_1' - j_2 \leq d-1$ we delete ρ and add the atomic formula x_{j_1}. That is, on the array j_1' we must apply a Type 2 replacement, and we also must apply such a replacement on one of the arrays j_1 or j_2; but we cannot apply simultaneously a Type 2 replacement on j_1' and j_2 so we can only apply such replacements on j_1 and j_1'. Also, by similar reasons, if we have two formulas $\rho = x_{j_1}$ and $\phi = x_{j_1'}$ **xor** $x_{j_2'}$ with $j_2' - j_1 \geq d$ and $j_1' - j_1 \leq d-1$ we delete ϕ and add the atomic formula $x_{j_2'}$. Finally, if we have two formulas $\rho = x_{j_1}$ and $\phi = x_{j_1'}$ with $j_1' - j_1 \geq d-1$ we have no solution (two Type 2 replacements should be applied in d consecutive lines) and set $\phi_M = 0$. Once all the possible such modifications are applied, we get ϕ_M as the conjunction of all these formulas.

Recalling the intuitive explanations given in the previous sections, algorithms *Formula* and *Filter* implement the second step of our solution. They put together the restrictions reflected by the boolean formulas, and detect whether some of these restrictions are in conflict.

At this point, we claim that Problem 3 has a solution for the input sequence of arrays M if and only if there exists a truth assignment of the variables $x_1, \ldots, x_m$ that makes ϕ_M equal to 1. Clearly, instead of each formula x_i **xor** x_j that appears in ϕ_M we can write the formula $(x_i \vee x_j) \wedge (\overline{x_i} \vee \overline{x_j})$. Hence, ϕ_M is a formula in 2-Conjunctive Normal Form, that one can solve efficiently with an algorithm for the 2-CNF-SAT Problem, in linear time [17].

A more careful analysis shows that, the special form of the formula ϕ_M, as it results from Remark 4 and the modifications done in algorithm *Formula*, allows us to decide whether it is satisfiable more easily. More precisely, ϕ_M is not decidable when it equals 0, while, otherwise, a truth assignment that makes $\phi_M = 1$ is obtained as follows: if the clause (x_i) is present in ϕ_M, then we assign $x_i = 1$, delete these variables from ϕ_M, and for the rest of the variables that appear in the formula we choose an arbitrary assignment that makes all the clauses true. Intuitively, we choose a valid assignment for the first d variables, and this is propagated in the rest of the set. Once we found an assignment of the variables x_i, $1 \leq i \leq p$, that makes $\phi_M = 1$, the set of equalities U indicates exactly which symbols are replaced by holes. More precisely, the symbol $M_i[t]$ remains unchanged when $x_i = 0$ and $M_i[t] = M_i[1]$ or when $x_i = x_i^t = 1$.

In order to compute the time complexity of the approach described above we discuss several implementation details. To begin with, we make the usual assumption (for the RAM-model) that the time needed to read (access the memory location containing) a symbol from V or to compare two such symbols is constant. The size of an input of Problem 3 is proportional with the total number of elements of the arrays that form the sequence M, times the size of the representation of a symbol of V (denoted here size(V)), plus the number of bits needed to represent d. In other words, the size of the input is $\mathcal{O}(\text{size}(V)mq + \log d)$.

Under these assumptions, it is not hard to see that a direct implementation of the above algorithms leads to a complexity of $\mathcal{O}(mqd)$. However, we show that by using some efficient data structures one can attain a complexity of $\mathcal{O}(mq)$. More precisely, we show that the formula ϕ_M can be constructed in linear time and, since it has $\mathcal{O}(mq)$ clauses and variables, deciding its satisfiability, and finding an assignment of the variables that makes $\phi_M = 1$, also takes linear time. Hence, deciding if Problem 3 has a solution, and effectively finding one, is done in $\mathcal{O}(mq)$ time. The proof consists in several observations which ensure that each of the previously described procedures is implemented in $\mathcal{O}(mq)$ time.

First, the sequence M^b can be constructed in time $\mathcal{O}(mq)$, canonically.

The key idea of our implementation is used in the algorithm $InitialFormula$. Let us show how one computes efficiently the sets L_i and the formulas ϕ_i for all i. Fix $i \in \{1, \ldots, m-d+1\}$ and define, for each $k \in \{1, \ldots, q\}$, an ordered queue Q_k that contains, at the moment when L_i is computed, the values $\{i_1, \ldots, i_t\}$, such that $M^b_{i_j}[k] = 0$ and $0 \leq i_j - i < d$, for all $j \in \{1, \ldots, t\}$. Furthermore, define, for each $k \in \{1, \ldots, q\}$, a variable p_k that stores the cardinality of the queue Q_k. If any p_k is greater than 2 when L_i is computed, we decide that $\phi_i = 0$. Otherwise, we can determine in $\mathcal{O}(q)$ time the elements that appear in all the non-empty queues Q_k, and return them as the set L_i. Moreover, the formulas ϕ_i are computed in $\mathcal{O}(q)$ time. Note that, when $i = 1$, the queues Q_k with $k \in \{1, \ldots, q\}$, are constructed by simply traversing the arrays $M_1, \ldots, M_d$ in $\mathcal{O}(dq)$ time. When we move from computing L_i and the formula ϕ_i, to computing L_{i+1} and the formula ϕ_{i+1}, in order to update the queues, we have to delete the minimum element from all these queues, if this element equals i, and add to the queue Q_k the element $i + d$ for all values of k such that $M^b_{i+d}[k] = 0$. Again, this operations take $\mathcal{O}(q)$ time. Following these observations, the time needed to compute all formulas ϕ_i, for $i \in \{1, \ldots, m - d + 1\}$, is $\mathcal{O}(mq)$.

The algorithm $Simplify$ is implemented in linear time, as it basically sorts the initial formulas by a bucket-sort strategy, while the verifications done in $Filter$ are performed efficiently using ideas and data-structures similar to those presented above for $InitialFormula$. Finally, one can see from the explanations provided above that the algorithm $Formula$ runs in $\mathcal{O}(m)$ time: once we order the formulas with $Simplify$ we just have to look at consecutive formulas and see if a modification of the described type must be applied or not; we can modify each formula at most once, we may delete it, or we just leave it alone, so the total running time of this function is linear in m. We notice that, if at any point ϕ is the formula on top of the stack S and x_j its second variable, then all the variables x_i that appear in formulas from S verify $i < j$.

Summarizing, Problem 3 can be decided in linear time $\mathcal{O}(mq)$. A solution for this Problem can also be obtained in the same time complexity.

3.2 The Main Problem

We show now how to solve Problem 2 using the previously discussed solution of Problem 3, in time $\mathcal{O}(|w|)$. The time complexity of our approach does not depend on either the size of V, d, nor p.

If $d \leq p$, then define $p+d-1$ arrays M such that M_i contains all symbols $w[x]$ with $x \geq 1$ and $x \equiv i \pmod p$, while M_{p+j} contains the symbols $w[p+x]$ with $x \geq 1$ and $x \equiv j \pmod p$ for $1 \leq i \leq p$ and $1 \leq j \leq d-1$. Finally, construct ϕ_M as previously explained. Clearly, the first $(n \pmod p)$ arrays have $q = \lceil \frac{n}{p} \rceil$ elements, the next p arrays (or, if $p + d - 1 \leq p + (n \pmod p)$, the remaining arrays) have $q-1$ elements, and the remaining arrays have $q-2$ elements. So, in the formalism of Problem 3, the constant k equals 2, $m = p+d-1$, and $q = \lceil \frac{n}{p} \rceil$.

We associate to each M_i a variable x_i, as previously explained, for all $i \in \{1, \ldots, p+d-1\}$. Note that if $j \equiv i \pmod p$ and $j > i$ then the variables x_i and x_j are not independent, as substituting some of the symbols of the array M_i with holes implies a substitution of the symbols of M_j with holes. This is because M_j contains only symbols of w that appear also in M_i. To capture the dependency in our formula, we use the set of equalities U, produced by algorithm *Filter*, and we extend it to obtain a new set of equalities U', as follows. Generally, for some $i \leq p$ the formulas $\overline{x}_i = x_i^1$, $\overline{x}_{i+p} = x_{i+p}^1$ (i.e., applying Type 2 replacements on the arrays i or $i+p$ means that the first symbol is not preserved), and $\overline{x}_i^1 \neq x_{i+p}^1$ when $M_i[1] \neq M_{i+p}[1]$, will be added to U'. Assume now that we have $i \leq p$ and $x_i = x_i^\ell$ and $x_{i+p} = x_{i+p}^{\ell'}$ in U; we also add to U' the equalities $\overline{x}_i^\ell = x_{i+p}^{\ell'}$ if $M_i[\ell] \neq M_{i+p}[\ell']$, $\overline{x}_i^\ell = x_{i+p}^1$ if $M_i[\ell] \neq M_{i+p}[1]$, and $\overline{x}_i^1 = x_{i+p}^{\ell'}$ if $M_i[1] \neq M_{i+p}[\ell']$ (these formulas say that the replacements on array i and on array $i+p$ must leave the same symbol unchanged). Next, we conjunct ϕ_M and all formulas in U', to obtain a new formula ϕ'_M.

Problem 2 has a solution if and only if there exists a truth assignment of the variables $\{x_1, \ldots, x_{p+d-1}\} \cup \{x_i^j \mid 1 \leq j \leq q, 1 \leq i \leq p\}$ that gives $\phi'_M = 1$. This is decidable, for instance, by an algorithm solving the 2-SAT problem. A solution is constructed applying *Type 2* replacements to all M_i with $x_i = 1$ and *Type 1* replacements to the rest of the arrays, followed by corresponding substitutions in the input word. For the *Type 2* replacements, the symbols not replaced by $\diamond$'s are the symbols $M_i[t]$, where the equality $x_i = x_i^t$ is in U.

If $d > p$, we first define the sequence of arrays M, such that, for $i \leq p$, M_i contains all symbols $w[x]$ with $x \geq 1$ and $x \equiv i \pmod p$. Note that, although several symbols of an array can be replaced with holes, it is not the case for any two consecutive such symbols, since this violates the d-validity condition. Thus, there are at most two possibilities to choose the symbol that is not replaced by $\diamond$ in the array: if it contains two different symbols, one of them must be replaced with hole, and the other not. Choosing a symbol a in the first array that has at least two different symbols, determines on all arrays M_i of M a symbol a_i that remains unchanged: the symbol that is found in the respective array on the same position as the position of a in the array we transformed. Thus, all the symbols of M that are not replaced with holes are already determined. It remains to check if the obtained word is d-valid. If so, the problem has a solution; otherwise, one tries to keep unchanged a different symbol in the initial array. If we cannot find a valid replacement in this way either, then the problem has no solution.

It is not hard to see that the time needed to solve the problem, in this case, is also $\mathcal{O}(n)$, and does not depend on neither the size of V, d nor p. Clearly, this

time is optimal. In conclusion, we have shown Theorem 2. We stress out that this theorem improves the result in [12], where Problem 2 was solved in time $\mathcal{O}(nd)$. Note that our approach is different from the one in the cited paper.

Finally, we mention an optimization problem related to Problem 2: given a word $w \in V^*$ and two positive integers d and p with $d, p < |w|$, construct a p-periodic d-valid partial word contained in w, and having a minimum number of holes. An optimal solution for this problem is based on the above solution for Problem 2 and a dynamic programming strategy:

Theorem 3. *Given a word w, and positive integers d and p with $p < |w|$, the construction of a p-periodic d-valid partial word having minimum number of holes and contained in w can be done in $\mathcal{O}(|w|)$. A solution can be effectively obtained in the same time complexity.*

References

1. Choffrut, C., Karhumäki, J.: Combinatorics of words. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 329–438. Springer, Heidelberg (1997)
2. Lothaire, M.: Combinatorics on Words. Cambridge University Press, Cambridge (1997)
3. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press (2008)
4. Galil, Z., Seiferas, J.I.: Time-space optimal string matching. J. of Comput. and Syst. Sci. 26, 280–294 (1983)
5. Crochemore, M.: String-matching on ordered alphabets. Theor. Comput. Sci. 92(1), 33–47 (1992)
6. Blanchet-Sadri, F., Anavekar, A.R.: Testing primitivity on partial words. Discr. Appl. Math. 155(3), 279–287 (2007)
7. Blanchet-Sadri, F., Chriscoe, A.: Local periods and binary partial words: an algorithm. Theoret. Comput. Sci. 314(1-2), 189–216 (2004)
8. Cole, R., Hariharan, R.: Verifying candidate matches in sparse and wildcard matching. In: STOC 2002, pp. 592–601 (2002)
9. Fischer, M.J., Paterson, M.S.: String matching and other products. In: Complexity of Computation, SIAM-AMS Proceedings, vol. 7, pp. 113–125 (1974)
10. Clifford, P., Clifford, R.: Simple deterministic wildcard matching. Inform. Proc. Lett. 101(2), 53–54 (2007)
11. Manea, F., Mercaş, R.: Freeness of partial words. Theoret. Comput. Sci. 389(1-2), 265–277 (2007)
12. Blanchet-Sadri, F., Mercaş, R., Rashin, A., Willett, E.: An answer to a conjecture on overlaps in partial words using periodicity algorithms. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 188–199. Springer, Heidelberg (2009)
13. Diaconu, A., Manea, F., Tiseanu, C.: Combinatorial queries and updates on partial words. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 96–108. Springer, Heidelberg (2009)

14. Leupold, P.: Languages of partial words - how to obtain them and what properties they have. Grammars 7, 179–192 (2004)
15. Leupold, P.: Partial words for DNA coding. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 224–234. Springer, Heidelberg (2005)
16. Knuth, D.E.: The Art of Computer Programming Fundamental Algorithms, vol. 1. Addison-Wesley, Reading (1968)
17. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inform. Proc. Lett. 8(3), 121–123 (1979)

State Complexity of Operations on Input-Driven Pushdown Automata

Alexander Okhotin[1,2,⋆] and Kai Salomaa[3]

[1] Department of Mathematics, University of Turku, 20014 Turku, Finland
alexander.okhotin@utu.fi
[2] Academy of Finland
[3] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. The family of deterministic input-driven pushdown automata (IDPDA; a.k.a. visibly pushdown automata, a.k.a. nested word automata) is known to be closed under reversal, concatenation and Kleene star. As shown by Alur and Madhusudan ("Visibly pushdown languages", STOC 2004), the reversal and the Kleene star of an n-state IDPDA can be represented by an IDPDA with $2^{O(n^2)}$ states, while concatenation of an m-state and an n-state IDPDA is represented by an IDPDA with $2^{O((m+n)^2)}$ states. This paper presents more efficient constructions for the reversal and for the Kleene star, which yield $2^{\Theta(n \log n)}$ states, as well as an $m2^{\Theta(n \log n)}$-state construction for the concatenation. These constructions are optimal due to the previously known matching lower bounds.

1 Introduction

A subclass of deterministic pushdown automata called *input-driven pushdown automata* (IDPDA), in which the input letter determines whether the automaton should push a symbol, pop a symbol or leave the stack untouched, was introduced by Mehlhorn [13] in 1980s and had only one follow-up paper at the time [5]. Their systematic investigation was initiated more than twenty years later by Alur and Madhusudan [2], who renamed the model to *visibly pushdown automata*, proved that its deterministic and nondeterministic cases are equal in power, and established its closure under all basic operations. These results inspired an ongoing stream of research on the properties of this model [1,3,6,7,9,15]. Part of the literature has adopted yet another name for the same model: *nested word automata*.

Though deterministic and nondeterministic IDPDAs define the same class of languages, they differ in terms of succinctness of description. A pushdown automaton uses states of two types: internal states and pushdown symbols. The natural succinctness measure is the sum of the number of these states. The determinization blowup of nondeterministic IDPDAs was assessed by Alur and

⋆ Supported by the Academy of Finland under grant 134860.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 485–496, 2011.

Madhusudan [2,3], who proved that representing an n-state nondeterministic IDPDA in the worst case requires $2^{\Theta(n^2)}$ states in an equivalent deterministic IDPDA. Recently, the authors [16] defined an intermediate family of *unambiguous* IDPDAs and showed that transforming a nondeterministic automaton to an unambiguous one, as well as an unambiguous automaton to deterministic, requires $2^{\Theta(n^2)}$ states in each case.

The closure of the language family defined by input-driven pushdown automata under all basic language-theoretic operations, established by Alur and Madhusudan [2], leaves related succinctness questions. According to Alur and Madhusudan [2], the Kleene star of an n-state IDPDA can be represented by an IDPDA with $2^{O(n^2)}$ states, and the concatenation of an m-state and an n-state IDPDA is representable by an IDPDA with $2^{O((m+n)^2)}$ states. In each case, the construction proceeds by representing the result of the operation by a nondeterministic IDPDA, and then by determinizing the latter. For the reversal of an n-state IDPDA, Piao and Salomaa [17] presented a similar $2^{O(n^2)}$-state construction, that relies on constructing a nondeterministic IDPDA and determinizing it, and also gave an improved construction for the concatenation of two IDPDAs, which uses $m \cdot 2^{O(n^2)}$ states. At the same time, Piao and Salomaa [17] demonstrated lower bounds on the number of states required to represent these operations: the reversal and the Kleene star requires $2^{\Omega(n \log n)}$ states, and the concatenation requires $2^{\Omega(n \log n)}$ states as well.

This paper presents more efficient constructions for these three operations, and thus determines the asymptotically optimal number of states needed to represent them. It is shown that both the reversal and the star of an n-state IDPDA can be represented using only $2^{O(n \log n)}$ states, which coincides with the lower bound given by Piao and Salomaa [17]. For the concatenation, the proposed construction yields an IDPDA with $m2^{O(n \log n)}$ states. This result is accompanied by an $m2^{\Omega(n \log n)}$-state lower bound, which refines the lower bound by Piao and Salomaa [17].

2 Definitions

A *(deterministic) input-driven pushdown automaton* (IDPDA) [13,2] is a special case of a deterministic pushdown automaton, in which the input alphabet is split into three classes, Σ_{+1}, Σ_{-1} and Σ_0, and the type of the input symbol determines the type of the operation with the stack. For an input symbol in Σ_{+1}, the automaton always pushes one symbol onto the stack. If the input symbol is in Σ_{-1}, the automaton pops one symbol. Finally, for a symbol in Σ_0, the automaton may not use the stack: that is, neither modify it, nor even examine its contents.

Unless otherwise mentioned, the acronym IDPDA shall refer to a *deterministic* input-driven pushdown automaton, and when the computation is allowed to use nondeterminism, the model shall be referred to as a *nondeterministic IDPDA*.

Different names and different notation for these automata has been used in the literature. Most of the time, they are regarded as pushdown automata, and recent

literature often refers to them under an alternative name of *visibly pushdown automata* [1,2,3,6,7]. An alternative outlook at essentially the same definition regards this model as an automaton operating on nested words: a *nested word automaton* [3,9,17].

An IDPDA is formally defined over an *action alphabet*, which is a triple $\widetilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$, in which the components Σ_{+1}, Σ_{-1} and Σ_0 are finite disjoint sets. In the following, unless otherwise mentioned, Σ_{+1}, Σ_{-1} and Σ_0 always refer to components of an action alphabet, and their union is denoted by Σ. A string over $\widetilde{\Sigma}$ is an ordinary string over Σ, where each symbol is assigned a "type" depending on the component it belongs to.

Let Q denote the set of (internal) states of the automaton, with a subset of accepting states $F \subseteq Q$, let Γ be its pushdown alphabet, and let $\bot \in \Gamma$ be the initial pushdown symbol. For each input symbol $a \in \Sigma_{+1}$, the behaviour of the automaton is described by partial functions $\delta_a : Q \to Q$ and $\gamma_a : Q \to (\Gamma \setminus \{\bot\})$, which provide the next state and the symbol to be pushed onto the stack, respectively. For every $b \in \Sigma_{-1}$, there is a partial function $\delta_b : Q \times \Gamma \to Q$ specifying the next state, assuming that the given stack symbol is popped from the stack. For $c \in \Sigma_0$, the state change is described by a partial function $\delta_c : Q \to Q$. There is an additional condition that whenever the stack contains only one symbol (which shall be $\bot$), any attempts to pop this symbol will result in checking that it is there, but not actually removing it. Once the automaton processes the last symbol of the input, it accepts if the current state is in F, regardless of whether the stack is empty or not.

Most of the constructions in this paper manipulate functions, mainly functions from Q to Q, and some related notation ought to be introduced. For any sets X and Y, denote the set of functions from X to Y by Y^X. Such functions can be applied to subsets of X: if $f\colon X \to Y$ is a function, then $f(S) = \{\, f(x) \mid x \in S \,\}$ for any set $S \subseteq X$; similarly, the set of pre-images of a set $T \subseteq Y$ is denoted by $f^{-1}(T) = \{\, x \mid x \in X : f(x) \in T \,\}$. If $f\colon X \to Y$ and $g\colon Y \to Z$ are two functions, then their *composition* $g \circ f$ is a function from X to Z defined by $(g \circ f)(x) = g(f(x))$.

The set of *well-nested strings* over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$, denoted by L_{Dyck}, is defined by the following context-free grammar:

$$
\begin{aligned}
&S \to <S> && (< \in \Sigma_{+1},\ > \in \Sigma_{-1})\\
&S \to SS\\
&S \to c && (c \in \Sigma_0)\\
&S \to \varepsilon
\end{aligned}
$$

Computations of IDPDAs on well-nested strings have the most straight form. The automaton finishes reading such a string with the same stack contents as in the beginning, and it never attempts to pop any symbols underneath. The behaviour of an automaton on a well-nested string w can thus be characterized by a function $f_w\colon Q \to Q$, which maps the initial state of the automaton to its state after processing the string. The behaviour on a concatenation uv can be obtained as a function composition $f_{uv} = f_v \circ f_u$. Given an IDPDA A, one

can construct another IDPDA B with the set of states Q^Q, that calculates the behaviour of A on the longest last well-nested substring it reads; this idea will be used in several constructions in this paper.

The measure of succinctness of an IDPDA adopted in the literature is the combined number $|Q|+|\Gamma|$ of internal states and pushdown symbols, which may be regarded as two kinds of states. The *state complexity of a language* is the least value of $|Q|+|\Gamma|$ among the IDPDAs recognizing this language. The *state complexity of an operation* is a function mapping the state complexities of its arguments to the worst-case state complexity of the result.

Generally, state complexity lower bounds are established using more or less *ad hoc* methods [3,19]. Various lower bound criteria for the size of IDPDAs were given by Piao and Salomaa [17]. The following variant of those criteria is tailored for the specific purpose of proving the results in this paper.

Lemma 1. *Let A be an IDPDA over action alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ and let S be a set of strings over Σ_0. Suppose that there exists a word $w \in \Sigma^*$ such that*

(i) *For each $u \in S$, wu is a prefix of some word of $L(A)$, and,*
(ii) *for any $u_1, u_2 \in S$, $u_1 \neq u_2$, there exists $v \in \Sigma^*$ such that $wu_1v \in L(A)$ if and only if $wu_2v \notin L(A)$.*

Then the number of states of A is at least $|S|$.

Proof. By (i) for each $u \in S$, A reaches the end of wu in some state q_u. Since the strings of S contain only symbols of Σ_0, condition (ii) implies that for any $u_1 \neq u_2$ the states q_{u_1} and q_{u_2} need to be distinct. □

3 Reversal

The reversal of a string $w = a_1a_2 \dots a_\ell$ over an action alphabet $\widetilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$, with $\ell \geqslant 0$ and $a_i \in \Sigma$, is the string $w^R = a_\ell \dots a_2a_1$ over the *inverted alphabet* $\widetilde{\Sigma}^R = (\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$, in which the symbol types "+1" and "−1" are interchanged. The reversal of a language L over $\widetilde{\Sigma}$ is the language $L^R = \{\, w^R \mid w \in L \,\}$, viewed as a language over the inverted alphabet $\widetilde{\Sigma}^R$.

Consider an deterministic input-driven pushdown automaton recognizing a language L. Then a *nondeterministic* IDPDA recognizing the language L^R can be obtained by reversing all transitions and exchanging the sets of initial and accepting states [17]. However, the construction for determinizing a nondeterministic IDPDA implies only an $2^{O(n^2)}$ upper bound on the number of states of a deterministic IDPDA recognizing the reversal. A more efficient construction for reversing a given deterministic IDPDA is given in this section.

The construction is first presented in a simplified form applicable to an IDPDA A operating on well-nested strings. Under this assumption, an IDPDA for $L(A)^R$ can separately calculate the behaviour of A as a function from Q to Q on every nested level of brackets, transfer the behaviour through the stack upon reading a closing bracket, and combine behaviours on substrings upon reading an opening bracket.

Lemma 2. *Let A be an IDPDA over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ that accepts only well-nested strings, let Q be its set of states and let Γ be its pushdown alphabet. Then there exists an IDPDA C with the set of states Q^Q and the pushdown alphabet $Q^Q \times \Sigma_{-1}$ that recognizes the language $L(A)^R$ (under the assumption that C rejects upon reaching the end of the input with nonempty stack).*

Proof. The goal of the construction is to simulate the behaviour of A on a well-nested string w, upon reading the string w^R. Note that the input alphabet of C is now the inverted alphabet $(\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$. A state of C represents the behaviour of A on the reversal of the longest well-nested suffix of the string read so far.

The initial state of C is $q'_0 = id$, the identity function on Q. The transition on each symbol from Σ_0 is defined as a composition of the transition function of A on this symbol with the calculated behaviour of A on the previously read suffix:

$$\delta'_c(f) = f \circ \delta_c.$$

If $f\colon Q \to Q$ is the behaviour of A on the suffix u, then this transition computes the behaviour of A on cu.

Whenever C reads a symbol from Σ_{-1}, on which A pops, C must push. What it does is to push the calculated behaviour $f\colon Q \to Q$ on the suffix and the current symbol $> \in \Sigma_{-1}$ to the stack, and begin calculating a new behaviour on the deeper level of nesting:

$$\delta'_>(f) = id,$$
$$\gamma'_>(f) = (f, >).$$

By the time C reaches the matching bracket $< \in \Sigma_{+1}$, it will have the behaviour of A on the inner level calculated in its internal state $g\colon Q \to Q$. It pops from the stack the pair $(f, >)$, where f is the behaviour on the suffix and $>$ is the other previously read bracket. Let u denote the well-nested string between these brackets, the behaviour on which is g, and let v be the suffix, on which the behaviour is f. Now B has all the data to calculate the behaviour of A on $<u>v$ as follows:

$$\delta'_<(g, (f, >)) = f \circ h,$$

where the function $h : Q \to Q$, defined by

$$h(q) = \delta_>\big(g(\delta_<(q)), \gamma_<(q)\big),$$

represents the behaviour of A on $<u>$.

Claim. For every string w over $\widetilde{\Sigma}$, let f be the behaviour of A on the longest well-nested prefix of w. Then the automaton C, executed on w^R with the initial state id, finishes its computation in the state f.

The claim is proved by an induction on the length of w. The basis is $w = \varepsilon$, with the empty string as the longest well-nested prefix, and the behaviour of A on it is the identity function, which is the initial state of C. For the induction step, the proof is split into three cases, depending on the form of w:

- If w begins with a symbol $c \in \Sigma_0$, let $w = cuv$, where u is longest well-nested prefix of the rest of w. The first portion of the computation of C on $w^R = v^R u^R c^R$ is a computation on the shorter string $v^R u^R$. By the induction hypothesis for uv, the automaton C computes the behaviour of A on u. Let f be this behaviour. Then cu is the longest well-nested prefix of w, and the transition of C by c correctly computes the behaviour of A on cu as a composition $f \circ \delta_c$.
- Let the first symbol of w be $< \in \Sigma_{+1}$. Then the longest well-nested prefix of w is a string of the form $<u>v$, where u and v are well-nested and $> \in \Sigma_{-1}$. Let $w = <u>vx$ and note that the longest well-nested prefix of vx is v. The computation of C on $w^R = x^R v^R > u^R <$ begins with the computation on $x^R v^R$ and, by the induction hypothesis, calculates the behaviour of A on v. Denote this behaviour by f. Next, C reads $>$ and pushes the pair $(f, >)$ to the stack, and afterwards, C processes the well-nested substring u^R. Applying the induction hypothesis to the ensuing computation of C—that is, to the string $u>vx$ with the longest well-nested prefix u—shows that C calculates the behaviour A on u. Finally, C reads $<$, pops the pair $(f, >)$ from the stack, uses the behaviour of A on u stored in the internal state to calculate the behaviour h of A on $<u>$, and finally combines it with the behaviour of A on v as $f \circ h$. This is the behaviour of A on $<u>v$, which is the longest well-nested prefix of w.
- Assume that w begins with a symbol $> \in \Sigma_{-1}$. Then the longest well-nested prefix of w is ε, and the computation of C on w^R ends with transition by $>$, which sets the internal state to *id*.

It is left to define the set of accepting states of C as

$$F' = \{\, f \colon Q \to Q \mid f(q_0) \in F \,\},$$

where q_0 is the initial state of A and F is the set of accepting states of A. Then, for every well-nested string w, the automaton C, executed on w^R computes the behaviour f of A on w, and accepts if and only if $f(q_0) \in F$, that is, if and only if A accepts w. If the input string is not well-nested, then C either reaches an undefined transition by the bottom stack symbol $\perp$, or finishes reading the input with nonempty stack. □

In the general case of IDPDAs operating on not necessarily well-nested strings, the construction is slightly extended.

Lemma 3. *For every IDPDA A over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ with a set of states Q and a pushdown alphabet Γ, there exists an IDPDA C over the inverted alphabet $(\Sigma_{-1}, \Sigma_{+1}, \Sigma_0)$ with the set of states $Q^Q \times 2^Q$ and the pushdown alphabet $Q^Q \times 2^Q \times \Sigma_{-1}$ that recognizes the language $L(A)^R$.*

Proof. Given a string uv, where u is its longest well-nested prefix, the automaton C is given a string $v^R u^R$ and should simulate the computation of A on uv. The goal of the construction is that C calculates (i) the behaviour of A on u, and (ii) the set of states, beginning from which A would accept the string uv.

The initial state of C is $q'_0 = (id, F)$, where F is the set of accepting states of A.

On any symbol $c \in \Sigma_0$, its transitions are:

$$\delta'_c\big((f,S)\big) = \big(f \circ \delta_c, \delta_c^{-1}(S)\big).$$

Every symbol $> \in \Sigma_{-1}$, which is a closing bracket for A, is therefore an opening bracket for C, and its transitions are defined as follows:

$$\delta'_>\big((f,S)\big) = (id, \{\, q \mid \delta_>(q,\bot) \in S \,\}),$$
$$\gamma'_>\big((f,S)\big) = (f, S, >).$$

Each A's opening bracket $< \in \Sigma_{+1}$ is regarded by C as a closing bracket, and is processed as follows:

$$\delta'_<\big((g,T),(f,S,>)\big) = (f \circ h, h^{-1}(S)),$$

where $h : Q \to Q$ is defined by

$$h(q) = \delta_>\big(g(\delta_<(q)), \gamma_<(q)\big).$$

And if the stack is empty (that is, $<$ has no matching right bracket):

$$\delta'_<\big((g,T),\bot\big) = (id, \delta_<^{-1}(T))$$

Claim. For every string w, the automaton C, after reading w^R, reaches a state (f,S), where f is the behaviour of A on the longest well-nested prefix of w, and S is the set of all such states q, that A, having begun a computation in the state q with the empty stack, accepts after reading w.

Finally, the set of accepting states of C is defined as

$$F' = \{\, (f,S) \mid q_0 \in S \,\},$$

where q_0 is the initial state of A. □

A matching lower bound for reversal is already known.

Proposition 1 (Piao and Salomaa [17]). *Let* $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$, $\Sigma_0 = \{a,b,c\}$. *For* $n \geqslant 1$, *the language*

$$L_n = \bigcup_{u \in \{a,b\}^{\lceil \log n \rceil}} u<(\{a,b\}^*c)^{n-1}uc(\{a,b\}^*c)^*>$$

has an IDPDA with $O(n)$ *states, while any IDPDA for its reversal requires at least* $2^{\Omega(n \log n)}$ *states.*

This shows that the construction in Lemma 3 is optimal up to a constant multiple in the exponent, and thus the state complexity of reversal has been determined as follows.

Theorem 1. *The state complexity of reversal of IDPDAs is* $2^{\Theta(n \log n)}$.

4 Concatenation

Given an m-state IDPDA and an n-state IDPDA, one can represent their concatenation by a nondeterministic IDPDA with $m+n$ states, and then determinize it to obtain $2^{O((m+n)^2)}$ states [2]. An improved construction given below directly yields a deterministic IDPDA, and this IDPDA contains only $m2^{O(n \log n)}$ states. Like in the previous section, this construction is first presented in an idealized form, in which all strings are well-nested, an no unmatched brackets may appear.

Lemma 4. *Let A and B be IDPDAs over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ that accept only well-nested strings, let P and Q be their respective sets of states, let Γ and Ω be their pushdown alphabets. Then there exists an IDPDA C with the set of states $P \times (2^Q \cup Q^Q)$ and the pushdown alphabet $\Gamma \times (2^Q \cup Q^Q) \times \Sigma_{+1}$ that recognizes the language $L(A) \cdot L(B)$.*

Proof. The first component of all states of C is used to simulate the operation of A. The states from $P \times 2^Q$ are used for well-nested prefixes of the input. After reading a well-nested prefix w, the automaton C should reach a state with a second component $S \subseteq Q$, containing all states of B reached on strings in $\{ u \mid uw \in L(A) \}$. After reading a prefix w that is not well-nested, C should calculate a state with the second component $f \colon Q \to Q$ representing the behaviour of B on the longest well-nested suffix of w.

Let π and δ be the transition functions of A and B, respectively. Let μ and ν be their push functions on Σ_{+1}, let p_0 and q_0 be their initial states, and let F_A and F_B be their sets of accepting states.

The initial state of C is $(p_0, \{ q_0 \mid \text{if } p_0 \in F_A \})$.

Its transitions on the symbols outside all brackets, for $p \in P$, $S \subseteq Q$ and $c \in \Sigma_0$, are:

$$\delta'_c\big((p,S)\big) = \big(\pi_c(p),\ \delta_c(S) \cup \{q_0 \mid \text{if } \pi_c(p) \in F_A\}\big).$$

When C enters the first level of brackets by a symbol $< \in \Sigma_{+1}$, it continues simulating A in the first component, and switches to computing the behaviour of B in the second component:

$$\begin{aligned} \delta'_<\big((p,S)\big) &= (\pi_<(p), id), \\ \gamma'_<\big((p,S)\big) &= \big(\mu_<(p),\ S,\ <\big), \end{aligned}$$

where $id : Q \to Q$ is the identity function. The state S and the symbol $<$ are stored in the stack, along with the stack symbol of the simulated automaton A.

When C returns from the first level of brackets by a symbol $> \in \Sigma_{+1}$, it has the behaviour of B on the substring inside the brackets computed, and can combine it with the behaviour on the brackets $<$ (popped from the stack) and $>$ (read from the input) to form a function $h : Q \to Q$, defined by

$$h(q) = \delta_>(g(\delta_<(q)), \nu_<(q)).$$

Then C can apply this function to the set S popped from the stack as follows:

$$\delta'_{>}\big((p,g),(s,S,<)\big) = \big(\pi_{>}(p,s),\ h(S)\big).$$

Transitions inside the brackets, on $c \in \Sigma_0$, are

$$\delta'_{c}\big((p,f)\big) = (\pi_c(p),\ \delta_c \circ f).$$

Going into the next level of brackets: for $p \in P$, $f : Q \to Q$ and $< \in \Sigma_{+1}$,

$$\begin{aligned}\delta'_{<}\big((p,f)\big) &= (\pi_{<}(p), id),\\ \gamma'_{<}\big((p,f)\big) &= \big(\mu_{<}(p),\ f,\ <\big).\end{aligned}$$

Returning into the previous level of brackets, which is not the first level: for $> \in \Sigma_{-1}$, $p \in P$, $f : Q \to Q$, $s \in \Gamma$, $g : Q \to Q$ and $< \in \Sigma_{+1}$,

$$\delta'_{>}\big((p,g),(s,f,<)\big) = \big(\pi_{>}(p,s),\ g \circ f\big),$$

where $h : Q \to Q$ is defined by

$$h(q) = \delta_{>}\big(g(\delta_{<}(q)), \nu_{<}(q)\big).$$

Claim. Upon reading a well-nested prefix w of an input string, the automaton C enters a state (p,S), where p is the state reached by A on w, and S is the set of all such states $q \in Q$ that $w = uv$ for some $u \in L(A)$ and $v \in \Sigma^*$ with $\delta(q_0, v) = q$.

Upon reading a not well-nested prefix w, the automaton C enters a state (p,f), where p is the state reached by A on w, and f is the behaviour of B on the longest well-nested suffix of w.

Let all states $(p,S) \in P \times 2^Q$ with $S \cap F_B \neq \varnothing$ be the accepting states of C. Then, by the above claim, C accepts a string w if and only if $w = uv$ for some $u \in L(A)$ and $v \in L(B)$, and therefore $L(C) = L(A)L(B)$. Unlike the automaton constructed in Lemma 2, here the automaton C is also able to reject strings other than well-nested. □

The next lemma presents the construction for concatenation of IDPDAs of the general form.

Lemma 5. *Let A and B be any IDPDAs over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$, let P and Q be their respective sets of states, let Γ and Ω be their pushdown alphabets. Then there exists a IDPDA C with the set of states $P \times 2^Q \times 2^Q \times Q^Q$ and the pushdown alphabet $\Gamma \times 2^Q \times 2^Q \times Q^Q \times \Sigma_{+1}$ that recognizes the language $L(A) \cdot L(B)$.*

The automaton C simulates A in the first component of its state, and calculates the behaviour of B on the last well-nested suffix of the input in the fourth component. The second and the third components are both sets of such states $q \in Q$, that the string read so far is a concatenation of a string in $L(A)$ with a string, on

which B goes from q_0 to q. The third component handles all such factorizations, where the suffix processed by B is a concatenation of well-nested strings and closing brackets from Σ_{-1}. The second component refers to factorizations with the suffix of any other form. The details of the construction are omitted due to space constraints.

A $2^{\Omega(n \log n)}$-state lower bound on the state complexity of concatenation of an m-state IDPDA and an n-state IDPDA was given by Piao and Salomaa [17]. This lower bound does not provide any dependence of the state complexity of concatenation on the complexity of the first language. The following more elaborate version refines the result of Piao and Salomaa [17] to reflect this dependence.

Lemma 6. *Let $\Sigma_0 = \{a, b, \#\}$, $\Sigma_{+1} = \{<\}$ and $\Sigma_{-1} = \{>\}$. Then, for every $m, n \geqslant 1$, the language*

$$K_m = \left\{ w \in \{a, b, \#, <, >\}^* \;\middle|\; |w|_b \equiv 0 \pmod{m} \right\}$$

has a DFA with m states, and the language

$$L_n = \bigcup_{k,\ell \in \{1,\ldots,n\}} a^k b^* \#(a^* b^* \#)^* < (a^* b^* \#)^k a^\ell b^* \#(a^* b^* \#)^* a^\ell > a^k$$

has an IDPDA with $O(n)$ states and n pushdown symbols, while any IDPDA for their concatenation $K_m L_n$ requires at least mn^n states.

Theorem 2. *The state complexity of concatenation of IDPDAs is $m \cdot 2^{\Theta(n \log n)}$.*

Using a variant of the languages L_n from Lemma 6, we get a lower bound for the state complexity of square.

Lemma 7. *Let $\Sigma_0 = \{a, \#\}$, $\Sigma_{+1} = \{<\}$ and $\Sigma_{-1} = \{>\}$. Then, for every $n \geqslant 1$, the language*

$$L_n = (a^* \#)^* \cup \bigcup_{k,\ell \in \{1,\ldots,n\}} a^k \#(a^* \#)^* < (a^* \#)^k a^\ell \#(a^* \#)^* a^\ell > a^k$$

has an IDPDA with $O(n)$ states and n pushdown symbols, while any IDPDA for the language $L_n \cdot L_n$ requires at least n^n states.

Theorem 3. *The state complexity of square of IDPDAs is $2^{\Theta(n \log n)}$.*

5 Kleene Star

The concatenation $L(A)L(B)$ was recognized by simulating A as it is, along with keeping track of all possible computations of B following a prefix in $L(A)$. Every time the simulated A would accept, one more computation of B was added to the set.

The below construction for the star is derived from the one for the concatenation. There is no automaton A this time, but multiple computations of B are traced in the same way as before. Whenever one of them would accept, the set is augmented with another computation of B.

Lemma 8. *Let B be an IDPDAs over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ that accepts only well-nested strings, let Q be its set of states. Then there exists a IDPDA C with the set of states $2^Q \cup Q^Q$ and the pushdown alphabet $(2^Q \cup Q^Q) \times \Sigma_{+1}$ that recognizes the language $L(A)^*$.*

The construction is extended to the full case of IDPDAs not restricted to well-nested strings as follows.

Lemma 9. *Let B be any IDPDA over an alphabet $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$, let Q be its set of states, and let Γ be its pushdown alphabet. Then there exists a IDPDA C with the set of states $2^Q \times 2^Q \times Q^Q$ and the pushdown alphabet $\Gamma \times 2^Q \times 2^Q \times Q^Q \times \Sigma_{+1}$ that recognizes the language $L(A)^*$.*

The construction is similar to the one for the concatenation, and is not included in this extended abstract.

This establishes a $2^{O(n \log n)}$-state upper bound on the state complexity of the star for IDPDAs. A matching lower bound is already known from Salomaa [19], who presented an IDPDA A with $O(n)$ states and stack symbols, such that the total number of states and stack symbols in any IDPDA recognizing $L(A)^*$ is at least $2^{n \log n}$. These results are combined to the following asymptotic estimation.

Theorem 4. *The state complexity of Kleene star of IDPDAs is $2^{\Theta(n \log n)}$.*

6 Conclusion

The complexity of all basic operations on both deterministic (IDPDA) and nondeterministic (NIDPDA) input-driven pushdown automata has now been determined. In the following table, it is compared to the similar results on two basic types of finite automata recognizing regular languages.

	DFA	NFA	IDPDA	NIDPDA
$\cup$	mn [12]	$m+n+1$ [10]	$\Theta(mn)$ [17]	$m+n+O(1)$ [2]
$\cap$	mn [12]	mn [10]	$\Theta(mn)$ [17]	$\Theta(mn)$ [9]
$\sim$	n	2^n [4]	n	$2^{\Theta(n^2)}$ [16]
$\cdot$	$m \cdot 2^n - 2^{n-1}$ [12]	$m+n$ [10]	$m2^{\Theta(n \log n)}$	$m+n+O(1)$ [2]
2	$n \cdot 2^n - 2^{n-1}$ [18]	$2n$ [8]	$2^{\Theta(n \log n)}$	$n+O(1)$ [2]
$*$	$\frac{3}{4}2^n$ [12]	$n+1$ [10]	$2^{\Theta(n \log n)}$	$n+O(1)$ [2]
R	2^n [11]	$n+1$ [10]	$2^{\Theta(n \log n)}$	$n+O(1)$ [2]

Investigating the complexity of operations on *unambiguous IDPDAs* [16] is suggested for future work. Since descriptional complexity questions are already difficult for unambiguous finite automata [14], this task might be nontrivial as well.

The authors are grateful to the anonymous referees for valuable comments, many of which could not yet be addressed due to space and time constraints.

References

1. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1102–1114. Springer, Heidelberg (2005)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: ACM Symposium on Theory of Computing STOC 2004, Chicago, USA, June 13–16, pp. 202–211 (2004)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM 56, 3 (2009)
4. Birget, J.C.: Partial orders on words, minimal elements of regular languages, and state complexity. Theoretical Computer Science 119, 267–291 (1993)
5. von Braunmuhl, B., Verbeek, R.: Input-driven languages are recognized in $\log n$ space. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 40–51. Springer, Heidelberg (1983)
6. Chervet, P., Walukiewicz, I.: Minimizing variants of visibly pushdown automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 135–146. Springer, Heidelberg (2007)
7. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 214–226. Springer, Heidelberg (2010)
8. Domaratzki, M., Okhotin, A.: State complexity of power. Theoretical Computer Science 410, 24–25 (2009)
9. Han, Y.-S., Salomaa, K.: Nondeterministic state complexity of nested word automata. Theoretical Computer Science 410, 2961–2971 (2009)
10. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. International Journal of Foundations of Computer Science 14, 1087–1102 (2003)
11. Leiss, E.L.: Succinct representation of regular languages by Boolean automata. Theoretical Computer Science 13, 323–330 (1981)
12. Maslov, A.N.: Estimates of the number of states of finite automata. Soviet Mathematics Doklady 11, 1373–1375 (1970)
13. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980)
14. Okhotin, A.: Unambiguous finite automata over a unary alphabet. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 556–567. Springer, Heidelberg (2010)
15. Okhotin, A.: Comparing linear conjunctive languages to subfamilies of the context-free languages. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 431–443. Springer, Heidelberg (2011)
16. Okhotin, A., Salomaa, K.: Descriptional Complexity of Unambiguous Nested Word Automata. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 414–426. Springer, Heidelberg (2011)
17. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. Theoretical Computer Science 410, 3290–3302 (2009)
18. Rampersad, N.: The state complexity of L^2 and L^k. Information Processing Letters 98, 231–234 (2006)
19. Salomaa, K.: Limitations of lower bound methods for deterministic nested word automata. Information and Computation 209, 580–589 (2011)

Conflict Packing Yields Linear Vertex-Kernels for k-FAST, k-dense RTI and a Related Problem*

Christophe Paul, Anthony Perez, and Stéphan Thomassé

LIRMM - Université Montpellier 2, CNRS - France

Abstract. We develop a technique that we call *Conflict Packing* in the context of kernelization [7]. We illustrate this technique on several well-studied problems: Feedback Arc Set in Tournaments, Dense Rooted Triplet Inconsistency and Betweenness in Tournaments. For the former, one is given a tournament $T = (V, A)$ and seeks a set of at most k arcs whose reversal in T results in an acyclic tournament. While a linear vertex-kernel is already known for this problem [6], using the Conflict Packing allows us to find a so-called *safe partition*, the central tool of the kernelization algorithm in [6], with simpler arguments. Regarding the Dense Rooted Triplet Inconsistency problem, one is given a set of vertices V and a dense collection $\mathcal{R}$ of rooted binary trees over three vertices of V and seeks a rooted tree over V containing all but at most k triplets from $\mathcal{R}$. Using again the Conflict Packing, we prove that the Dense Rooted Triplet Inconsistency problem admits a linear vertex-kernel. This result improves the best known bound of $O(k^2)$ vertices for this problem [16]. Finally, we use this technique to obtain a linear vertex-kernel for Betweenness in Tournaments, where one is given a set of vertices V and a dense collection $\mathcal{R}$ of *betweenness triplets* and seeks an ordering containing all but at most k triplets from $\mathcal{R}$. To the best of our knowledge this result constitutes the first polynomial kernel for the problem.

1 Introduction

The concept of *fixed parameter algorithms* [13] has been introduced to cope with NP-Hard problems. For a given (parameterized) problem, the goal is to identify a parameter k, independent from the data-size n, which captures the exponential growth of the complexity cost to solve the problem at hand. That is the complexity of such an (FPT) algorithm is $f(k) \cdot n^{O(1)}$, where f is an arbitrary computable function. As one of the most powerful techniques to design efficient fixed parameter algorithms, *kernelization algorithms* [7] have attracted a lot of attention during the last few years. A kernelization algorithm transforms, in polynomial time (via *reduction rules*), an arbitrary instance (I, k) of a parameterized problem into an equivalent instance (I', k') with the property that the

* Research supported by the AGAPE project (ANR-09-BLAN-0159) and the Phylariane project (ANR-08-EMER-011-01).

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 497–507, 2011.

parameter k' and the size $|I'|$ of the reduced instance only depend on k. The smaller the size of the reduced instance (called *kernel*) is, the faster the problem can be solved. Indeed, once reduced the instance can be efficiently tackled with any exact algorithms (e.g. bounded search tree or exponential time algorithms). In this paper, we develop and push further a kernelization technique used on a few parameterized problems [9,24], called *Conflict Packing*. Combined with a polynomial time algorithm that computes an accurate vertex partition (called *safe partition* in [6]), Conflict Packing yields linear vertex-kernels. In this extended abstract, we illustrate the Conflict Packing technique on three parameterized problems. We first obtain a linear vertex-kernel for Feedback Arc Set in Tournaments. While such a kernel was already known to exist [6], our proofs are simpler and shorter. Then we obtain the main result of this paper, namely the first linear vertex-kernel for Dense Rooted Triplet Inconsistency, improving the best known bound of $O(k^2)$ vertices [16]. Finally we apply the technique on Betweenness in Tournaments and obtain a linear vertex-kernel. No polynomial kernel was known before [18].

Feedback Arc Set in Tournaments *(k-FAST).* Let $T = (V, A)$ be a tournament on n vertices, *i.e.* an oriented graph obtained from an arbitrary orientation of the complete (undirected) graph, and k be an integer. The task is to check whether there exists a subset of at most k arcs of A whose reversal transforms T into an acyclic (i.e. transitive) tournament. In other words, k-FAST consists in computing a linear vertex ordering $v_1 \dots v_n$ having at most k *backward arcs* ($v_i v_j$ with $i > j$). It is well-known that a tournament is transitive if and only if it does not contain a directed triangle (circuit on 3 vertices). The k-FAST problem is a well studied problem from the combinatorial [14,21] as well as from the algorithmic point of view [3,19]. It is known to be NP-complete [3,11], but fixed parameter tractable [4,18,20]. The first kernelization algorithms for k-FAST [4,12] yield $O(k^2)$ vertex-kernels. Recently, a kernel with at most $(2+\epsilon)k$ vertices has been proposed [6]. More precisely, using a PTAS which computes a linear vertex ordering with at most $(1+\epsilon)k$ backward arcs, the authors of [6] show how to find in polynomial time an ordered vertex partition, called *safe partition* $\mathcal{P}$, of T. Roughly speaking, a vertex partition is *safe* if the backward arcs whose extremities lie in different part can be reversed independently from the others (inside the parts). We prove that the Conflict Packing technique (which provides a lower bound on the number of editions to be made) can be used to compute such a partition.

Dense Rooted Triplet Inconsistency *(k-dense RTI).* In phylogenetics, a classical problem consists in testing whether a collection $\mathcal{R}$ of rooted binary trees over three leaves of a set V, called *rooted triplets*, is *consistent*: does there exist a binary tree T on leaves V such that every rooted triplet $\{a, b, c\}$ of $\mathcal{R}$ is homeomorphic to the subtree of T spanning $\{a, b, c\}$? This problem can be solved in polynomial time [1]. When $\mathcal{R}$ is not consistent, different optimization problems can be considered: removing a minimum number of leaves or removing a minimum number of rooted triplets (see e.g. [10,23]). We consider the parameterized version of the latter problem in the dense case, called k-dense RTI, where one

is given a dense collection of rooted triplets $\mathcal{R}$ and an integer k (the parameter) and seeks a rooted tree over V containing all but at most k rooted triplets from $\mathcal{R}$. It is known that when $\mathcal{R}$ is dense, i.e. contains exactly one rooted triplet for every triple of leaves (or vertices), then it is consistent with a binary tree if and only if it does not contain any conflict on four leaves [5,16]. The k-DENSE RTI problem is known to be NP-complete [10] but fixed parameter tractable [15,16], the fastest algorithm running in time $O(n^4) + 2^{O(k^{1/3}logk)}$ [16]. Moreover, [16] provided a quadratic vertex-kernel for k-DENSE RTI. However, unlike for k-FAST, no PTAS nor constant approximation algorithm is known [16]. Using *Conflict Packing* enables us to obtain a linear vertex-kernel for this problem. This result improves the best known bound of $O(k^2)$ vertices [16].

BETWEENNESS IN TOURNAMENTS *(k-BIT).* In the k-BIT problem, one is given a set of vertices V, a dense collection $\mathcal{R}$ of *betweenness triplets* defined over V and an integer k. A *betweenness triplet* t defined over $\{a, b, c\}$ *chooses* one of its vertices (say b), and we say that an ordering σ over $\{a, b, c\}$ *contains* t if b is *between* a and c in σ. The aim of the k-BIT problem is to compute an ordering of V containing all but at most k triplets from $\mathcal{R}$. The k-BIT problem is NP-Complete [2] but fixed-parameter tractable [18,22]. Using *Conflict Packing* we obtain a linear vertex-kernel for this problem, which is to the best of our knowledge the first polynomial kernel for this problem [18].

Outline. We first illustrate the Conflict Packing technique on the k-FAST problem, proving how to compute a so-called safe partition in polynomial time (Section 2). Next, we generalize the results to the k-DENSE RTI problem (Section 3). Finally, we give another example where this technique can be applied, namely k-BIT (Section 4).

2 Linear Vertex-Kernel for k-FAST

Preliminaries. Let $T = (V, A)$ be a tournament. We write uv whenever the arc of A between vertices u and v is oriented from u to v. If $V' \subseteq V$, then $T[V'] = (V', A')$ is the subtournament induced by V', that is $A' = \{uv \in A \mid u \in V', v \in V'\}$. If $A' \subseteq A$, then $T[A'] = (V', A')$ denotes the oriented graph where $V' \subseteq V$ contains the vertices incident to some arc of A'. A tournament $T = (V, A)$ is *transitive* if for every triple of vertices u, v, w such that $uv \in A$ and $vw \in A$, then $uw \in A$. A *directed triangle* is a circuit of size three, i.e. a set of vertices $\{u, v, w\}$ such that $\{uv, vw, wu\} \subseteq A$.

Lemma 1 (Folklore). *Let $T = (V, A)$ be a tournament. Then the following properties are equivalent: (i) T is acyclic; (ii) T is transitive; (iii) T does not contain any directed triangle.*

Clearly T is transitive if and only if there exists an ordering σ on V such that for every $u \in V$ and $v \in V$ with $\sigma(u) < \sigma(v)$ (also denoted $u <_\sigma v$) then $uv \in A$. Such an ordering is called *transitive*. We use $T_\sigma = (V, A, \sigma)$ to denote a tournament whose vertices are ordered with respect to some ordering σ. An arc

$vu \in A$ such that $u <_\sigma v$ is called *backward* in T_σ. In other words, T is transitive if and only if there exists a total ordering σ of its vertices such that T_σ does not contain any backward arc. Hereafter, a directed triangle will be called a *conflict*. Our kernel uses the following rule from [4].

Rule 1. *Remove any vertex v that does not belong to any conflict.*

Certificate and safe partition. The following definitions are adapted from notions introduced in [6]. Let $e = vu$ be a backward arc of an ordered tournament T_σ. The *span* of e is the set of vertices $span(e) = \{w \in V \mid u <_\sigma w <_\sigma v\}$. If $w \in span(e)$ is a vertex not incident to any backward arc, then $c(e) = \{u, v, w\}$ is a *certificate* of e. Observe that $c(e)$ induces a conflict. By convention, when speaking of an arc e of a certificate c we mean that e belongs to $T[c]$. If $F \subseteq A$ is a set of backward arcs of T_σ, we can *certify* F whenever there exists a set $c(F) = \{c(f) : f \in F\}$ of arc-disjoint certificates (i.e. for every distinct e and f of F, $|c(e) \cap c(f)| \leqslant 1$). If $T_\sigma = (V, A, \sigma)$, then $\mathcal{P} = \{V_1, \ldots, V_l\}$ is an *ordered partition* of T_σ if it is a partition of V and for every $i \in [l]$, V_i is a set of consecutive vertices in σ. By convention, if $i < j$, then for every $u \in V_i$ and $v \in V_j$ we have $u <_\sigma v$. We denote by $A_O = \{uv \in A \mid u \in V_i, v \in V_j, i \neq j\}$ the subset of *outer-arcs*. We say that an ordered partition $\mathcal{P} = \{V_1, \ldots, V_l\}$ is a *safe partition* of an ordered tournament $T_\sigma = (V, A, \sigma)$ if A_O contains at least one backward arc and if it is possible to certify the backward arcs of $T_\sigma[A_O]$ only with outer-arcs of A_O. The following rule was central in the kernel of [6]:

Rule 2 (Safe partition). *Let T_σ be an ordered tournament, and $\mathcal{P} = \{V_1, \ldots, V_l\}$ be a safe partition of T_σ with F the set of backward arcs of $T_\sigma[A_O]$. Then reverse all the arcs of F and decrease k by $|F|$.*

Conflict packing. Our kernelization algorithm applies the two rules above. The basic idea to identify a safe partition in polynomial time is to greedily compute a maximal packing of arc-disjoint conflicts $\mathcal{C}$, called *conflict packing*. Then, by maximality, removing from T the vertices $V(\mathcal{C})$ covered by $\mathcal{C}$ yields a transitive subtournament $T' = T[V \setminus V(\mathcal{C})]$. We use the transitive ordering σ' of T' to define an ordering σ on V. We prove that if $V \setminus V(\mathcal{C})$ is large enough (with respect to parameter k) then a safe partition $\mathcal{P}$ of T_σ can be identified. By rule 2, the set F of backward arcs of $T_\sigma[A_O]$ can thus be reversed. We first give a bound on the number of vertices covered by a conflict packing $\mathcal{C}$. An instance of k-FAST is *positive* if there exists a set of at most k arcs whose reversal lead to a transitive tournament.

Lemma 2. *Let $T = (V, A)$ be a positive instance of k-FAST and $\mathcal{C}$ be a conflict packing of T. Then $|V(\mathcal{C})| \leq 3k$.*

Lemma 3 (Conflict Packing). *Let $T = (V, A)$ be an instance of k-FAST and $\mathcal{C}$ a conflict packing of T. There exists an ordering of T whose backward arcs uv are such that $u, v \in V(\mathcal{C})$.*

Theorem 1. *The k-FAST problem admits a kernel with at most $4k$ vertices that can be computed in polynomial time.*

Proof. Let $T = (V, A)$ be a positive instance of k-FAST reduced under Rule 1. We greedily (hence in polynomial time) compute a conflict packing $\mathcal{C}$ of T and let σ be an ordering of V obtained through Lemma 3. Consider the bipartite graph $B = (I \cup G, E)$ where (i) $G = V \setminus V(\mathcal{C})$, (ii) there is a vertex i_{vu} in I for every backward arc vu of T_σ and (iii) $i_{vu}w \in E$ if $w \in G$ and $\{u, v, w\}$ is a certificate of vu. Observe that any matching in B of size at least $k + 1$ would correspond to a conflict packing (*i.e.* a collection of arc-disjoint conflicts) of size at least $k+1$, which cannot be. Hence a minimum vertex cover D of B has size at most k [8]. We denote $D_1 = D \cap I$ and $D_2 = D \cap G$. Assume that $|V| > 4k$. Then since $|D_2| \leqslant k$ and $|V(\mathcal{C})| \leqslant 3k$ (by Lemma 2), $G \setminus D_2 \neq \emptyset$. Let $\mathcal{P} = \{V_1, \ldots, V_l\}$ be the ordered partition of T_σ such that every part V_i consists of either a vertex of $G \setminus D_2$ or a maximal subset of consecutive vertices (in σ) of $V \setminus (G \setminus D_2)$.

Claim 1. *$\mathcal{P}$ is a safe partition of T_σ.*

Proof. Let w be a vertex of $G \setminus D_2$. By Lemma 3, w is not incident to any backward arc. As T is reduced under Rule 1, there must exist a backward arc $e = vu$ such that $w \in span(e)$. It follows that A_O contains at least one backward arc. Let $e = vu \in A_O$ be a backward arc of σ. By construction of $\mathcal{P}$, there exists a vertex $w \in (G \setminus D_2) \cap span(e)$. Then $\{u, v, w\}$ is a certificate of e and $i_{vu}w$ is an edge of B. Observe that as D is a vertex cover and $w \notin D_2$, the vertex i_{vu} has to belong to D_1 to cover the edge $i_{vu}w$. Thereby the subset $I' \subseteq I$ corresponding to the backward arcs of A_O is included in D_1. Finally, we argue that I' can be matched into $G \setminus D_2$ in B. Assume there exists $I'' \subseteq I'$ such that $|I''| > |N(I'') \cap (G \setminus D_2)|$. As there is no edge in B between $I \setminus D_1$ and $G \setminus D_2$ (D is a vertex cover of B), the set $D' = (D \setminus I'') \cup (N(I'') \cap (G \setminus D_2))$ is a vertex cover of B and $|D'| < |D|$: contradicting the minimality of D. Thereby for every subset $I'' \subseteq I'$, we have $|I''| \leqslant |N(I'') \cap (G \setminus D_2)|$. By Hall's theorem [17], I' can be matched into $G \setminus D_2$. As every vertex of $G \setminus D_2$ is a singleton in $\mathcal{P}$, the existence of the matching shows that the backward arcs of A_O can be certified using arcs of A_O only, and hence $\mathcal{P}$ is safe. ◇

Hence if $|V| > 4k$, there exists a safe partition that can be computed in polynomial time, and we can reduce the tournament using Rule 2. We then apply Rule 1 and repeat the previous steps until we either do not find a safe partition or $k < 0$. In the former case we know that $|V| \leqslant 4k$; in the latter case, we return a small trivial NO-instance. This concludes the proof. □

3 Linear Vertex-Kernel for k-DENSE RTI

The kernelization algorithm for k-DENSE RTI follows the same lines than the kernelization algorithm for k-FAST. It involves two rules: the first removes *irrelevant* leaves and the second deals with a *safe partition* of the instance. The first rule was already used to obtain a quadratic kernel in [16]. As an instance of k-DENSE RTI is constituted of triplets that choose one vertex (observe that

an instance of k-FAST can be seen as couples that choose one vertex), we have to adapt the notions of *conflict*, *certificate* and *safe partition*. To prove our safe partition rule, we use again the Conflict Packing technique.

Preliminaries. A *rooted triplet* t is a rooted binary tree on a set of three leaves $V(t) = \{a, b, c\}$. We write $t = ab|c$ if a and b are siblings of a child of the root of t, the other child of the root being c. We also say that t *chooses* c. An instance of k-DENSE RTI is a pair $R = (V, \mathcal{R})$, where $\mathcal{R}$ is a set of rooted triplets on V. We only consider *dense* instances, that is $\mathcal{R}$ contains exactly one rooted triplet for every triple of V. For a subset $S \subseteq V$, we define $\mathcal{R}[S] = \{t \in \mathcal{R} \mid V(t) \subseteq S\}$ and $R[S] = (S, \mathcal{R}[S])$. A rooted binary tree is defined *over* a set V if the elements of V are in one-to-one correspondence with the leaves of T. Hereafter the elements of V are called *leaves* and the term *nodes* stands for internal nodes of T. By $T_{|S}$, with $S \subseteq V$, we denote the rooted binary tree over S which is homeomorphic to the subtree of T spanning the leaves of S. Let $t \in \mathcal{R}$ be a rooted triplet and T be a tree over V. Then t is *consistent with* T if $T_{|V(t)} = t$, and *inconsistent* otherwise. A set of rooted triplets $\mathcal{R}$ is *consistent* if there exists a rooted binary tree T over V such that every $t \in \mathcal{R}$ is consistent with respect to T. If such a tree does not exist, then $\mathcal{R}$ is *inconsistent*. A *conflict* C is a subset of V such that $\mathcal{R}[C]$ is inconsistent. If a dense set of rooted triplets $\mathcal{R}$ is consistent, then there exists a unique binary tree T in which every rooted triplet of $\mathcal{R}$ is consistent.

Lemma 4 ([16]). *Let $R = (V, \mathcal{R})$ be an instance of k-DENSE RTI. The following properties are equivalent: (i) $\mathcal{R}$ is consistent; (ii) $\mathcal{R}$ contains no conflict on four leaves; (iii) $\mathcal{R}$ contains no conflict of the form $\{ab|c, cd|b, bd|a\}$ or $\{ab|c, cd|b, ad|b\}$.*

It follows that, as in k-FAST where it was enough to consider directed triangles as conflict, we can restrict our attention to conflicts on sets of four leaves. Hereafter, the term of *conflict* is only used on sets of four leaves. Our kernelization algorithm uses the following rule from [16].

Rule 3. *Remove any leaf $v \in V$ that does not belong to any conflict.*

Certificate. An *embedded* instance of k-DENSE RTI is a triple $R_T = (V, \mathcal{R}, T)$ such that $\mathcal{R}$ is a dense set of rooted triplets on V and T is a rooted binary tree over V. When dealing with an embedded instance R_T, the inconsistency of a rooted triplet always refers to the tree T. If x is a node of T, then T_x denotes the subtree of T rooted in x. Given three leaves $\{a, b, c\}$, we define $span(t)$ as the set of leaves of V contained in $T_{lca(\{a,b,c\})}$, where lca stands for *least common ancestor*. Moreover, given $S \subseteq V$ we define $R_T[S] = (S, \mathcal{R}[S], T_{|S})$. Finally, *editing* an inconsistent rooted triplet $t = ab|c$ *w.r.t.* T means replacing t with the rooted triplet on $\{a, b, c\}$ consistent w.r.t. T. As mentioned earlier, our kernelization algorithm only uses conflicts on sets of four leaves. The following lemma describes more precisely the topology of such conflicts.

Lemma 5. *Let $R_T = (V, \mathcal{R}, T)$ be an embedded instance of k-DENSE RTI. Let $\{a, b, c, d\}$ be a set of leaves such that $t = bc|a$ is the only inconsistent rooted triplet of $R_T[\{a, b, c, d\}]$. Then $\{a, b, c, d\}$ is a conflict if and only if $d \in span(t)$.*

In the following, given an embedded instance $R_T = (V, \mathcal{R}, T)$ of k-DENSE RTI, a conflict containing exactly one rooted triplet inconsistent with T is called a *simple conflict*. We now formally define the notion of *certificate* for an embedded instance R_T. Let t be a rooted triplet inconsistent with T. If $d \in span(t)$ does not belong to any inconsistent rooted triplet then $c(t) = V(t) \cup \{d\}$ is a *certificate* of t. Observe that $c(t)$ induces a simple conflict. By convention, when speaking of a rooted triplet t of a certificate c we mean that t belongs to $R[c]$. If $\mathcal{F} \subseteq \mathcal{R}$ is a set of rooted triplets inconsistent with T, we can *certify* $\mathcal{F}$ whenever it is possible to find a set $c(\mathcal{F}) = \{c(t) : t \in \mathcal{F}\}$ of *triplet-disjoint* certificates (i.e. for every distinct t and t', $|c(t) \cap c(t')| \leqslant 2$).

Safe partition reduction rule. Let $R_T = (V, \mathcal{R}, T)$ be an embedded instance of k-DENSE RTI. We say that $\mathcal{P} = \{T_1, \ldots, T_l\}$ is a *tree partition* of V if there exist l nodes and leaves $x_1, \ldots, x_l$ of T such that: (i) for every $i \in [l]$ $T_i = T_{x_i}$ and (ii) the set of leaves in $\cup_{i=1}^{l} T_{x_i}$ partition V. A tree partition of R_T naturally distinguishes two sets of rooted triplets: $\mathcal{R}_I = \{t \in \mathcal{R} \mid \exists i \in [l]\ V(t) \subseteq V(T_i)\}$ and $\mathcal{R}_O = \mathcal{R} \setminus \mathcal{R}_I$. Let us call a rooted triplet of $\mathcal{R}_O$, an *outer triplet*.

Definition 1. *Let $R_T = (V, \mathcal{R}, T)$ be an embedded instance of k-DENSE RTI and $\mathcal{P} = \{T_1, \ldots, T_l\}$ a tree partition of R_T such that $\mathcal{R}_O$ contains at least one triplet inconsistent with T. Then $\mathcal{P}$ is a* safe partition *if it is possible to certify the rooted triplets of $\mathcal{R}_O$ inconsistent with T only with rooted triplets of $\mathcal{R}_O$.*

We show that it is possible to reduce any embedded instance which has a safe partition. Rouhgly speaking, we prove that the editions in $\mathcal{R}_O$ are independent from the editions in $\mathcal{R}_I$.

Rule 4 (safe partition). *Let $R_T = (V, \mathcal{R}, T)$ be an embedded instance of k-DENSE RTI and $\mathcal{P}$ be a safe partition of T with $\mathcal{F}$ the set of rooted triplets of $\mathcal{R}_O$ inconsistent with T. Edit every rooted triplet $t \in \mathcal{F}$ w.r.t. T and decrease k by $|\mathcal{F}|$.*

Lemma 6. *The safe partition rule (Rule 4) is sound.*

Conflict packing. The remaining problem is now to either compute in polynomial time a safe partition if one exists or bound the size of the instance with respect to k. To that end, we use the conflict packing technique as for k-FAST. Observe that the aim of a conflict packing is to provide a lower bound on the number of editions required to obtain a consistent instance. In the context of k-DENSE RTI, two conflicts may share a rooted triplet t but still require two distinct editions. To see this, let $\{a, b, c, d, e\}$ be a set of leaves and consider the following conflicts: $C = \{ab|c, ac|d, ad|b, cd|b\}$ and $C' = \{ed|c, ed|b, bc|e, bd|c\}$. Observe first that C remains a conflict for any choice of $\{b, c, d\}$. Since C and C' only have this rooted triplet in common, no edition on C' can solve C. Hence (at least) two distinct editions are require to solve both C and C'. Due to this remark, we refine our definition of conflict packing as follows. A leaf a belonging to a conflict $\{a, b, c, d\}$ is a *seed* if $\{a, b, c, d\}$ is a conflict for any choice of $\{b, c, d\}$. A *conflict packing* is a maximal sequence of conflicts $\mathcal{C} = \{C_1, C_2, \ldots, C_l\}$ such that for every $2 \leq i \leq l$:

– Either C_i intersects $\cup_{1 \leqslant j<i} C_j$ on at most two leaves or,
– C_i has a unique leaf not belonging to $\cup_{1 \leqslant j<i} C_j$, which is a seed of C_i.

Lemma 7. *Let $R = (V, \mathcal{R})$ be a positive instance of k-*DENSE RTI *and $\mathcal{C}$ be a conflict packing of R. Then $l \leq k$ and $|V(\mathcal{C})| \leq 4k$.*

So let us now consider a conflict packing $\mathcal{C}$ (which can be computed greedily in polynomial time). Observe that $R[V \setminus V(\mathcal{C})]$ is consistent by maximality of $\mathcal{C}$. This means that there exists a unique tree T such that every rooted triplet of $\mathcal{R}[V \setminus V(\mathcal{C})]$ is consistent with T. We will use that tree T to embed R and compute a safe partition.

Lemma 8 (Conflict packing). *Let $R = (V, \mathcal{R})$ be an instance of k-*DENSE RTI *and $\mathcal{C}$ a conflict packing of R. There exists an embedded tree T of R such that every rooted triplet t inconsistent with T is such that $V(t) \subseteq V(\mathcal{C})$.*

Proof. The leaves of $G = V \setminus V(\mathcal{C})$ are called *good* leaves. As already observed $R[G]$ is consistent with a unique tree T'. Notice that for every leaf $a \in V(\mathcal{C})$, $R_a = R[G \cup \{a\}]$ is also consistent (otherwise $\mathcal{C}$ would not be maximal). Thereby there exists a unique binary tree T_a such that every rooted triplet t of R_a is consistent with T_a. In other words T' contains a unique tree edge $e = xz$ which can be subdivided into xyz to attach the leaf a to node y. Hereafter the edge e will be called the *locus* of a. The maximality argument on $\mathcal{C}$ also implies that for any pair of leaves a and b in $V(\mathcal{C})$, $R_{ab} = R[G \cup \{a, b\}]$ is consistent. If a and b have different loci, then R_{ab} is clearly consistent with the tree T obtained from T' by inserting a and b in their respective loci. It remains to consider the case where a and b have the same locus. Let $e = xy$ be a tree edge of T' such that x is the child of y and let $B_e \subseteq V(\mathcal{C})$ be the subset of leaves whose locus is e. Given $a, b \in B_e$, we define the following binary relations $<_e$ and $\sim_e$ on B_e as follows: $a <_e b$ if there exists $c \in G$ such that $ac|b \in \mathcal{R}$ and $a \sim_e b$ if neither $a <_e b$ nor $b <_e a$. Using the maximality of $\mathcal{C}$ we will prove that $<_e$ is a strict weak ordering (i.e. $<_e$ is a transitive and asymmetric relation and $\sim_e$ is transitive). This implies that the equivalence classes of $\sim_e$ partition the leaves of B_e and are totally ordered by $<_e$.

Claim 2. *The relation $<_e$ is a strict weak ordering.*

We can now describe how the tree T is build from T'. For every tree edge $e = xy$ with x a child of y such that $B_e \neq \emptyset$ we proceed as follows. Let $B_1 \dots B_q$ be the equivalence classes of $\sim_e$ such that $B_i <_e B_j$ for $1 \leqslant i < j \leqslant q$. The tree edge e is subdivided into the path $x, z_1 \dots, z_q, y$. For every $i \in [q]$, if B_i contains a unique leaf a, then a is attached to node z_i. Otherwise, a new node w_i is attached to z_i and we add an arbitrary binary tree (rooted in w_i) over the leaves of B_i. We now prove that T has the desired property. Let $t = \{a, b, c\}$ be any triplet of $\mathcal{R}$, and assume first that $V(t) \subseteq G$. Then t is consistent by construction. Next, assume w.l.o.g. that $V(t) \cap V(\mathcal{C}) = \{a\}$: then t is consistent with T since R_a is consistent and a has been inserted to its locus. Finally, assume $V(t) \cap V(\mathcal{C}) = \{a, b\}$. If a and b have different loci then t is clearly consistent with T. Now, if a and b

have the same locus e then t is consistent since a and b have been added to e according to the strict weak ordering $<_e$. It follows that any triplet of T such that $V(t) \cap G \neq \emptyset$ is consistent with T. □

We now state the main result of this section. Its proof follows the same lines than the proof of Theorem 1. In particular, the safe partition is built using the same bipartite graph and matching arguments.

Theorem 2. *The k-DENSE RTI problem admits a kernel with at most $5k$ vertices.*

4 Linear Vertex-Kernel for k-BIT

Preliminaries. A *betweenness triplet* t defined over a set of three vertices $\{a, b, c\}$ *chooses* one of its vertices. We write $t = abc$ to illustrate that t chooses b. An instance of k-BIT is a pair $B = (V, \mathcal{R})$ where $\mathcal{R}$ is a set of betweenness triplets defined over V. We only consider *dense* instances, that is $\mathcal{R}$ contains exactly one triplet for every triple of V. Let $t = abc \in \mathcal{R}$ be a betweenness triplet (or triplet for short) and σ be an ordering on V. Then t is *consistent* with σ if $a <_\sigma b <_\sigma c$ or $c <_\sigma b <_\sigma a$. A set of triplets $\mathcal{R}$ is *consistent* if there exists an ordering σ on V such that every $t \in \mathcal{R}$ is consistent with σ. If such an ordering does not exist, then $\mathcal{R}$ is *inconsistent*. A conflict C is a subset of V such that $\mathcal{R}[C]$ is inconsistent. Given an instance $B = (V, \mathcal{R})$, an *edition* for a triplet $t \in \mathcal{R}$ is a modification of its choosen vertex. A set $\mathcal{F}$ of edited triplets of $\mathcal{R}$ is an *edition* for B if the edition of every $t \in \mathcal{R}$ leads to a consistent instance. We use $B_\sigma = (V, \mathcal{R}, \sigma)$ to denote an instance of k-BIT fixed under some ordering σ. Given an ordered instance, we use the same notations than in Section 3 for embedded instances.

Lemma 9. *Let $B = (V, \mathcal{R})$ be an instance of k-BIT. Then B is consistent if and only if B does not contain any conflict on 4 vertices.*

Sunflower reduction rule. The main difference with the previous section lies in the definition of *simple conflict* for an ordered instance $B_\sigma = (V, \mathcal{R}, \sigma)$: given an inconsistent triplet t and a vertex d that does not belong to any inconsistent triplet, we do not need to require that d belongs to $span(t)$ to obtain a conflict. As indicated by Lemma 11, *any* such vertex can be used to form a conflict with $V(t)$. In particular, this result allows us to replace the *Safe Partition* rule with a *Sunflower*-based reduction rule. A *sunflower* $\mathcal{S}$ is a set of conflicts $\{C_1, \ldots, C_m\}$ pairwise intersecting in exactly one triplet t. We say that t is the *centre* of $\mathcal{S}$.

Lemma 10 (Sunflower Lemma). *Let $B = (V, \mathcal{R})$ be an instance of k-BIT. Let $\mathcal{S} = \{C_1, \ldots, C_m\}$, $m > k$ be a sunflower of centre t. Any edition of size at most k has to edit t.*

Observe that there exist two ways to edit the centre t. By setting $m > 2k$ we can fix this and obtain a quadratic vertex-kernel for k-BIT (by adapting techniques from [16]). Nevertheless, this is not enough to obtain a *linear* vertex kernel. To that aim, we combine Conflict Packing and a sunflower rule on *simple conflicts*.

Lemma 11. *Let $B_\sigma = (V, \mathcal{R}, \sigma)$ be an ordered instance of k-BIT. Let $\{a, b, c, d\}$ be a set of vertices such that $bca \in \mathcal{R}$ is the only inconsistent triplet of $B_\sigma[\{a, b, c, d\}]$. Then $\{a, b, c, d\}$ is a conflict.*

Given an ordered instance $B_\sigma = (V, \mathcal{R}, \sigma)$, a triplet $t = \{a, b, c\}$ inconsistent with σ and a vertex d that does not belong to any inconsistent triplet, the set $V(t) \cup \{d\}$ is called a *simple conflict*. A sunflower $\mathcal{S} = \{C_1, \ldots, C_m\}$ of B_σ is *simple* if the C_i's are simple conflicts and if the centre of $\mathcal{S}$ is the only triplet inconsistent with σ for every C_i, $1 \leqslant i \leqslant m$. The soundness of the Simple sunflower rule follows from Lemmas 10 and 11.

Rule 5 (Simple sunflower). *Let $B_\sigma = (V, \mathcal{R}, \sigma)$ be an ordered instance of k-BIT. Let $\mathcal{S} = \{C_1, \ldots, C_m\}$, $m > k$ be a simple sunflower of centre t. Edit t w.r.t. σ and decrease k by 1.*

Conflict Packing. Using Conflict Packing and the Simple sunflower rule, we prove that the k-BIT problem can be solved in polynomial time on instances whose parameter is such that $k < (|V|/5)$. The existence of this algorithm directly implies the existence of a linear vertex-kernel for k-BIT, as stated in Corollary 1. The definition of a conflict packing $\mathcal{C}$, involving the notion of *seed*, is the same than the one given in Section 3. As in Section 3, given a positive instance of k-BIT we have $|V(\mathcal{C})| \leqslant 4k$. Given a conflict packing $\mathcal{C}$, we can compute in polynomial time an ordering σ such that any triplet t inconsistent with σ verifies $V(t) \subseteq V(\mathcal{C})$. Providing that $V \setminus V(\mathcal{C})$ is large enough (w.r.t. parameter k), we prove how to compute a simple sunflower on $B_\sigma = (V, \mathcal{R}, \sigma)$.

Theorem 3. *Let $B = (V, \mathcal{R})$ be an instance of k-BIT such that $k < (|V|/5)$. There exists an algorithm that either computes an edition of size at most k or answers* No *in polynomial time.*

Corollary 1. *The k-BIT problem admits a kernel with at most $5k$ vertices.*

References

1. Aho, A., Sagiv, Y., Szymansk, T., Ullman, J.: Inferring a tree from lowest common ancestor with an application to the optimization of relational expressions. SIAM Journal on Computing 10(3), 405–421 (1981)
2. Ailon, N., Alon, N.: Hardness of fully dense problems. Inf. Comput. 205(8), 1117–1129 (2007)
3. Alon, N.: Ranking tournaments. SIAM J. Discrete Math. 20(1), 137–142 (2006)
4. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 49–58. Springer, Heidelberg (2009)
5. Bandelt, H.-J., Dress, A.: Reconstructing the shape of a tree from observed dissimilarity data. Advances in Applied Mathematics 7, 309–343 (1986)
6. Bessy, S., Fomin, F.V., Gaspers, S., Paul, C., Perez, A., Saurabh, S., Thomassé, S.: Kernels for feedback arc set in tournaments. In: FSTTCS 2009. LIPIcs, vol. 4, pp. 37–47 (2009)

7. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
8. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland, Amsterdam (1976)
9. Brügmann, D., Komusiewicz, C., Moser, H.: On generating triangle-free graphs. Electronic Notes in Discrete Mathematics 32, 51–58 (2009)
10. Byrka, J., Guillemot, S., Jansson, J.: New results on optimizing rooted triplets consistency. Discrete Applied Mathematics 158(11), 1136–1147 (2010)
11. Charbit, P., Thomassé, S., Yeo, A.: The minimum feedback arc set problem is NP-hard for tournaments. Combin. Probab. Comput. 16(1), 1–4 (2007)
12. Dom, M., Guo, J., Hüffner, F., Niedermeier, R., Truß, A.: Fixed-parameter tractability results for feedback set problems in tournaments. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 320–331. Springer, Heidelberg (2006)
13. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
14. Erdös, P., Moon, J.W.: On sets on consistent arcs in tournaments. Canad. Math. Bull. 8, 269–271 (1965)
15. Guillemot, S., Berry, V.: Fixed-parameter tractability of the maximum agreement supertree problem. IEEE/ACM Trans. Comput. Biology Bioinform. 7(2) (2010)
16. Guillemot, S., Mnich, M.: Kernel and fast algorithm for dense triplet inconsistency. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 247–257. Springer, Heidelberg (2010)
17. Hall, P.: On representatives of subsets. J. London Math. Soc. 10(37), 26–30 (1935)
18. Karpinski, M., Schudy, W.: Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 3–14. Springer, Heidelberg (2010)
19. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: STOC 2007, pp. 95–103 (2007)
20. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. Theor. Comput. Sci. 351(3), 446–458 (2006)
21. Reid, K.D., Parker, E.T.: Disproof of a conjecture of Erdös and Moser on tournaments. J. Combin. Theory 9, 225–238 (1970)
22. Saurabh, S.: Chromatic coding and universal (hyper-)graph coloring families. Parameterized Complexity News, 49–58 (June 2009)
23. Semple, C., Steel, M.: Phylogenetics. Oxford Lecture Series in Mathematics and Its Applications, vol. 24. Oxford University Press, Oxford (2003)
24. Wang, J., Ning, D., Feng, Q., Chen, J.: An improved kernelization for P_2-packing. Inf. Process. Lett. 110(5), 188–192 (2010)

Transduction on Kadanoff Sand Pile Model Avalanches, Application to Wave Pattern Emergence*

Kevin Perrot and Eric Rémila

Université de Lyon
LIP (UMR 5668 CNRS-ENS de Lyon, Université Lyon 1),
46 allée d'Italie 69364 Lyon Cedex 7, France
{kevin.perrot,eric.remila}@ens-lyon.fr

Abstract. Sand pile models are dynamical systems describing the evolution from N stacked grains to a stable configuration. It uses local rules to depict grain moves and iterate it until reaching a fixed configuration from which no rule can be applied. The main interest of sand piles relies in their *Self Organized Criticality* (SOC), the property that a small perturbation — adding some sand grains — on a fixed configuration has uncontrolled consequences on the system, involving an arbitrary number of grain fall. Physicists L. Kadanoff *et al* inspire KSPM, a model presenting a sharp SOC behavior, extending the well known *Sand Pile Model*. In KSPM(D), we start from a pile of N stacked grains and apply the rule: $D-1$ grains can fall from column i onto the $D-1$ adjacent columns to the right if the difference of height between columns i and $i+1$ is greater or equal to D. This paper develops a formal background for the study of KSPM fixed points. This background, resumed in a finite state word transducer, is used to provide a plain formula for fixed points of KSPM(3).

Keywords: Discrete dynamical system, self-organized criticality, sand pile model, transducer.

1 Introduction

Sand pile models were introduced in [1] as systems presenting a critical self-organized behavior, a property of dynamical systems having critical points as attractors. In the scope of sand piles, starting from an initial configuration of N stacked grains the local evolution of particles is described by one or more iteration rules. Successive applications of such rules alter the configuration until it reaches an attractor, namely a stable state from which no rule can be applied. SOC property means those attractors are critical in the sense that a small perturbation — adding some more grains — involves an arbitrary deep reorganization of the system. Sand pile models were well studied in recent years ([8],[4],[5],[16]).

* Partially supported by IXXI (Complex System Institute, Lyon) and ANR projects Subtile and MODMAD.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 508–519, 2011.

In [11], Kadanoff proposed a generalization of classical models closer to physical behavior of sand piles in which more than one grain can fall from a column during one iteration. Informally, Kadanoff sand pile model with parameter D and N grains is a discrete dynamical system, which initial configuration is composed of N stacked grains, moving in discrete space and time according to a transition rule : if the height difference between column i and $i+1$ is greater or equal to D, then $D-1$ grains can fall from column i to the $D-1$ adjacent columns on the right (see figure 1).

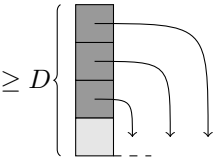


Fig. 1. KSPM(D) transition rule

In [10], the authors show that the set of reachable configurations endowed with the order induced by the successor relation has a lattice structure, in particular it has a unique *fixed point*.

More formally, sand pile models we consider are defined on the space of ultimately null decreasing integer sequences. Each integer represents a column of stacked sand grains and transition rules describe how grains can move from columns. Let $h = (h_0, h_1, h_2, \dots)$ denote a *configuration* of the model, where each integer h_i is the number of grains on column i. Configurations can also be given as height difference $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$, where for all $i \geq 0$, $\sigma_i = h_i - h_{i+1}$. We will use this latter representation throughout the paper, within the space of ultimately null non-negative integer sequences.

Definition 1. *The Kadanoff sand pile model with parameter D, KSPM(D), is defined by:*

- *A set of* configurations*, consisting in ultimately null non-negative integer sequences.*
- *A set of* transition rules *: we have a transition from a configuration σ to a configuration σ' on column i, and we note $\sigma \xrightarrow{i} \sigma'$ if*
  - $\sigma'_{i-1} = \sigma_{i-1} + D - 1$ *(for $i \neq 0$)*
  - $\sigma'_i = \sigma_i - D$,
  - $\sigma'_{i+D-1} = \sigma_{i+D-1} + 1$
  - $\sigma'_j = \sigma_j$ *for* $j \notin \{i-1, i, i+D-1\}$.

Remark that according to the definition of the transition rules, a condition for σ' to be a configuration is that $\sigma_i \geq D$. We note $\sigma \to \sigma'$ when there exists an integer i such that $\sigma \xrightarrow{i} \sigma'$. The transitive closure of $\to$ is denoted by $\xrightarrow{*}$. A *strategy* is a sequence of nonnegative integers $s = (s_1, \dots, s_T)$. We say that σ' is *reached* from σ via s when $\sigma \xrightarrow{s_1} \sigma'' \xrightarrow{s_2} \dots \xrightarrow{s_T} \sigma'$, and we note $\sigma \xrightarrow{s} \sigma'$. We also say, for each integer t such that $0 < t \leq T$, that the column s_t *is fired* at *time* t in s (informally, the index of the sequence is interpreted as time). For any strategy s and any nonnegative integer i, we state $|s|_i = \#\{t | s_t = i\}$. Let s^0, s^1 be two strategies such that $\sigma \xrightarrow{s^0} \sigma^0$ and $\sigma \xrightarrow{s^1} \sigma^1$. We have the equivalence: $[\forall\, i, |s^0|_i = |s^1|_i] \Leftrightarrow \sigma^0 = \sigma^1$.

We say that a configuration σ is *stable*, or a *fixed point* if no transition is possible from σ. A basic property of the KSPM model is the *diamond property.*

If there exist two distinct integers i and j such that $\sigma \xrightarrow{i} \sigma'$ and $\sigma \xrightarrow{j} \sigma''$, then there exists a configuration σ''' such that $\sigma' \xrightarrow{j} \sigma'''$ and $\sigma'' \xrightarrow{i} \sigma'''$. From this property, one can easily check that, for each configuration σ, there exists a unique stable configuration, denoted by $\pi(\sigma)$, such that $\sigma \xrightarrow{*} \pi(\sigma)$. Moreover, for any configuration σ' such that $\sigma \xrightarrow{*} \sigma'$, we have the *convergence property*: $\pi(\sigma') = \pi(\sigma)$ (see [10] for details).

Let N be a fixed integer. This paper is devoted to the structure of $\pi((N, 0^\omega))$ (where 0^ω stands for an infinity of 0's. $\pi((N, 0^\omega))$ is denoted $\pi(N)$, for simplification). Informally, our goal is to know what finally happens, starting from a configuration where all grains are in the first column.

The main interest of our approach is to provide a new tool to study fixed points: a *deterministic finite state word transducer*. (a transducer is essentially a finite state automaton, which outputs a word during each transition). The idea is the following, we concentrate on a set I of $D-1$ consecutive columns and constructs a set of states, according to the possible values of configurations on I. We then study how the way grains fall from the left part into I (input word) is related to the way grains fall from I to the right part (output word). The word transducer is formally defined in section 2.3. Using this transducer, an application to the case $D = 3$ is also proposed in section 2.3. This application leads to:

Theorem 1. *In KSPM(3), there exists a column number $i(N)$ in $O(\log N)$ such that:*

$$\pi(N)_{[i(N),\infty[} = (2,1)^*[0](2,1)^*0^\omega$$

where $\sigma_{[i,j]}$ is the subsequence $(\sigma_i, \ldots, \sigma_j)$, and $[0]$ stands for at most one 0.

This result can be interpreted as a kind of spatial emergence in a complex system. On a short length, the structure of the sand pile is complex, but a very regular shape is issued from the complexity.

Describing and proving regularity properties, for models issued from basic dynamics is a present challenge for physicists, mathematicians, and computer scientists. there exists a lot of conjectures, issued from simulations, on discrete dynamical systems with simple local rules (sandpile model [3] or chip firing games, but also rotor router [12], the famous Langton ant [6] [7]...) but very few results have actually been proved.

As regards KSPM(D), the *prediction problem* (namely, the problem of computing the fixed point) has been proven in [13] to be in $\mathbf{NC}^3$ for the one dimensional case (the model of our purpose), which means that the time needed to compute the fixed point is in $O(\log^3 N)$ where N is the number of grains, and **P**-complete when the dimension is ≥ 3. In this paper we give a straightforward characterization, describing asymptotically completely fixed points. A recent study ([9]) showed that in the two dimensional case the avalanche problem (given a configuration σ and a column i on which we add one grain, does it have an influence on index j?) is **P**-complete, which points out an inherently sequential behavior.

2 Avalanches and Transducer

2.1 Previous Results about Avalanches

Let σ be a configuration, $\sigma^{\downarrow 0}$ is the configuration obtained by adding one grain on column 0. In other words, if $\sigma = (\sigma_0, \sigma_1, \dots)$, then $\sigma^{\downarrow 0} = (\sigma_0+1, \sigma_1, \dots)$. Clearly, for any strategy s, if $\sigma \xrightarrow{s} \sigma'$, then we have $\sigma^{\downarrow 0} \xrightarrow{s} \sigma'^{\downarrow 0}$. In particular, since we have: $(N, 0^\omega) \xrightarrow{*} \pi(N)$, we get: $(N, 0^\omega)^{\downarrow 0} \xrightarrow{*} \pi(N)^{\downarrow 0}$, i.e. $(N+1, 0^\omega) \xrightarrow{*} \pi(N)^{\downarrow 0}$. The model is convergent, therefore we get the inductive formula:

$$\pi(\pi(N)^{\downarrow 0}) = \pi(N+1).$$

In the following, we will use the inductive approach described above, which consists in computing $\pi(N+1)$ by first computing $\pi(N)$, then adding a grain on the origin column, and process all possible transitions until a fixed point is reached. For initialization, $\pi(0) = 0^\omega$.

The convergence property also allows to only consider leftmost strategies. A strategy s such that $\sigma \xrightarrow{s} \sigma'$ is called *leftmost* if it is the minimal strategy from σ to σ' according to lexicographic order. A leftmost strategy is such that at each iteration, the leftmost possible transition is performed. The k^{th} *avalanche* s^k is the leftmost strategy from $\pi(k-1)^{\downarrow 0}$ to $\pi(k)$. For our iterative approach, we need to describe avalanches. In a previous work [14], we provide a simplified description of an avalanche. This description is a core result toward the construction of the transducer in section 2.3. A first insight shows that any column is fired at most once within an avalanche. The formal statement of the avalanche description requires one more definition, even though it is graphically simple. For any sequence $x = (x_1, \dots, x_n)$ and any integers i, j with $1 \le i \le j \le n$, we denote $x_{[i,j]} = (x_i, \dots, x_j)$.

Given an avalanche $s^k = (s^k_1, \dots, s^k_T)$, a column s^k_t is a *peak* if and only if $s^k_t > \max s^k_{[1,t-1]}$. Intuitively, peaks are columns where an avalanche progresses rightward.

Proposition 1. *[14] Let $s = (s_1, \dots, s_T)$ be the k^{th} avalanche and $(p_1 \dots, p_q)$ be its sequence of peaks. Assume that there exists a column l, such that for each column i with $l \le i < l + D - 1$, $i \in s$. Then for any column p such that $p \ge l + D - 1$,*

$$p \text{ is a peak of } s \iff \pi(k-1)_p = D - 1 \text{ and } \exists i \text{ s.t. } p_i < p \le p_i + D - 1$$

Furthermore, let $p_i = s_t$, with $p_i \ge l + D - 1$, be a peak. Then

$$T \ge t + p_i - p_{i-1} - 1 \text{ and for all } t' \text{ s.t. } t < t' \le t + p_i - p_{i-1} - 1,\ s_{t'} = s_{t'-1} - 1$$

A graphical representation of this statement is given on figure 2.

The theorem and therefore the simplified description applies starting from a certain column l which depends on parameters of the model D and N. We say that the k^{th} avalanche s^k is *dense starting at l and ending at m* when m is the greatest fired column ($\forall\ i > m,\ i \notin s^k$) and any column between l and m

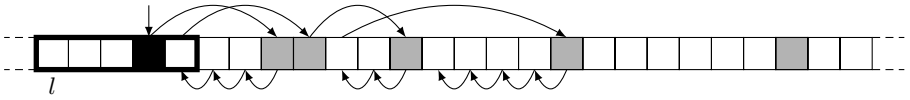


Fig. 2. Illustration of Proposition 1 with $D = 6$. Surrounded columns l to $l + D - 2$ are supposed to be fired. Black column is the greatest peak strictly lower than $\ell + D - 1$ before the avalanche. A column is grey if and only if its value is $D-1$. Following arrows depicts the avalanche.

included has been fired ($\forall\ l \leq i \leq m,\ i \in s^k$). A consequence of Proposition 1 is that the avalanche s^k considered is dense starting at l, where l denotes the parameter in the statement of Proposition 1. We define the *global density column* $L(D, N)$ as the minimal column such that for any avalanche s^k, with $k \leq N$, s^k is dense starting at $L(D, N)$. When parameters D and N are fixed, we sometimes simply denote L.

Proposition 2. *[14] The global density column $L(3, N)$ is in $O(\log N)$.*

In KSPM(3), a trivial bounding of the maximal non empty column of $\pi(N)$ shows that it is in $\Theta(\sqrt{N})$, so proposition 1 describes asymptotically completely avalanches used to construct the fixed point. We come back on this point in section 3.

2.2 Successive Avalanches

When the k^{th} avalanche is dense starting at l and ending at m, for each column i such that $l+D-1 \leq i < m$, columns $i-D+1$, i and $i+1$ are fired within the k^{th} avalanche. Therefore, $\pi(k)_i = \pi(k-1)_i^{\downarrow 0} = \pi(k-1)_i$. Moreover, $\pi(k)_j = \pi(k-1)_j$ for $j > m + D - 1$. An intuitive consequence is that two consecutive avalanches are similar. This intuition is formally stated in this section.

Let s^k denote the k^{th} avalanche of KSPM(D). We recall that the *global density column* $L(D, N)$ of KSPM(D) is the minimal column such that for any avalanche s^k, with $k \leq N$, s^k is dense starting at $L(D, N)$. We also define $\Phi(D, N) = (\phi^1, \ldots, \phi^n)$, the subsequence of $(s^1, \ldots, s^N)$ reaching column $L(D, N) + D - 1$. Formally, $s^k \in \Phi(D, N) \iff L(D, N) + D - 1 \in s^k$. $\Phi(D, N)$ is called the *sequence of long avalanches up to N* of KSPM(D).

We also define the sequence $(\mu^0, \mu^1, ..., \mu^n)$ of configurations such that $\mu^0 = \pi(0) = 0^\omega$, and for each integer k such that $\phi^k = s^m$, we have $\mu^k = \pi(m)$.

The definition of long avalanche is motivated by the property above, which says that the effect of such an avalanche is easy to compute on the right of the global density column.

Remark 1. In KSPM(D), if s^k is a long avalanche, whose sequence of peaks is denoted by P^k (the largest peak being $\max P^k$), from proposition 1 we have:

- $\pi(k)_{\max P^k} = \pi(k-1)_{\max P^k} - D + 1 = 0$,
- $\pi(k)_i = \pi(k-1)_i$ for $L(D, N) + D - 1 \leq i < \max P^k$,

– $\pi(k)_i = \pi(k-1)_i + 1$ for $\max P^k < i \leq \max P^k + D - 1$,
– $\pi(k)_i = \pi(k-1)_i$ for $i > \max P^k + D - 1$.

This result is a clear application of transition rules (for each considered column i we know which columns of the set $\{i - D + 1, i, i + 1\}$ are fired in s^k, so we can update the configuration). In other words, the main element that we need to compute (the right part of) $\pi(k)$ from $\pi(k-1)$ is $\max P^k$.

Lemma 1. *In KSPM(D), let L be the global density column of N, and $\Phi = (\phi^1, \ldots, \phi^n)$ its sequence of long avalanches up to N. Let $k < n$, and P^k (resp. P^{k+1}) be the sequence of peaks i of ϕ^k (resp. ϕ^{k+1}) such that $i \geq L + 2(D-1)$. The largest peak of P^k is denoted by $\max P^k$. We have:*

$$P^k \setminus \{\max P^k\} \quad = \quad P^{k+1} \cap [\![L + 2(D-1), \max P^k[\![$$

The lemma above can be seen as follows: $|P^{k+1}| \geq |P^k| - 1$, and the $|P^k| - 1$ first elements of P^{k+1} and P^k are equal. Informally, the peak sequence can increase in arbitrary manner, but can decrease only peak after peak.

Proof. Let κ, κ' be two integers such that ϕ^k is the κ^{th} avalanche, and ϕ^{k+1} is the κ'^{th} avalanche. For each column i such that $i \in [\![L + D - 1, \max P^k[\![$, we have $i - D + 1, i, i + 1 \in \phi^k$ and therefore $\pi(\kappa)_i = \pi(\kappa - 1)_i$.

By definition of long avalanches, any avalanche s between ϕ^k and ϕ^{k+1} stops before $L + D - 1$, *i.e.* for all $i \geq L + D - 1$, $i \notin s$. Combining it with previous remark, we have for all κ'' such that $\kappa \leq \kappa'' < \kappa'$

$$\text{for all } i \in [\![L + D - 1, L + 2(D-1)[\![\ , \pi(\kappa'')_i \geq \pi(\kappa - 1) \tag{1}$$

$$\text{for all } i \in [\![L + 2(D-1), \max P^k[\![\ , \pi(\kappa'')_i = \pi(\kappa - 1) \tag{2}$$

because columns within interval $[\![L + D - 1, L + 2(D-1)[\![$ can gain height difference when a column within $[\![L, L + D - 1[\![$ is fired. This is in particular true for $\kappa'' = \kappa' - 1$. We now study the κ'^{th} avalanche ϕ^{k+1}. From relation (1) and since $\pi(\kappa' - 1)$ is a fixed point, for all $i \in [\![L + D - 1, L + 2(D-1)[\![$, $\pi(\kappa - 1)_i = D - 1 \Rightarrow \pi(\kappa' - 1)_i = D - 1$. Let Q^k (resp. Q^{k+1}) be the sequence of peaks i of ϕ^k (resp. ϕ^{k+1}) such that $L + D - 1 \leq i < L + 2(D-1)$ Using proposition 1, we therefore get

$$Q^k \quad \subseteq \quad Q^{k+1} \tag{3}$$

Let $I = [\![L + 2(D-1), \max P^k[\![$. From relation (2)

$$\text{for all } i \in I,\ \pi(\kappa - 1)_i = D - 1 \iff \pi(\kappa' - 1)_i = D - 1 \tag{4}$$

We now eventually prove the conclusion of the lemma. Let $p_{\underline{i}} = \min\{i \in P^k\}$, from proposition 1 we equivalently have $p_{\underline{i}} = \min\{i' \in I | \pi(\kappa - 1)_i = D - 1\}$ (the existence of $p_{\underline{i}}$ is a hypothesis of the lemma). Let $p'_{\underline{i'}} = \min\{i' \in P^{k+1}\} = \min\{i' \in I | \pi(\kappa' - 1)_{i'} = D - 1\}$ (the existence of $p'_{\underline{i'}}$ is given by subset relation (3)), using relation (4) we have $p'_{\underline{i'}} = p_{\underline{i}}$.

Other peaks within I are obviously equal from proposition 1 and relation (2) with $\kappa'' = \kappa' - 1$. □

2.3 Transducer

We now exploit the similarity between successive avalanches. Informally, we will cut configurations into intervals $I_1, I_2, \ldots$ of size $D-1$ and study each of them and their interactions when considering an avalanche. Given three successive intervals I_{i-1}, I_i and I_{i+1}, we construct a finite state word transducer which computes the influence of I_i on I_{i+1}, knowing the influence of I_{i-1} on I_i and the value of the configuration in I_i. The main idea to use transducers is that the value of any interval I_i with $i > 0$ in $\pi(0)$ is 0^{D-1}, so we can relate temporally emergent patterns arising from transduction iterations to spatially emergent patterns on stable configurations.

The *interval* I_i is the column sequence $((D-1)i, (D-1)i+1, ..., (D-1)i+D-2))$. We call *state* of an interval I_i of a fixed point π its value $(\pi_{(D-1)i}, \pi_{(D-1)i+1} \cdots, \pi_{(D-1)i+D-2})$. Hence, each interval state is an element of the set $\mathcal{S} = \{0, 1, ..., D-1\}^{D-1}$.

We fix an interval I_i such that $(D-1)i \geq L(N) + 3(D-1)$. The largest peak j of ϕ^k, such that $j < (D-1)i$, is denoted by $p(i,k)$. The *type* $\alpha(i,k)$ of the long avalanche ϕ^k on I_i is defined as follows.

- if $p(i,k) \in I_{i-1}$, then $\alpha(i,k) = p(i,k) \mod [D-1]$;
- if $p(i,k) \notin I_{i-1}$, then $\alpha(i,k) = \epsilon$.

Therefore, the set of possible types is $\mathcal{T} = \{\epsilon, 0, 1, ..., D-2\}$. We say that two long avalanches are *i-similar* if they have the same type for i. Note that if a long avalanche ϕ^k changes the state of I_i, then from remark 1 there necessarily exists a peak of ϕ^k in the interval I_{i-1}.

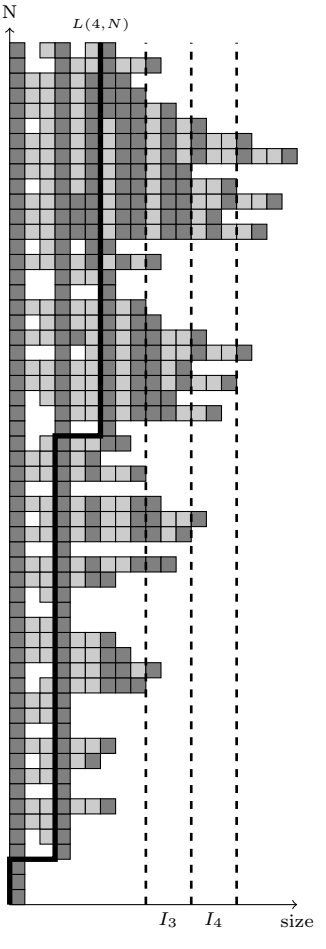


Fig. 3. D=4. Long avalanches up to 500, one by line. The global density column is lined in bold black. A light grey square is a fired column, a dark grey square is a peak. The sequence of types of 4-influent subsequences up to 500 is $0, 1, 2, 0, 1, 2, 0, 2, 1, 0$.

We now divide the sequence Φ of long avalanches up to N into maximal length subsequences of the type $(\phi^k, \phi^{k+1}, ..., \phi^{k''})$ such that, for each integer k' such that $k \leq k' < k''$, $\phi^{k'}$ and $\phi^{k'+1}$ are i-similar. Such a subsequence is called an *i-subsequence*. An i-subsequence is said of type α for i when the type of each avalanche of the subsequence is α. When α is not the empty word ϵ, we say that the subsequence

is *i-influent*. Remark that, from Lemma 1, each $(i+1)$-influent subsequence is contained in an i-influent subsequence. See figure 3 for an example of i-influent subsequence.

Lemma 2. *Let $\Phi_{[k,k'']} = (\phi^k, \phi^{k+1}, ..., \phi^{k''})$ be a subsequence of type α for i, with $k'' \leq n$, and with I_i an interval whose columns are greater than $L+3(D-1)$. Given the state $(a_0, a_1, ..., a_{D-2})$ of I_i in the configuration μ^{k-1}, and α, one can compute, with no need of more knowledge:*

- *the state $(a'_0, a'_1, ..., a'_{D-2})$ of I_i in the configuration $\mu^{k''}$,*
- *the sequence of types of the successive $i+1$-influent subsequences contained in $\Phi_{[k,k'']}$*

Proof. This is obvious when the type of the subsequence is ϵ, since there is no change and the $i+1$-subsequence contained in $(\phi^k, \ldots, \phi^{k''})$ is also ϵ.

The computation is simple when there is no integer m such that $0 \leq m \leq \alpha$ and $a_m = D-1$. In this case, the peak $p(i,k)$ is the last peak of ϕ^k, thus $\mu^k_{p(i,k)} = 0$, which gives that $p(i,k)$ is not a peak of ϕ^{k+1}, thus the subsequence is reduced to a singleton which is not $i+1$-influent (second part of the result). For $(D-1)i \leq j \leq p(i,k)+D-1$, we have $\mu^k_j = \mu^{k-1}_j + 1$, and for $p(i,k)+D-1 < j < (D-1)(i+1)$, we have $\mu^k_j = \mu^{k-1}_j$. Thus, we have $a'_m = a_m + 1$ for $0 \leq m \leq \alpha$ and $a'_m = a_m$ for $\alpha < m \leq D-2$ (first part of the result).

Otherwise, ϕ^k contains a peak in I_i. Let $q(i,k)$ denote the largest one. The column $q(i,k)$ is the largest j such that $\mu^{k-1}_j = D-1$ and $j < (D-1)(i+1)$. Thus $q(i,k) \mod D-1$ is the largest m such that $a_m = D-1$. In this case, ϕ^k starts an $i+1$-subsequence of type $q(i,k)$. Consider the following long avalanches. From Lemma 1, while $q(i,k)$ remains a peak of $\phi^{k'}$, $p(i,k)$ also remains a peak of $\phi^{k'}$. From Remark 1, while $q(i,k)$ is not the last peak of $\phi^{k'}$, the state of I_i remains invariant. So the first avalanche $\phi^{k'}$ that changes the state of I_i is the one whose last peak is $q(i,k)$ (this avalanche exists from the hypothesis: $k'' < n$). We have $\mu^{k'}_{q(i,k)} = 0$, which closes the $i+1$-subsequence of type $q(i,k)$. We also have $\mu^{k'}_j = \mu^k_j + 1$ for $q(i,k) < j < (D-1)(i+1)$, and $\mu^{k'}_j = \mu^k_j$ for $p(i,k) \leq j < q(i,k)$. This gives the state of I_i for $\mu^{k'}$ (as in the previous case, this can be rewritten to show that this state can be expressed only from α and $(a_0, a_1, ..., a_{D-2})$) and proves that $p(i,k) = p(i,k'+1)$.

The argument above can be repeated as long as we have a column j of I_i whose current value is $D-1$. When there is no more such column, the peak $p(i,k)$ is deleted (its value becomes 0) by the next long avalanche which is necessarily $\phi^{k''}$ from the maximality of i-similar subsequences. □

The algorithm below gives the exact computation. From the state of an interval I_i and an avalanche type on I_i, f returns the greatest fired peak in I_i, and g computes the new state of I_i and appends the result of f to a sequence of types on interval I_{i+1}. g recursively calls itself, anticipating the i-similarity of successive avalanches when $\max P^k$ lies on the right of interval i.

Input: a non empty type α and an interval state $A = (a_0, \ldots, a_{D-2})$.
Data structure: a sequence w of types.
Functions:

$f : \mathcal{S} \times \mathcal{T} \setminus \{\epsilon\} \to \mathcal{T}$	$g : \mathcal{S} \times \mathcal{T} \setminus \{\epsilon\} \times T^* \to \mathcal{S} \times T^*$
$f(A, \alpha) :=$	$g(A, \alpha, w) :=$
if	**match** $f(A, \alpha)$ **with**
$(\{m \leq \alpha \mid a_m = D-1\} \neq \emptyset)$	$\quad \mid \epsilon \to (a_0 + 1, \ldots, a_\alpha + 1, a_{\alpha+1}, \ldots, a_{D-2}), w)$
then	$\quad \mid p \to$
$\quad \max\{m \mid a_m = D-1\}$	$g((a_0, \ldots, a_{p-1}, 0, a_{p+1} + 1, \ldots, a_{D-2} + 1), \alpha, w :: p)$
else	
$\quad \epsilon$	

Computation: $(A, \alpha) \mapsto g(A, \alpha, \epsilon)$

The algorithm above allows to define a deterministic finite state transducer $\mathfrak{T}$ (see for example [2]). $\mathfrak{T}$ is a 5-tuple $(Q, \Sigma, \Gamma, I, \delta)$ where the set of states Q is $\mathcal{S}$, the input and output alphabets (resp. Σ and Γ) are equal to $A = T \setminus \{\epsilon\} = \{0, \ldots, D-2\}$, the transition function δ has type $Q \times \Sigma \to Q \times \Gamma^*$ and is defined by the algorithm above: $\delta(q, \alpha) = Computation(q, \alpha)$. The initial state is $(0, 0, ..., 0)$, and we do not need to define a final state. The image of a word u by $\mathfrak{T}$ is denoted by $t(u)$.

If α is the type of an i-influent subsequence for a fixed integer i, then the sequence of types of the corresponding $i+1$-influent subsequences (*i.e.* subsequences where considered avalanches lie) is $t(\alpha)$. Thus, if u is the sequence of types of consecutive i-influent subsequences for a fixed integer i, then $t(u)$ is the sequence of types of the corresponding $i+1$-influent subsequences. Note that the last considered avalanche may not be the last one of the last $i+1$-influent subsequence.

For the lowest interesting value, $D = 3$, the transducer $\mathfrak{T}$ can easily be drawn. This diagram is given on figure 4. For readability, we write a and b instead of, respectively, 0

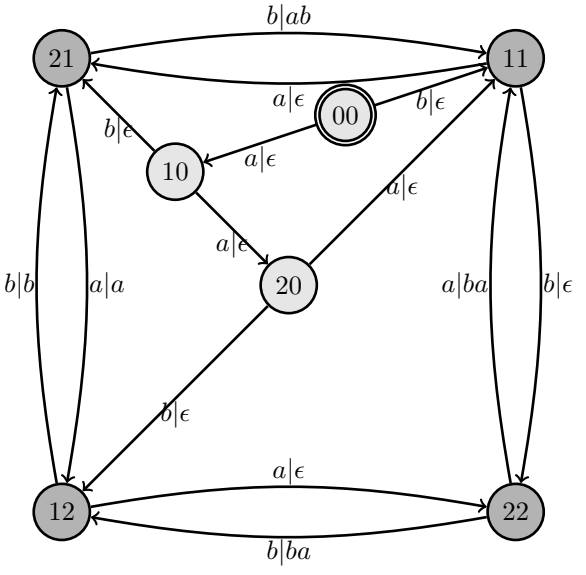


Fig. 4. Transducer for $D = 3$ - Edges are labelled $x|u$, where $x \in \mathcal{A}$ is the type to the current interval (input) and $u \in \mathcal{A}^*$ is the resulting sequence of types applied to the next interval (output). For example, $t(abaaaaab) = abaab$. Remark that, for $n > 0$, we have: $t((ab)^n) = (ab)^{n-1}$.

and 1, for the alphabet of the transducer, and we omit the drawing of states which are not connected with the initial one and are not useful for the computation of $t(u)$, for any word u.

The transducer has three transient states, (00, 10 and 20) and four recurrent states (11, 12, 21 and 22) organized in a cycle. A non trivial analysis of this transducer is given in appendix A. The result is stated on the lemma below.

Lemma 3. *[$D = 3$] For any k there exists n in $O(\log k)$ such that for all u of length k, $t^n(u)$ is a prefix of $(ab)^\omega$.*

2.4 From Words to Waves

The lemma above can be used to describe fixed point configurations, noticing that $|u| \leq N$, as follows:

Proposition 3. *In KSPM(D), let L be the global density column of N and I_i be an interval whose columns are greater than $L + 3(D-1)$. Assume that the sequence of types of i-influent subsequences of long avalanches up to N is*

$$(0, \ldots, D-2)^x (0, \ldots, p), \text{ with } x \geq 0 \text{ and } p \leq D-2.$$

Let y be the size of the last subsequence of type p. We have $y \leq x+1$, and

$$\pi(N)_{[i,\infty[} = \begin{cases} (p, \ldots, 1)(D-1, \ldots, 1)^{x-y} 0 (D-1, \ldots, 1)^y 0^\omega & \text{if } y < x+1 \\ (p+1, \ldots, 1)(D-1, \ldots, 1)^x 0^\omega & \text{if } y = x+1 \end{cases}$$

Proof (sketch). It is a trivial induction on avalanches. We concentrate on the right part of fixed points: $\pi(k)_{[i,\infty[}$. Initially for $k = 0$, it is equal to 0^ω. The $D-1$ first i-influent subsequences lead to $D-1, \ldots, 1, 0^\omega$. Then from $(D-1, \ldots, 1)^x$, using lemma 1 to predict the size of each i-influent subsequence, we have that a sequence of type $(0, \ldots, D-2)$ corresponds to exactly $(x+1)(D-1)$ long avalanches, and the behavior verifies the following invariant : the $((x+1)p+y)^{th}$ long avalanche, $0 \leq p < D-1$ and $0 < y \leq x+1$, has type p and lead to

$$\begin{cases} p, p-1, \ldots, 1, (D-1, \ldots, 1)^{x-y} 0 (D-1, \ldots, 1)^y 0^\omega & \text{if } y \leq x; \\ p+1, p, \ldots, 1, (D-1, \ldots, 1)^x 0^\omega & \text{if } y = x+1. \end{cases}$$ □

3 Conclusion

Let us sum up results on KSPM(3). We introduced the finite state transducer which, given a sequence of types (associated with a sequence of long avalanches) on an interval I_i, outputs the sequence of types (associated with the same sequence of long avalanches) on interval I_{i+1}. We proved in a previous paper [14] that the global density column $L(3, N)$ is in $O(\log N)$, and therefore that toward the study of the fixed point $\pi(N)$, the word transducer applies starting from an interval I_j with j in $\Omega(\log N)$. Lemma 3 shows that iterating $O(\log |u|)$ times the transducer function t outputs a prefix of $(ab)^\omega$, from any input sequence u. An upper bound for the size of any input word (sequence of types)

in KSPM(3) is N. As a consequence, there exists an index k in $O(\log N)$ such that the sequence of types associated with subsequences of long avalanches up to N on interval I_{j+k} is a prefix of $(ab)^\omega$. Finally, proposition 3 converts the temporal emergence of regularities when we iterate t into a spatial emergence of a wave pattern. It points out that as soon as a sequence of types which is a prefix of $(ab)^\omega$ is applied on an interval, then on the right of that interval $\pi(N)$ is a wave. A simple framing of the maximal non-empty column of $\pi(N)$ shows that it is in $\Theta(N)$, therefore the wave $(2,1)^*[0](2,1)^*$ describes asymptotically completely fixed points of KSPM(3) obtained starting from a finite number of stacked grains. This concludes the proof of Theorem 1.

We hope a generalization of this result to any parameter D, confirming experiments:

Conjecture 1. For a fixed parameter D, there exists a column number $i(N)$ in $O(\log N)$ such that: $\pi(N)_{[i(N),\infty[} = (D-1, D-2, \ldots, 2, 1)^*[0](D-1, D-2, \ldots, 2, 1)^*0^\omega$

We name this pattern a *wave* for when your draw the corresponding sand pile, it looks like waves. Toward this aim, a possible outline is decomposed into two subproblems: one is to provide a general formula in $O(\log N)$ for the global density column, allowing the use of transducers from that index; a second is a general study of KSPM(D) transducers resulting in the experimentally checked emergence of balanced outputs, then using proposition 3 we eventually conclude. Unfortunately from $D = 4$, transducers lack of human-readability for their number of state is D^{D-1}. Nevertheless, one may look for core properties on built transducers in order to deduce regular pattern emergence.

References

1. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized criticality. Phys. Rev. A 38(1), 364–374 (1988)
2. Berstel, J., Boasson, L.: Transductions and context-free languages. Teubner (Ed.), pp. 1–278 (1979)
3. Dartois, A., Magnien, C.: Results and conjectures on the sandpile identity on a lattice. In: Morvan, M., Rémila, É. (eds.) Discrete Models for Complex Systems, DMCS 2003. DMTCS Proceedings, vol. AB, pp. 89–102. Discrete Mathematics and Theoretical Computer Science (2003)
4. Durand-Lose, J.O.: Parallel transient time of one-dimensional sand pile. Theor. Comput. Sci. 205(1-2), 183–193 (1998)
5. Formenti, E., Masson, B., Pisokas, T.: Advances in symmetric sandpiles. Fundam. Inform. 76(1-2), 91–112 (2007)
6. Gajardo, A., Moreira, A., Goles, E.: Complexity of langton's ant. Discrete Applied Mathematics 117(1-3), 41–50 (2002)
7. Gale, D., Propp, J., Sutherland, S., Troubetzkoy, S.: Further travels with my ant. Mathematical Entertainments column, Mathematical Intelligencer 17, 48–56 (1995)
8. Goles, E., Kiwi, M.A.: Games on line graphs and sand piles. Theor. Comput. Sci. 115(2), 321–349 (1993)

9. Goles, E., Martin, B.: Computational Complexity of Avalanches in the Kadanoff Two-dimensional Sandpile Model. In: TUCS (ed.) Proceedings of JAC 2010 Journées Automates Cellulaires 2010, Turku Finland, pp. 121–132 (December 2010); F.1.1
10. Goles, E., Morvan, M., Phan, H.D.: The structure of a linear chip firing game and related models. Theor. Comput. Sci. 270(1-2), 827–841 (2002)
11. Kadanoff, L.P., Nagel, S.R., Wu, L., Zhou, S.-m.: Scaling and universality in avalanches. Phys. Rev. A 39(12), 6524–6537 (1989)
12. Levine, L., Peres, Y.: Spherical asymptotics for the rotor-router model in z d. Indiana Univ. Math. J, 431–450 (2008)
13. Moore, C., Nilsson, M.: The computational complexity of sandpiles. Journal of Statistical Physics 96, 205–224 (1999); 10.1023/A:1004524500416
14. Perrot, K., Rémila, E.: Avalanche structure in the kadanoff sand pile model. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 427–439. Springer, Heidelberg (2011), http://arxiv.org/abs/1101.5940
15. Perrot, K., Rémila, É.: Transduction on Kadanoff Sand Pile Model Avalanches. Application to Wave Pattern Emergence (2011), http://arxiv.org/abs/1106.2670
16. Phan, T.H.D.: Two sided sand piles model and unimodal sequences. ITA 42(3), 631–646 (2008)

Appendix

A Analysis of the Transducer for $D = 3$

In this appendix we provide a sketched[1] analysis of the transducer for $D = 3$, leading to a proof of lemma 3. Note that though we consider maximal length subsequences of long avalanches, input words for the transducer may contain arbitrary numbers of successive occurrences of a and b since we consider only i-influent subsequences.

Definition 2 (Height). *The height h of a finite word $u \in A^*$ is $h(u) = ||u|_a - |u|_b|$ where $|u|_x$ is the number of occurrences of the letter x in u.*

Lemma 4. *For any finite word $v \in \mathcal{L}$, we have: $h(t(v)) \leq \frac{h(v)}{4} + 1$*

Corollary 1. *Given a word $u \in \mathcal{A}^*$ of length l, there exists an $n(l)$ in $O(\log l)$ such that $t^{n(l)}(u)$ is a prefix of $(ab)^\omega$.*

Let us remark that $l < N$ for any input word u so corollary 1 apply for actual sand pile behavior. We therefore have a strong property on words emerging from iterations of the transducer function t : they are exponentially quickly prefixes of $(ab)^\omega$.

[1] A version with full appendix is available, see [15].

Problems Parameterized by Treewidth Tractable in Single Exponential Time: A Logical Approach

Michał Pilipczuk

Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Poland
michal.pilipczuk@students.mimuw.edu.pl

Abstract. We introduce a variant of modal logic, named EXISTENTIAL COUNTING MODAL LOGIC (ECML), which captures a good number of problems known to be tractable in single exponential time when parameterized by treewidth. It appears that all these results can be subsumed by the theorem that model checking of ECML admits an algorithm with such complexity. We extend ECML by adding connectivity requirements and, using the Cut&Count technique introduced by Cygan et al. [4], prove that problems expressible in the extension are also tractable in single exponential time when parameterized by treewidth; however, using randomization. The need for navigational character of the introduced logic is informally justified by a negative result that two expository problems involving non-acyclic local conditions, C_l-VERTEX DELETION and GIRTH $> l$ VERTEX DELETION for $l \geq 5$, do not admit such a robust algorithm unless Exponential Time Hypothesis fails.

1 Introduction

The notion of treewidth, extensively used by Robertson and Seymour in their proof of Wagner's Conjecture [20], in recent years proved to be an excellent tool for capturing characteristics of certain graph classes. Of particular interest are algorithmic applications of treewidth. Many problems, while hard in general, become robustly tractable, when the input graph is of bounded treewidth — a usual technique is based on construction of a dynamic programming algorithm on the tree decomposition. When combined with the graph-theoretical properties of treewidth, the approach leads to a number of surprisingly efficient algorithms, including approximation [6,9], parameterized [7,18] and exact algorithms [12,22]. In most cases, the dynamic program serves as a subroutine that solves the problem, when the treewidth turns out to be small.

The tractability of problems parameterized by treewidth can be generalized into a meta-theorem of Courcelle [3]: there exists an algorithm that, given a MSO formula φ and a graph G of treewidth t, tests whether φ is true in G in time $f(|\varphi|, t)|G|$ for some function f. Courcelle's Theorem can be viewed as a generalization of Thatcher and Wright Theorem about equivalence of MSO on finite trees and tree automata; in fact, in the proof one constructs an analogous tree automaton working on the tree decomposition. Unfortunately, similarly to other theorems regarding MSO and automata equivalence, the function f, which is in fact the time needed to process automaton's production, can depend non-elementary on $|\varphi|$ and t [13,23]. Therefore, a lot of effort

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 520–531, 2011.

has been invested in actual construction of the dynamic programming algorithms mimicking the behaviour of a minimal bottom-up automaton in order to obtain solutions that can be considered efficient and further used as robust subroutines. One approach, due to Arnborg et al. [2], is extending MSO by maximisation or minimisation properties, which corresponds to augmenting the automaton with additional counters. In many cases, the length of the formula defining the problem can be reduced to size independent of the expected size, yielding a $f(t)|G|^{O(1)}$ time algorithm. Unfortunately, careful analysis of the algorithm shows that the obtained function f can be still disastrous; however, for many concrete problems the algorithm can be designed explicitly and the complexity turns out to be satisfactory. For example, for the expository VERTEX COVER problem, the book by Kleinberg and Tardos gives an algorithm with running time $4^t|G|^{O(1)}$ [14], while the book by Niedermeier contains a solution with complexity $2^t|G|^{O(1)}$ [19].

Recently, Lokshtanov et al. [16] initiated a deeper study of currently best dynamic programming routines working in single exponential time in terms of treewidth, i.e., with complexity $c^t|G|^{O(1)}$ for some constant c. For a number of problems they proved them to be probably optimal: a faster solution would yield a better algorithm for CNF-SAT than exhaustive search. One can ask whether the phenomenon is more general: the straightforward dynamic programming solution reflecting the seemingly minimal automaton is optimal under believed assumptions. This question was stated by the same set of authors in [17] for a number of problems based on connectivity requirements, like CONNECTED VERTEX COVER or HAMILTONIAN PATH. For these, the considered routines work in time $2^{O(t \log t)}|G|^{O(1)}$, and a matching lower bound for one such problem, DISJOINT PATHS, was already established [17].

Surprisingly, the answer turned out to be negative. Very recently, Cygan et al. [4] introduced a technique called Cut&Count that yields single exponential in terms of treewidth Monte-Carlo algorithms for a number of connectivity problems, thus breaking the expected limit imposed by the size of the automaton. The results also include several intriguing lower bounds: while problems that include minimization of the number of connected components of the solution are tractable in single exponential time in terms of treewidth, similar tractability results for maximization problems would contradict *Exponential Time Hypothesis*. Recall that *Exponential Time Hypothesis* (ETH) states that the infinimum of such c that there exists a c^n algorithm solving 3CNF-SAT (n is the number of variables), is greater than 1.

A natural question arises: what properties make a problem tractable in single exponential time in terms of treewidth? Can we obtain a logical characterization, similar to Courcelle's Theorem?

Our contribution. We introduce a model of logic, named EXISTENTIAL COUNTING MODAL LOGIC (ECML), which captures a good number of problems known to admit an algorithm running in single exponential time in terms of treewidth. The model consists of a variation of modal logic, encapsulated in a framework for formulating algorithmic problems. We prove that model checking of ECML formulas is tractable in single exponential time, when parameterized by treewidth. In addition to solving the decision problem, the algorithm can actually count the number of solutions. The result generalizes a number of explicit dynamic programming routines (for example [1,5,10,11,21]), however yielding significantly worse constants in the bases of exponents.

Furthermore, we extend the ECML by connectivity requirements in order to show that the tractability result for ECML can be combined with the Cut&Count technique of Cygan et al. Again, we are able to show similar tractability for all the problems considered in [4], however with significantly worse constants in the bases of exponents.

Finally, we argue that the introduced logic has to be in some sense navigational or acyclic, by showing intractability in time $2^{o(p^2)}|G|^{O(1)}$ under ETH of two model problems involving non-acyclic local requirements, C_l-VERTEX DELETION and GIRTH $> l$ VERTEX DELETION for $l \geq 5$, where p is the width of a given path decomposition.

Outline. In Section 2, we introduce the notation and recall the well-known definitions. We try to follow the notation from [4] whenever it is possible. In Section 3, we introduce the model of logic. Section 4 contains the main tractability result, while Section 5 treats of combining it with the Cut&Count technique. In Section 6, we prove the intractability results under ETH. Section 7 is devoted to concluding remarks and suggestions on the further study.

2 Preliminaries and Notation

2.1 Notation

Let $G = (V, E)$ be a (directed) graph. By $V(G)$ and $E(G)$ we denote the sets of vertices and edges (arcs) of G, respectively. Let $|G| = |V(G)| + |E(G)|$. For a vertex set $X \subseteq V(G)$ by $G[X]$ we denote the subgraph induced by X. For an edge set $X \subseteq E$, by $V(X)$ denote the set of the endpoints of the edges from X, and by $G[X]$ — the subgraph $(V(X), X)$. Note that for an edge set X, $V(G[X])$ may differ from $V(G)$.

In a directed graph G by connected components we mean the connected components of the underlying undirected graph. For a subset of vertices or edges X of G, we denote by $|\mathtt{cc}(X)|$ the number of connected components of $G[X]$.

A monoid is a semigroup with a neutral element (a *unit*). The unit of a monoid M is denoted by e_M. Throughout the paper all monoids are commutative, therefore we choose to denote the operations by $+$ and refer to them as to *additions*. We treat the natural numbers $\mathbb{N}$ (nonnegative integers) also as a monoid with operation $+$ and unit 0.

2.2 Treewidth and Pathwidth

Definition 1 (Tree Decomposition, [20]). *A* tree decomposition *of a (undirected or directed) graph G is a tree $\mathbb{T}$ in which each vertex $x \in \mathbb{T}$ has an assigned set of vertices $B_x \subseteq V$ (called a* bag*) such that $\bigcup_{x \in \mathbb{T}} B_x = V$ with the following properties:*

- *for any $uv \in E$, there exists an $x \in \mathbb{T}$ such that $u, v \in B_x$.*
- *if $v \in B_x$ and $v \in B_y$, then $v \in B_z$ for all z on the path from x to y in $\mathbb{T}$.*

The *treewidth* $tw(\mathbb{T})$ of a tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one. The treewidth of a graph G is the minimum treewidth over all possible tree decompositions of G. A *path decomposition* is a tree decomposition that is a path. The pathwidth of a graph is the minimum width over all path decompositions.

We use a modified version of tree decomposition from [4], called *nice tree decomposition*, which is more suitable for development of dynamic programs. The idea of adjusting the tree decomposition to algorithmic needs comes from Kloks [15].

Definition 2 (Nice Tree Decomposition, Definition 2.3 of [4]). *A* nice tree decomposition *is a tree decomposition with one special bag r called the* root *with $B_r = \emptyset$ and in which each bag is one of the following types:*

- ***Leaf bag**: a leaf x of $\mathbb{T}$ with $B_x = \emptyset$.*
- ***Introduce vertex bag**: an internal vertex x of $\mathbb{T}$ with one child vertex y for which $B_x = B_y \cup \{v\}$ for some $v \notin B_y$. This bag is said to* introduce *v.*
- ***Introduce edge bag**: an internal vertex x of $\mathbb{T}$ labeled with an edge $uv \in E$ with one child bag y for which $u, v \in B_x = B_y$. This bag is said to* introduce *uv.*
- ***Forget bag**: an internal vertex x of $\mathbb{T}$ with one child bag y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$. This bag is said to* forget *v.*
- ***Join bag**: an internal vertex x with two child vertices y and z with $B_x = B_y = B_z$.*

We additionally require that every edge in E is introduced exactly once.

The main differences between standard nice tree decompositions used by Kloks [15] and this notion are: emptiness of leaf and root bags and usage of introduce edge bags.

As Cygan et al. observed in [4], given an arbitrary tree decomposition, a nice tree decomposition of the same width can be found in polynomial time. Therefore, we can assume that all our algorithms are given a tree decomposition that is nice.

Having fixed the root r, we associate with each node x of a tree decomposition $\mathbb{T}$ a set $V_x \subseteq V$, where a vertex v belongs to V_x iff there is a bag y which is a descendant of x in $\mathbb{T}$ with $v \in B_y$ (we follow convention that x is its own descendant). We also associate with each bag x of $\mathbb{T}$ a subgraph of G_x defined as follows:

$$G_x = (V_x, E_x = \{e \mid e \text{ is introduced in a descendant of } x\}).$$

As every edge is introduced exactly once, for each join bag x with children y, z, E_x is a disjoint sum of E_y and E_z.

3 The Model of Logic

We begin with introducing a notion of a *recognizable set* (for reference, see [8]).

Definition 3. *A set $S \subseteq \mathbb{N}$ is called* recognizable *iff there exists a finite monoid M, a set $F \subseteq M$ and homomorphism $\alpha_S : \mathbb{N} \to M$ such that $S = \alpha_S^{-1}(F)$.*

The notion of recognizable sets coincides with semilinear sets over $\mathbb{N}$. To better understand the intuition behind it, let us state the following simple fact.

Lemma 4 ([8]). *A set $S \subseteq \mathbb{N}$ is recognizable iff it is ultimately periodic, i.e., there exist positive integers N, k such that $n \in S \Leftrightarrow n + k \in S$ for all $n \geq N$.*

Intuitively, the main property of recognizable sets that will be useful, is that one can represent the behaviour of a nonnegative integer with respect to the operation of addition by one of finitely many values — the elements of the monoid.

Now, we are ready to introduce the syntax and semantics of ECML. We will do this in two steps. First, we introduce the inner, modal part of the syntax. Then, we explain how this part is to be put into the context of quantification over subsets of vertices and edges, thus creating a framework for defining problems.

3.1 The Inner Logic

The inner logic is called COUNTING MODAL LOGIC (CML). Of course, instead of a modal paradigm we could introduce an equivalent variation of guarded first order logic; however, modality seems to better capture the character of properties that can be expressed. Therefore, we choose this option.

A formula ψ of CML is evaluated in a certain vertex v of a (directed) graph G supplied by a vector of subsets of vertices $\overline{X}$ and a vector of subsets of edges $\overline{Y}$, of lengths p, q respectively. If ψ is true in vertex v of graph G, we will denote it by $G, \overline{X}, \overline{Y}, v \models \psi$. We begin with the syntax of CML for undirected graphs, defined by the following grammar:

$$\psi := \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \Rightarrow \psi \mid \psi \Leftrightarrow \psi \mid \mathbb{X} \mid \mathbb{Y} \mid \Diamond^S \psi \mid \Box^S \psi$$
$$\mathbb{X} := X_1 \mid X_2 \mid \ldots \mid X_p$$
$$\mathbb{Y} := Y_1 \mid Y_2 \mid \ldots \mid Y_q$$

The boolean operators are defined naturally. Let us firstly discuss the modal quantifiers $\Diamond^S$ and $\Box^S$. By definition, S has to be a recognizable set. We define the semantics of $\Diamond^S$ in the following manner: we say that $G, \overline{X}, \overline{Y}, v \models \Diamond^S \psi$ iff the number of neighbours w of vertex v satisfying $G, \overline{X}, \overline{Y}, w \models \psi$ belongs to S. The quantifier $\Box^S$ is somewhat redundant, as we say that $G, \overline{X}, \overline{Y}, v \models \Box^S \psi$ iff $G, \overline{X}, \overline{Y}, v \models \neg\Diamond^S \neg\psi$. To shorten notation, we use $\Diamond$ for $\Diamond^{\mathbb{N}^+}$ and $\Box$ for $\Box^{\mathbb{N}^+}$, where $\mathbb{N}^+$ is the set of positive integers. Thus, the definitions of $\Diamond$ and $\Box$ coincide with the natural way of introducing these quantifiers in other modal logics: $\Diamond\psi$ means that ψ has to be true in at least one neighbour, while $\Box\psi$ means that ψ has to be true in all the neighbours. Observe that the evaluation of the formula can be viewed as a process of walking on the graph — each time we evaluate a modal quantifier we move to a neighbour of the current vertex. Thus, after the first modal quantification there is a well specified edge that was used to directly access the current vertex from his neighbour.

Operators $\mathbb{X}$ can be viewed as unary predicates, checking whether the vertex, in which the formula is evaluated, belongs to a particular X_i. Formally, $G, \overline{X}, \overline{Y}, v \models X_i$ iff $v \in X_i$. Operators $\mathbb{Y}$ play the same role for edges — they check, whether the edge that was used to directly access the vertex belongs to a particular Y_j. Therefore, we narrow ourselves only to such formulas that use operators $\mathbb{Y}$ under some quantification.

We extend the logic to directed graphs by defining the neighbour to be a vertex that is adjacent via an arc, with no matter which direction. We introduce two new operators belonging to $\mathbb{Y}$: $\downarrow$ and $\uparrow$. The $\downarrow$ operator is true iff the arc that was used to directly access the current vertex is directed towards it, while $\uparrow$ is true iff it is directed towards the neighbour. Note that the new operators are significantly different from other operators in $\mathbb{Y}$, as they are not symmetrical from the point of view of the endpoints.

3.2 The Outer Logic

Let an *instance* be a quadruple $(G, \overline{FX}, \overline{FY}, \overline{k})$: a (directed) graph $G = (V, E)$ together with a vector of fixed subsets of vertices $\overline{FX}$, a vector of fixed subsets of edges $\overline{FY}$ and a vector of integer parameters $\overline{k}$. In most cases the fixed sets are not used, however they can be useful to distinguish subsets of vertices or edges of the graph that are

given in the input, like, for example, terminals in the STEINER TREE problem. Let $\mathcal{K}$ be a class of instances: a set of instances with the same lengths of vectors $\overline{FX}, \overline{FY}, \overline{k}$. We say that $\mathcal{K}$ is expressible in ECML iff belonging to $\mathcal{K}$ is equivalent to satisfying a fixed formula φ of the following form:

$$\varphi = \exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v (G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v) \models \psi \right].$$

Here:

- $\overline{X}, \overline{Y}$ are vectors of existentially quantified sets of vertices and edges, respectively;
- ϕ is an arbitrary quantifier-free arithmetic formula over the parameters, cardinalities of sets of vertices and edges of G and cardinalities of fixed and quantified sets;
- ψ is a CML formula evaluated on the graph G supplied with all the fixed and quantified sets.

We say that formulas of this form belong to EXISTENTIAL COUNTING MODAL LOGIC (ECML).

Example 5. The VERTEX COVER problem, given an undirected graph G and an integer k, asks whether there exists a set of at most k vertices such that every edge has at least one endpoint in the set. This can be reformulated as following: if a vertex is not chosen, then all its neighbours have to be chosen. Thus, the class of YES instances of VERTEX COVER can be expressed in ECML using the following formula:

$$\exists_{X \subseteq V} (|X| \leq k) \wedge \forall_v G, X, v \models (\neg X \Rightarrow \Box X).$$

Example 6. The r-DOMINATING SET problem, given an undirected graph G and an integer k, asks whether there exists a set of at most k vertices such that every vertex is at distance at most r from a vertex belonging to the set. The class of YES instances of r-DOMINATING SET can be expressed in ECML using the following formula:

$$\exists_{X \subseteq V} (|X| \leq k) \wedge \forall_v G, X, v \models \underbrace{(X \vee \Diamond (X \vee \Diamond (X \vee \ldots \Diamond (X \vee \Diamond X) \ldots)))}_{r \text{ quantifications}}.$$

4 Tractability of Problems Expressible in ECML

We are ready to prove the main result of the paper: the tractability of $\mathcal{K}$-RECOGNITION problem. The algorithm is based on the technique of prediction, useful in the construction of more involved dynamic programming routines on various types of decompositions. For an example, see the tractability result of Demaine et al. for r-DOMINATING SET [5] that is in fact a prototype of the constructed algorithm.

$\mathcal{K}$-RECOGNITION
Input: An instance $I = (G, \overline{FX}, \overline{FY}, \overline{k})$
Question: Does $I \in \mathcal{K}$?

Theorem 7. *If the class of instances $\mathcal{K}$ is expressible in* ECML*, then there exists an algorithm that, given an instance I along with a tree decomposition of G of width t, solves* $\mathcal{K}$-RECOGNITION *in time $c^t|G|^{O(m)}$ for some constant c depending on the class $\mathcal{K}$, where m is the number of quantified sets in the formula φ defining $\mathcal{K}$. Moreover, the algorithm can also compute the number of vectors $\overline{X}, \overline{Y}$ satisfying the formula φ.*

Before we proceed to the proof, let us strongly underline the fact that the theorem yields fixed parameter tractability of problems expressible in ECML, when the parameter is the treewidth only. The polynomial factor depends on the length of the formula (in fact, only on the number of quantified sets, which is in most cases small), so we obtain a different FPT algorithm for every ECML expressible class.

Proof. As was already mentioned in Section 2, we may assume that the given tree decomposition is a nice tree decomposition.

Let $\varphi = \exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v (G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v) \models \psi\right]$ be the formula defining the class $\mathcal{K}$ of instances of form $(G, \overline{FX}, \overline{FY}, \overline{k})$. Denote by p_0, q_0, p_1, q_1 lengths of vectors $\overline{FX}$, $\overline{FY}$, $\overline{X}$, $\overline{Y}$ respectively, where $m = p_1 + q_1$. We show the algorithm for computing the number of possible solutions $\overline{X}, \overline{Y}$; testing the outcome against zero solves the decision problem.

Firstly, the algorithm counts the cardinalities of fixed sets from vectors $\overline{FX}, \overline{FY}$ and introduces them into the arithmetic formula ϕ along with parameters and the numbers of vertices and edges of G. Then, the algorithm branches into $(1 + |V|)^{p_1}(1 + |E|)^{q_1}$ subroutines: in each it fixes the expected cardinalities of quantified sets from vectors $\overline{X}, \overline{Y}$. The algorithm executes only the branches with cardinalities satisfying ϕ and at the end sums up obtained numbers of solutions. This operation yields only a $|G|^{O(m)}$ blow-up of the running time, so we may assume that the expected cardinalities of all the quantified sets are precisely determined. Let us denote by $\overline{\mathbf{x}}$, $\overline{\mathbf{y}}$ vectors of expected cardinalities of $\overline{X}$, $\overline{Y}$, respectively.

As $\Box^S$ quantifier can be expressed by $\Diamond^S$ quantifier, we may assume that ψ uses only $\Diamond^S$ quantifiers. Consider all subformulas $\psi_1, \psi_2, \ldots, \psi_l$ of ψ beginning with a quantifier, denote $\psi_i = \Diamond^{S_i} \psi_i'$. Let us define S_i as $\alpha_{S_i}^{-1}(F_i)$ for homomorphism $\alpha_{S_i} : \mathbb{N} \to M_i$, finite monoid M_i and $F_i \subseteq M_i$. Let $\mathcal{H} = \prod_{i=1}^{l} M_i$ be a product monoid.

Let us denote $\mathcal{X} = \{0,1\}^{p_1}$, $\mathcal{P} = \{0,1\}^l$. Furthermore, let $\mathcal{I} = \mathcal{H} \times \mathcal{P} \times \mathcal{X}$. Intuitively, $\mathcal{I}$ is a set of possible information that can be stored about a vertex. The information consists of: *history*, an element of $\mathcal{H}$; *prediction*, a binary vector from $\mathcal{P}$ indicating, which formulas ψ_i are predicted to be true in a vertex; and *alignment*, a binary vector from $\mathcal{X}$ indicating, to which quantified sets X_i a vertex belongs.

Before we proceed to the formal description of the algorithm, let us give some intuition about what will be happening. The history is an element of the product monoid, used to count already introduced neighbours satisfying certain formulas ψ_i'. The additive structure on $\mathcal{H}$ enables us to update the history during introduce edge and join steps. However, while determining satisfaction of subformulas ψ_i in vertices of the graph, for the vertices in the bag we have to know their 'type' in the whole graph, not just the influence of already introduced part. Therefore, we introduce prediction: the information,

which subformulas are predicted to be true in a vertex in the whole graph. When doing updates while introducing edges we can access the predicted values, however when forgetting a vertex we have to ensure that its history is consistent with the prediction.

Let $\mathcal{R}$ be the set of solutions, i.e., pairs of vectors $\overline{X}, \overline{Y}$ for which ψ is satisfied in every vertex and satisfying constraints imposed on cardinalities of the sets. For a node x of the tree decomposition let $s \in \mathcal{I}^{B_x}$ be an *information evaluation*. We denote $s(v) = (h_v, \pi_v, b_v)$, where $v \in B_x$. Let $\overline{\tau}, \overline{\sigma}$ be vectors of integers of lengths p_1, q_1 respectively, satisfying $0 \leq \tau_i \leq \mathbf{x}_i$ and $0 \leq \sigma_j \leq \mathbf{y}_j$ for all $1 \leq i \leq p_1$, $1 \leq j \leq q_1$. Let us define $\mathcal{R}_x(\overline{\tau}, \overline{\sigma}, s)$: the set of partial solutions consistent with vectors $\overline{\tau}, \overline{\sigma}$ and information evaluation s. By this we mean the set of pairs of vectors $\overline{X}, \overline{Y}$ of subsets of vertices and edges of G_x respectively, such that the following conditions are satisfied.

- $|X_i| = \tau_i$ for $1 \leq i \leq p_1$, $|Y_j| = \sigma_j$ for $1 \leq j \leq q_1$.
- Every $v \in B_x$ belongs to exactly those X_i, for which the i-th coordinate of b_v is 1.
- In all $v \in G_x \setminus B_x$ the formula ψ is satisfied, when evaluated in G_x supplied with quantified and fixed sets. However, when evaluating some formula ψ_j in a vertex $w \in B_x$ we access the corresponding coordinate in the prediction π_w instead of actually evaluating it in G_x.
- For all $v \in B_x$ the number of neighbours of v in G_x satisfying the formula ψ'_i maps in α_{S_i} to the i-th coordinate of h_v, where ψ'_i is evaluated in the neighbour as if it was accessed directly from v. Again, boolean values of formulas ψ_j in the vertices of B_x are taken from the prediction instead of truly evaluated.

Observe that $\mathcal{R} = \mathcal{R}_r(\overline{\mathbf{x}}, \overline{\mathbf{y}}, \emptyset)$. The number of possible vectors $\overline{\tau}, \overline{\sigma}$ and information evaluations s is bounded by $|\mathcal{I}|^t |G|^{O(m)}$, so it suffices to show a dynamic program that computes $A_x(\overline{\tau}, \overline{\sigma}, s) = |\mathcal{R}_x(\overline{\tau}, \overline{\sigma}, s)|$ for all possible arguments in a bottom-up fashion. It is not hard to implement the performance of the routine for every type of a bag. The details of an algorithm running in $|\mathcal{I}|^{2t} |G|^{O(m)}$ time are described in the full version of the paper.

5 Adding Connectivity Requirements

We extend ECML by connectivity requirements. We say that an arithmetic formula $\phi(\overline{x}, y)$ is *monotone* over y iff $\phi(\overline{x}, y) \Rightarrow \phi(\overline{x}, y')$ for $y \geq y'$. In ECML+C, the arithmetic formula ϕ can also depend on $|\mathtt{cc}(FX_i)|$, $|\mathtt{cc}(FY_j)|$, $|\mathtt{cc}(X_i)|$, $|\mathtt{cc}(Y_j)|$, numbers of connected components of fixed and quantified sets. The dependence on the quantified part, variables $|\mathtt{cc}(X_i)|$ and $|\mathtt{cc}(Y_j)|$, is however restricted to be monotone, i.e., if y is the variable of ϕ that corresponds to the number of connected components of some quantified set, then ϕ has to be monotone over y. The need of monotonicity can be justified by a number of lower bounds for problems involving maximization of the number of connected components, due to Cygan et al. [4].

It appears that we can combine the Cut&Count technique with the dynamic programming routine described in Section 4 in order to obtain similar tractability of problems defined in ECML+C. Unfortunately, application of the technique gives us the tractability of only the decision problem. To the best of author's knowledge, extending the Cut&Count technique to counting problems is an open question, posted in [4].

Theorem 8. *If the class of instances $\mathcal{K}$ is expressible in* ECML+C*, then there exists a Monte-Carlo algorithm that, given the instance I along with a tree decomposition of G of width t, solves* $\mathcal{K}$-RECOGNITION *in time $c^t|G|^{O(m)}$ for some constant c depending on the class $\mathcal{K}$, where m is the number of quantified sets in the formula defining $\mathcal{K}$. The algorithm cannot produce false positives and produces false negatives with probability at most $\frac{1}{2}$.*

Again, we strongly underline the fact that the theorem yields an FPT algorithm with only treewidth as a parameter for every class $\mathcal{K}$ separately, as the polynomial factor depends on the formula defining the class.

The proof is a straightforward translation of the proof of Theorem 7 to the language of Cut&Count. Due to space limitations it can be found in the full version of the paper.

6 The Necessity of Acyclicity

We prove the intractability results for two expository problems involving non-acyclic local requirements.

C_l-VERTEX DELETION
Input: An undirected graph G and an integer k
Question: Is it possible to remove at most k vertices from G so that the remaining vertices induce a graph without cycles of length l?

GIRTH $> l$ VERTEX DELETION
Input: An undirected graph G and an integer k
Question: Is it possible to remove at most k vertices from G so that the remaining vertices induce a graph without cycles of length at most l?

Observe that local conditions involved in these problems are not closed under unraveling of the graph, which is a necessary condition to be expressible in CML. From this we can informally infer that in any approach of this kind the language of defining local requirements has to be in some sense navigational, as it cannot capture even the simplest properties not closed under bisimulation.

Theorem 9. *Assuming ETH, there is no $2^{o(p^2)}|G|^{O(1)}$ time algorithm for C_l-*VERTEX DELETION *nor for* GIRTH $> l$ VERTEX DELETION *for any $l \geq 5$. The parameter p denotes the width of a given path decomposition of the input graph.*

As a path decomposition of width p is also a tree decomposition of width p, the result is in fact stronger than analogous for treewidth instead of pathwidth. Before we proceed to the proof, note that both these problems admit a simple $2^{O(t^2)}|G|^{O(1)}$ dynamic programming algorithm, where t is the width of a given tree decomposition. In the state, one remembers for every pair of vertices of bag B_x, whether in G_x they can be connected via paths of length $1, 2, \ldots, l-1$ disjoint with the solution.

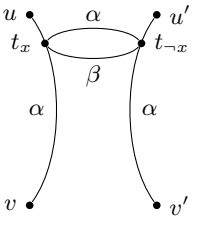
We present a polynomial-time reduction that given a 3CNF-SAT instance: a formula φ in 3CNF over n variables and consisting of m clauses, produces a graph G along with its path decomposition of width $O(\sqrt{n})$ and an integer k, such that

- if φ is satisfiable then (G, k) is a YES instance of GIRTH $> l$ VERTEX DELETION;
- if (G, k) is a YES instance of C_l-VERTEX DELETION then φ is satisfiable.

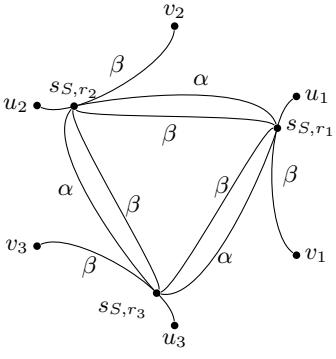
As every YES instance of GIRTH $> l$ VERTEX DELETION is also a YES instance of C_l-VERTEX DELETION, the constructed instance (G, k) is equivalent to given instance of 3CNF-SAT both when considered as an instance of C_l-VERTEX DELETION and of GIRTH $> l$ VERTEX DELETION. Thus, existence of an algorithm for C_l-VERTEX DELETION or GIRTH $> l$ VERTEX DELETION running in $2^{o(p^2)}|G|^{O(1)}$ time would yield an algorithm for 3CNF-SAT running in $2^{o(n)}(n+m)^{O(1)}$ time, contradicting ETH. We can assume that each clause in φ contains exactly three literals by copying some of them if necessary.

Let us choose $\alpha = \lfloor \frac{l-1}{2} \rfloor$, $\beta = \lceil \frac{l+1}{2} \rceil$. Thus, following conditions are satisfied: $2 \leq \alpha < \beta$, $\alpha + \beta = l$, $2\beta > l$, $2\alpha + 4 > l$.

Now we show the construction of the instance. The proof of its soundness and the bound on pathwidth can be found in the full version of the paper.



(a) Variable gadget Q_x



(b) Clause gadget C_S

Construction. We begin the construction by creating two sets of vertices A, B, each consisting of $\lceil \sqrt{2n} \rceil$ vertices. As $|A \times B| \geq 2n$, let us take any injective function $\psi : L \to A \times B$, where L is the set of literals over the variables of the formula φ, i.e., symbols x and $\neg x$ for all variables x.

For every variable x we construct a *variable gadget* Q_x in the following manner. Let $\psi(x) = (u, v)$ and $\psi(\neg x) = (u', v')$ (u and u' or v and v' may possibly coincide). Connect u with v and u' with v' via paths of length α. Denote the inner vertices of the paths that are closest to u and u' by t_x and $t_{\neg x}$ respectively. Connect t_x with $t_{\neg x}$ via two paths: one of length α and one of length β. Note that these two paths form a cycle of length l. The gadget consists of all the constructed paths along with vertices u, u', v, v'.

Now, for every clause $S = r_1 \vee r_2 \vee r_3$, where r_1, r_2, r_3 are literals, we construct the *clause gadget* C_S in the following manner. Let $\psi(r_i) = (u_i, v_i)$ for $i = 1, 2, 3$ (u_i or v_i may possibly coincide). For $i = 1, 2, 3$ connect u_i with v_i via a path of length β, and denote inner vertices of these paths that are closest to u_i by s_{S,r_i}. Connect each pair (s_{S,r_1}, s_{S,r_2}), (s_{S,r_2}, s_{S,r_3}), (s_{S,r_3}, s_{S,r_1}) via two paths: one of length α and one of length β. Thus, we connect $s_{S,r_1}, s_{S,r_2}, s_{S,r_3}$ by a triple of cycles of length l. The gadget consists of all the constructed paths together with vertices u_i, v_i.

We conclude the construction by setting $k = n + 2m$.

7 Conclusions and Open Problems

In this paper we introduced a logical formalism based on modality, EXISTENTIAL COUNTING MODAL LOGIC, capturing a good number of well-studied problems known to be tractable in single exponential time when parameterized by treewidth. We proved that testing, whether a fixed ECML formula is true in a given graph, admits an algorithm with complexity $c^t|G|^{O(1)}$, where t is the width of given tree decomposition. We extended ECML by connectivity properties and obtained a similar tractability result using the Cut&Count technique of Cygan et al. [4]. The need for navigational character of the logic was informally justified by a negative result under ETH that two model problems with non-acyclic requirements are not solvable in $2^{o(p^2)}|G|^{O(1)}$ time, where p is the width of a given path decomposition.

One open question is to breach the gap in the presented negative result. For $l = 3$, C_l-VERTEX DELETION is solvable in single exponential time in terms of treewidth, while for $l \geq 5$ our negative result states that such a robust solution is unlikely. To the best of author's knowledge, for $l = 4$ there are no matching lower and upper bounds.

Secondly, there are problems that admit a single exponential algorithm when parameterized by treewidth, but are not expressible in ECML. One example could be K_l-VERTEX DELETION, that, given a graph G along with an integer k, asks whether there exists a set of at most k vertices that hits all the subgraphs K_l. A dynamic program for this problem running in time $2^t|G|^{O(1)}$ can be constructed basing on the observation, that for every subclique of a graph there has to be a bag fully containing it. Can we find an elegant extension of ECML that would capture also such type of problems?

Acknowledgments. The author would like to thank Mikołaj Bojańczyk for invaluable help with the logical side of the paper, as well as Marek Cygan, Marcin Pilipczuk and Jakub Onufry Wojtaszczyk for many helpful comments on the algorithmic part.

References

1. Alber, J., Niedermeier, R.: Improved tree decomposition based algorithms for domination-like problems. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 221–233. Springer, Heidelberg (2002)
2. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms 12, 308–340 (1991)
3. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Information and Computation 85(1), 12–75 (1990)
4. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. CoRR, abs/1103.0534 (2010)
5. Demaine, E., Fomin, F., Hajiaghayi, M., Thilikos, D.: Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, p. 190. Springer, Heidelberg (2003)
6. Demaine, E.D., Hajiaghayi, M.T.: The bidimensionality theory and its algorithmic applications. The Computer Journal 51(3), 292–302 (2008)
7. Dorn, F., Fomin, F.V., Thilikos, D.M.: Fast subexponential algorithm for non-local problems on graphs of bounded genus. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 172–183. Springer, Heidelberg (2006)

8. Eilenberg, S.: Automata, Languages, and Machines. Academic Press, Inc., Orlando (1974)
9. Eppstein, D.: Diameter and treewidth in minor-closed graph families. Algorithmica 27(3), 275–291 (2000)
10. Fiorini, S., Hardy, N., Reed, B., Vetta, A.: Planar graph bipartization in linear time. In: Proc. 2nd GRACO, Electronic Notes in Discrete Mathematics, pp. 265–271. Elsevier, Amsterdam (2005)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. Springer, Heidelberg (2006)
12. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. Algorithmica 54(2), 181–207 (2009)
13. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Annals of Pure and Applied Logic, 215–224 (2002)
14. Kleinberg, J., Tardos, É.: Algorithm Design. Addison-Wesley, Reading (2005)
15. Kloks, T.: Treewidth, Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
16. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: 22st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011 (2011)
17. Lokshtanov, D., Marx, D., Saurabh, S.: Slightly superexponential parameterized problems. In: 22st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011 (2011)
18. Mölle, D., Richter, S., Rossmanith, P.: Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. Theory of Computing Systems 43(2), 234–253 (2008)
19. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2006)
20. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. Journal of Combinatorial Theory, Series B 36(1), 49–64 (1984)
21. van Rooij, J.M.M., Bodlaender, H.L., Rossmanith, P.: Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 566–577. Springer, Heidelberg (2009)
22. van Rooij, J.M.M., Nederlof, J., van Dijk, T.C.: Inclusion/Exclusion Meets Measure and Conquer. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 554–565. Springer, Heidelberg (2009)
23. Weyer, M.: Modifizierte parametrische Komplexitätstheorie. Dissertation, Albert-Ludwigs-Universität Freiburg (2007)

Distributed Synthesis for Regular and Contextfree Specifications

Wladimir Fridman and Bernd Puchala

RWTH Aachen University, Germany
{fridman@automata,puchala@logic}.rwth-aachen.de

Abstract. We adress the controller synthesis problem for distributed systems with regular and deterministic contextfree specifications. Our main result is a complete characterization of the decidable architectures for local specifications. This extends existing results on local specifications in two directions. First, we consider arbitrary, not necessarily acyclic, architectures and second, we allow deterministic contextfree specifications. Moreover, we show that as soon as one considers global deterministic contextfree specifications, even very simple architectures are undecidable.

1 Introduction

Open non-terminating reactive systems are computing systems which continously interact with an environment. Such systems are modeled as infinite games between a controller for the system and the environment. As the behavior of the environment is usually not constrained a priori in such settings, it is considered as being antagonistic. Non-terminating reactive systems have first been considered in the context of switching circuits. Church's Synthesis Problem [2] is to decide whether there exists a switching circuit such that all possible input/output behaviors of the circuit satisfy a given specification and, if such a circuit exists, it should be constructed effectively. The first solution to this problem has been given by Büchi and Landweber [1] who showed that for any specification formulated in monadic second order logic over words, the synthesis problem is decidable and finite state solutions can be constructed effectively.

Since then, non-terminating reactive system have received growing attention in computer science. Such systems naturally capture many settings where a given plant should be controlled in such a way, that any constrained system behavior satisfies a certain specification. Moreover, the prospect of being able to construct such systems automatically from a given specification, rather than verifying a system that has already been built, has led to intensive research on the controller synthesis problem [8,12,14] and to extensions of the basic setting in various directions. For example, other specification formalisms have been considered like temporal logics [11] and contextfree specifications [16], the systems have been extended to distributed systems which consist of several components [13], and stochastic versions of reactive systems have been investigated [4].

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 532–543, 2011.

We consider distributed systems with regular and contextfree specifications. Such a distributed system is specified by an architecture which consists of a set of processes and channels via which the processes can communicate. Distributed systems have first been considered in [13] where it has been shown that in general, the distributed controller synthesis problem is undecidable for specifications from the linear time temporal logic LTL. Moreover, for pipelines, a special class of acyclic architectures, decidability has been proved for LTL specifications. In [9] the decidability results have been extended to certain classes of architectures with cycles and to specifications from the branching time logic CTL. Finally, in [5], a full characterization of the decidable architectures has been given by means of certain patterns of information flow, called information forks: Two processes form an information fork if they are incomparably informed and the controller synthesis problem for an architecture is decidable if, and only if, it does not contain an information fork. This holds for both LTL and CTL specifications.

In [10], the concept of local specifications was introduced. There, any system process has an individual specification which defines the correct behaviors of just this process instead of the whole system. Obviously, any set of regular local specifications can be transformed into a regular global specification, so all decidability results for global regular specifications hold for local regular specifications as well. However, there are architectures, called two-flanked pipelines, which contain an information fork but are decidable for local regular specifications [10]. Moreover, for the class of acyclic architectures, a characterization of the decidable architectures was established for regular specifications.

We extend this result to architectures which may contain cycles and to specifications which are regular or deterministic contextfree. Notice that in the case of global specifiations, backward-channels (channels from processes with a lower level of information to better informed processes) are futile, so architectures without information forks can be transformed into a normal form which does not have cycles [5]. These techniques do not, however, preserve local specifications and in fact, in the case of local specifications, backward-channels can be significant, as they may increase the access of the local specifications to the overall (global) system behavior. Therefore, to deal with cycles in the case of local specifications, one has to use different methods. Also, processes not reachable from the environment cannot be eliminated in general, so a decidable architecture may consist of several basic decidable subarchitectures. Our analysis is centered around those two structural aspects and the higher expressive power of deterministic contextfree specifications. In Section 4 we first prove decidability and undecidability results for some special classes of architectures, followed by a complete characterization of the decidable architectures in Section 5.

In Section 3, we also show that as soon as one considers global deterministic contextfree specifications, even very simple architectures become undecidable. For architectures with only one system process, [16] has shown that they are decidable for deterministic contextfree specifictions. We show that this is not the case for architectures with at least two system processes or at least one channel from the environment which cannot be read by any system process.

2 Preliminaries

We denote the boolean alphabet by $\mathbb{B} = \{\top, \bot\}$. For any alphabet Σ and words $\alpha, \beta \in \Sigma^* \cup \Sigma^\omega$ and $n \in \mathbb{N}$ we write $\alpha(n)$ for the n-th letter of α, $\alpha\restriction_n = \alpha(0)\dots\alpha(n-1)$ and $\alpha \sqsubseteq \beta$ if α is a prefix of β. For an integer $k > 0$ let $[k]$ denote the set $\{0, ..., k-1\}$. For a cartesian product $A = A_0 \times \ldots \times A_{n-1}$ and $I \subseteq [n]$ we denote $A_I = \prod_{i\in I} A_i$. Moreover, $\mathrm{Pr}_I(a) = (a_i)_{i\in I}$ for an element $a = (a_0, \ldots, a_{n-1}) \in A$, $\mathrm{Pr}_I(\alpha) = \mathrm{Pr}_I(\alpha(0))\mathrm{Pr}_I(\alpha(1))\dots$ for a word $\alpha \in A^* \cup A^\omega$ and $\mathrm{Pr}_I(L) = \{\mathrm{Pr}_I(\alpha) \mid \alpha \in L\}$ for a language $L \subseteq A^* \cup A^\omega$. However, usually we do not refer to an explicit ordering of the components of a cartesian product and write Pr_{A_I} instead of Pr_I. If A has certain identical components, this is of course not unambigious, but it will be clear from the context to which components the operator projects. Moreover, if $\alpha \in X^\omega$ and $\beta \in Y^\omega$, $\alpha^\frown\beta \in (X \times Y)^\omega$ denotes the ω-word with $(\alpha^\frown\beta)(i) = (\alpha(i), \beta(i))$ for all $i \in \mathbb{N}$. For a function $\sigma\colon \Sigma^* \to \Sigma'$, the ω-language which is generated by σ over a language $L_{\mathrm{in}} \subseteq \Sigma^\omega$ is $\{\sigma(\alpha\restriction_0)\sigma(\alpha\restriction_1)\dots \mid \alpha \in L_{\mathrm{in}}\} \subseteq (\Sigma')^\omega$. For any functions $f\colon A \to B$ and $g\colon B \to C$ the composition $f \circ g\colon A \to C$ is defined by $(f \circ g)(a) = g(f(a))$.

Architectures. An architecture $\mathfrak{A} = (P, C, r)$ consists of the following components. $P = \{p_{\mathrm{env}}\} \cup P_{\mathrm{sys}}$ is the set of processes where $p_{\mathrm{env}} = p_0$ takes the role of the environment and $P_{\mathrm{sys}} = \{p_1, \ldots, p_n\}$ with $n \geq 1$ are system processes. Moreover, $C = \bigcup_{p\in P} C_p$ is the set of channels where the sets C_p are pairwise disjoint and $r\colon C \to P$ is a function, assigning for each channel a process which reads it such that $r(C_p) \subseteq P_{\mathrm{sys}}$ for all $p \in P_{\mathrm{sys}}$. We assume that, for all $p \in P_{\mathrm{sys}}$, $r^{-1}(p) \neq \emptyset$ and $C_p \neq \emptyset$, i.e., each system process has at least one input and one output channel. So basically, an architecture is a directed graph with multi-egdes.

An architecture $\mathfrak{A}$ is called connected, if P_{sys} induces a connected subgraph of $\mathfrak{A}$. Consider a set $Q \subseteq P_{\mathrm{sys}}$. If the subgraph of $\mathfrak{A}$ induced by $Q \cup \{p_{\mathrm{env}}\}$ is an architecture, then we denote this architecture by $\mathfrak{A}(Q)$ and we say that $\mathfrak{A}(Q)$ is the subarchitecture of $\mathfrak{A}$ induced by Q. (Notice that not each set $Q \subseteq P_{\mathrm{sys}}$ induces a subarchitecture of $\mathfrak{A}$.) An architecture $\mathfrak{A}'$ is a subarchitecture of $\mathfrak{A}$ if $\mathfrak{A}' = \mathfrak{A}(Q)$ for some set $Q \subseteq P_{\mathrm{sys}}$.

For $p \in P$, $H_p = \{c \in C_p | r(c) = p\}$ are called hidden channels of p, i.e., they cannot be read by any other process. C_{p_0} are external input channels and $H_{p_0} \subseteq C_{p_0}$ are hidden input channels. $\bigcup_{i=1}^n C_{p_i} \setminus H_{p_i}$ are internal communication channels and the channels $\bigcup_{i=1}^n H_{p_i}$ are used to model external output channels.

We say that process p sends information to process $p' \neq p$ if there is some channel $c \in C_p$ such that $p' = r(c)$. Process p is called reachable, if there is a directed path from p_{env} to p. Process p is better informed than $p' \neq p$ if p is reachable and each directed path from p_{env} to p' goes through p. Notice that a process may send information to another process via multiple channels. However, if more convenient, we can assume w.l.o.g. that there is at most one such channel.

An architecture $\mathfrak{A} = (P, C, r)$ is called pipeline (two-flanked pipeline) with backward-channels if $r(C_{p_0}) = \{p_1\}$ ($r(C_{p_0}) = \{p_1, p_n\}$ in case of a two-flanked pipeline) and, for $i \in [n] \setminus \{0\}$, $r(C_{p_i}) \subseteq \{p_j \mid 0 < j \leq i+1\}$ (see Figure 1).

Fig. 1. Pipeline and two-flanked pipeline with backward-channels

A channel $c \in C_{p_i}$ with $r(c) = p_j$ is called backward-channel if $0 < j < i$ and it is called forward-channel if $j = i+1$. Moreover, $\mathfrak{A}$ is called (two-flanked) pipeline if it has no backward-channels.

A labeling $(\Sigma_c)_{c\in C}$ for $\mathfrak{A}$ assigns to any channel $c \in C$ a nonempty finite set Σ_c of signals which can be sent along c. We define, for every $p \in P$, the input and output alphabets of p as $\Sigma^p_{\text{in}} = \prod_{c\in r^{-1}(p)} \Sigma_c$ and $\Sigma^p_{\text{out}} = \prod_{c\in C_p} \Sigma_c$. The local alphabet of p is $\Sigma^p = \Sigma^p_{\text{in}} \times \Sigma^p_{\text{out}}$. The global system alphabet is defined as $\Sigma^{\mathfrak{A}} = \prod_{c\in C} \Sigma_c = \prod_{i=0}^{n} \Sigma^{p_i}_{\text{out}}$. At each point in time i every process p writes a letter $\alpha_p(i) \in \Sigma^p_{\text{out}}$ to the corresponding channels $c \in C_p$. A global system behavior is an ω-word $\alpha = \alpha_{p_0} \frown \ldots \frown \alpha_{p_n}$ from $(\Sigma^{\mathfrak{A}})^\omega$. For $p \in P_{\text{sys}}$, the local process behavior of p is $\beta_p \frown \alpha_p$, where $\beta_p = \Pr_{\Sigma^p_{\text{in}}}(\alpha)$,

A global system specification is a language $L \subseteq (\Sigma^{\mathfrak{A}})^\omega$ consisting of all correct system behaviors. A local specification for process p is a language $L_p \subseteq (\Sigma^p)^\omega$. For a collection $(L_{p_1}, \ldots, L_{p_n})$ of local specifications for the system processes, the corresponding global system specification is the language $L \subseteq (\Sigma^{\mathfrak{A}})^\omega$ such that $\Pr_{\Sigma^p}(L) = L_p$ for any system process $p \in P_{\text{sys}}$.

A local strategy for process p maps a local input history of process p to the next output symbol of process p, i. e., it is a function $\sigma_p \colon (\Sigma^p_{\text{in}})^* \to \Sigma^p_{\text{out}}$. The local behavior $\beta_p \frown \alpha_p$ of process p is consistent with σ_p, if $\alpha_p(i) = \sigma_p(\beta_p{\restriction_i})$ for all $i \in \mathbb{N}$. For a language $L_{\text{in}} \subseteq (\Sigma^p_{\text{in}})^\omega$, the local strategy σ_p is called winning on L_{in} if any local behavior $\beta_p \frown \alpha_p$ of p with $\beta_p \in L_{\text{in}}$ which is consistent with σ_p is in L_p. It is called winning for process p, if it is winning on $(\Sigma^p_{\text{in}})^\omega$.

A joint strategy for $p_1, \ldots, p_n$ is a tuple $\sigma = (\sigma_{p_1}, \ldots, \sigma_{p_n})$ where each σ_{p_i} is a local strategy for process p_i. A global system behavior $\alpha = \alpha_{p_0} \frown \ldots \frown \alpha_{p_n}$ is consistent with σ, if the local process behavior of each system process p is consistent with σ_p. The strategy σ is winning, if any system behavior which is consistent with σ is in the global system specification L. Notice that a joint strategy which consists of local winning strategies is also winning for L. The converse is, however, not true in general: a local strategy for a process p which is part of a joint winning strategy is not necessarily locally winning as the inputs that p receives from other system processes are constrained by their local strategies.

A specification $L \subseteq (\Sigma^{\mathfrak{A}})^\omega$ is realizable in an architecture $\mathfrak{A}$ with labeling $(\Sigma_c)_{c\in C}$ if there is a joint winning strategy for processes $p_1, \ldots, p_n$. The realizability problem is to decide, given an architecture $\mathfrak{A}$, a labeling $(\Sigma_c)_{c\in C}$ and a specification $L \subseteq (\Sigma^{\mathfrak{A}})^\omega$, whether L is realizable in $\mathfrak{A}$. For a class $\mathcal{L}$ of specifications we say that an architecture $\mathfrak{A}$ is decidable for specifications from $\mathcal{L}$ if the realizability problem is decidable for the fixed architecture $\mathfrak{A}$ when specifications are restricted to $\mathcal{L}$.

Specifications. We consider regular and deterministic contextfree specifications. Regular specifications are those which can be recognized by parity automata. Notice that deterministic, nondeterministic and alternating parity automata over words have all the same expressive power [15]. Deterministic contextfree specifications are those which can be recognized by deterministic parity pushdown automata (parity DPDA) [3], i.e., finite state automata which additionally have access to a stack-memory. We also consider deterministic 1-counter specifications, i.e., recognizable by parity DPDA with only a single stack-symbol. We assume standard concepts and notations for these automata. Notice that deterministic contextfree languages form a proper subclass of contexfree languages and that, while games with deterministic contextfree winning condition are decidable [16], nondeterministic contextfree games are undecidable (see, e.g., [6]).

Trees and Tree-Automata. For a finite alphabet Σ and a finite set X, a Σ-labeled X-tree is a function $t\colon X^* \to \Sigma$. We use alternating parity tree automata (parity ATA) and nondeterministic parity pushdown tree automata (parity NPDTA) on such trees. Again, we assume standard concepts and notations for these automata [7,14]. Notice that for any parity ATA $\mathcal{A}$ there is an equivalent nondeterministic parity tree automaton (parity NTA) $\mathcal{N}$, that means, $L(\mathcal{A}) = L(\mathcal{N})$ [15]. Moreover, like for parity NTA, the nonemptiness problem for parity NPDTA is decidable [7]. Furthermore, for any parity NTA $\mathcal{A}$ and any parity NPDTA $\mathcal{B}$, there is a parity NPDTA $\mathcal{C}$ such that $L(\mathcal{C}) = L(\mathcal{A}) \cap L(\mathcal{B})$. We also use the widening operator $\mathrm{wide}(t, Y)$ on a Σ-labeled X-tree t which yields the Σ-labeled $X \times Y$-tree $t' = \mathrm{wide}(t, Y)$ with $t'(x, y) = t(x)$. For any parity NTA $\mathcal{A}$ over Σ-labeled $X \times Y$-trees, there is a parity NTA $\mathcal{B}$ over Σ-labeled X-trees, which accepts a tree t if, and only if, $\mathrm{wide}(t, Y) \in L(\mathcal{A})$ [8].

3 Global Specifications

Theorem 1. *The realizability problem for global deterministic contextfree specifications is undecidable for an architecture $\mathfrak{A}$ if, and only if, $|P| > 2$ or $H_{p_0} \neq \emptyset$.*

Proof. (Sketch) We proceed by a reduction from the Post's Correspondence Problem PCP. Let $p_1, p_2 \in P_{sys}$ and let $c_{\mathrm{in}} \in C_{p_0}$. W.l.o.g., assume $r(c_{\mathrm{in}}) = p_1$. Moreover, let $c_{\mathrm{out}} \in C_{p_2}$. We silence all other channels by defining $\Sigma_c = \{\sharp\}$ for all $c \in C \setminus \{c_{\mathrm{in}}, c_{\mathrm{out}}\}$. Now given an instance $\mathcal{I} = ((u_0, v_0), \ldots, (u_{m-1}, v_{m-1}))$ of the PCP over an alphabet Θ, consider the specification L which consists of all words $\alpha_{\mathrm{in}} \frown \alpha_{\mathrm{out}} \frown \alpha'$ where $\alpha_{\mathrm{in}} \in \{U, V\}^\omega$ and $\alpha_{\mathrm{out}} = \sharp i_k \ldots i_1 \sharp w \sharp \Sigma_{c_{\mathrm{out}}}^\omega$ with $i_j \in [m]$ and $w \in \Theta^*$ such that $w = u_{i_1} \ldots u_{i_k}$ if $\alpha_{\mathrm{in}}(0) = U$ and $w = v_{i_1} \ldots v_{i_k}$ if $\alpha_{\mathrm{in}}(0) = V$. Clearly, L can be recognized by a parity DPDA and the system processes have a joint winning strategy if, and only if, $\mathcal{I}$ has a solution. The crucial point is that the process which produces α_{out} cannot observe the first letter of α_{in}. This is also the case if α_{in} is sent via some channel from H_{p_0}. □

Remark 2. By a reduction from the halting problem for 2-register machines one can show that Theorem 1 also holds for deterministic 1-counter specifications.

4 Local Specifications

From now on, we consider architectures with specifications given by collections of local specifications, one for each system process. For the class of acylic architectures, it has been shown in [10], that the realizability problem for local regular specifications is decidable if, and only if, each conntected subarchitecture is a subarchitecture of a two-flanked pipeline. We continue the investigation by classifying the decidable architectures for the more general case where cycles are allowed and the local specifications may also be deterministic contextfree. In this section, we first prove decidability and undecidability results for some special classes of architectures.

4.1 Decidability

Pipelines with Backward-Channels. Let $\mathfrak{A} = (P, C, r)$ be a pipeline with backward-channels, let $(\Sigma_c)_{c\in C}$ be a labeling of $\mathfrak{A}$ and let $L_{p_1}, \ldots, L_{p_n}$ be local specifications for the system processes where $L_{p_1}, \ldots, L_{p_{n-1}}$ are regular and L_{p_n} is regular or deterministic contextfree. For any system process p_i we define the accumulated output alphabet $\Sigma_{\text{out}}^{\geq i} := \prod_{j\geq i} \Sigma_{\text{out}}^{p_j}$ which labels all the output channels of all processes p_j with $j \geq i$ and the alphabet $\Sigma_{\text{out}}^{b,p_i}$ which labels all the backward-channels and external output channels of process p_i. Moreover, for $0 \leq i < n$, Σ_i denotes the alphabet on the channels from p_i to p_{i+1}.

To prove decidability, we adopt the $\mathbb{B}$-labeled trees used in [10] to represent communication languages of the processes, that means, sets of infinite sequences of signals which can be sent along certain channels in the given architecture. Given an alphabet Σ, a $\mathbb{B}$-labeled Σ-tree t represents the ω-language $L^\omega(t) = \{\alpha \in \Sigma^\omega \mid t(\alpha\restriction_k) = \top \text{ for all } k \in \mathbb{N}\}$. Now if such a tree t represents in fact a communication language, then the $\top$-labelled nodes of t form a nonempty subtree of t, containing the root of t. More formally, t has the following properties: (C1) $t(\epsilon) = \top$, (C2) if $t(u) = \bot$, then $t(ua) = \bot$ for all $a \in \Sigma$ and (C3) if $t(u) = \top$, then $t(ua) = \top$ for some $a \in \Sigma$. We call $\mathbb{B}$-labeled Σ-trees which have the properties (C1) - (C3) communication trees over Σ and we denote the set of all such trees by $\mathbb{T}_c(\Sigma)$. Notice that for $t \in \mathbb{T}_c(\Sigma)$, $\{u \in \Sigma^* \mid t(u) = \top\} = \{u \in \Sigma^* \mid u \sqsubseteq \alpha \text{ for some } \alpha \in L^\omega(t)\}$.

Now, given a tree t_{in} which represents input sequences that a process receives and a tree t_{out} which represents output sequences that the process may write, we define the strategy product $t_{\text{in}} \hookrightarrow t_{\text{out}}$ of t_{in} and t_{out} as a set of trees t, each of which defines an assignment of input sequences from $L^\omega(t_{\text{in}})$ to output sequences from $L^\omega(t_{\text{out}})$, so it yields a strategy $\sigma(t)$ for the process. Formally, for a tree $t_{\text{in}} \in \mathbb{T}_c(\Sigma)$ and a tree $t_{\text{out}} \in \mathbb{T}_c(\Sigma')$, the strategy product $t_{\text{in}} \hookrightarrow t_{\text{out}}$ is defined as the set of all $\mathbb{B}$-labeled $\Sigma \times \Sigma'$-trees t such that: (S1) if $t_{\text{in}}(u) = \bot$ or $t_{\text{out}}(v) = \bot$ then $t(u^\frown v) = \bot$, (S2) if $t_{\text{in}}(u) = \top$ then there is exactly one $v \in (\Sigma')^{|u|}$ such that $t(u^\frown v) = \top$ and (S3) if $t(u^\frown v) = \top$ then there is some $b \in \Sigma'$ such that for all $a \in \Sigma$ with $t_{\text{in}}(ua) = \top$ we have $t(ua^\frown vb) = \top$ and $t(ua^\frown vc) = \bot$ for $c \in \Sigma' \setminus \{b\}$. Notice that in fact for any such t, $\text{Pr}_\Sigma(L^\omega(t)) = L^\omega(t_{\text{in}})$ and $\text{Pr}_{\Sigma'}(L^\omega(t)) \subseteq L^\omega(t_{\text{out}})$. Moreover, notice that $t_{\text{in}} \hookrightarrow t_{\text{out}} \subseteq \mathbb{T}_c(\Sigma \times \Sigma')$.

The key argument for the decidability result for pipelines with backward-channels is that a system process p_i is better informed than any system process p_j with $j > i$. In particular, a strategy for p_i needs only to depend on the input that it receives from the previous process p_{i-1} and not on the input received via backward-channels. Due to this observation, the following definition of an extended local strategy is meaningful. For $1 \leq i \leq n$, an extended local strategy for process p_i is a tuple $\sigma_{\geq i} = (\sigma_i, \ldots, \sigma_n)$ of functions $\sigma_j : (\Sigma_{i-1})^* \to \Sigma_{\text{out}}^{p_j}$, i.e., it takes the local input history of process p_i, ignoring the backward-channels read by p_i, and yields the next output symbol for each process p_j with $j \geq i$. Such a strategy is called locally winning on inputs from $L_{\text{in}} \subseteq \Sigma_{i-1}^{\omega}$, if each global system behavior α of $\mathfrak{A}$ with $\text{Pr}_{\Sigma_{i-1}}(\alpha) \in L_{\text{in}}$ which is consistent with $\sigma_{\geq i}$ fulfills L_{p_i}, i.e., $\text{Pr}_{\Sigma^{p_i}}(\alpha) \in L_{p_i}$. The main technical argument how these strategies can be used to decide the realizability problem for $\mathfrak{A}$ is given in the following Lemma.

Lemma 3. *For any $1 \leq i < n$ there is a parity NTA $\mathcal{N}_i$ over $\mathbb{B}$-labeled $\Sigma_{out}^{\geq i}$-trees which accepts a tree $t_{out} \in \mathbb{T}_c(\Sigma_{out}^{\geq i})$ if, and only if, there are a $\mathbb{B}$-labeled Σ_{i-1}-tree $t_{in} \in \mathbb{T}_c(\Sigma_{i-1})$, a tree $t \in t_{in} \hookrightarrow t_{out}$ and strategies $\sigma_1, \ldots, \sigma_{i-1}$ for processes $p_1, \ldots, p_{i-1}$ such that $\sigma(t)$ is locally winning on $L^{\omega}(t_{in})$ and*

- *$\sigma_1 \circ Pr_{\Sigma_1} \circ \ldots \circ \sigma_{i-1} \circ Pr_{\Sigma_{i-1}}$ generates a language $L \subseteq L^{\omega}(t_{in})$ over Σ_0^{ω}*
- *$(\sigma_1, \ldots, \sigma_{i-1}, \sigma(t))$ is winning for $p_1, \ldots, p_i$.*

Proof. (Sketch) We prove this by induction on i. We omit the base case $i = 1$ and consider only the case $i > 1$. For this, let $\mathcal{N}_{i-1}$ be a parity NTA over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq i-1}$-trees according to the induction hypothesis. Then there is a parity NTA $\mathcal{N}'_{i-1}$ over $\mathbb{B}$-labeled $\Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i}$-trees which accepts a tree t if, and only if, $\text{wide}(t, \Sigma_{\text{out}}^{b,p_{i-1}}) \in L(\mathcal{N}_{i-1})$. Now we construct a parity ATA $\mathcal{A}_i$ over $\mathbb{B}$-labeled $\Sigma_{\text{out}}^{\geq i}$-trees which, roughly, works as follows: Running on a tree t_{out}, in each step, $\mathcal{A}_i$ guesses an output signal $b \in \Sigma_{\text{out}}^{\geq i}$ and a set $\emptyset \neq X \subseteq \Sigma_{i-1}$ of possible input signals and sends, for each $(x, y) \in \Sigma_{i-1} \times \Sigma_{\text{out}}^{\geq i}$, a copy into direction y. If $x \in X$ and $y = b$, it sends a $\top$-copy, otherwise it sends a $\bot$-copy. Moreover, if $\mathcal{A}_i$ encounters a $\bot$-symbol in t_{out} when being in a $\top$-copy (that means, output b should not have been chosen in the previous step according to t_{out}), it immediately rejects. In this way, $\mathcal{A}_i$ guesses a tree $t_{\text{in}} \in \mathbb{T}_c(\Sigma_{i-1})$ and a tree $t \in t_{\text{in}} \hookrightarrow t_{\text{out}}$. While doing so, $\mathcal{A}_i$ simulates $\mathcal{N}'_{i-1}$ on t and it simulates a deterministic parity automaton recognizing L_{p_i} on all paths $\alpha \in L^{\omega}(t)$. Therefore, $\mathcal{A}_i$ accepts iff $t \in L(\mathcal{N}'_{i-1})$ and $\sigma(t)$ is locally winning on $L^{\omega}(t_{\text{in}})$. Now using the induction hypothesis, one can show that $\mathcal{A}_i$ fulfills all conditions of the lemma. Moreover, there is a parity NTA $\mathcal{N}_i$ with $L(\mathcal{N}_i) = L(\mathcal{A}_i)$, which concludes the proof. □

For the last process p_n, we need the following Lemma.

Lemma 4. *There is a parity NPDTA $\mathcal{N}_n$ over $\mathbb{B}$-labeled $\Sigma_{n-1} \times \Sigma_{out}^{p_n}$-trees which accepts a tree t if, and only if, $t \in t_{in} \hookrightarrow t_{out}$ for some $t_{in} \in \mathbb{T}_c(\Sigma_{n-1})$ and some $t_{out} \in \mathbb{T}_c(\Sigma_{out}^{p_n})$ such that $\sigma(t)$ is locally winning on $L^{\omega}(t_{in})$.*

Now by applying a widening argument for $\Sigma_{out}^{b,p_{n-1}}$, similar as in the proof of Lemma 3, one obtains an automaton $\mathcal{N}'_{n-1}$ from $\mathcal{N}_{n-1}$ and there is a parity NPDTA $\mathcal{N}$ recognizing the intersection of $L(\mathcal{N}'_{n-1})$ and $L(\mathcal{N}_n)$, where $\mathcal{N}_n$ is the parity NPDTA obtained from Lemma 4. Then one can show that $(L_{p_1}, \ldots, L_{p_n})$ is realizable in $\mathfrak{A}$ if, and only if, $L(\mathcal{N}) \neq \emptyset$ and as nonemptiness of $L(\mathcal{N})$ is decidable, the decidability result is established.

Theorem 5. *The realizability problem for $\mathfrak{A}$ is decidable if $L_{p_1}, \ldots, L_{p_{n-1}}$ are regular and L_{p_n} is regular or deterministic contextfree.*

Two-Flanked Pipelines with Backward-Channels. Let $\mathfrak{A} = (P, C, r)$ be a two-flanked pipeline with backward-channels, let $(\Sigma_c)_{c \in C}$ be a labeling of $\mathfrak{A}$ and let $L_{p_1}, \ldots, L_{p_n}$ be regular local specifications for the system processes.

First we consider the case where there are no backward-channels from the last process. The main idea and the constructions are essentially the same as for the case of pipelines with backward-channels. Clearly, the construction has to be adapted to account for the fact, that processes p_i for $i < n$ can determine the decisions of all processes p_j with $i \leq j < n$, but not those of process p_n. However, since these decisions are not relevant for the satisfaction of the local specification of p_i, it is not necessary that process p_i makes those decision.

So, an extended local strategy for p_i is now a tuple $\sigma_{\geq i} = (\sigma_i, \ldots, \sigma_{n-1})$ of functions $\sigma_j \colon (\Sigma_{i-1})^* \to \Sigma_{\text{out}}^{p_j}$ and the accumulated output alphabets $\Sigma_{out}^{\geq i}$ have to be adapted accordingly. Of course it has also to be taken into account, that a strategy for process p_n depends not only on the input from p_{n-1} but also on a part of the input from the environment.

Now Lemma 3 holds just as before with the new definition of $\Sigma_{\text{out}}^{\geq i}$ and the new notion of an extended local strategy. Lemma 4 however, has to be reformulated.

Lemma 6. *There is a parity NTA $\mathcal{N}_n$ over $\mathbb{B}$-labeled Σ_{n-1}-trees which accepts a tree $t_{in} \in \mathbb{T}_c(\Sigma_{n-1})$ if, and only if, there is a local strategy for process p_n which is locally winning on $\{\alpha \in (\Sigma_{in}^{p_n})^\omega \mid Pr_{\Sigma_{n-1}}(\alpha) \in L^\omega(t_{in})\}$.*

Theorem 7. *The realizability problem for regular local specifications is decidable for $\mathfrak{A}$ if there is no backward-channel from the last process p_n.*

Now we prove decidability for the case $n = 2$, where we also have backward-channels from the last process, so let $P = \{p_0, p_1, p_2\}$, let $\Gamma_i = \prod_{c \in C_0, r(c) = p_i} \Sigma_c$, $\Sigma_1 = \prod_{c \in C_1, r(c) = p_2} \Sigma_c$ and Σ_2 analogous. For convenience, we omit the external output channels of p_1 and p_2, it is straightforward how to account for these channels in the proof given below, if they are present.

For the proof, we extend the $\mathbb{B}$-labeled trees as here, ω-languages over $\Sigma = \Sigma_1 \times \Sigma_2$ are represented, but we want to maintain access to the components. A $\mathbb{B}^2$-labeled Σ-tree t represents the language $L^\omega(t) \subseteq \Sigma^\omega$ with $\alpha \in L^\omega(t)$ if, and only if, for each finite prefix w of α we have $t(w) = (\top, \top)$. However, more information is stored in such a tree: the components $\Pr_{\Sigma_i}(\alpha)$ may depend on each other which is expressed in the inidividual components of the $\mathbb{B}$-tuples.

If $t(u^\frown v) = (\top, \bot)$, then this tells us that $v\restriction_{|v|-1}$ may be answered by u but $u\restriction_{|u|-1}$ may not be answered by v, analogous for $t(u^\frown v) = (\bot, \top)$. Clearly this is different from just saying that $u^\frown v$ will not occur.

Of course, any such tree t which in fact represents a joint output language of p_1 and p_2 has properties analogous to the properties (C1) - (C3) of the communication trees $\mathbb{T}_c(\Sigma)$. However, as we don't need those properties in the decidability proof, we do not require them explicitly.

Theorem 8. *The realizability problem for regular local specifications is decidable for any two-flanked pipeline with backward-channels and $n = 2$.*

Proof. (Sketch) We construct two parity ATA $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\mathbb{B}^2$-labeled Σ-trees which, roughly, work as follows. When running on a tree t, at each step, $\mathcal{A}_1$ guesses an output signal $b \in \Sigma_1$ and sends, for any possible input signal $(x, y) \in \Gamma_1 \times \Sigma_2$, a copy into direction (b, y). If the corresponding node is labeled with $\bot$ in the Σ_1-component (that means, output b should not be chosen in this situation according to t), then $\mathcal{A}_1$ rejects immediately and if it is labeled with $\bot$ in the Σ_2-component (that means, input y will not occur in the next step according to t), $\mathcal{A}_1$ goes into a special accepting state. This way, $\mathcal{A}_1$ guesses a local strategy for p_1 on all inputs from Γ_1 and those inputs from Σ_2 which it may receive according to the intended joint strategy for p_1 and p_2. Moreover, on all paths which are consistent with this strategy, it simulates a deterministic parity automaton, recognizing the local specification L_{p_1}. The automaton $\mathcal{A}_2$ works analogously and thus one can show that $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \neq \emptyset$ iff a joint winning strategy for p_1 and p_2 exists. □

4.2 Undecidability

First, if there are at least two connected processes which may have a deterministic 1-counter specification, then those two processes can directly simulate a 2-register machine. As the halting problem for such machines is undecidable, we obtain the following result.

Theorem 9. *The realizability problem is undecidable for any connected architecture with at least two local deterministic 1-counter specifications.*

The next result restricts the decidable architectures which may have one deterministic 1-counter specification.

Theorem 10. *The realizability problem is undecidable for any architecture with two connected processes $p \neq p'$ such that p is reachable and has a deterministic 1-counter specification and p' is not better informed than p.*

Proof. (Sketch) We proceed by a reduction from the halting problem for 2-register machines, so let $\mathcal{R}$ be a 2-register machine. We only consider the architecture consisting of p_{env} and the two system processes p, p' where p_{env} sends information to p, p sends information to p' and p' has an external output channel. The proof idea can easily be adapted to all other cases. The local specification of p' requires that p' writes a sequence of configurations of $\mathcal{R}$ and that its output

symbol equals the input symbol that it receives from p in each step. Notice that those symbols have to be chosen simultaneously. In each step, the environment may trigger the 1-counter automaton recognizing L_p to check whether the next two configurations are in the successor relation w.r.t. one of the registers. For this, the environment has two special symbols s_j for $j \in \{1, 2\}$ where j determines which of the registers should be checked. Obviously, the system processes have a joint winning strategy iff $\mathcal{R}$ does not halt. □

Notice that to apply the automata constructions from the decidability proofs of Section 4.1 to an architecture as in Theorem 10, one has to construct alternating pushdown tree automata since the process with the pushdown specification is not the one with the lowest level of information. However, nonemptiness of alternating pushdown tree automata is not decidable.

The remaining two results concern regular local specifications and we just state them here without giving proofs. Essentially, those results can be proved by combining and refining ideas from [13], [5] and [10].

Theorem 11. *The realizability problem for regular local specifications is undecidable for any architecture with a reachable process p_1 such that p_1 sends information to processes $p_2 \neq p_3$ which are not better informed than p_1.*

Theorem 12. *The realizability problem for regular local specifications is undecidable for any architecture with at least two incomparably informed processes p_1 and p_2 which are both reachable such that there is a process $p_3 \notin \{p_1, p_2\}$ which is reachable from both p_1 and p_2.*

5 Characterization

Now we characterize the exact classes of architectures which are decidable for local regular specifications and, for each such decidable class, we determine the exact set of processes which may have a deterministic contextfree specification such that decidability still holds. Notice that an architecture which is already undecidable for regular local specifications is clearly undecidable if we additionally allow deterministic contextfree specifications. Since for local specifications the realizability problem for some architecture is decidable if, and only if, it is decidable for every connected subarchitecture, w.l.o.g. we restrict our attention to connected architectures. We also note that in the case of local specifications the hidden channels of the environment are futile.

In the following, we denote the class of all pipelines with backward-channels by $\mathcal{K}_1$ and the class of all two-flanked pipelines with backward-channels which have either only two system processes or which do not have a backward-channel from the last process by $\mathcal{K}_2$. Moreover, for an architecture $\mathfrak{A}$, $M(\mathfrak{A})$ denotes the set of all system processes of $\mathfrak{A}$ which are not reachable. Notice that $P_{sys} \setminus M(\mathfrak{A})$ induces a subarchitecture of $\mathfrak{A}$.

Theorem 13. *Let $\mathfrak{A} = (P^{\mathfrak{A}}, C^{\mathfrak{A}}, r^{\mathfrak{A}})$ be a connected architecture. Then $\mathfrak{A}$ is decidable for regular local specifications if, and only if, any connected subarchitecture*

of $\mathfrak{A}(P^{\mathfrak{A}}_{sys} \setminus M(\mathfrak{A}))$ *is in* $\mathcal{K}_1 \cup \mathcal{K}_2$*. Moreover,* $\mathfrak{A}$ *remains decidable for deterministic contextfree specifications if, and only if, one of the following conditions hold.*

(1) $\mathfrak{A} \in \mathcal{K}_1$ *and* $L_{p_1}, \ldots, L_{p_{n-1}}$ *are regular.*
(2) There is a $p \in M(\mathfrak{A})$ *such that* $L_{p'}$ *is regular for all* $p' \in P^{\mathfrak{A}}_{sys} \setminus \{p\}$*.*

Proof. (Sketch) We prove here only that $\mathfrak{A}$ is decidable, if any connected subarchitecture of $\mathfrak{A}(P^{\mathfrak{A}}_{sys} \setminus M(\mathfrak{A}))$ is in $\mathcal{K}_1 \cup \mathcal{K}_2$ (see Figure 2) and there is a $p \in M(\mathfrak{A})$ such that $L_{p'}$ is regular for all $p' \in P_{sys} \setminus \{p\}$. For this, we define $M := M(\mathfrak{A})$ and $C_M := \bigcup_{p \in M} C_p$ and we consider some labeling $(\Sigma_c)_{c \in C}$ of $\mathfrak{A}$.

First, let $\tilde{L}_p = \{\alpha \in \Sigma_M^\omega \mid \mathrm{Pr}_{\Sigma^p}(\alpha) \in L_p\}$ for $p \in M$ and let $\tilde{L}_M = \bigcap_{p \in M} \tilde{L}_p$. Then $\alpha_M \in \tilde{L}_M$ iff there is a joint strategy $\sigma_M = (\sigma_p)_{p \in M}$ for the processes in M such that any global system behavior β of $\mathfrak{A}$ which is consistent with σ_M fulfills all local specifications of the processes in M and $\mathrm{Pr}_{\Sigma_M}(\beta) = \alpha_M$. Moreover, as at most one specification L_p for $p \in M$ is deterministic contextfree and all others are regular, $\tilde{L}_M$ is deterministic contextfree.

Now consider any connected subarchitecture $\mathfrak{B} = (P, C, r)$ of $\mathfrak{A}(P_{sys} \setminus M)$ and let $C_{M \to \mathfrak{B}} = C_M \cap (r^{\mathfrak{A}})^{-1}(P)$. As all specifications L_p for $p \in P^{\mathfrak{A}}_{sys} \setminus M$ are regular, it suffices to consider the case where $\mathfrak{B}$ is a two-flanked pipeline with backward-channels and here, we only consider the case where $\mathfrak{B}$ has no backward-channel from the last process. We define the architecture $\widehat{\mathfrak{B}} = (P, \widehat{C}, \widehat{r})$ as follows. The channels from M to P are simulated by new channels $C_{M \to \mathfrak{B}} \subseteq \widehat{C}$ of process p_{n-1}, via which p_{n-1} sends information to the respective receipients of the original channels. Moreover, p_{n-1} has a set of duplicate channels $C^d_{M \to \mathfrak{B}}$ which are read by process p_n and the specification $L_{p_{n-1}}$ requires that the information sent along the channels $C_{M \to \mathfrak{B}}$ is the same as the information sent along the channels $C^d_{M \to \mathfrak{B}}$.

Now let $\mathcal{N}_{n-1}$ be the parity NTA over $\mathbb{B}$-labeled $\widehat{\Sigma}^{\geq n-1}_{\mathrm{out}}$-trees according to Lemma 3, applied to $\widehat{\mathfrak{B}}$. Then there is a parity NTA $\mathcal{N}'_{n-1}$ over $\mathbb{B}$-labeled $\widehat{\Sigma}_{n-1}$-trees which accepts a tree t iff $\mathrm{wide}(t, \widehat{\Sigma}^{b,p_{n-1}}_{\mathrm{out}}) \in L(\mathcal{N}_{n-1})$. Moreover, let $\mathcal{N}_n$ be the parity NTA over $\mathbb{B}$-labeled $\widehat{\Sigma}_{n-1}$-trees according to Lemma 6, applied to $\widehat{\mathfrak{B}}$. Then there is a parity NTA $\mathcal{N}^{\mathfrak{B}}$ over $\mathbb{B}$-labeled $\widehat{\Sigma}_{n-1}$-trees such that $L(\mathcal{N}^{\mathfrak{B}}) = L(\mathcal{N}'_{n-1}) \cap L(\mathcal{N}_n)$. Furthermore, one can construct an alternating parity automaton $\mathcal{A}^{\mathfrak{B}}$ such that $\mu \in L(\mathcal{A}^{\mathfrak{B}})$ iff there is a tree $t \in \mathbb{T}_c(\widehat{\Sigma}_{n-1})$ such that $\mathrm{Pr}_{\Sigma^d_{M \to \mathfrak{B}}}(L^\omega(t)) = \{\mu\}$ and $t \in L(\mathcal{N}^{\mathfrak{B}})$. Using this, one can furthermore show that $\mu \in L(\mathcal{A}^{\mathfrak{B}})$ iff there is a strategy $\sigma = (\sigma_1, \ldots, \sigma_n)$ for processes $p_1, \ldots, p_n$ in the architecture $\mathfrak{A}$ such that any global system behavior β of $\mathfrak{A}$ which is consistent with σ and fulfills $\mathrm{Pr}_{\Sigma_{M \to \mathfrak{B}}}(\beta) = \mu$, fulfills all local specifications of the processes $p_1, \ldots, p_n$.

Finally, for any connected subarchitecture $\mathfrak{B}$ of $\mathfrak{A}(P_{sys} \setminus M)$ let $\tilde{L}(\mathcal{A}^{\mathfrak{B}}) = \{\alpha \in \Sigma_M^\omega \mid \mathrm{Pr}_{\Sigma_{M \to \mathfrak{B}}}(\alpha) \in L(\mathcal{A}^{\mathfrak{B}})\}$ and $L = \bigcap_{\mathfrak{B}} \tilde{L}(\mathcal{A}^{\mathfrak{B}}) \cap \tilde{L}_M$. Then $L \neq \emptyset$ iff the system processes have a joint winning strategy. As L is deterministic contextfree, emptiness of L can be decided. □

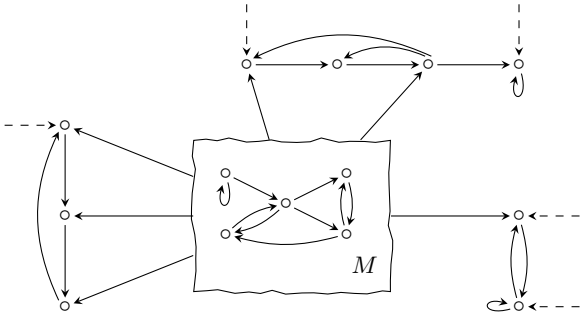


Fig. 2. Generic decidable architecture with non-reachable processes M

References

1. Büchi, J.R., Landweber, L.H.: Solving Sequential Conditions by Finite-State Strategies. Trans. Amer. Math. Soc. 138, 367–378 (1969)
2. Church, A.: Applications of Recursive Arithmetic to the Problem of Circuit Synthesis. Sum. of the Sum. Inst. of Symb. Log. I, 3–50 (1957)
3. Cohen, R.S., Gold, A.Y.: Theory of Omega-Languages. I. Characterizations of Omega-Context-Free Languages. J. Comput. Syst. Sci. 15(2) (1977)
4. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent Reachability Games. In: FOCS 1998 (1998)
5. Finkbeiner, B., Schewe, S.: Uniform Distributed Synthesis. In: LICS 2005, pp. 321–330. IEEE, Los Alamitos (2005)
6. Finkel, O.: Topological Properties of Omega Context-Free Languages. Theor. Comput. Sci. 262(1) (2001)
7. Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown Specifications. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, Springer, Heidelberg (2002)
8. Kupferman, O., Vardi, M.Y.: Church's Problem Revisited. Bulletin of Symbolic Logic 5(2) (1999)
9. Kupferman, O., Vardi, M.Y.: Synthesizing Distributed Systems. In: LICS 2001 (2001)
10. Madhusudan, P., Thiagarajan, P.S.: Distributed controller synthesis for local specifications. In: Yu, Y., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 396–407. Springer, Heidelberg (2001)
11. Pnueli, A.: The Temporal Logic of Programs. In: FOCS 1977. IEEE, Los Alamitos (1977)
12. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: POPL 1989, pp. 179–190 (1989)
13. Pnueli, A., Rosner, R.: Distributed Reactive Systems are Hard to Synthesize. In: FOCS 1990, pp. 746–757. IEEE, Los Alamitos (1990)
14. Rabin, M.O.: Automata on Infinite Objects and Church's Problem. Amer. Math. Soc., Providence (1972)
15. Safra, S.: On the Complexity of Omega-Automata. In: FOCS 1988 (1988)
16. Walukiewicz, I.: Pushdown Processes: Games and Model Checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 62–74. Springer, Heidelberg (1996)

Geometric Graphs with Randomly Deleted Edges - Connectivity and Routing Protocols

K. Krzywdziński and K. Rybarczyk

Adam Mickiewicz University, Poznań, Poland

Abstract. In the article we study important properties of random geometric graphs with randomly deleted edges which are natural models of wireless ad hoc networks with communication constraints. We concentrate on two problems which are most important in the context of theoretical studies on wireless ad hoc networks. The first is how to set parameters of the network (graph) to have it connected. The second is the problem of an effective message transmition i.e. the problem of construction of routing protocols in wireless networks. We provide a thorough mathematical analysis of connectivity property and a greedy routing protocol. The models we use are: an intersection of a random geometric graph with an Erdős–Rényi random graph and an intersection of a random geometric graph with a uniform random intersection graph. The obtained results are asymptotically tight up to a constant factor.

1 Introduction

1.1 Background

A random geometric graph is a widely used model of wireless ad hoc networks. A wireless ad hoc network is a collection of devices randomly deployed over a given area and equipped with radio transmitters. This type of networks have proved to have many applications in environment and industrial monitoring, security, cell phone networks etc. It is generally assumed that in wireless ad hoc networks direct communication between two nodes is possible if they are mutually in their transmission range, thus the usage of random geometric graphs seems natural. In random geometric graphs vertices are located in the Euclidean plane according to a given probability distribution and two vertices v and w are connected by an edge if their Euclidean distance is not greater than some constant r (for a monograph on random geometric graphs see [21]). The vertices of the model represent devices and edges arise if appropriate devices are in mutual transmission range. A random geometric graph, though concise enough to promote strong theoretical results, frequently does not apply directly to networks in realistic settings. In fact, wireless links in real networks may be extremely unreliable. Moreover, the connections are sometimes constrained by security requirements. In reality the message transmission between devices may be obstructed and the direct communication is possible only if additional conditions are also fulfilled. Therefore it is justified to introduce new graph models which

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 544–555, 2011.

are considerably closer to reality than a random geometric graph. In the context of theoretical studies on wireless networks two questions concerning new models seem to be of main importance: how to set parameters of the network to have it connected and whether routing protocols are effective in the network. Both of them are related to the message transmission problem in wireless ad hoc networks, which is the main subject of this article.

1.2 Related Work

The most important results on connectivity of random geometric graphs were obtained by Gupta and Kumar in [14] and Penrose (see [21] and references therein). In [14] Gupta and Kumar considered a random graph in which n vertices are uniformly distributed in a disk of a unit area and two vertices are connected by an edge if they are at distance at most $r(n)$. It was shown that if $\pi r^2(n) = (\ln n + c(n))/n$ and $c(n) \to \infty$ then the probability that an instance of the considered random graph model is connected tends to 1 as n tends to infinity. In concluding remarks Gupta and Kumar proposed a generalised random geometric graph, which models wireless ad hoc network with independent link constraints. In the model two vertices at distance at most $r(n)$ are connected with probability $p(n)$. Gupta and Kumar conjectured that if

$$\pi p(n) r^2(n) = \frac{\ln n + c(n)}{n} \tag{1}$$

and $c(n) \to \infty$ then a random geometric graph with independently deleted edges is connected with probability tending to one as $n \to \infty$. Properties of the model were later studied for example in [9] and [26]. In [26] Yi et al. showed a Poisson approximation of the number of isolated nodes in a random geometric graph with independently deleted edges for r and p as in (1) and $c(n) \to c$. The result implies that as $c(n) \to -\infty$ a random geometric graph with independently deleted edges is disconnected with probability tending to one as $n \to \infty$. To the best of our knowledge no sufficient conditions for connectivity are known. The model proposed by Gupta and Kumar is also related to the so–called "bluetooth model" (see [22] and references therein). However the results proved in this article are much stronger than those which would follow from the results concerning "bluetooth model" and the models comparison.

Sometimes link constraints in a wireless network are not independent and another model should be used. This is the case of wireless sensor networks with random key predistribution. The widely recognised solution to the problem of security of the message transmission in those networks is so–called random key predistribution introduced in [12]. So far, the studies on wireless sensor networks with random key predistribution reveal either experimental results (see for example [6,12,15]) or concentrate only on the security aspects, neglecting a limited transmission range (i.e. properties of uniform random intersection graphs are studied, see for example [1,2,8,23]). As it is pointed out in [8] the model with both security and transmission limitations is an intersection of a random geometric graph and a uniform random intersection graph. Its variant has only

recently been analysed in rigorous mathematical manner in [17]. However in [17] an additional assumption on deployment knowledge is made and the techniques used cannot be applied to the model studied in this article.

The connectivity of a wireless network implies possibility of the message transmition from any device to any other device. However it yields another question: how this message should be send? This is a folklore result that the simplest method of the message transmission in wireless ad hoc network is forwarding a routed message to the neighbour closest (in the sense of Euclidian distance, angle or other) to the destination. Such greedy forwarding, basing on the topology of the network graph, is called geographic routing. In geographic routing two main assumptions are made. The first one, that each network device possesses information about its own and about its neighbours positions. The second, that the source of a message is provided with the position of the destination. As any device knows only about its immediate neighbours, there is often insufficient information for it to make a good decision on the forwarding direction and a packet may get trapped. Already numerous attempts to enhance the greedy algorithm in general setting have been proposed (see for example [5,19,20]). Routing in the network with unreliable links has been also studied (see for example [27]). The mathematical analysis of the greedy routing protocol in random geometric graph was the subject of [25]. However, to the best of our knowledge none of the results considered analytical analysis of the greedy routing protocol in wireless networks with unreliable links.

1.3 Our Contribution

In the article we analyse in detail theoretical models of wireless networks with independent link constraints and wireless sensor networks with random key predistribution. We maintain the assumption that in the graph model a link between vertices is possible if they are at distance at most r, at the same time making an additional one, that the link may be obstructed with a certain probability. We analyse in detail asymptotic properties of two models. First of the considered models is a random geometric graph with independently disappearing edges, proposed in [14]. In the second model the edge deletion depends on the structure of a uniform random intersection graph. More precisely, we analyse the models which are an intersection of a random geometric graph with an Erdős–Rényi random graph (see [14]) and an intersection of a random geometric graph with a uniform random intersection graph (see [8]).

For the first time we present a thorough mathematical analysis of a greedy routing protocol in wireless networks with link constraints. Moreover we provide a solution to the conjecture posed in [14] concerning connectivity of the random geometric graph model with independently deleted edges (an intersection of a random geometric graph with Erdős–Rényi random graph). In Theorem 4 an answer is given and the obtained result is tight up to a constant factor. In addition in Theorem 3 we state sufficient conditions for connectivity of the model of wireless sensor network with random key predistribution (an intersection of a random geometric graph with a uniform random intersection graph). This is

the first analytical result which takes into account both security and transmission range constraints in this type of networks. Moreover we show how to set parameters of the network with communication constraints in such way that the greedy routing protocol effectively delivers the message to the destination. We also set an upper bound on the length of transmission path in the presented greedy routing protocol.

1.4 The Article Organisation

The article is organised as follows. In Section 2 we introduce the models and provide a version of the greedy protocol which is considered in the article. Section 3 discusses the routing protocol in networks modelled by a graph with independent communication constraints, whereas Section 4 shows a sketch of analogous results for the model of wireless sensor network with random key predistribution.

2 Preliminaries

2.1 Definitions of the Models

In further considerations we make an assumption that $\mathbb{D}$ is a disk of a unit area. However it should be pointed out that the results apply also to a wider class of areas after rescaling and repeating the proofs. We assume that $V = V_n$ is a set of n points independently, uniformly distributed on $\mathbb{D}$, $0 < r = r(n) < 1/\sqrt{\pi}$, $p = p(n) \in [0;1]$ are real numbers and $d = d(n)$ is a positive integer. The assumption $r(n) \leq 1/\sqrt{\pi}$ is a natural one, since otherwise radius of $\mathbb{D}$ would be smaller than transmission range in the modelled network. Moreover, for any two points $v, v' \in \mathbb{D}$, by $\|v, v'\|$ we denote their Euclidian distance.

Let $G_r(n)$ be a random geometric graph, i.e. a graph with the vertex set $V = V_n$ and the edge set $E(G_r(n)) = \{vv' : v, v' \in V \text{ and } \|v, v'\| \leq r\}$.

We denote by $G_r(n,p)$ an instance of the graph model suggested by Gupta and Kumar in [14]. A random graph $G_r(n,p)$ is obtained form $G_r(n)$ by independently deleting each edge with probability $1-p$ (i.e. each edge stays in a graph independently with probability p). Notice that in $G_r(n,p)$ the edge set is an intersection of the edge sets of $G_r(n)$ and an Erdős–Rényi random graph $G(n,p)$, in which each edge appears independently with probability p (see [3,16]).

The second considered model represents a wireless sensor network with random key predistribution. As noticed in [8] (see also [17]) it is an intersection of $G_r(n)$ and a uniform random intersection graph $\mathcal{G}(n,m,d)$. In a uniform random intersection graph $\mathcal{G}(n,m,d)$ each vertex $v \in V$ chooses a subset from an auxiliary arbitrarily given feature set $W = W_n$ of cardinality $m = m(n)$. More precisely, given $d = d(n)$, each vertex $v \in V$ is attributed a subset of features $S(v)$ chosen uniformly at random from all $d(n)$–element subsets of W. A uniform random intersection graph is a graph with the vertex set $V = V_n$ and the edge set $E(\mathcal{G}(n,m,d)) = \{vv' : v, v' \in V \text{ and } S(v) \cap S(v') \neq \emptyset\}$. In fact $\mathcal{G}(n,m,d)$ is a variant of a widely studied random intersection graph. The model of a random intersection graph was studied for the first time in [18,24] and its generalised

version was introduced in [13]. By $\mathcal{G}_r(n, m, d)$ we denote a graph with the vertex set V_n and an edge set $E(G_r(n)) \cap E(\mathcal{G}(n, m, d))$, where $G_r(n)$ and $\mathcal{G}(n, m, d)$ are independent (i.e. their edge sets are constructed independently).

2.2 Greedy Routing Protocol

Notice that in $G_r(n)$ a set of n' vertices that are mutually at distance at most r form a clique. In contrast those vertices in $G_r(n, p)$ and $\mathcal{G}_r(n, m, d)$ behave like $G(n', p)$ and $\mathcal{G}(n', m, d)$, respectively. Therefore, in $G_r(n, p)$ and $\mathcal{G}_r(n, m, d)$, in a close neighbourhood of the destination the network does not have any features of a random geometric graph and the geographic routing would fail. We propose a new routing protocol, which is a combination of a modified geographic compass routing and the standard Breadth First Search procedure (BFS). Recall that in geographic routing we assume that: each network device possesses information about its own and about its neighbours positions and the source of a message is provided with the position of the destination.

The algorithm parameter $0 < \varepsilon < 1/2$ is chosen prior to algorithm start and remains constant throughout the execution. Let G be a graph representing the network and r be a transmission range. Denote by s the source, by t the destination and by $G[t]$ a subgraph of G induced on the vertices at distance at most r from t (in the transmission range of t). The algorithm CompassPlus is as follows:

First s has a token, which contains a message.

1. Denote by v a vertex in possession of the token.
2. (Compass Routing Mode) If $\|v, t\| > r$, then send the token to a neighbour w of v at distance at least εr, such that the measure of the angle $\angle tvw$ is minimised.
3. (BFS Routing Mode) If $\|v, t\| \leq r$, then use BFS on $G[t]$ to route the message (i.e. use a path established by the Breath First Search procedure).

3 Geographic Routing in $G_r(n, p)$

In further considerations by the length of a path we mean the number of its edges in the graph. Moreover by $D(v)$ we denote a disk of radius r and centre in a point $v \in \mathbb{D}$. In this section we prove the following theorem.

Theorem 1. *Let $C > 8$, $p = p(n) \in [0; 1]$ and $r = r(n) \in (0; 1/\sqrt{\pi}]$. If*

$$\pi r^2 p \geq C \frac{\ln n}{n}, \tag{2}$$

then $G_r(n, p)$ is connected with probability tending to 1 as $n \to \infty$.

Moreover if $0 < \varepsilon < \min\{\sqrt{1 - 8/C}, 1/2\}$, then with probability tending to 1 as $n \to \infty$ CompassPlus *delivers information in $G_r(n, p)$ in at most $2\|s, t\|/(\varepsilon r(n)) + \ln(\pi r^2 n/6)/\ln\ln(\pi r^2 n/6) + 4$ steps between any two vertices s, t in some set of size $n(1 - o(1))$.*

It follows by the result from [26] that if $\pi r^2 p \leq C\frac{\ln n}{n}$ and $C < 1$ then with probability $1-o(1)$ $G_r(n,p)$ is disconnected. Therefore r and p fulfilling equation

$$\pi r^2 p = C\frac{\ln n}{n}. \tag{3}$$

establish a threshold function for the connectivity and efficient delivery property in $G_r(n,p)$ and the obtained result is tight up to a constant factor. Since augmenting value of p or r may only make a graph more connected in further considerations we concentrate on the case given by (3) for $C > 8$.

The proof of Theorem 1 proceeds as follows. In Lemma 1 we prove that with probability close to one during Compass Routing Mode of CompassPlus a message is delivered from s to some vertex contained in $D(t)$. Moreover we give an upper bound on the length of the path that a message have to pass from s to a vertex in $D(t)$. In Lemma 2 we show that with probability $1-o(1)$ the message may be sent from any vertex from $D(t)$ to t using BFS Routing Mode of CompassPlus and we give an upper bound on the length of the path. In Lemmas 3 and 4 we finish the proof of the first part of Theorem 1. Lemma 5 implies the second part of Theorem 1.

Lemma 1. *Let $0 < \varepsilon \leq 1/2$ and (3) with $C > 8$ be fulfilled. Then with probability at least $1 - O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right)$ for any $s \in V(G_r(n,p))$ and any point $t \in \mathbb{D}$ CompassPlus in Compass Routing Mode constructs in $G_r(n,p)$ a path of length at most $2\|s,t\|/(\varepsilon r(n))$ between s and some vertex contained in $D(t)$.*

Proof. Denote by $\mathcal{A}$ the event: "For all $v,t \in V(G_r(n,p))$ there exists w such that, w is a neighbour of v at distance at least εr from v, $\angle tvw$ is minimised and

$$\|w,t\| < \|v,t\| - (\varepsilon r)/2." \tag{4}$$

Notice that

$$\mathbb{P}(\mathcal{A}) = 1 - O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right) \tag{5}$$

implies the thesis. Namely $\mathcal{A}$ implies that for all $s,t \in V(G_r(n,p))$ all steps of Compas Routing Mode of CompassPlus are possible and in each step a distance between t and a vertex in possession of the message is shortened by $(\varepsilon r)/2$. Let $s = v_0, v_1, \ldots, v_l \in D(t)$ be a path constructed by Compas Routing Mode. If for all $1 \leq i \leq l$ we have $\|v_i,t\| < \|v_{i-1},t\| - (\varepsilon r)/2$, thus inductively $\|v_l,t\| < \|v_0,t\| - l(\varepsilon r)/2 = \|s,t\| - l(\varepsilon r)/2$. Therefore $l \leq 2\|s,t\|/(\varepsilon r(n))$ and the constructed path is of length at most $2\|s,t\|/(\varepsilon r(n))$.

Therefore it remains to prove (5). Notice that $D(v)$ may be divided by 4 lines coming through v (4 diameters of $D(v)$ pairwise creating angles 45^o, 90^o and 135^o) into 8 equal parts. All parts are $1/8$ of the disk $D(v)$ and all of them may be enumerated by $1,\ldots,8$. Given the division, if k–th part is contained in $\mathbb{D}$ we denote by $D_{k,\varepsilon}(v)$ a subset of its points, which are at distance at least εr from v (see Figure 1). We also have to take into consideration border conditions. Therefore if k–th part of the division is not contained entirely in $\mathbb{D}$, we rotate

it around v until it is contained in $\mathbb{D}$ and then denote by $D_{k,\varepsilon}(v)$ its subset of points, which are at distance at least εr from v (see $D_{5,\varepsilon}(v')$ on Figure 1). Notice that either $t \in D(v)$ or the segment (v,t) intersects at least one of the sets $D_{k,\varepsilon}(v)$, $k = 1,\ldots,8$. Therefore either $t \in D(v)$ (equivalently $v \in D(t)$) or the next vertex in possession of the token should lie on some sector $D_{k,\varepsilon}(v)$. For example on Figure 1, three vertices in $D(v')$ do not belong to any sector $D_{k,\varepsilon}(v)$, but they cannot be intermediate vertices in Compass Routing Mode. However they may be t. Moreover, if $t \notin D(v)$, (v,t) intersects $D_{k,\varepsilon}(v)$ and $w \in D_{k,\varepsilon}(v)$ then w is at distance at least εr from v and $\angle tvw$ is at most 45^o (see Figure 2).

Now we bound the probability that for all $v \in V$ and all $1 \le k \le 8$ there exists a neighbour of v in $D_{k,\varepsilon}(v)$. Let $N_{k,\varepsilon}(v)$ be the set of neighbours of v in $G_r(n,p)$ contained in $D_{k,\varepsilon}(v)$. Then for any constant $0 \le \varepsilon < 1/2$

$$\begin{aligned}
\mathbb{P}\left(\exists_v \exists_k N_{k,\varepsilon}(v) = \emptyset\right) &\le \sum_{v\in V} \sum_{1\le k\le 8} \mathbb{P}\left(N_{k,\varepsilon}(v) = \emptyset\right) \\
&\le 8n\left(1 - \tfrac{(1-\varepsilon^2)\pi r^2 p}{8}\right)^{n-1} \\
&\le (1+o(1))8\exp\left(-n\tfrac{(1-\varepsilon^2)C\ln n}{8n} + \ln n\right) \\
&= O\left(n^{-\frac{C(1-\varepsilon^2)-8}{8}}\right).
\end{aligned}$$

Therefore with probability $1 - O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right)$ for all $v \in V$ and all $1 \le k \le 8$ a set $D_{k,\varepsilon}(v)$ contains a neighbour of v, i.e.

$$\mathbb{P}\left(\forall_v \forall_k N_{k,\varepsilon}(v) \neq \emptyset\right) = 1 - O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right).$$

Now we shall prove that event $\forall_v \forall_k N_{k,\varepsilon}(v) \neq \emptyset$ implies event $\mathcal{A}$. If event $\forall_v \forall_k N_{k,\varepsilon}(v) \neq \emptyset$ occurs then for all $v,t \in V$ such that $\|v,t\| \ge r$ a step in Compas Routing Mode of CompassPlus is always possible. Now it remains to show that under $\forall_v \forall_k N_{k,\varepsilon}(v) \neq \emptyset$, if in a step w receives the message form v then (4) is true. Consider an instance of $G_r(n,p)$ such that for all $v \in V$ and

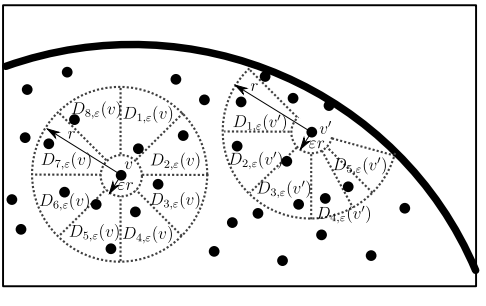


Fig. 1. Division into $D_{k,\varepsilon}(v)$

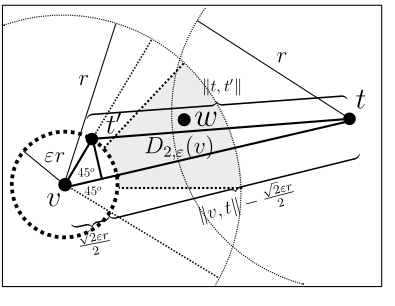


Fig. 2. One step of CompassPlus

all $1 \le k \le 8$ there exists a neighbour of v in $D_{k,\varepsilon}(v)$ (i.e. for any v and any k a set $N_{k,\varepsilon}(v)$ is nonempty). Let k be such that the segment (v,t) intersects $D_{k,\varepsilon}(v)$. By above assumptions there exists w – a neighbour of v, in $D_{k,\varepsilon}(v)$. Let t' be a point at distance εr from v and such that $\angle tvt' = 45^o$ (see Figure 2). It is easy to see that

$$\text{either } D_{k,\varepsilon}(v) \subseteq D(t) \text{ (i.e. } w \in D(t)) \quad \text{ or } \|w,t\| \le \|t',t\|.$$

Therefore for any v and w, if $\|v,t\| \ge r$ and $0 < \varepsilon < 1/2$, then (see Figure 2)

$$\begin{aligned}\|w,t\|^2 &\le \|t',t\|^2 \le \left(\|v,t\| - \tfrac{\sqrt{2}\varepsilon r}{2}\right)^2 + \tfrac{1}{2}(\varepsilon r)^2 \\ &= \left(\|v,t\| - \tfrac{\varepsilon r}{2}\right)^2 - \varepsilon r\left((\sqrt{2}-1)\|v,t\| - \tfrac{3}{4}\varepsilon r\right) < \left(\|v,t\| - \tfrac{\varepsilon r}{2}\right)^2.\end{aligned}$$

Thus if $\|v,t\| \ge r$ and $\varepsilon < 1/2$, then

$$\|w,t\| < \|v,t\| - (\varepsilon r)/2.$$

Thus $\forall_v \forall_k N_{k,\varepsilon}(v) \ne \emptyset$ implies $\mathcal{A}$ and we have

$$\mathbb{P}(\mathcal{A}) \ge \mathbb{P}(\forall_v \forall_k N_{k,\varepsilon}(v) \ne \emptyset) = 1 - O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right). \qquad \square$$

A simple corollary of the above lemma is the result stated below. It is only a little less tight than this from [25] but may be shown by much simpler methods. Moreover here the parameter $o(1)$ from [25] is replaced by $O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right)$ (constant may be calculated effectively after careful insight into the proof of Lemma 1) and the length of the path is estimated.

Theorem 2. *Let $0 < \varepsilon \le 1/2$ and $\pi r^2 = C \ln n/n$, where $C > 8$. Then with probability at least $1 - O\left(n^{-(C(1-\varepsilon^2)-8)/8}\right)$ for any $s,t \in V(G_r(n))$* CompassPlus *constructs in $G_r(n)$ a path of length at most $2\|s,t\|/(\varepsilon r(n)) + 1$ between s and t.*

Proof. Follows immediately by Lemma 1 and fact that $G_r(n)$ is $G_r(n,1)$. $\square$

Now we concentrate on the BFS Routing Mode. In fact we prove that with probability close to one $G[t]$ is connected and have small diameter (as a graph) for almost all $t \in V(G_r(n,p))$. This proves that in BFS Routing Mode of CompassPlus the message is delivered in short time.

For any graph G and any vertices $v, v' \in V(G)$, by $dist(v,v')$ we denote their distance in G (i.e the length of the shortest path between v and v' in G).

Lemma 2. *Let $C > 8$ and (3) be fulfilled, $v \in V(G_r(n,p))$ and $G[v]$ be a subgraph of $G_r(n,p)$ induced on vertices contained in $D(v)$. Then with probability $1 - o(1)$ (where $o(1)$ is uniformly bounded over all $v \in V(G_r(n,p))$) $G[v]$ is connected and for all $v' \in V(G[v])$ we have*

$$dist(v',v) \le \ln(\pi r^2 n/6)/\ln\ln(\pi r^2 n/6) + 4.$$

Proof. For any $v \in V(G_r(n,p))$ the figure $\mathbb{D} \cap D(v)$ may be divided into 6 (or less) connected, not necessarily disjoint parts of diameter (in the sense of Euclidian norm) at most r and area $\pi r^2/6$, such that each of them contains v. For example if $D(v) \subseteq \mathbb{D}$, then $D(v)$ may be divided into 6 equal parts by 3 lines containing v pairwise making angle 60^o. Denote by $G_i(v)$, $1 \le i \le 6$, a subgraph of $G_r(n,p)$ induced on the vertices contained in the i-th part of the division.

For any $v \in V(G_r(n,p))$ and $1 \le i \le 6$ denote by $N = N(i,v) = |V(G_i(v)) \setminus \{v\}|$ a random variable counting the vertices of $G_i(v)$ except v. Surely N is binomialy distributed $\mathrm{Bin}(n-1, P)$, where $P = \pi r^2/6$. Therefore

$$\mathbb{E}N = (n-1)P = (1+o(1))nr^2\pi/6 \to \infty, \tag{6}$$

$$\mathrm{Var}N = P(1-P)(n-1) \le \mathbb{E}N. \tag{7}$$

Let $J = \left[(\pi r^2(n-1))(1-(nr^2)^{-1/3})/6 \,;\, (\pi r^2(n-1))(1+(nr^2)^{-1/3})/6 \right]$.

By Chebyshev's inequality

$$\mathbb{P}(N \notin J) = \mathbb{P}\left(|N - \mathbb{E}N| > \mathbb{E}N(nr^2)^{-1/3}\right) \le \tfrac{VarN}{(\mathbb{E}N)^2(nr^2)^{-2/3}} = o(1).$$

For all $\bar{n} \in J$ we have $\bar{n} = (1+o(1))\pi nr^2/6$ and by (3)

$$p = \frac{C}{6}\frac{\ln n}{\frac{\pi r^2 n}{6}} \ge \frac{4}{3}\frac{\ln \frac{\pi r^2 n}{6}}{\frac{\pi r^2 n}{6}} = (1+o(1))\frac{4}{3}\frac{\ln(\bar{n}+1)}{\bar{n}+1}.$$

Denote by $\mathcal{A}_G$ an event that $\mathrm{diam}G \le \ln(\pi r^2 n/6)/\ln\ln(\pi r^2 n/6) + 4$ and G is connected. Notice that all the vertices in $G_i(v)$ are pairwise in transmission range. Therefore, since $4/3 > 1$, by classical results on the connectivity and the diameter of an Erdős–Rényi random graph (see [3,4,7,10,11]) for $G_i = G_i(v)$ we have $\mathbb{P}\left(\mathcal{A}_{G_i} \;\middle|\; N = \bar{n}\right) = 1 - o(1)$, where $o(1)$ is uniformly bounded over all values $\bar{n} \in J$. Denote by $\mathcal{A}^C_{G_i}$ a complement of the event $\mathcal{A}_{G_i}$, then

$$\mathbb{P}\left(\mathcal{A}^C_{G_i}\right) \le \textstyle\sum_{\bar{n}\in J} \mathbb{P}\left(\mathcal{A}^C_{G_i} \;\middle|\; N = \bar{n}\right)\mathbb{P}(N = \bar{n}) + \mathbb{P}(N \notin J) = o(1),$$

thus $\mathbb{P}\left(\exists_{1\le i\le 6}\mathcal{A}^C_{G_i(v)}\right) \le \sum_{i=1}^{6}\mathbb{P}\left(\mathcal{A}^C_{G_i(v)}\right) = o(1)$. Therefore with probability $1-o(1)$ all graphs $G_i(v)$, $1 \le i \le 6$, are connected. Each vertex from $D(v)$ is contained in at least one of the graphs $G_i(v)$, therefore the lemma follows. □

Lemma 3. *Let $C > 8$, (3) be fulfilled, $t \in \mathbb{D}$ be any point and $G[t]$ be a subgraph of $G_r(n,p)$ induced on the vertices contained in $D(t)$. Then $G[t]$ is connected with probability $1-o(1)$.*

Proof. The proof is analogous to the proof of the above lemma. We only have to replace v by t and omit the assumption that v is a vertex of $G_i(v)$. Moreover we have to add $G_7(t)$ – a graph induced on the vertices contained in the circle with centre in t and area equal $\pi r^2/6$. Then we can prove that with probability $1-o(1)$ all graphs $G_i(v)$, $1 \le i \le 7$, are connected. This implies that $G[t]$ is connected (it is a union of $G_i(v)$). □

Lemma 4. *Let $C > 8$ and (3) be fulfilled. Then $G_r(n,p)$ is connected with probability tending to 1 as $n \to \infty$.*

Proof. Let t be the centre of $\mathbb{D}$. By Lemma 3 $G[t]$ is connected with probability $1 - o(1)$. If we set $0 < \varepsilon < \min(\sqrt{1-8/C}, 1/2)$, then by Lemma 1 with probability $1 - o(1)$ every vertex from $V(G_r(n,p))$ is connected by a path with at least one vertex from $V(G[t])$. Therefore any two vertices from $V(G_r(n,p))$ are connected by a path with probability tending to 1 as $n \to \infty$. □

Lemma 5. *Let $C > 8$, $0 < \varepsilon < \min(\sqrt{1-8/C}, 1/2)$ and (3) be fulfilled. Then with probability tending to 1 as $n \to \infty$* CompassPlus *delivers information in $G_r(n,p)$ in at most $2\|s,t\|/(\varepsilon r(n)) + \ln(\pi r^2 n/6)/\ln\ln(\pi r^2 n/6) + 4$ steps between any two vertices s,t in some set of size $n(1+o(1))$.*

Proof. Let $t \in V(G_r(n,p))$. Denote $G[t]$ as in Lemma 2 and by $\mathcal{A}_t$ event that for all $v' \in V(G[t])$ we have $dist(v',t) \le \ln(\pi r^2 n/6)/\ln\ln(\pi r^2 n/6) + 4$ and $G[t]$ is connected. Let $\delta(n) = \max_{t \in V(G_r(n,p))} \mathbb{P}(\mathcal{A}_t^c)$, where $\mathcal{A}_t^c$ is a complement of $\mathcal{A}_t$ By Lemma 2

$$\delta(n) = o(1) \tag{8}$$

Let $X = \sum_{t \in V(G_r(n,p))} X_t$, where X_t is an indicator random variable of the event $\mathcal{A}_t^c$. Then $\mathbb{E}X = \sum_t \mathbb{E}X_t \le \delta(n)n$ and by Markov's inequality

$$\mathbb{P}\left(X \ge \sqrt{\delta(n)}n\right) \le \frac{\mathbb{E}X}{\sqrt{\delta(n)}n} = \sqrt{\delta(n)} = o(1) \tag{9}$$

Therefore the number of destination vertices such that the routing may fail at the BFS Routing Mode with probability $1 - o(1)$ is at most $\sqrt{\delta(n)}n = o(n)$. This combined with Lemma 1 finishes the proof. □

4 Geographic Routing in $\mathcal{G}_r(n,m,d)$

Analogous results to those presented in the previous section may be obtained for $\mathcal{G}_r(n,m,d)$.

Theorem 3. *Let $d \ge 2$ and $m = m(n) \to \infty$ be such that*

$$\frac{\pi r^2 d^2}{m} \ge C\frac{\ln n}{n} \text{ and } C > 8. \tag{10}$$

Then with probability tending to 1 as $n \to \infty$ a graph $\mathcal{G}_r(n,m,d)$ is connected. Moreover if $0 < \varepsilon < \min(\sqrt{1-8/C}, 1/2)$ with probability $1 - o(1)$ CompassPlus *delivers information in at most $2\|s,t\|/(\varepsilon r(n)) + (1+o(1))\ln(\pi r^2 n/6)/\ln\ln(\pi r^2 n/6)$ steps between any two vertices $s,t \in V(\mathcal{G}_r(n,m,d))$ in some set of size $n(1-o(1))$.*

Proof (Sketch of the proof). In order to obtain the above result we have to repeat the reasoning from the previous section with two minor modifications.

In the proof of Lemma 1 we have only to make changes in calculation from the first equation. We should notice that, for values of d as in (10), a vertex v' lying in $D(v)$ is not connected by an edge with v with probability $\binom{m-d}{d}/\binom{m}{d} = 1-(1+o(1))d^2/m$ independently of all other vertices. Therefore $\mathbb{P}\left(N_{k,\varepsilon}(v) = \emptyset\right) \leq \exp\left(-(1+o(1))(1-\varepsilon^2)\pi r^2 d^2/m\right)$.

Moreover in the proofs of Lemmas 2 and 3 instead of results on connectivity and the diameter stated in [3,4,7,10,11] we should use results from [23]. In order to prove only the connectivity result for the case $m = n^\alpha$ (α – a constant) we may also use the result from [1]. □

5 Concluding Remarks

We have shown how to set the parameters of a wireless ad hoc network with independent communication constraints and wireless sensor network with random key predistribution in such a way that they are connected and messages' transmission in greedy manner functions well with probability close to 1. For that purpose we have used the models which are random geometric graphs with randomly deleted edges and gave asymptotic results on their properties. We have also mentioned the necessary condition for disconnectivity of the network, which shows that the obtained result is tight up to a constant factor. Moreover we have set an upper bound on the length of transmission paths in the presented greedy routing protocol.

Acknowledgements. We thank an anonymous referee for the remark on the "bluetooth model". This work is partially supported by the Ministry of Science and Higher Education grant NN206 017 32/2452 and partially supported by the Ministry of Science and Higher Education grant NN206565740.

References

1. Blackburn, S.R., Gerke, S.: Connectivity of the uniform random intersection graph. Discrete Mathematics 309(16), 5130–5140 (2009)
2. Bloznelis, M., Jaworski, J., Rybarczyk, K.: Component evolution in a secure wireless sensor network. Networks 53(1), 19–26 (2009)
3. Bollobás, B.: Random Graphs. Academic Press, London (1985)
4. Bollobás, B.: The diameter of random graphs. IEEE Trans. Inform. Theory 36, 285–288 (1990)
5. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless network. Wireless Networks 7(6), 609–616 (2001)
6. Chan, H., Perrig, A., Song, D.: Random key predistribution schemes for sensor networks. In: SP 2003: Proceedings of the 2003 IEEE Symposium on Security and Privacy, Washington, DC, USA, pp. 197–213. IEEE Computer Society, Los Alamitos (2003)
7. Chung, F., Lu, L.: The diameter of sparse random graphs. Adv. in Appl. Math. 26(4), 257–279 (2001)
8. Di Pietro, R., Mancini, L.V., Mei, A., Panconesi, A., Radhakrishnan, J.: Sensor networks that are provably resilient. In: Proc. 2nd IEEE Int Conf Security Privacy Emerging Areas Commun Networks (SecureComm 2006), Baltimore, MD (2006)

9. Diaz, J., Petit, J., Serna, M.: Faulty random geometric networks. Parallel Processing Letters 10(4), 343–357 (2000)
10. Erdős, P., Rényi, A.: On random graphs I. Publ. Math. Debrecen 6, 290–297 (1959)
11. Erdős, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hungar. Acad. Sci. 5, 17–61 (1960)
12. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: CCS 2002: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 41–47. ACM Press, New York (2002)
13. Godehardt, E., Jaworski, J.: Two models of random intersection graphs for classifcation. In: Studies in Classifcation, Data Analysis and Knowledge Organization, pp. 67–81. Springer, Heidelberg (2003)
14. Gupta, P., Kumar, P.R.: Critical Power for Asymptotic Connectivity in Wireless Networks. In: Critical Power for Asymptotic Connectivity in Wireless Networks, pp. 547–566. Birkhauser, Basel (1998)
15. Hwang, J., Kim, Y.: Revisiting random key pre-distribution schemes for wireless sensor networks. In: SASN 2004: Proceedings of the 2nd ACM Workshop on Security of ad Hoc and Sensor Networks, pp. 43–52. ACM, New York (2004)
16. Janson, S., Łuczak, T., Ruciński, A.: Random Graphs. Wiley, Chichester (2001)
17. Jaworski, J., Ren, M., Rybarczyk, K.: Random key predistribution for wireless sensor networks using deployment knowledge. Computing 85(1–2), 57–76 (2009)
18. Karoński, M., Scheinerman, E.R., Singer-Cohen, K.B.: On random intersection graphs: The subgraph problem. Combinatorics, Probability and Computing 8, 131–159 (1999)
19. Karp, B., Kung, H.T.: GPSR: Greedy perimeter stateless routing for wireless networks. In: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, pp. 243–254 (2000)
20. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-case optimal and average-case efficient geometric ad-hoc routing. In: Proceedings of the 4th ACM International Symposium on Mobile ad Hoc Networking & Computing, pp. 267–278 (2003)
21. Penrose, M.: Random Geometric Graphs. Oxford University Press, Oxford (2003)
22. Pettarin, A., Pietracaprina, A., Pucci, G.: On the expansion and diameter of bluetooth-like topologies. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 528–539. Springer, Heidelberg (2009)
23. Rybarczyk, K.: Diameter, connectivity, and phase transition of the uniform random intersection graph. Discrete Mathematics (2011), doi:10.1016/j.disc.2011.05.029.
24. Singer-Cohen, K.B.: Random intersection graphs. PhD Thesis, Department of Mathematical Sciences, The Johns Hopkins University (1995)
25. Wan, P.-J., Yi, C.-W., Yao, F., Jia, X.: Asymptotic critical transmission radius for greedy forward routing in wireless ad hoc networks. In: Proceedings of the 7th ACM International Symposium on Mobile ad Hoc Networking and Computing, MobiHoc 2006, pp. 25–36. ACM, New York (2006)
26. Yi, C.-W., Wan, P.-J., Lin, K.-W., Huang, C.-H.: Asymptotic distribution of the number of isolated nodes in wireless ad hoc networks with unreliable nodes and links. Discrete Mathematics, Algorithms, and Applications 2(1), 107–124 (2010)
27. Zamalloa, M.Z., Seada, K., Krishnamachari, B., Helmy, A.: Efficient geographic routing over lossy links in wireless sensor networks. ACM Transactions on Sensor Networks (TOSN) 4(3), 1–33 (2008)

Untimed Language Preservation in Timed Systems*

Ocan Sankur

LSV, CNRS & ENS Cachan, France
sankur@lsv.ens-cachan.fr

Abstract. Timed automata are a model that is extensively used in formal verification of real-time systems. However, their mathematical semantics is an idealization which assumes perfectly precise clocks, but does not correspond to real hardware. In fact, it is known that imprecisions, however small they may be, may yield extra behaviours. Several works concentrated on a relaxation of the semantics of timed automata to model the imprecisions of the clocks. Algorithms were given, first for safety, then for richer linear-time properties, to decide the *robustness* of timed systems, that is, the existence of a bound on the imprecisions under which the system satisfies a given property. In this work, we study a stronger notion of robustness: we show how to decide whether the untimed language of a timed automaton is preserved under small enough imprecisions, and provide a bound on the imprecision parameter.

1 Introduction

Timed automata [2] are a well established model in real-time systems design. These allow the modeling, model-checking and synthesis of systems with timing constraints, and several mature tools are today available. Implementing such mathematical models on physical machines is an important step in practical verification, and it is a challenging problem in timed systems. In fact, it is known that perturbations on clocks, however small they may be, can lead to a different semantics than the mathematical semantics in timed automata [15,11] (see Fig. 1 below). One way of modelling these perturbations is to *enlarge* all the guards, that is to transform any timing constraint of the form "$x \in [a, b]$" into "$x \in [a - \Delta, b + \Delta]$" for some parameter $\Delta > 0$. In fact, the resulting semantics is an overapproximation of a concrete implementation semantics, called the *program semantics* studied in [12], which corresponds to the execution of timed automata by a simple microprocessor with a digital clock. An important problem is then to determine whether a given timed automaton is *implementable*, that is, whether its implementation is correct with respect to a given property, in a fast enough hardware. For this purpose, *robust model-checking*, which asks for the existence of a parameter Δ under which the resulting enlarged semantics is

* This work has been partly supported by projects DOTS (ANR-06-SETI-003) and ImpRo (ANR-10-BLAN-0317).

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 556–567, 2011.

correct, was proven decidable for safety properties [15,11] and for richer linear-time properties [8,9]. If robust model-checking succeeds, then the given timed automaton is implementable in a fast enough hardware in the sense of [12].

In this work, we study a stronger notion of robustness which requires untimed language equivalence between the original and the enlarged automaton for some value of the parameter Δ. We call such timed automata *language robust*. In particular, if a timed automaton is language robust, then any (untimed) language based property (such as linear-time properties) proven for $[\![\mathcal{A}]\!]$ will be preserved in $[\![\mathcal{A}_\Delta]\!]$ for small enough Δ, hence also in the program semantics mentioned above. We show that language robustness is decidable in 2EXPTIME for general timed automata, and we identify a class for which it can be decided in PSPACE. Note that the high complexity of our algorithm in the general case is not surprising, since deciding untimed language inclusion for timed automata is already a computationally hard problem (we recently showed this to be EXPTIME-hard [10]). In order to establish our results, we revisit the results of [11] and generalize these to a more general setting, taking into account the untimed languages (Section 4). Then, we prove a Ramsey-like combinatorial theorem on directed paths, which has an independent interest (Section 5). The proof of the main result (Section 6) combines these independent results.

Related work. A closely related line of work considers clock drifts, where clocks can have different rates [15,12,11,3,13] (this is equivalent to enlargement under some assumptions [11]). A solution based on modelling the perturbations using timed automata was suggested in [1], but their approach suffer from the fact that the verification results obtained in some platform may not hold in a faster (or more precise) platform (this holds in our setting, see Subsection 2.3). Other notions of robustness have been investigated, mainly to *remove* "isolated" or "unlikely" behaviours using topological and probabilistic methods (*e.g.* [14,5]), but these do not make the link with physical implementations.

2 Preliminaries

2.1 Timed Automata

A *labelled timed transition system (LTTS)* is a tuple $(S, s_0, \Sigma, \to)$, where S is the set of *states*, $s_0 \in S$ the initial state, Σ a finite alphabet, and $\to \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ the *transition* relation.

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For a subset $R \subseteq \mathcal{C}$ and a valuation v, we write $v[R \leftarrow 0]$ for the valuation defined by $v[R \leftarrow 0](x) = v(x)$ for $x \in \mathcal{C} \setminus R$ and $v[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. For two valuations u and v, we let $d_\infty(u, v) = \max_{x \in \mathcal{C}} |u(x) - v(x)|$.

Given a clock set $\mathcal{C}$, a *guard* is a formula generated by the grammar $\Phi_\mathcal{C} ::= k \leq x \mid x \leq k \mid \Phi_\mathcal{C} \wedge \Phi_\mathcal{C}$, where k ranges over $\mathbb{Q}_{\geq 0}$ and x over $\mathcal{C}$. We define the *enlargement* of a guard by $\Delta \in \mathbb{Q}_{\geq 0}$ as $\langle k \leq x \rangle_\Delta = k - \Delta \leq x$, and

$\langle x \leq k \rangle_\Delta = x \leq k + \Delta$, for $x, y \in \mathcal{C}$ and $k \in \mathbb{Q}_{\geq 0}$. The enlargement of a guard g, denoted by $\langle g \rangle_\Delta$, is obtained by enlarging all atomic formulas.

A valuation v *satisfies* a guard g, denoted $v \models g$, if all formulas are satisfied when each $x \in \mathcal{C}$ is replaced by $v(x)$.

Definition 1. *A timed automaton $\mathcal{A}$ is a tuple $(\mathcal{L}, \mathcal{C}, \Sigma, l_0, E)$, consisting of finite sets $\mathcal{L}$ of* locations*, $\mathcal{C}$ of* clocks*, Σ of* labels*, $E \subseteq \mathcal{L} \times \Phi_\mathcal{C} \times \Sigma \times 2^\mathcal{C} \times \mathcal{L}$ of edges, and $l_0 \in \mathcal{L}$, the* initial location*. An edge $e = (l, g, \sigma, R, l')$ is also written as $l \xrightarrow{g,\sigma,R} l'$, where g is called the* guard *of e.*

A timed automaton $\mathcal{A}$ is *integral* if all constants that appear in its guards are integers. For any $\Delta \in \mathbb{Q}_{\geq 0}$, $\mathcal{A}_\Delta$ will denote the timed automaton where all guards are enlarged by Δ. We will refer to finite automata with no clocks as *untimed* automata.

Definition 2. *The semantics of a timed automaton $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$ is an LTTS denoted by $[\![\mathcal{A}]\!]$, over alphabet Σ, whose state space is $\mathcal{L} \times \mathbb{R}^\mathcal{C}_{\geq 0}$. The initial state is $(l_0, \mathbf{0})$, where $\mathbf{0}$ denotes the valuation where all clocks have value 0. Transitions are defined as $(l, v) \xrightarrow{\tau} (l, v + \tau)$ for any state (l, v) and $\tau \geq 0$, $(l, v) \xrightarrow{\sigma} (l', v')$, for any edge $l \xrightarrow{g,\sigma,R} l'$ in $\mathcal{A}$ such that $v \models g$ and $v' = v[R \leftarrow 0]$.*

A *run* of $[\![\mathcal{A}]\!]$ is a finite or infinite sequence $\rho = (q_i, \tau_i, \sigma_i)_{i \geq 0}$ where q_i is a state of $[\![\mathcal{A}]\!]$, $\tau_i \in \mathbb{R}_{\geq 0}$ and $\sigma_i \in \Sigma$, such that for each $i \geq 0$, $q_i \xrightarrow{\tau_i} q'_i \xrightarrow{\sigma_i} q_{i+1}$ for some state q'_i. A run is *initialized* if $q_0 = (l_0, \mathbf{0})$. A *trace* is a word in $\Sigma^* \cup \Sigma^\omega$. We say that the above run ρ *follows the trace* $\sigma_0 \sigma_1 \ldots$, which is denoted by $\mathsf{tr}(\rho)$. We define $(\rho)_i = q_i$, the i-th state of ρ, and $\mathsf{first}(\rho) = q_0$ its first state. If ρ is finite, then we denote by $\mathsf{last}(\rho)$ its last state. We denote by $\rho_{i \ldots j}$ the run defined by ρ between states of indices i and j. Let $L([\![\mathcal{A}]\!])$ denote the set of (untimed) traces of the initialized finite and infinite runs of $[\![\mathcal{A}]\!]$. For any state q of $[\![\mathcal{A}]\!]$, $L([\![\mathcal{A}]\!], q)$ will denote the traces of the runs that start at q. The *length* of a finite run ρ is the length of its trace, and is denoted by $|\rho|$.

We define the usual notion of *regions* and *region automaton* [2]. Consider an integral timed automaton $\mathcal{A}$ with clock set $\mathcal{C}$. Let K be the largest constant that appears in its guards. For any $(l, u), (l', v) \in \mathcal{L} \times \mathbb{R}^C_{\geq 0}$, we let $(l, u) \simeq (l, v)$ if, and only if, $l = l'$ and

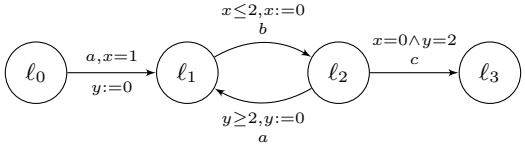


Fig. 1. The following is an example run of the above automaton: $(\ell_0, 0, 0) \xrightarrow{1} (\ell_1, 1, 1) \xrightarrow{a} (\ell_1, 1, 0) \xrightarrow{0.9} (\ell_1, 1.9, 0.9) \xrightarrow{b} (\ell_2, 0, 0.9) \xrightarrow{1.4} (\ell_2, 1.4, 2.3) \xrightarrow{a} (\ell_1, 1.4, 0)$. It can be seen that location ℓ_3 is not reachable in the runs of $[\![\mathcal{A}]\!]$. In fact, $L([\![\mathcal{A}]\!]) = (ab)^* + (ab)^* a + (ab)^\omega$. See [15].

$$\begin{aligned}
&\text{either } \lfloor u(x)\rfloor = \lfloor v(x)\rfloor \text{ or } u(x), v(x) > K, && \forall x \in \mathcal{C},\\
&\text{and } \mathsf{frac}(u(x)) = 0 \iff \mathsf{frac}(v(x)) = 0, && \forall x \in \mathcal{C},\\
&\text{and } \mathsf{frac}(u(x)) < \mathsf{frac}(u(y)) \iff \mathsf{frac}(v(x)) < \mathsf{frac}(v(y)), && \forall x, y \in \mathcal{C},
\end{aligned}$$

where $\mathsf{frac}(\cdot)$ denotes the fractional part. The equivalence class of a state (l, v) for the relation $\simeq$ is denoted by $\mathsf{reg}((l, v)) = \{(l, u) \mid (l, u) \simeq (l, v)\}$, and called a *region* of $\mathcal{A}$. Note that our definition of regions includes locations. It is known that $\simeq$ has finite index [2]. For any region r, let $\overline{r}$ denote its topological closure. A region is *bounded* if it is a bounded subset of $\mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$. One can associate to $\mathcal{A}$, a finite (untimed) automaton $\mathcal{R}(\mathcal{A})$, called the *region automaton*, whose states are the regions of $\mathcal{A}$, and which has an edge of label $\sigma \in \Sigma$ from region r to r' whenever $q \xrightarrow{d} q'' \xrightarrow{\sigma} q'$ in $[\![\mathcal{A}]\!]$ for some $d \geq 0$ and $q \in r$ and $q' \in r'$. A *path* of $\mathcal{R}(\mathcal{A})$ is a sequence $\pi = (r_i, \sigma_i)_{i \geq 0}$ where r_i is a region and $\sigma_i \in \Sigma$, such that for each $i \geq 0$, $\mathcal{R}(\mathcal{A})$ has an edge from r_i to r_{i+1} with label σ_i. The i-th state of path π is denoted by $(\pi)_i$. Path π is *initialized* if $r_0 = \mathsf{reg}((l_0, \mathbf{0}))$. The *trace* of π is the word $\sigma_0\sigma_1 \ldots$, which is denoted by $\mathsf{tr}(\pi)$. The set of traces of the finite or infinite paths of $\mathcal{R}(\mathcal{A})$ is denoted by $L(\mathcal{R}(\mathcal{A}))$. A finite path $\pi = (r_i, \sigma_i)_{0 \leq i \leq n}$ is a *cycle* if $r_0 = r_n$. A run $\rho = (q_i, \tau_i, \sigma_i)_{i \geq 0}$ of $[\![\mathcal{A}]\!]$ *follows a path* $(r_i, \sigma'_i)_{i \geq 0}$ of $\mathcal{R}(\mathcal{A})$ if $q_i \in \overline{r_i}$ and $\sigma_i = \sigma'_i$ for all $i \geq 0$. It is known that for any path π of $\mathcal{R}(\mathcal{A})$, there is a run of $[\![\mathcal{A}]\!]$ that follows π, starting from any state in $\mathsf{first}(\pi)$ and conversely. In particular, $L(\mathcal{R}(\mathcal{A})) = L([\![\mathcal{A}]\!])$ [2].

2.2 Restrictions on Timed Automata

Following [11,15], we defined timed automata with closed and *rectangular* guards (that is, we do not have *diagonal* constraints such as $k \leq x - y \leq l$). We also assume that all clocks are bounded above by some constant M. Considering closed guards is natural in our setting, since we are interested in the behaviour of the systems under positive enlargement. Assuming rectangular guards and bounded clocks is not restrictive in terms of expressiveness, but has an effect on the size of the models ([7]). As in [11,15,4], the only real restriction is the following. We consider timed automata where all clocks are reset at least once along any cycle of the region automaton; these are called *progress cycles*. A sufficient condition for a timed automaton to have only progress cycles is that any cycle of the underlying finite automaton resets all clocks at least once ([4]).

Although we prove our results for general timed automata with progress cycles, we also identify a subclass for which we improve the complexity of the problem we study. We call a timed automaton *concise* if its region automaton is deterministic (that is, from all states of the region automaton, there is at most one outgoing edge per label)[1].

[1] It would actually suffice to define conciseness by requiring that all states that satisfy the guards of edges with the same label to be language-equivalent (that the same untimed language is recognized). In fact, in this case, the region automaton can be made deterministic by leaving one (arbitrary) edge per label at each state.

2.3 Robustness

By definition of the enlargement, we have $L([\![\mathcal{A}_\Delta]\!]) \subseteq L([\![\mathcal{A}_{\Delta'}]\!])$ for any $\Delta \leq \Delta'$, and in particular $L([\![\mathcal{A}]\!]) \subseteq L([\![\mathcal{A}_\Delta]\!])$ for any $\Delta \geq 0$. We are interested in the inverse inclusion, which does not always hold as we also noted in the introduction. In fact, if $\mathcal{A}$ denotes the timed automaton given in Figure 1, then for any $\Delta > 0$, all long enough words in $(ab)^*c$ belong to $L([\![\mathcal{A}_\Delta]\!])$ but not to $L([\![\mathcal{A}]\!])$ ([11]). (In fact location ℓ_3 is reachable in $[\![\mathcal{A}_\Delta]\!]$ iff $\Delta = 0$). Intuitively, this is due to the fact that at each cycle ab, the imprecisions can add up in $[\![\mathcal{A}_\Delta]\!]$, and the values of the clocks can deviate from what they can normally be in $[\![\mathcal{A}]\!]$.

Definition 3 (Language-robustness). *A timed automaton $\mathcal{A}$ is* language-robust *if there exists $\Delta > 0$ such that $L([\![\mathcal{A}]\!]) = L([\![\mathcal{A}_\Delta]\!])$.*

Informally, $\mathcal{A}$ is language-robust if $[\![\mathcal{A}_\Delta]\!]$ has no extra behaviour than $[\![\mathcal{A}]\!]$ for some $\Delta > 0$, in terms of untimed language. Observe that whenever $L([\![\mathcal{A}_\Delta]\!]) \subseteq L([\![\mathcal{A}]\!])$, we also have $L([\![\mathcal{A}_{\Delta'}]\!]) \subseteq L([\![\mathcal{A}]\!])$ for any $\Delta' < \Delta$. This is a desirable property, called "faster is better" [12,1], which means that once we prove the correctness of the system for some Δ, it remains correct on any faster platform.

3 Main Result

Theorem 1. *Let $\mathcal{A}$ be any timed automaton with progress cycles, and W the size of its region automaton. Let $K = W$ if $\mathcal{A}$ is concise, and $K = 2^W$ otherwise, and fix any $N_0 \geq 15 \cdot W \cdot |\mathcal{C}|^2 \cdot 2^{(|\mathcal{C}|+1)^2} \cdot (K+1)^2$. Then, there exists $\Delta > 0$ such that $L([\![\mathcal{A}_\Delta]\!]) = L([\![\mathcal{A}]\!])$ if and only if $L([\![\mathcal{A}_{\frac{1}{N_0}}]\!]) = L([\![\mathcal{A}]\!])$.*

Our main result, that is, the decidability of language-robustness is a direct corollary of the previous theorem. In fact, $\mathcal{A}_{\Delta_0}$ can be transformed into a (language-) equivalent integral automaton by multiplying all constants by $\frac{1}{\Delta_0}$. We will denote by $\mathcal{R}(\mathcal{A}_{\Delta_0})$ the region automaton of the corresponding integral timed automaton. We can then check whether $\mathcal{R}(\mathcal{A}_{\Delta_0})$ and $\mathcal{R}(\mathcal{A})$ recognize the same untimed language. We obtain the following complexity results.

Corollary 1. *For concise timed automata with progress cycles, language robustness can be decided in PSPACE. For general timed automata with progress cycles, language robustness can be decided in 2EXPTIME.*

Proof. Consider a concise timed automaton $\mathcal{A}$, and let $\mathcal{R}(\mathcal{A})$ denote its region automaton. Let $\mathcal{R}(\mathcal{A})^c$ denote the complement of $\mathcal{R}(\mathcal{A})$ (which recognizes the complement of $L(\mathcal{R}(\mathcal{A}))$). Then, one can decide whether $L(\mathcal{A}_{\Delta_0}) \cap L(\mathcal{R}(\mathcal{A})^c) \neq \emptyset$ in polynomial space. In fact, the states of both $\mathcal{R}(\mathcal{A}_{\Delta_0})$ and $\mathcal{R}(\mathcal{A})^c$ can be encoded in polynomial space (for Δ_0 given by the theorem for concise $\mathcal{A}$). Then, the usual non-deterministic procedure (*e.g.* [2]) that guesses an accepting path in the product of these can be carried out in polynomial space.

Now, consider an arbitrary timed automaton $\mathcal{A}$. Then the encoding of regions of $\mathcal{A}_{\Delta_0}$ requires exponential space, for Δ_0 given for this case. We describe an alternating exponential space procedure (which is doubly exponential time) to decide $L(\mathcal{A}_{\Delta_0}) \subseteq L(\mathcal{R}(\mathcal{A}))$. The procedure starts by the initial states of $\mathcal{R}(\mathcal{A}_{\Delta_0})$ and $\mathcal{R}(\mathcal{A})$. At each step, it first universally guesses a successor of $\mathcal{R}(\mathcal{A}_{\Delta_0})$, then it existentially guesses a successor of $\mathcal{R}(\mathcal{A})$. The machine can count the length of such an execution up to 2^W, which is the number of regions of the former system (above which any path contains a cycle). Thus, this procedure accepts iff for each path in $\mathcal{R}(\mathcal{A}_{\Delta_0})$, there is a path with the same trace in $\mathcal{R}(\mathcal{A})$. □

In the rest of this paper, we present the proof of Theorem 1. We start with a study of the properties of enlarged timed automata (Section 4), and some combinatorial results (Section 5), then give the proof (Section 6).

4 Properties of the Semantics under Enlargement

Let us fix a timed automaton $\mathcal{A}$ with $C > 0$ clocks. We start with the following result, which is a direct corollary of Lemma 3.14 we proved in [17]. It states that for any $\Delta > 0$, the trace of any run of $[\![\mathcal{A}_\Delta]\!]$ of length less then $O(\lfloor \frac{1}{\Delta} \rfloor)$ can be followed in $\mathcal{R}(\mathcal{A})$ too. An immediate implication is that if the length of the runs are fixed a priori, then a small enough enlargement has no effect on the behaviour of timed automata (in terms of untimed language). Figure 2 illustrates the construction of the following lemma.

Lemma 1 ([17]). *Fix any $n \in \mathbb{N}$ and $\Delta > 0$ such that $\Delta \leq \frac{1}{5nC^2}$. Let ρ be any run of $[\![\mathcal{A}_\Delta]\!]$. Then, for all $1 \leq i_0 \leq |\rho|$, there exists a region, denoted by $H(\rho, i_0, n)$, included in $\overline{\mathsf{reg}((\rho)_{i_0})}$, such that for all regions $r \subseteq \overline{H(\rho, i_0, n)}$, there is a path π of $\mathcal{R}(\mathcal{A})$ over the trace $\mathsf{tr}(\rho_{i_0 \dots \min(i_0+n,|\rho|)})$, with $(\pi)_1 = r$ and $\overline{(\pi)_j} \cap \overline{H(\rho, i_0 + j - 1, n)} \neq \emptyset$ for all $1 \leq j \leq |\pi|$.*

We are now interested in "long" or infinite runs. In [11], it is shown that for some timed automaton $\mathcal{A}$ (*e.g.* the one in Fig. 1) some regions that are not reachable in $[\![\mathcal{A}]\!]$ become entirely reachable in $[\![\mathcal{A}_\Delta]\!]$, for any $\Delta > 0$ (See also [15] for a similar analysis under *clock drifts*). An analysis of the behaviour of $[\![\mathcal{A}_\Delta]\!]$ shows that this is due to the accumulation of the "error" of Δ along some cycles of the region automaton. They give a characterization of those cycles of $\mathcal{R}(\mathcal{A})$ which cause this behaviour and get a decision procedure for safety properties. In this section, we revisit the analysis of the cycles of $\mathcal{R}(\mathcal{A})$ under enlargement, and prove the same results in a slightly more general setting. Roughly, we show that, the states that are reachable in $[\![\mathcal{A}_\Delta]\!]$ by repeating a single cycle are also reachable by repeating particular sets of cycles in any order. Our proofs follow [11].

A state whose valuation has integer components is called a *vertex*. For any region r, we denote by $V(r)$ the set of vertices of $\overline{r}$. Given a cycle π from a region r, we define the relation $\nu(\pi) \subseteq V(r) \times V(r)$, the *vertex map of* π, where $(q, q') \in \nu(\pi)$ if and only if there is a run in $[\![\mathcal{A}]\!]$, from q to q' following π.

Note that for all $q \in V(r)$, there exists at least one $q' \in V(r)$ such that $(q, q') \in \nu(\pi)$, and $q'' \in V(r)$ such that $(q'', q) \in \nu(\pi)$ by the well-known properties of regions ([11]). Two cycles π_1 and π_2 are *equivalent* if $\mathsf{first}(\pi_1) = \mathsf{first}(\pi_2)$ and $\nu(\pi_1) = \nu(\pi_2)$. Let π be a cycle of $\mathcal{R}(\mathcal{A})$ starting at some region r. For any $k > 0$, we let $V_{\pi,k} = \{q \in V(r) \mid (q, q) \in \nu(\pi)^k\}$. This is the set of vertices $q \in \overline{\mathsf{first}(\pi)}$ for which there are runs following π^k that start and end at q. We define the convex hull of the union of these sets as $L_\pi = \mathsf{convex\text{-}hull}(\bigcup_{k>0} V_{\pi,k})$. It is clear from the definition of L_π that $L_\pi = L_{\pi'}$ for any equivalent cycles π and π'. The main result of this section is the following lemma, which generalizes Theorem 23 in [11] (see also Lemma 7.10 in [15]).

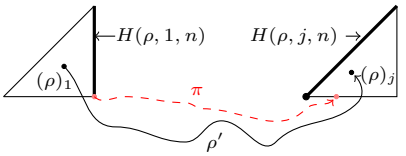


Fig. 2. A run ρ of $[\![\mathcal{A}_\Delta]\!]$. The leftmost triangle represents $\mathsf{reg}((\rho)_1)$, and the rightmost one $\mathsf{reg}((\rho)_j)$ (their corners, edges and interiors are subregions). By Lemma 1, there is a region $H(\rho, 1, n)$ such that starting from any region in $\overline{H(\rho, 1, n)}$, one can construct a path π of length n (the red dashed curve) such that $\overline{(\pi)_j}$ intersects $\overline{H(\rho, j, n)}$ for all j.

Lemma 2. *Let $\pi_1, \ldots, \pi_p$ be equivalent cycles of $\mathcal{R}(\mathcal{A})$ that start in region r, and consider any $\Delta > 0$. Then, there exists $k > 0$ such that for any $q, q' \in \overline{r}$, and any word $w \in \{\pi_1, \ldots, \pi_p\}^k$ there is a run in $[\![\mathcal{A}_\Delta]\!]$ from q to q' on word w.*

To prove this lemma, we first show that L_{π_1} is backward and forward reachable in $[\![\mathcal{A}]\!]$, from any point of $\overline{r}$, by iterating at least C times any of the equivalent cycles in any order (Lemma 5), and that any pair of points in L_{π_1} can be connected by a run of $[\![\mathcal{A}_\Delta]\!]$, again by iterating these cycles (Lemma 7).

A natural property of runs of timed automata is that convex combinations of two runs yield a run over the same word, as shown in the following lemma. We denote $\lambda(l, v) = (l, \lambda v)$ where $\lambda \in \mathbb{R}_{\geq 0}$, v is a valuation and l a location.

Lemma 3 ([11, Lemma 24], and [15, Lemma 7.1]). *Let π be a path in $\mathcal{R}(\mathcal{A})$, and let ρ and ρ' be runs in $[\![\mathcal{A}]\!]$ that follow π. Then for all $\lambda \in [0, 1]$, there exists a run ρ'' of $[\![\mathcal{A}]\!]$ following π, such that $(\rho'')_i = \lambda(\rho)_i + (1 - \lambda)(\rho')_i$ for all $1 \leq i \leq n$.* □

The following proposition provides a bound on the number of vertices of regions. It also implies that from each region, there is a finite number of cycles with pairwise distinct vertex maps. Remember that all clocks are bounded above by some constant, so we only need to consider bounded regions.

Lemma 4 ([11, Lemma 14]). *Any bounded region has at most $C+1$ vertices. Any point $u \in \mathbb{R}^C_{\geq 0}$ is a convex combination of the vertices of $\mathsf{reg}(u)$.* □

The following lemma states that, L_π is backward and forward reachable from any state of $\overline{\mathsf{first}(\pi)}$, by repeating at least C times cycles equivalent to π.

Lemma 5. *Let $\pi_1, \ldots, \pi_p$ be equivalent cycles of $\mathcal{R}(\mathcal{A})$, that all start in region state r, and fix any $q \in \overline{r}$. Then, for any $k \geq C$ and any path $w \in \{\pi_1, \ldots, \pi_p\}^k$,*

there exists $q_1, q_2 \in L_{\pi_1}$ *and runs* ρ_1 *and* ρ_2 *of* $[\![\mathcal{A}]\!]$ *that follow* w*, such that* $\mathsf{first}(\rho_1) = q$ *and* $\mathsf{last}(\rho_1) = q_1$*; and* $\mathsf{first}(\rho_2) = q_2$ *and* $\mathsf{last}(\rho_2) = q$.

Proof. We first prove the statement when q is a vertex. As we already noted above, for all $v \in V(r)$, there exists at least one $v' \in V(r)$ such that $(v, v') \in \nu(\pi_1)$, and the number of vertices is at most $C+1$ by Lemma 4, so by repeating C times any cycles among $\pi_1, \ldots, \pi_p$, we get a sequence of vertices $v_1, \ldots, v_{C+1}$ such that $(v_i, v_{i+1}) \in \nu(\pi_1)$. But then, $v_i = v_j$ for some $i < j$, thus we have $v_i, v_{i+1}, \ldots, v_j \in L_{\pi_1}$. Now, we can extend the sequence $v_1 \ldots v_j$ to k vertices by repeating the cycle $v_i v_{i+1} \ldots v_j$. Clearly, this run can be constructed following any path w since π_i's are equivalent.

Consider now an arbitrary point q in $\overline{r}$. By Lemma 4, q can be written as a convex combination of the vertices of r. Let $v_1, \ldots, v_m$ denote the vertices of r, and $\lambda_1, \ldots, \lambda_m \geq 0$ be such that $\lambda_1 + \ldots + \lambda_m = 1$ and $q = \lambda_1 v_1 + \ldots + \lambda_m v_m$. As we showed above, for any v_i, there is a run in $[\![\mathcal{A}]\!]$ that follows w, from v_i to some vertex $v_i' \in L_{\pi_1}$. Lemma 3 yields the desired run. □

Lemma 6. *Let* $\pi_1, \ldots, \pi_p$ *be equivalent cycles in* $\mathcal{R}(\mathcal{A})$*. Then there exists* $m > 0$ *such that for all paths* $w \in \{\pi_1, \ldots, \pi_p\}^m$*, and for all* $q \in L_{\pi_1}$*, there is a run* ρ *in* $[\![\mathcal{A}]\!]$ *from* q *to* q*, following* w.

Proof. By definition of L_{π_1}, any $z \in L_{\pi_1}$ is a convex combination of a set of vertices v_i in L_{π_1}. But, for any vertex $v_i \in L_{\pi_1}$, there exists $m_i > 0$ such that $(v_i, v_i) \in \nu(\pi_1)^{m_i}$. So, there exists $m > 0$ such that $(v_i, v_i) \in \nu(\pi_1)^m$ for all $1 \leq i \leq k$. Now, the convex combination of these runs yield the desired run from q to q, by Lemma 3. □

The following lemma shows that any pair of states in L_π can be connected by a run in $[\![\mathcal{A}_\Delta]\!]$.

Lemma 7 ([11, Lemma 29]). *Let* π *be a cycle of* $\mathcal{R}(\mathcal{A})$ *that starts in region* r *and let* $q \in L_\pi \cap \overline{r}$*. Then for any* $\Delta > 0$*, and any* $q' \in \overline{r}$ *such that* $d_\infty(q, q') \leq \frac{\Delta}{2}$*, there is a run, in* $[\![\mathcal{A}_\Delta]\!]$*, from* q *to* q' *following* π. □

Proof (of Lemma 2). By Lemma 5, repeating at least C times the cycles $\pi_1, \ldots, \pi_p$ suffices to reach some point $q_1 \in L_{\pi_1}$. The same lemma provides a point $q_2 \in L_{\pi_1}$ which is backward reachable from q' by repeating C times any of these cycles. It follows from the definition of regions that for any pair of points q_1, q_2 that belong to a same region, one can find points $q_1 = u_0, u_1, \ldots, u_N = q_2$ in $\overline{r}$ where $N = \lceil \frac{1}{\Delta} \rceil$ such that $d_\infty(u_i, u_{i+1}) \leq \frac{\Delta}{2}$ for all $0 \leq i \leq N-1$. Let $m > 0$ be the bound provided by Lemma 6. Now, q_2 can be reached from q_1, by a run over any word $\{\pi_1, \ldots, \pi_p\}^{mN}$ by Lemma 7 (applied N times to pairs (u_i, u_{i+1})). □

5 Some Combinatorial Tools

In this section, we prove a Ramsey-like theorem for colored directed paths, which gives a lower bound on the length of monochromatic subpaths contained in these (Subsection 5.1). This improves, by an exponential, the result provided by a direct application of Ramsey's theorem [16]. In Subsection 5.2, we give a simple property on untimed finite automata accepting *ultimately universal* languages.

5.1 A Ramsey-Like Theorem for Directed Paths

A *directed graph* is a pair $G = (V, E)$ where V is a finite set of *nodes* and $E \subseteq V \times V$, is the set of *edges*. A graph G is complete if for all $i, j \in V$ either $(i, j) \in E$ or $(j, i) \in E$. A graph is a *linearly-ordered complete graph*, if for some (strict) linear order $\prec$ on V, $(i, j) \in E$ iff $i \prec j$. The *degree* of a node v is $d(v) = |\{u \mid (v, u) \in E \text{ or } (u, v) \in E\}|$. Two nodes u and v are *connected* in the graph G if there exists a sequence $u = s_0, s_1, \ldots, s_k = v$ of nodes such that $(s_i, s_{i+1}) \in E$ or $(s_{i+1}, s_i) \in E$ for all $0 \leq i \leq k-1$. A graph is connected if all its nodes are connected. A *subgraph* of $G = (V, E)$ is a graph (V', E') such that $V' \subseteq V$ and $E' \subseteq E$. A *connected component* is a maximal connected subgraph. A *directed path* of G is a sequence of nodes $v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq n-1$, and its *length* is n. A *r-coloring* of a graph $G = (V, E)$ is a function $E \to \{1, \ldots, r\}$ that associates to each edge a color from $\{1, \ldots, r\}$. A path is *monochromatic* if all its edges are assigned the same color.

Our result is based on the following theorem from [6].

Theorem 2 ([6]). *Let G be a connected directed graph over n nodes such that, for some h, every node v satisfies $d(v) \geq h$. Then G contains a directed path of length* $\min(n, h+1)$. □

The main result of this subsection is the following theorem.

Theorem 3. *Let $G = (V, E)$ be a linearly-ordered complete graph over n nodes given with an r-coloring of its edges. Then G contains a monochromatic directed path of length $\lfloor \sqrt{n/r - 2} \rfloor - 1$.*

Proof. Fix $h = \lfloor \sqrt{n/r - 2} \rfloor - 1$. For each $1 \leq i \leq r$, define subgraph G_i which contains exactly the edges colored by i. Then, for each G_i, define G'_i by removing any node v (and any edge containing v) whose degree in G_i is less than h. In G'_i, any node has either none or at least h edges of color i. Let G' be the union of all G'_i's. To define G', we remove at most $(h-1)rn$ edges ($h-1$ edges per color and node). Thus, G' has at least $\binom{n}{2} - (h-1)rn$ edges. Then, one of the G'_i's has at least $\binom{n}{2}/r - (h-1)n$ edges, say G'_{i_0}. We show that G'_{i_0} has a connected component C with at least $\sqrt{(n-1)/r - 2(h-1)}$ nodes. In fact, consider a graph with N nodes and M edges. Let K denote the number of nodes of the largest connected component. Since there are at most N connected components, each of them containing at most $\binom{K}{2}$ edges, we get that $M \leq N\binom{K}{2}$, which implies $\sqrt{2M/N} \leq K$. Applying this to G'_{i_0}, we get the desired connected component C. Now, by construction of G'_{i_0}, all nodes of C have degree at least h in C (in fact, by maximality, all edges of G'_{i_0} adjacent to the nodes of C are also included in C). Then, by Theorem 2, C contains a directed path of length at least $\min(\sqrt{(n-1)/r - 2(h-1)}, h)$. But the first term of the min is greater than or equal to h by the choice of h. So, there is a directed path of length at least $\lfloor \sqrt{n/r - 2} \rfloor - 1$ in G'_{i_0}, and this is a monochromatic path in G. □

5.2 Ultimately Universal Languages

Let Σ denote a finite alphabet. We call a regular language $L \subseteq \Sigma^*$ *ultimately universal* if there exists $k \geq 0$ such that $\bigcup_{l \geq k} \Sigma^l \subseteq L$.

Lemma 8. *Let L be an ultimately universal language recognized by a deterministic untimed automaton with n locations. Then $\bigcup_{l \geq n-1} \Sigma^l \subseteq L$.*

6 Proof of the Theorem

Fix any timed automaton $\mathcal{A}$ with $C \geq 1$ clocks. Let W denote the number of regions of $\mathcal{A}$. Let $\mathcal{D}$ denote a deterministic untimed automaton such that $L(\mathcal{D}) = L(\mathcal{R}(\mathcal{A}))$ (say, obtained by minimization), and let K denote the size of $\mathcal{D}$, which is at most 2^W ($K \leq W$ if $\mathcal{A}$ is concise). We let $M = 2^{(C+1)^2}\big((K+1)^2 + 2\big)$, $n \geq WM$ and fix any $\Delta_0 \leq \frac{1}{5nC^2}$. The theorem states that if $L([\![\mathcal{A}]\!]) = L([\![\mathcal{A}_\Delta]\!])$ for some $\Delta > 0$, then $L([\![\mathcal{A}]\!]) = L([\![\mathcal{A}_{\Delta_0}]\!])$. The only interesting case is when $\Delta < \Delta_0$ since otherwise $L([\![\mathcal{A}]\!]) \subseteq L([\![\mathcal{A}_{\Delta_0}]\!]) \subseteq L([\![\mathcal{A}_\Delta]\!])$ and the theorem follows. So let us suppose that $L([\![\mathcal{A}]\!]) = L([\![\mathcal{A}_\Delta]\!])$ for some $\Delta < \Delta_0$. Let ρ be a run of $[\![\mathcal{A}_{\Delta_0}]\!]$. We will show that $\mathsf{tr}(\rho) \in L(\mathcal{D}) = L([\![\mathcal{A}]\!])$, which will prove the theorem.

Lemma 9. *Any path π of $\mathcal{R}(\mathcal{A})$ of length at least n can be factorized as $\pi = \pi_1\tau_1\tau_2 \ldots \tau_{K-1}\pi_2$ where π_1, π_2 are paths, and τ_i's are equivalent cycles.*

Proof. Since $n \geq W \cdot M$, by the Pigeon-hole principle, π contains a factor $t = t_1 \ldots t_M$ such that $\mathsf{first}(t_1) = \mathsf{first}(t_j)$ for all j. We apply Theorem 3 to get a further factorization of t. Consider a directed graph of the usual linear order $<$ over $\{1, \ldots, M\}$. To each edge (j, k) of the graph, where $j < k$, we assign as *color*, the vertex map $\nu(t_j t_{j+1} \ldots t_k)$. The number of colors is then bounded by $2^{(C+1)^2}$. Applying Theorem 3, we get that t contains a factor $\tau_1 \ldots \tau_{K-1}$, where $\tau_1 = t_{j_1} t_{j_1+1} \ldots t_{j_2}, \tau_2 = t_{j_2} t_{j_2+1} \ldots t_{j_3}, \ldots, \tau_{K-1} = t_{j_{K-1}} \ldots t_{j_K}$, for some $j_1 < j_2 < \ldots < j_K$, such that $\nu(\tau_1) = \nu(\tau_j)$ for all $1 \leq j \leq K$. □

Lemma 10. *Let $\pi = \pi_1\tau_1\tau_2 \ldots \tau_{K-1}\pi_2$ be a path of $\mathcal{R}(\mathcal{A})$ where π_1 and π_2 are paths and τ_i's are equivalent cycles. Then, there exists $k_0 > 0$ such that for all $q \in \overline{\mathsf{first}(\pi)}$, $q' \in \overline{\mathsf{last}(\tau_{K-1})}$, $k \geq k_0$, and any word $w \in \mathsf{tr}(\pi_1) \cdot (\mathsf{tr}(\tau_1) + \ldots + \mathsf{tr}(\tau_{K-1}))^k$, there is a run ρ' of $[\![\mathcal{A}_\Delta]\!]$ over w with $\mathsf{first}(\rho) = q$ and $\mathsf{last}(\rho) = q'$.*

The proof of the previous lemma uses Lemma 7 (See Appendix). We are now ready to prove our main theorem. The reader may follow the proof in Figure 3.

Proof (of Theorem 1). Consider ρ and the constants as defined above, and notice that $\Delta_0 \leq \frac{1}{N_0}$. Let $H(\rho, i, n)$ be the regions given by Lemma 1 for all $i \geq 0$. We will inductively construct the desired run γ of $\mathcal{D}$ with $\mathsf{tr}(\gamma) = \mathsf{tr}(\rho)$. At step i of the induction, we will define $\gamma_{\alpha_i \ldots \alpha_{i+1}}$ for some increasing sequence $(\alpha_i)_{i \geq 0}$ with $\alpha_0 = 1$. When constructing γ, we will also construct an auxiliary run ρ' of

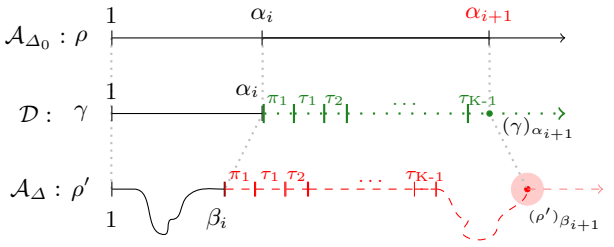


Fig. 3. An induction step in the proof of Theorem 1. First, ρ' is extended following path $\pi_1\tau_1\tau_2\ldots$ (shown in red dashed line), and for any long enough repetition of cycles $\tau_1,\ldots,\tau_K$, $\overline{H(\rho,\alpha_{i+1},n)}$ (shown in pink filled circle) can be reached. Then, γ is extended to $(\gamma)_{\alpha_{i+1}}$ (shown in green loosely dotted line).

$[\![\mathcal{A}_\Delta]\!]$ in parallel, defining $\rho'_{\beta_i\ldots\beta_{i+1}}$ at each step i, for some increasing sequence $(\beta_i)_{i\geq 0}$ with $\beta_0 = 1$, and ensuring that $(\rho')_{\beta_i} \in \overline{H(\rho,\alpha_i,n)}$ and $L([\![\mathcal{A}_\Delta]\!],(\rho')_{\beta_i}) \subseteq L(\mathcal{D},(\gamma)_{\alpha_i})$ for all $i \geq 0$.

– For $i = 0$, since ρ is an initialized run, we have $(\rho)_1 = (l_0,\mathbf{0})$ so $H(\rho,1,n) = \mathsf{reg}(()l_0,\mathbf{0})$. We have $\alpha_0 = \beta_0 = 1$, $(\rho')_1 = (l_0,\mathbf{0})$ and $(\gamma)_1$ is the initial state of $\mathcal{D}$. We have $L([\![\mathcal{A}_\Delta]\!],\rho'_1) \subseteq L(\mathcal{D},\gamma_1)$ by hypothesis.

– For any $i \geq 1$, suppose by induction that γ is defined between indices 1 and α_i and that $\rho'_{\beta_i} \in \overline{H(\rho,\alpha_i,n)}$. We will choose $\alpha_{i+1} > \alpha_i$ and $\beta_{i+1} > \beta_i$, and first define $\rho'_{\beta_i\ldots\beta_{i+1}}$ such that $(\rho')_{\beta_{i+1}} \in \overline{H(\rho,\alpha_{i+1},n)}$, then define $\gamma_{\alpha_i\ldots\alpha_{i+1}}$. Let π be the path of $\mathcal{R}(\mathcal{A})$ which starts at $\mathsf{reg}((\rho')_{\beta_i})$, given by Lemma 1 for the run $\rho_{\alpha_i\ldots|\rho|}$. If ρ is finite and $|\rho| - \alpha_i \leq n$, then $\mathcal{D}$ has a run from γ_{α_i} on word $\mathsf{tr}(\pi)$ (since $L([\![\mathcal{A}_\Delta]\!],\rho'_{\alpha_i}) \subseteq L(\mathcal{D},\gamma_{\alpha_i})$) and we are done. Suppose now that ρ is either infinite, or $|\rho| - \alpha_i > n$. Then $|\pi| = n$, and by Lemma 9, π can be decomposed into $\pi = \pi_1\tau_1\ldots\tau_{K-1}\pi_2$ where τ_i's are equivalent cycles. We let $\alpha_{i+1} > \alpha_i$ such that $\mathsf{last}(\tau_{K-1})$ is the $(\alpha_{i+1}-\alpha_i)$-th state of π. $[\![\mathcal{A}_\Delta]\!]$ has a run from $(\rho')_{\beta_i}$ to some $z \in \mathsf{first}(\tau_1)$ following π_1 (in fact, $(\rho')_{\beta_i} \in \mathsf{first}(\pi_1)$). By construction of π, there exists $z' \in \overline{H(\rho,\alpha_{i+1},n)} \cap \overline{\mathsf{last}(\tau_{K-1})} \neq \emptyset$, and by Lemma 10, for any $k \geq k_0$ and any word $w \in (\mathsf{tr}(\tau_1)+\ldots+\mathsf{tr}(\tau_{K-1}))^k$, there is a run, in $[\![\mathcal{A}_\Delta]\!]$, from z to z' following trace w. Let $\rho''(w)$ denote the run thus constructed from $(\rho')_{\beta_i}$ to z' on $\mathsf{tr}(\pi_1)\cdot w$. We let β_{i+1} s.t. $\rho'_{\beta_i\ldots\beta_{i+1}}(w) = \rho''(w)$ for an arbitrary w.

Now, $\mathcal{D}$ has a run from $(\gamma)_{\alpha_i}$ to some state q_0 over trace $\mathsf{tr}(\pi_1)$ because $L([\![\mathcal{A}_\Delta]\!],(\rho')_{\beta_i}) \subseteq L(\mathcal{D},(\gamma)_{\alpha_i})$. Let $\mathcal{D}'$ denote the finite untimed automaton obtained from $\mathcal{D}$ by designating q_0 as the initial state, and all states q_f such that $L([\![\mathcal{A}_\Delta]\!],z') \subseteq L(\mathcal{D},q_f)$ for some $w \in (\mathsf{tr}(\tau_1)+\ldots+\mathsf{tr}(\tau_{K-1}))^k$, $k \geq k_0$, as final states. There is at least one final state because $L(\mathcal{D}) = L([\![\mathcal{A}_\Delta]\!])$ and $\mathcal{D}$ is deterministic. Let there be an edge in $\mathcal{D}'$ with label $\mathsf{tr}(\tau_i)$ from state q to q' whenever there is a path in $\mathcal{D}$ from q to q' over word $\mathsf{tr}(\tau_i)$. Observe that $\mathcal{D}'$ is still deterministic. Since $\rho''(w)$ is defined for any $w \in (\mathsf{tr}(\tau_1)+\ldots+\mathsf{tr}(\tau_{K-1}))^k$, $k \geq k_0$, $\mathcal{D}'$, defined over alphabet $\{\mathsf{tr}(\tau_1),\ldots,\mathsf{tr}(\tau_{K-1})\}$, is ultimately universal. But then,

by Lemma 8, $\mathcal{D}'$ accepts any word in $\{\mathsf{tr}(\tau_1), \ldots, \mathsf{tr}(\tau_{K-1})\}^{K-1}$, and in particular $\mathsf{tr}(\tau_1 \ldots \tau_{K-1})$. Therefore, there is a run in $\mathcal{D}$ from $(\gamma)_{\alpha_i}$ to some state $(\gamma)_{\alpha_{i+1}}$ following $\mathsf{tr}(\tau_1 \ldots \tau_{K-1})$, which satisfies $L([\![\mathcal{A}_\Delta]\!], (\rho')_{\beta_{i+1}}) \subseteq L([\![\mathcal{A}]\!], (\gamma)_{\alpha_{i+1}})$. □

Acknowledgement. I am grateful to Patricia Bouyer and Nicolas Markey for useful discussions on earlier versions of this paper.

References

1. Altisen, K., Tripakis, S.: Implementation of timed automata: An issue of semantics or modeling? In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 273–288. Springer, Heidelberg (2005)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., La Torre, S., Madhusudan, P.: Perturbed timed automata. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 70–85. Springer, Heidelberg (2005)
4. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. Symposium on System Structure and Control, pp. 469–474. Elsevier, Amsterdam (1998)
5. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Probabilistic and topological semantics for timed automata. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 179–191. Springer, Heidelberg (2007)
6. Bermond, J., Germa, A., Heydemann, M., Sotteau, D.: Longest paths in digraphs. Combinatorica 1, 337–341 (1981)
7. Bouyer, P., Chevalier, F.: On conciseness of extensions of timed automata. Journal of Automata, Languages and Combinatorics 10(4), 393–405 (2005)
8. Bouyer, P., Markey, N., Reynier, P.-A.: Robust model-checking of linear-time properties in timed automata. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 238–249. Springer, Heidelberg (2006)
9. Bouyer, P., Markey, N., Reynier, P.-A.: Robust analysis of timed automata *via* channel machines. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 157–171. Springer, Heidelberg (2008)
10. Brenguier, R., Sankur, O.: Hardness of untimed language universality (submitted, 2011)
11. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robust safety of timed automata. Formal Methods in System Design 33(1-3), 45–84 (2008)
12. De Wulf, M., Doyen, L., Raskin, J.-F.: Almost ASAP semantics: From timed models to timed implementations. Formal Aspects of Computing 17(3), 319–341 (2005)
13. Dima, C.: Dynamical properties of timed automata revisited. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 130–146. Springer, Heidelberg (2007)
14. Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust timed automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)
15. Puri, A.: Dynamical properties of timed systems. Discrete Event Dynamic Systems 10(1-2), 87–113 (2000)
16. Ramsey, F.P.: On a problem in formal logic. Proc. London Math. Soc (3) 30, 264–286 (1930)
17. Sankur, O.: Model-checking robuste des automates temporisés via les machines à canaux. Master's thesis, Ecole Normale Supérieure (2010)

Lower Bounds for Linear Decision Trees via an Energy Complexity Argument

Kei Uchizawa[1,⋆] and Eiji Takimoto[2,⋆⋆]

[1] Graduate School of Information Sciences, Tohoku University,
Aramaki Aoba-aza 6-6-05, Aoba-ku, Sendai, 980-8579, Japan
[2] Department of Informatics, Graduate School of Information Science and
Electrical Engineering, Kyushu University, 744 Motooka,
Nishi-ku, Fukuoka 819-0395, Japan
uchizawa@ecei.tohoku.ac.jp, eiji@i.kyushu-u.ac.jp

Abstract. A linear decision tree is a binary decision tree in which a classification rule at each internal node is defined by a linear threshold function. In this paper, we consider a linear decision tree T where the weights $w_1, w_2, ..., w_n$ of each linear threshold function satisfy $\sum_i |w_i| \leq w$ for an integer w, and prove that if T computes an n-variable Boolean function of large unbounded-error communication complexity (such as the Inner-Product function modulo two), then T must have $2^{\Omega(\sqrt{n})}/w$ leaves. To obtain the lower bound, we utilize a close relationship between the size of linear decision trees and the energy complexity of threshold circuits; the energy of a threshold circuit C is defined to be the maximum number of gates outputting "1," where the maximum is taken over all inputs to C. In addition, we consider threshold circuits of depth $\omega(1)$ and bounded energy, and provide two exponential lower bounds on the size (i.e., the number of gates) of such circuits.

1 Introduction

A linear decision tree is a binary decision tree in which a classification rule at each internal node is defined by a linear function so that right and left edges correspond to ≥ 0 and < 0, respectively. The complexity of a linear decision tree is usually measured by the depth and the size, where the depth is the length of the longest path from the root to a leaf and the size is the number of leaves. The depth and size are reasonable measures of time and space required for the corresponding algorithm, respectively. In previous research, the depth complexity of linear decision trees is extensively studied, and lower bounds are obtained for many problems [3–5, 8, 15]. In particular, Gröger and Turán obtain a linear lower bound on the depth of the linear decision trees computing the Inner-Product function IP_n [8]. However, the size complexity is less understood especially for Boolean functions. Since the depth is a lower bound on the size,

⋆ Supported by MEXT Grant-in-Aid for Young Scientists (B) No.23700003.
⋆⋆ Supported by MEXT Grant-in-Aid for Scientific Research (B) No.23300003.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 568–579, 2011.

the linear lower bound on the depth given in [8] yields a linear lower bound on the size. To the best of our knowledge, this is the largest known lower bound on the size of linear decision trees computing an explicit Boolean function.

In this paper, we restrict ourselves to the case where the weights of each linear functions are not too large, and show an exponential lower bound on the size of linear decision trees for a large class of Boolean functions including the Inner-Product function IP_n. More precisely, we prove that any linear decision tree computing an n-variable Boolean function of unbounded-error communication complexity $\Omega(n)$ has size $2^{\Omega(\sqrt{n})}/w$, provided that the linear threshold function at each internal node has a weight vector $(w_1, w_2, ..., w_n)$ such that $\sum_i |w_i| \leq w$; hence we have an exponential lower bound on the size if $w = 2^{o(\sqrt{n})}$. For the model of ternary decision trees computing a function $f : \mathbb{R}^n \to \{0, 1\}$ where each internal node gives ternary classification into "> 0", "$= 0$", or "< 0", exponential lower bounds are known even if the classification rule is defined by a high degree polynomial [2, 9]. However, they consider ternary decision trees computing a real function, and hence their results do not immediately imply lower bounds for binary linear decisions computing a Boolean function. In fact, the explicit function used to derive the exponential lower bound on the size of ternary decision trees in [9] can be computed by a binary linear decision tree of polynomial size. In the paper [5], Fleischer obtained an exponential lower bound on the size of binary linear decision trees, but they consider a real function too. For a simpler and more standard model of binary decision trees in which the classification rule is defined by a Boolean variable, Wegener derives an exponential lower bound on the size of the trees computing the Parity function [19].

To obtain our lower bound, we utilize a close relationship between the size of linear decision trees and the energy complexity of threshold circuits; a threshold circuit is a combinatorial circuit of threshold gates, and the energy of a threshold circuit C is defined to be the maximum number of gates outputting "1," where the maximum is taken over all inputs to C. More precisely, we use the following fact given by Uchizawa *et al.* [17]: if a function f cannot be computed by any threshold circuit of size $O(l)$, energy $O(\log l)$ and weight $O(w)$, then f cannot be computed by any linear decision tree of size $O(l)$ and weight $O(w)$ either. Thus, a lower bound on the size of threshold circuits of small energy and weight implies a lower bound on the size of linear decision trees. Using the unbounded-error communication complexity argument, we prove that if a threshold circuit C of energy e and weight w computes IP_n, then the size s of C is $s = 2^{\Omega(n/e)}/w$, which suffices to provide the desired lower bound for linear decision trees. Our result appears to be the first application of the notion of energy complexity of a threshold circuit for a computational model other than a threshold circuit, and shows a new method to obtain a lower bound: if a computational model can be simulated by a threshold circuit of small energy and weight, then we have a lower bound for the model.

In addition, we obtain lower bounds on the size (i.e., the number of gates) of threshold circuits of depth $\omega(1)$ and bounded energy. The above lower bound $2^{\Omega(n/e)}/w$ immediately yields an exponential lower bound on the size of threshold

circuits of energy $n^{o(1)}$ and weight $2^{o(n)}$. Note that this lower bound is independent of depth. We also consider the case where weights of threshold gates are unrestricted, and provide an exponential lower bound on the size of threshold circuits of depth $n^{o(1)}$ and energy $O(1)$. These two exponential lower bound are of independent interest, since they contrast with the known super-polynomial lower bounds for threshold circuits which require the depth to be a constant with further restrictions on fan-in, weight, or energy [1, 6, 7, 10–12, 14, 18].

2 Definitions

2.1 Linear Decision Trees

Let g be a *linear threshold function* with n inputs, weights $w_1, w_2, \ldots, w_n$ and a threshold t. Then, for every input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, $g(\boldsymbol{x}) = \mathrm{sign}\,(\sum_{i=1}^{n} w_i x_i - t)$ where $\mathrm{sign}(z) = 1$ if $z \geq 0$ and $\mathrm{sign}(z) = 0$ if $z < 0$. We assume throughout the paper that the weights and threshold of every threshold function are integers. A *linear decision tree* T computing a Boolean function of n variables is a binary decision tree in which each internal node is labeled by a linear threshold function of the n variables and each leaf is labeled by 0 or 1. For a given input $\boldsymbol{x} \in \{0,1\}^n$, the output $T(\boldsymbol{x})$ of T is determined by the following procedure starting from the root until reaching a leaf: if the linear threshold function at the current node outputs 0 for the input $\boldsymbol{x}$, then go to the left child; otherwise go the right. If the leaf reached is labeled by $z \in \{0,1\}$, then $T(\boldsymbol{x}) = z$. The *size* l of T is defined to be the number of leaves in T. We say that T has weight w if weights $w_1, w_2, \ldots, w_n$ of each threshold function in T satisfies $\sum_i |w_i| \leq w$.

2.2 Threshold Circuits

A *threshold gate* with an arbitrary number k of inputs computes a linear threshold function of k inputs. A *threshold circuit* is a directed acyclic graph where each internal node is a threshold gate or an input variable. The *size* s of a threshold circuit is defined to be the number of threshold gates in the circuit.

Let C be a threshold circuit computing a Boolean function f of n variables $x_1, x_2, \ldots, x_n$, and have size s. Let $g_1, g_2, \ldots, g_s$ be the gates in C, where $g_1, g_2, \ldots, g_s$ are topologically ordered with respect to the underlying directed acyclic graph of C. We regard the output of g_s as the *output* $C(\boldsymbol{x})$ *of* C, and call the gate g_s the *top gate* of C. A threshold circuit C *computes* a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if $C(\boldsymbol{x}) = f(\boldsymbol{x})$ for every input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$. We say that C has *weight* w if the sum of the absolute values of the weights for the n input variables $x_1, x_2, \ldots, x_n$ of each gate in C is at most w. Note that only the weights of the input corresponding to $x_1, x_2, \ldots, x_n$ are taken into account. The *level* of a gate in C is the length of the longest directed path to the gate from an input node. The *depth* d of C is the level of the top gate g_s of C. We define the *energy* e of a threshold circuit C as the maximum number of gates outputting "1", where the maximum is taken

over all inputs $\boldsymbol{x} \in \{0,1\}^n$ [17]. Thus, $e = \max_{\boldsymbol{x}\in\{0,1\}^n} \sum_{i=1}^{s} g_i(\boldsymbol{x})$, where $g_i(\boldsymbol{x})$ is the output of g_i for $\boldsymbol{x} \in \{0,1\}^n$. Clearly $0 \leq e \leq s$. It should be noted that the inequality $d \leq e$ does not necessarily holds for threshold circuits of depth d and energy e. For example, the Parity function of n variables can be computed by a threshold circuit of size $O(n)$, depth $O(n)$, and energy two [16].

2.3 Communication Complexity

Consider a game of two players, say Alice and Bob, with a Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, where Alice and Bob have unlimited computational power. Alice receives an input $\boldsymbol{x} \in \{0,1\}^n$ and Bob does an input $\boldsymbol{y} \in \{0,1\}^n$. Alice and Bob exchange bits according to a protocol, and try to compute the value $f(\boldsymbol{x}, \boldsymbol{y})$. The *cost* of a protocol is defined to be the maximum number of exchanged bits in the protocol. There are several variants of communication complexity measures of Boolean functions. In this paper, we consider the following three of them.

Definition 1. *The* deterministic communication complexity *of $f(\boldsymbol{x}, \boldsymbol{y})$, denoted by $D(f)$, is defined to the minimum cost over all the deterministic protocols that compute $f(\boldsymbol{x}, \boldsymbol{y})$ for every input $\boldsymbol{x} \times \boldsymbol{y} \in \{0,1\}^n \times \{0,1\}^n$.*

Definition 2. *Alice and Bob can use an unlimited "private" source of random bits. The* unbounded-error communication complexity *of $f(\boldsymbol{x}, \boldsymbol{y})$, denoted by $U(f)$, is defined to be the minimum cost over all the randomized protocols that compute $f(\boldsymbol{x}, \boldsymbol{y})$ correctly with probability strictly greater than $1/2$ for every input $\boldsymbol{x} \times \boldsymbol{y} \in \{0,1\}^n \times \{0,1\}^n$.*

Definition 3. *Alice and Bob share an unlimited "public" source of random bits. For each real number ϵ, $0 \leq \epsilon < 1/2$, the* bounded-error communication complexity *of $f(\boldsymbol{x}, \boldsymbol{y})$, denoted by $R_\epsilon(f)$, is defined to be the minimum cost over all the randomized protocols that compute $f(\boldsymbol{x}, \boldsymbol{y})$ correctly with probability $1 - \epsilon$ for every input $\boldsymbol{x} \times \boldsymbol{y} \in \{0,1\}^n \times \{0,1\}^n$.*

The Inner-Product function IP_n of $2n$ variables is defined as follows. For every pair of inputs $\boldsymbol{x} = (x_1, x_2, \cdots, x_n) \in \{0,1\}^n$ and $\boldsymbol{y} = (y_1, y_2, \cdots, y_n) \in \{0,1\}^n$, $IP_n(\boldsymbol{x}, \boldsymbol{y}) = x_1 y_1 \oplus x_2 y_2 \oplus \cdots \oplus x_n y_n$, where $\oplus$ denotes the XOR function. It is known that IP_n has large unbounded-error and bounded-error communication complexity:

Proposition 1 ([6, 13]). *$U(IP_n) = \Omega(n)$, and $R_{\frac{1}{2}-\delta}(IP_n) = \Omega(n + \log \delta)$ for every number δ, $0 < \delta \leq 1/2$.*

3 Lower Bounds for Linear Decision Trees

Our main result is the following lower bound on size of linear decision trees:

Theorem 1. *Let f be a Boolean function of n variables such that $U(f) = \Omega(n)$. If a linear decision tree T of weight w computes f, then the size of T is $2^{\Omega(\sqrt{n})}/w$.*

By Proposition 1 and Theorem 1, we immediately obtain the following exponential lower bound on the size of linear decision trees.

Corollary 1. *If a linear decision tree T of weight $w = 2^{o(\sqrt{n})}$ computes IP_n, then the size of T is $2^{\Omega(\sqrt{n})}$.*

We below give a proof of Theorem 1. In the paper [17], Uchizawa *et al.* show that there is close relationship between linear decision trees and threshold circuits of small energy, as follows.

Lemma 1 ([17]). *Assume that a Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a linear decision tree of l leaves and weight w. Then f can also be computed by a threshold circuit C of size $O(l)$, energy $O(\log l)$ and weight $O(w)$.*

In other words, if a function f cannot be computed by any threshold circuit of size $O(l)$, energy $O(\log l)$ and weight $O(w)$, then f cannot be computed by any linear decision tree of size $O(l)$ and weight $O(w)$. Thus, a lower bound for threshold circuits implies a lower bound for linear decision trees. The following theorem gives the desired lower bound.

Lemma 2. *Let f be a Boolean function of n variables such that $U(f) = \Omega(n)$. If f can be computed by a threshold circuit C of energy e and weight w, then the size s of C is $s = 2^{\Omega(n/e)}/w$.*

Combining Lemma 1 and 2, we can easily prove Theorem 1 as follows.

Proof of Theorem 1. Let T be a linear decision tree that computes IP_n and has size l and weight w. If $\log l \geq \sqrt{n}$, we are done. Consider the other case

$$\log l < \sqrt{n}. \tag{1}$$

Then Lemma 1 implies that IP_n can be computed by a threshold circuit C of size $s = O(l)$, energy $e = O(\log l)$ and weight $O(w)$. By Theorem 1, we have $l \geq 2^{\Omega(n/\log l)}/w$, and hence Eq. (1) implies that $l = 2^{\Omega(\sqrt{n})}/w$. □

Thus, it suffices to prove Lemma 2. We prove Lemma 2 by an unbounded-error communication complexity argument used in [7]. The following lemma summarizes the argument.

Lemma 3. *Assume that a Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be represented by a threshold function of a number k of Boolean functions $f_1, f_2, \ldots, f_k$ with weights $w_1, w_2, \ldots, w_k$, that is,*

$$f(\boldsymbol{x}) = sign\left(\sum_{i=1}^{k} w_i f_i(\boldsymbol{x})\right) \tag{2}$$

for every input $\boldsymbol{x} \in \{0,1\}^n$. Then

$$U(f) \leq \max_i D(f_i) + O(\log k). \tag{3}$$

We omitted the proof of Lemma 3 due to the page limitation.

Lemma 3 implies that we can obtain an upper bound on the unbounded-error communication complexity of a Boolean function f by expanding f to a linear combination of Boolean functions $f_1, f_2, \ldots, f_k$. The following lemma plays a key role in our proof, and give such Boolean functions.

Lemma 4. *Assume that a Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a threshold circuit C of size s, depth d, energy e and weight w. Then f can be represented by a threshold function of a number k of depth-2 threshold circuits $C_1, C_2, \ldots, C_k$ with weights $w_1, w_2, \ldots, w_k$, where*

$$k = \sum_{i=0}^{e} \binom{s}{i} \leq s^e + 1. \tag{4}$$

That is,

$$f(\boldsymbol{x}) = sign\left(\sum_{i=1}^{k} w_i C_i(\boldsymbol{x})\right) \tag{5}$$

for every input $\boldsymbol{x} \in \{0,1\}^n$. Besides, for every index i, $1 \leq i \leq k$, C_i has size

$$s_i \leq e + 1, \tag{6}$$

and weight w. Moreover, the weights $w_1, w_2, \ldots, w_k$ satisfy

$$\sum_{i=1}^{k} |w_i| \leq 3s^{2(d+1)^{e+1}}. \tag{7}$$

Remark. The above lemma has a quite similar form to Lemma 2 in [18]. The main difference between them is that the right hand side of Eq. (7) is $3s^{2(d+1)^{e+1}}$ while that of the corresponding equation of Lemma 2 in [18] is $2s^{3(e+1)^d}$. Note that the depth d and energy e symmetrically appear in the exponents in the formulas. This difference is critical to yield an exponential lower bound on the size of threshold circuits of depth $\omega(1)$ in Section 5. While the proof idea of the above lemma is mostly based on that of Lemma 2 in [18], we in fact require a new idea to obtain Eq. (7).

We prove Lemma 4 in the next section. Lemma 3 and 4 immediately imply Lemma 2, as follows.

Proof of Lemma 2. Let f be a Boolean function of $2n$ variables such that

$$U(f) = \Omega(n). \tag{8}$$

Assume that f can be computed by a threshold circuit C of size s, depth d, energy e and weight w. Lemma 4 implies that f can be represented by a threshold function of a number k of threshold circuits $C_1, C_2, \ldots, C_k$ with weights $w_1, w_2, \ldots, w_k$ satisfying Eqs. (4)–(7). For each integer i, $1 \leq i \leq k$, let f_i be the function that C_i computes, then Eq. (3) implies that

$$U(f) \leq \max_i D(f_i) + O(\log k). \tag{9}$$

Let s_i be the size of C_i. Since C_i has weight w, Alice and Bob can compute the output of every gate in C_i by such a protocol that one of the players sends in binary representation the sum of the products between the weights and the inputs of the gate. Hence we have $D(f_i) = O(s_i \log w)$ for every integer i, $1 \leq i \leq k$. Thus, Eqs. (4), (6) and (9) imply that

$$U(f) = O(s_i \log w + \log k) = O(e(\log w + \log s)) \tag{10}$$

By Eqs. (8) and (10), we obtain the desired result. □

4 Proof of Lemma 4

In this section, we prove Lemma 4. Assume that a function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a threshold circuit C of size s, depth d, energy e, and weight w. Let C consist of the gates $g_1, g_2, \ldots, g_s$. One may assume that $g_1, g_2, \ldots, g_s$ are topologically ordered with respect to the underlying directed acyclic graph of C, and that g_s is the output gate of C. For each index i, $1 \leq i \leq s$, we denote by $w_{i,1}, w_{i,2}, \ldots, w_{i,n}$ the weights of the gate g_i for the inputs $x_1, x_2, \ldots, x_n$ and by $w_{i,g_1}, w_{i,g_2}, \ldots, w_{i,g_s}$ the weights of g_i for the outputs of the gates $g_1, g_2, \ldots, g_s$, respectively. Since the gates $g_1, g_2, \ldots, g_s$ are topologically ordered, we have $w_{i,g_i} = w_{i,g_{i+1}} = \cdots = w_{i,g_s} = 0$. We denote by t_i the threshold of g_i. Thus, the output $g_i(\boldsymbol{x})$ of g_i for $\boldsymbol{x} \in \{0,1\}^n$ is represented as

$$g_i(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j=1}^{i-1} w_{i,g_j} g_j(\boldsymbol{x}) - t_i\right). \tag{11}$$

For each gate g_i, $1 \leq i \leq s$, we denote by $\mathrm{lev}(g_i)$ the level of the gate g_i in C. We shall present threshold circuits and weights satisfying Eqs. (4)–(7).

Define $\mathbb{S}$ as a family of subsets of $\{1, 2, \ldots, s\}$ such that $\mathbb{S} = \{S \subseteq \{1, 2, \ldots, s\} \mid 0 \leq |S| \leq e\}$. For every set $S \in \mathbb{S}$, we construct a depth-2 threshold circuit C_S consisting of $|S| + 1$ gates as follows. In the first level, the circuit C_S contains a gate g_i^S for every index $i \in S$ that computes

$$g_i^S(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j \in S} w_{i,g_j} - t_i\right) \tag{12}$$

for every input $\boldsymbol{x} \in \{0,1\}^n$. In the second level, C_S contains a gate g_S computing AND of the outputs of the gates g_i^S for every $i \in S$, that is,

$$C_S(\boldsymbol{x}) = g_S(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i \in S} g_i^S(\boldsymbol{x}) - |S|\right). \tag{13}$$

If $S = \emptyset$, then $C_S(\boldsymbol{x}) = \operatorname{sign}(0) = 1$ for every input $\boldsymbol{x} \in \{0,1\}^n$.

For each set $S \in \mathbb{S}$, we give the weight w_S for the circuit C_S as follows. For $S = \emptyset$, let $w_S = -1$. For each set $S \in \mathbb{S} \backslash \{\emptyset\}$, assume that S contain the

indices $i_1, i_2, \ldots, i_{|S|}$ such that $1 \leq i_1 < i_2 < \cdots < i_{|S|} \leq s$. Then we denote by $\boldsymbol{v} = (v_1, v_2, \ldots, v_e)$ the *weight vector* for S defined as follows: For each index j, $1 \leq j \leq e$,

$$v_j = \begin{cases} d + 1 - \mathrm{lev}(g_{i_j}) & \text{if } 1 \leq j \leq |S|; \\ 0 & \text{if } |S| + 1 \leq j \leq e. \end{cases}$$

Clearly, we have $0 \leq v_j \leq d$, for every index j, $1 \leq j \leq e$. Then we define $[S]$ as the integer whose $(d+1)$-nary representation is the weight vector $\boldsymbol{v}$, that is,

$$[S] = \sum_{j=1}^{e} v_j (d+1)^{e-j}.$$

Then let

$$w_S = \begin{cases} k^{[S]} & \text{if } g_s \in S; \\ -k^{[S]} & \text{otherwise.} \end{cases} \tag{14}$$

where $k = |\mathbb{S}| = \sum_{i=0}^{e} \binom{s}{i}$.

Consequently, we obtain the following threshold function:

$$\mathrm{sign}\left(\sum_{S \in \mathbb{S}} w_S C_S(\boldsymbol{x}) \right). \tag{15}$$

In the rest of the section, we prove that the threshold function (15) satisfies Eqs. (4)–(7).

Since we have $k = |\mathbb{S}| = \sum_{i=0}^{e} \binom{s}{i} \leq s^e + 1$, Eq. (4) holds. Clearly, for every set $S \in \mathbb{S}$, $\mathrm{size}(C_S) = |S| + 1 \leq e + 1$, and hence Eq. (6) holds. Moreover, it holds that

$$[S] \leq \sum_{j=1}^{e} d \cdot (d+1)^{e-i} \leq (d+1)^e$$

for every $S \in \mathbb{S}$. Hence

$$\begin{aligned} \sum_{S \in \mathbb{S}} |w_S| &= \sum_{S \in \mathbb{S}} k^{[S]} \\ &\leq k \cdot k^{(d+1)^e} \\ &\leq 3s^{2(d+1)^{e+1}}. \end{aligned}$$

We have thus proved Eq. (7). Below, we verify Eq. (5).

Consider an arbitrary and fixed input $\boldsymbol{x} \in \{0,1\}^n$. Let

$$S^* = \{i \in \{1, 2, \ldots, s\} \mid \text{the gate } g_i \text{ outputs 1 for } \boldsymbol{x} \text{ in } C\}.$$

Thus, for every index $i \in S^*$,

$$g_i(\boldsymbol{x}) = 1. \tag{16}$$

Let

$$\mathbb{F} = \{S \in \mathbb{S} \mid \forall i \in S, \text{ the gate } g_i^S \text{ outputs 1 for } \boldsymbol{x} \text{ in } C_S\},$$

then Eq. (13) implies that, for every set $S \in \mathbb{F}$,

$$C_S(\boldsymbol{x}) = g_S[\boldsymbol{x}] = 1. \tag{17}$$

Eqs. (11) and (12) imply that, for every index $i \in S^*$,

$$\begin{aligned} g_i(\boldsymbol{x}) &= \operatorname{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j=1}^{i-1} w_{i,g_j} g_j[\boldsymbol{x}] - t_i\right) \\ &= \operatorname{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j\in S^*} w_{i,g_j} - t_i\right) = g_i^{S^*}(\boldsymbol{x}) \end{aligned}$$

and hence Eq. (16) implies that $g_i^{S^*}(\boldsymbol{x}) = 1$. We thus have

$$S^* \in \mathbb{F}. \tag{18}$$

Eqs. (17) and (18) imply that

$$g_{S^*}[\boldsymbol{x}] = 1. \tag{19}$$

Then the following claim holds.

Claim. For every set $S \in \mathbb{F}\backslash\{S^*\}$, $[S] \leq [S^*] - 1$.

Proof idea. Let S be an arbitrary set in $\mathbb{F}$ such that

$$S \neq S^*. \tag{20}$$

Thus, for every index $i \in S$,

$$g_i^S(\boldsymbol{x}) = 1. \tag{21}$$

For each index j, $1 \leq j \leq d$, we define S_j as a set of indices i such that the gate g_i is in the level j of C, that is, $S_j = \{i \in S \mid \operatorname{lev}(g_i) = j\}$. Similarly, we define $S_j^* = \{i \in S^* \mid \operatorname{lev}(g_i) = j\}$.

By Eq. (20), there exists an index h such that, for every index j, $1 \leq j \leq h-1$,

$$S_j = S_j^* \tag{22}$$

and $S_h \neq S_h^*$. Let i be an arbitrary index in S_h. Then,

$$\begin{aligned} g_i^S(\boldsymbol{x}) &= \operatorname{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j\in S} w_{i,g_j} - t_i\right) \\ &= \operatorname{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j\in S_1\cup S_2\cup\cdots\cup S_{h-1}} w_{i,g_j} - t_i\right). \end{aligned} \tag{23}$$

By Eq. (22) and (23), we have

$$g_i^S(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j\in S^*} w_{i,g_j} - t_i\right)$$
$$= \mathrm{sign}\left(\sum_{j=1}^{n} w_{i,j}x_j + \sum_{j=1}^{i-1} w_{i,g_j} g_j(\boldsymbol{x}) - t_i\right) = g_i(\boldsymbol{x}) \tag{24}$$

Hence, by Eqs. (21) and (24) we have $g_i(\boldsymbol{x}) = 1$, and thus $i \in S_h^*$. Consequently, we have $S_h \subset S_h^*$, which implies the claim. □

We are now ready to prove Eq. (5). There are two cases to consider: (i) $f(\boldsymbol{x}) = C(\boldsymbol{x}) = 1$, and (ii) $f(\boldsymbol{x}) = C(\boldsymbol{x}) = 0$. We prove Eq. (5) only for the case (i), since the proof for the other case is similar. Consider an arbitrary input $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = C(\boldsymbol{x}) = 1$. In this case, it suffices to prove that

$$\sum_{S\in\mathbb{S}} w_S C_S(\boldsymbol{x}) \geq 0. \tag{25}$$

Eqs. (17) and (19) implies that

$$\sum_{S\in\mathbb{S}} w_S C_S(\boldsymbol{x}) = w_{S^*} + \sum_{S\in\mathbb{S}\setminus\{S^*\}} w_S C_S(\boldsymbol{x})$$
$$\geq w_{S^*} - \sum_{S\in\mathbb{F}\setminus\{S^*\}} |w_S|. \tag{26}$$

Since $C(\boldsymbol{x}) = g_s(\boldsymbol{x}) = 1$, we have $s \in S^*$, and hence Eqs. (14) and (26) imply that

$$\sum_{S\in\mathbb{S}} w_S C_S(\boldsymbol{x}) \geq k^{[S^*]} - \sum_{S\in\mathbb{F}\setminus\{S^*\}} k^{[S]}. \tag{27}$$

If $\mathbb{F} = \{S^*\}$, then we have $\sum_{S\in\mathbb{S}} w_S C_S(\boldsymbol{x}) \geq k^{[S^*]} > 0$, and hence Eq. (25) holds. If $\mathbb{F} \neq \{S^*\}$, then Claim 1 implies that $[S] \leq [S^*] - 1$ for every $S \in \mathbb{F}\setminus S^*$. Therefore, by Eq. (27) we have

$$\sum_{S\in\mathbb{S}} w_S C_S(\boldsymbol{x}) \geq k^{[S^*]} - (k-1)k^{[S^*]-1} > 0,$$

and hence Eq. (25) holds.

5 Lower Bounds for Threshold Circuits of Depth $\omega(1)$

In this section, we give two exponential lower bounds on the size of threshold circuits of depth $\omega(1)$. The first lower bound is immediately obtained from Lemma 2, as follows.

Theorem 2. *If IP_n can be computed by a threshold circuit C of size s, energy $e = n^{o(1)}$ and weight $w = 2^{o(n)}$, then $s = 2^{\Omega(n^{1-o(1)})}$.*

Note that the bound is, in fact, independent of depth.

The other lower bound is obtained from Lemma 3 by which arbitrary threshold circuit C computing a Boolean function f can be converted to a threshold function of a number k of depth 2 threshold circuits $C_1, C_2, \ldots, C_k$ of size at most $e + 1$. This lemma enable us to use a bounded-error communication complexity argument that yields a lower bound on the size of threshold circuits of depth three. More precisely, we obtain the following theorem.

Theorem 3. *Assume that f is a Boolean function of $2n$ variables such that*

$$R_{\frac{1}{2}-\delta}(f) = \Omega(n + \log \delta) \tag{28}$$

for every number δ, $0 < \delta \leq 1/2$. If a threshold circuit C of depth d and energy e computes f, then the size s of C satisfies

$$s = exp\left(\Omega\left(\frac{n}{e(d+1)^{(e+1)}}\right)\right) \tag{29}$$

We omit the proof of Theorem 3. Since IP_n satisfies Eq. (28), Theorem 3 immediately yields the following corollary.

Corollary 2. *If a threshold circuit C of depth $d = n^{o(1)}$ and energy $e = O(1)$ computes IP_n, then the size s of C satisfies $s = 2^{\Omega(n^{1-o(1)})}$.*

6 Conclusion

In this paper, we consider a binary linear decision tree T computing a Boolean function f, and prove that if T has weight w and f has large unbounded-error communication complexity, then T must have size $2^{\Omega(\sqrt{n})}/w$. Our result implies an exponential lower bound on the size of linear decision trees computing IP_n provided that $w = 2^{o(\sqrt{n})}$. The energy complexity of threshold circuits plays important role in our proof; and our result suggests that we can obtain a strong lower bound for some computational model if the model can be simulated by threshold circuits of small energy and weight.

In addition, we consider threshold circuits of depth $\omega(1)$ and bounded energy, and obtain an exponential lower bound on the size of threshold circuits of energy $n^{o(1)}$ and weight $2^{o(n)}$ and an exponential lower bound on the size of threshold circuits of depth $n^{o(1)}$ and energy $O(1)$.

References

1. Amano, K., Maruoka, A.: On the complexity of depth-2 circuits with threshold gates. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 107–118. Springer, Heidelberg (2005)
2. Björner, A., Lovász, L., Yao, A.C.C.: Linear decision trees: volume estimates and topological bounds. In: Procedings of the 24th ACM Symposyum on Theory of Computing, pp. 170–177 (1992)

3. Dobkin, D.P., Lipton, R.J.: On the complexity of computations under varying sets of primitives. Journal of Computer and System Sciences 3, 1–8 (1982)
4. Erickson, J.: Lower bounds for linear satisfiability problems. In: Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 388–395 (1995)
5. Fleischer, R.: Decision trees: Old and new results. Information and Computation 152, 44–61 (1999)
6. Forster, J.: A linear lower bound on the unbounded error probabilistic communication complexity. Journal of Computer and System Sciences 65, 612–625 (2002)
7. Forster, J., Krause, M., Lokam, S.V., Mubarakzjanov, R., Schmitt, N., Simon, H.U.: Relations between communication complexity, linear arrangements, and computational complexity. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 171–182. Springer, Heidelberg (2001)
8. Gröger, H.D., Turán, G.: On linear decision trees computing boolean functions. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 707–718. Springer, Heidelberg (1991)
9. Grigoriev, D., Karpinski, M., Yao, A.: An exponential lower bound on the size of algebraic decision trees for max. Computational Complexity 7(3), 193–203 (1998)
10. Hajnal, A., Maass, W., Pudlák, P., Szegedy, M., Turán, G.: Threshold circuits of bounded depth. Journal of Computer and Sysntem Sciences 46, 129–154 (1993)
11. Hansen, K.A., Podolskii, V.V.: Exact threshold circuits. In: Proceedings of the 25th Annual IEEE Conference on Computational Complexity, pp. 270–279 (2010)
12. Håstad, J., Goldmann, M.: On the power of small-depth threshold circuits. Computational Complexity 1, 113–129 (1993)
13. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)
14. Razborov, A., Wigderson, A.: $n^{\Omega(\log n)}$ lower bounds on the size of depth 3 threshold circuits with AND gates at the bottom. Information Processing Letters 45, 303–307 (1993)
15. Steele, J.M., Yao, A.C.: Lower bounds for algebraic decision trees. Journal of Algorithms 18, 86–91 (1979)
16. Suzuki, A., Uchizawa, K., Zhou, X.: Energy-efficient threshold circuits computing mod functions. In: Potanin, A., Viglas, T. (eds.) Proceedings of the 17th Computing: the Australasian Theory Symposium, Perth, Australia. CRPIT, ACS, vol. 119, pp. 105–110 (2011)
17. Uchizawa, K., Douglas, R., Maass, W.: On the computational power of threshold circuits with sparse activity. Neural Computation 18(12), 2994–3008 (2006)
18. Uchizawa, K., Takimoto, E.: Exponential lower bounds on the size of constant-depth threshold circuits with small energy complexity. Theoretical Computer Science 407(1-3), 474–487 (2008)
19. Wegener, I.: Optimal decision trees and one-time-only branching programs for symmetric boolean functions. Information and Control 62(2-3), 129–143 (1984)

Weak Cost Monadic Logic over Infinite Trees*

Michael Vanden Boom

Department of Computer Science, University of Oxford, UK
michael.vandenboom@cs.ox.ac.uk

Abstract. Cost monadic logic has been introduced recently as a quantitative extension to monadic second-order logic. A sentence in the logic defines a function from a set of structures to $\mathbb{N} \cup \{\infty\}$, modulo an equivalence relation which ignores exact values but preserves boundedness properties. The rich theory associated with these functions has already been studied over finite words and trees.

We extend the theory to infinite trees for the weak form of the logic (where second-order quantification is interpreted over finite sets). In particular, we show weak cost monadic logic is equivalent to weak cost automata, and finite-memory strategies suffice in the infinite two-player games derived from such automata. We use these results to provide a decision procedure for the logic and to show there is a function definable in cost monadic logic which is not definable in weak cost monadic logic.

1 Introduction

A fundamental result in the theory of regular languages is the equivalence between monadic second-order logic (MSO) and finite-state automata, which Büchi exploited in order to provide a decision procedure for the logic. Recently, Colcombet [3] has proposed a quantitative extension to MSO called cost monadic second-order logic (cost MSO). In this setting, a cost MSO sentence defines a function from some domain (like words or trees over a finite alphabet) to $\mathbb{N} \cup \{\infty\}$, modulo an equivalence relation $\approx$ which ignores exact values but preserves the existence of bounds over any subset of the domain. Mirroring the classical result, there is an equivalent automaton model called cost automata which can be used to help decide whether the functions definable by cost MSO sentences (over finite words [3] or finite trees [7]) are equivalent up to $\approx$.

This "theory of regular cost functions" is a strict extension to the theory of regular languages, which retains the equivalences, closure properties, and decidability of the classical theory. It captures the theory of regular languages since we can identify each language with its characteristic function mapping structures in the language to 0 and everything else to ∞; it is a strict extension since cost MSO can count some behaviour within the input structure (e.g. the number of positions labelled with some symbol). The theory has been studied over finite words [3,4] and finite trees [5,7]. This paper is an initial step in extending the theory to infinite trees when restricting to weak cost monadic logic (written cost WMSO) in which second-order quantifiers are interpreted over finite sets.

* Supported by the French ANR 2007 JCJC 0051 JADE.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 580–591, 2011.

1.1 Motivation

The motivation for studying the logic and automata considered in this paper comes from problems that can be reduced to questions of boundedness. The most famous language-theoretic problem like this is the "star-height problem": given a regular language L and natural number n, is there some regular expression (using concatenation, union, and Kleene star) for L which uses at most n nestings of Kleene-star operations? Hashiguchi [8] first showed that this problem is decidable over finite words and Kirsten [9] gave an alternative proof. Colcombet and Löding have shown that this problem is also decidable over finite trees [5].

In each case, finite-state automata enriched with counting features were used (distance, nested distance-desert, and cost automata). The star-height problem was then reduced to a question of "limitedness": given some automaton with counting features, is the function it computes bounded over all accepted structures? Because limitedness is a special case of deciding $\approx$, this theory of regular cost functions is a useful framework for reasoning about this sort of problem.

Kirsten [9] and Colcombet [3] cite problems in areas as diverse as speech recognition, databases, and model theory which have been solved using a similar approach. One open problem is the "parity-index problem". It asks: given a regular language L of infinite trees and $i < j$, is there a parity automaton using only priorities $\{i, i+1, \ldots, j\}$? Colcombet and Löding [6] have shown that this problem is reducible to limitedness for a "cost-parity automaton" over infinite trees, but the decidability of limitedness for these automata remains open. Thus, understanding the theory of regular cost functions over infinite trees is desirable.

1.2 Contributions

We show that the results over finite trees from [7] can be lifted to infinite trees when restricting to cost WMSO. The main contribution is proving cost WMSO is effectively equivalent to alternating "weak cost automata" and that these automata can be simulated by a type of non-deterministic automata. This is used to prove the relation $\approx$ is decidable for functions definable in cost WMSO. Another consequence (similar to the classical theory) is a separation result showing there is a function definable in MSO which is not definable in cost WMSO.

The main difficulty compared to the finite-tree case in [7] is the simulation result for weak cost automata. It relies on the fact that certain "weak cost games" (games derived from weak cost automata) admit finite-memory strategies. Proving this result over infinite trees is more difficult because of the interplay between the traditional acceptance condition and the cost features of the game.

1.3 Organisation

In Sect. 2 we provide background on trees, cost WMSO, and cost functions. We then present in Sect. 3 the general framework for reasoning about cost automata acting on infinite trees using infinite two-player games. Using this framework, we prove the results mentioned above for cost WMSO and weak cost automata in Sect. 4. We conclude in Sect. 5.

2 Cost Monadic Logic

2.1 Trees

Let $\mathbb{N}$ be the set of non-negative integers and $\mathbb{N}_\infty := \mathbb{N} \cup \{\infty\}$, ordered such that $0 < 1 < \cdots < \infty$. For $i \leq j$, we write $[i, j]$ to denote $\{i, i+1, \ldots, j\}$. We fix a finite alphabet $\mathbb{A}$ which is *ranked*: each symbol $a \in \mathbb{A}$ has some *arity* $j \in \mathbb{N}$ denoted by $a \in \mathbb{A}_j$. Let r be the maximum arity of the labels in $\mathbb{A}$. The set $\mathcal{T}_\mathbb{A}$ of infinite $\mathbb{A}$-*labelled ranked trees* is the set of partial functions $t : [1, r]^* \to \mathbb{A}$ such that the domain of t (denoted $pos(t)$) is prefix-closed, $t(\epsilon) \in \mathbb{A}$ is the label at the root, and if $t(x) \in \mathbb{A}_j$ then $t(xi)$ is defined if and only if $i \leq j$.

2.2 Cost Monadic Logic

Cost monadic second-order logic, or cost MSO, is a quantitative extension to MSO (see e.g., [14] for an introduction to monadic second-order logic). As usual, the logic can be defined over any relational structure, but we describe here the logic over $\mathbb{A}$-labelled ranked trees. In addition to first-order variables which range over nodes and second-order monadic variables which range over sets of nodes, cost MSO uses a single additional variable N called the *bound variable* which ranges over $\mathbb{N}$. The atomic formulas in cost MSO are the usual atomic formulas from MSO (namely, the membership relation $x \in X$ and relations $a(x, x_1, \ldots, x_k)$ which assert that $a \in \mathbb{A}_k$ is the label at position x with children $x_1, \ldots, x_k$ from left to right), as well as new predicates $|X| \leq N$ where X is any second-order variable and N is the bound variable. Arbitrary cost MSO formulas can be built inductively in the usual way by applying boolean connectives or by quantifying (existentially or universally) over first- or second-order variables. We additionally require that predicates of the form $|X| \leq N$ appear positively in the formula (i.e. within the scope of an even number of negations).

If we fix a value n for N, then the semantic of $|X| \leq N$ is what one would expect: the predicate is satisfied if and only if the valuation of X has cardinality at most n. If this value for N is not specified, then a sentence φ in cost monadic logic defines a function $[\![\varphi]\!]$, from $\mathcal{T}_\mathbb{A}$ to $\mathbb{N}_\infty$ (the natural numbers extended with a special infinity symbol ∞). The function is

$$[\![\varphi]\!](t) := \inf\{n : t, n \models \varphi\}$$

where $t, n \models \varphi$ if t satisfies φ when all occurrences of N take value n. By convention, $\inf \emptyset = \infty$, so in case φ is a pure MSO sentence (with no instances of the predicates $|X| \leq N$), $[\![\varphi]\!](t)$ is 0 if t satisfies the sentence φ and ∞ otherwise.

In Sect. 4, we will focus our attention on *cost weak monadic second-order logic* (written weak cost monadic logic or cost WMSO) which restricts the second-order quantification to finite sets, as usual. WMSO (and consequently cost WMSO) is still a very expressive logic (e.g. CTL embeds into it) but as we will see in Sect. 4, it has some nice properties that make working with the corresponding automata and games easier than in the full logic. We pause to give an example of a typical function definable in cost WMSO.

Example 1. Let $\mathbb{A} = \mathbb{A}_2 = \{a, b, c\}$. We seek to define the function which, for trees with infinitely many a's, outputs the maximum number of b's along a single branch (and otherwise assigns value ∞). A suitable cost WMSO sentence φ is

$$\forall X.\exists x.\big(\neg(x \in X) \wedge a(x)\big) \wedge \forall Z.\big((\forall z.(z \in Z \to b(z)) \wedge \mathit{chain}(Z)) \to |Z| \leq N\big).$$

where $\mathit{chain}(Z)$ is a WMSO formula asserting Z is totally ordered (and hence the nodes are on the same branch). We write $a(x)$ for $\exists x_1.\exists x_2.a(x, x_1, x_2)$.

The first conjunct is a typical WMSO formula: "infinitely many a's" is expressed as "for all finite sets of nodes, there is an a-labelled node outside". The least n that can be substituted for N to satisfy the second conjunct is exactly the bound on the number of b's along a single branch (∞ if there is no bound).

2.3 Cost Functions

Given cost WMSO sentences φ and ψ, we would like to be able to decide for any $t \in \mathcal{T}_\mathbb{A}$ whether $[\![\varphi]\!](t) \leq [\![\psi]\!](t)$ or $[\![\varphi]\!](t) = [\![\psi]\!](t)$. This is undecidable even over words by [10], so we must relax the relation being used. Following [4], we define relations $\preccurlyeq$ and $\approx$. Given $f, g : \mathcal{T}_\mathbb{A} \to \mathbb{N}_\infty$, we say g *dominates* f (written $f \preccurlyeq g$) if and only if for all $U \subseteq \mathcal{T}_\mathbb{A}$, $g(U)$ bounded implies $f(U)$ bounded. We write $f \approx g$ if and only if $f \preccurlyeq g$ and $g \preccurlyeq f$. Thus, $\preccurlyeq$ and $\approx$ are weakenings of $\leq$ and $=$ which ignore exact values of f and g, but do preserve boundedness properties.

If we want to be more precise about the relationship between f and g, we can annotate $\preccurlyeq$ and $\approx$ with a *correction function* $\alpha : \mathbb{N} \to \mathbb{N}$, a non-decreasing function which satisfies $\alpha(n) \geq n$ for all n. We write $f \preccurlyeq_\alpha g$ if $f(t) \leq \alpha(g(t))$ for all $t \in \mathcal{T}_\mathbb{A}$ (with the convention that $\alpha(\infty) = \infty$). Thus, α describes how much we may need to "stretch" g such that it dominates f. As an example, the function $|\cdot|_a$ which counts the number of a's in an infinite $\{a, b\}$-labelled tree is not $\approx$-equivalent to $|\cdot|_b$; however, $|\cdot|_a \approx_\alpha 2|\cdot|_a$ for $\alpha(n) = 2n$. To compare single values $m, n \in \mathbb{N}$, we also write $m \preccurlyeq_\alpha n$ if $m \leq \alpha(n)$.

With these relations, we can formally define a *cost function* F to be an equivalence class of $\approx$, but we will blur the distinction between a particular function $f : \mathcal{T}_\mathbb{A} \to \mathbb{N}_\infty$ and its equivalence class F. The natural decision procedure in this setting is the following: given two cost functions f and g, is $f \preccurlyeq g$? We remark that the classical language inclusion problem and the limitedness problem mentioned in the introduction can be seen as a special cases of this procedure.

3 Cost Games

3.1 Objectives

We will use a game-theoretic approach to tree automata (see e.g., [14]).

An *objective* O is a tuple $\langle \mathbb{C}, f, \mathit{goal}\rangle$ where $\mathbb{C}$ is a finite alphabet of actions, $f : \mathbb{C}^\omega \to \mathbb{N}_\infty$ maps sequences of actions to a value, and $\mathit{goal} \in \{\min, \max\}$ describes how a player seeks to optimize f. Switching the goal (from min to max or max to min) yields the dual objective $\overline{O}$ representing the aim of the opponent.

These objectives take the place of classical winning conditions. For example, with a *parity condition*, Eve wins if the maximum priority occurring

infinitely-often is even. This is described by the objective $\langle \Omega, P, \min \rangle$ where $\Omega \subseteq \mathbb{N}$ is a set of priorities and $P(u)$ is 0 if the maximum infinitely-occurring priority in u is even (∞ otherwise). Winning for Eve corresponds to minimizing P. The *Büchi condition* is a special case where $\Omega = [1, 2]$.

We want to enrich the classical objectives with counter actions such that values come from $\mathbb{N}_\infty$ (instead of only $\{0, \infty\}$). A counter γ is initially assigned value 0 and can be *incremented* $\mathtt{i}$, *reset* $\mathtt{r}$ to 0, *checked* $\mathtt{c}$, or left unchanged ε. We care about the value of the counter at the moment(s) when it is checked. Given a word u_γ over the alphabet $\{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{c}\}$, we define a set $C(u_\gamma) \subseteq \mathbb{N}$ which collects all of the checked values of γ. For instance, $C(\mathtt{iri}\varepsilon\mathtt{iicriic}) = \{2, 3\}$ since the first time the counter is checked it has value 3 and the second time it has value 2. In the case of a finite set of counters Γ and a word u over the alphabet $\{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{c}\}^\Gamma$, $C(u) := \bigcup_{\gamma \in \Gamma} C(u_\gamma)$ (u_γ is the γ-projection of u).

We will use three objectives which combine a classical parity condition with particular atomic counter actions and valuations. The *B-parity objective*[1] (over counters Γ and priorities Ω) is $Cost_B^{\Gamma,\Omega} := \langle \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma \times \Omega, cost_B^{\Gamma,\Omega}, \min \rangle$ where

$$cost_B^{\Gamma,\Omega}(u) := \sup(C(u) \cup \{P(u)\})$$

and $P(u)$ is the function described above which interprets the parity condition (on the projection of u to its last component). The atomic actions in this case are ε, $\mathtt{ic}$, and $\mathtt{r}$. For example, if $u = ((\mathtt{ic}, 2)(\mathtt{ic}, 2)(\varepsilon, 1)(\mathtt{r}, 2)(\mathtt{ic}, 1))^\omega$, then $cost_B^{\{1\},[1,2]}(u) = \sup(\{1, 2, 3\} \cup \{0\}) = 3$. The idea is that if the parity condition is satisfied (as in this example), then the value is the supremum of the checked counter values; otherwise, the counters are ignored and the value is ∞.

A useful variant of this B-objective is the *hB-parity objective*. In this case, the set of counters Γ is totally ordered and whenever a counter is incremented or reset, all lower counters are reset (we say the counters are *hierarchical*). Formally, we let $H_\Gamma := \{c \in \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma : c_\gamma \neq \varepsilon \text{ implies } c_{\gamma'} = \mathtt{r} \text{ for all } \gamma' < \gamma\}$ and then $Cost_{hB}^{\Gamma,\Omega} := \langle H_\Gamma \times \Omega, cost_B^{\Gamma,\Omega}, \min \rangle$.

The *S-parity objective* (over counters Γ and priorities Ω) has max as the goal and atomic actions ε, $\mathtt{i}$, $\mathtt{r}$, and $\mathtt{cr}$. It is $Cost_S^{\Gamma,\Omega} := \langle \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^\Gamma \times \Omega, cost_S^{\Gamma,\Omega}, \max \rangle$ where $cost_S^{\Gamma,\Omega}(u) := \inf(C(u) \cup \{\overline{P}(u)\})$ and $\overline{P}(u)$ is 0 (respectively, ∞) if $P(u)$ is ∞ (respectively, 0). In other words, if the parity condition is not satisfied then the counters are ignored and the value assigned is 0; otherwise, the minimum checked value is used (∞ if the counters are never checked).

3.2 Cost Games

A *cost game* $\mathcal{G} := \langle V, v_0, \delta, O \rangle$ consists of a set of positions V, an initial position $v_0 \in V$, an objective $O = \langle \mathbb{C}, f, goal \rangle$ for Eve, and a control function $\delta : V \to \mathcal{B}^+(\mathbb{C} \times V)$ (where $\mathcal{B}^+(\mathbb{C} \times V)$ is the set of positive boolean combinations, written as a disjunction of conjunctions of elements from $\mathbb{C} \times V$).

[1] This B and S notation was originally used in [2] to stand for counters which were bounded and strongly unbounded (whose limit tended towards infinity).

A *play* π is an infinite word $(v_i, c_{i+1}, v_{i+1})_{i\in\mathbb{N}} \in (V \times \mathbb{C} \times V)^\omega$ such that v_0 is the initial position and (c_{i+1}, v_{i+1}) appears in $\delta(v_i)$ for all $i \in \mathbb{N}$. Given a set σ of plays, let $\mathit{pref}(\sigma)$ denote the set of prefixes of plays in σ. We say $(v_0, c_1, v_1), \ldots, (v_j, c_{j+1}, v_{j+1}) \in \mathit{pref}(\sigma)$ is a partial play *ending* in v_{j+1} (by convention, we say $\epsilon \in \mathit{pref}(\sigma)$ ends in v_0). At a position $v \in V$, the positive boolean combination given by $\delta(v)$ can be viewed as a subgame in which Eve selects a disjunct in $\delta(v)$ and Adam selects a conjunct within this disjunct. For instance, if there is some partial play π ending in $v \in V$ with $\delta(v) = (c, v) \vee ((c', v') \wedge (c'', v''))$, then Eve can choose a disjunct, say $(c', v') \wedge (c'', v'')$, and Adam can choose one of the conjuncts in this disjunct, say (c'', v''). The play is then extended to $\pi \cdot (v, c'', v'')$ and c'' describes the cost for making this move.

A *strategy* σ for Eve is a set of plays σ such that if a partial play $\pi \in \mathit{pref}(\sigma)$ ends in position v, there must be some disjunct $(c'_1, v'_1) \wedge \ldots \wedge (c'_j, v'_j)$ in $\delta(v)$ such that for every conjunct (c'_i, v'_i) for $i \in [1, j]$, $\pi \cdot (v, c'_i, v'_i) \in \mathit{pref}(\sigma)$. We say σ is *positional* (or memoryless) if Eve's next move depends only on the current position rather than the history of the play (i.e. for all partial plays $\pi, \pi' \in \sigma$ ending in v, $\pi \cdot (v, c', v') \in \mathit{pref}(\sigma)$ if and only if $\pi' \cdot (v, c', v') \in \mathit{pref}(\sigma)$).

The objective $O = \langle \mathbb{C}, f, \mathit{goal}\rangle$ describes how to assign values. For a play $\pi = (v_i, c_{i+1}, v_{i+1})_{i\in\mathbb{N}}$, the value is $\mathit{val}(\pi) := f(\pi_\mathbb{C})$ where $\pi_\mathbb{C} := c_1 c_2 \ldots$. If *goal* is min, then the value of a strategy σ for Eve is $\mathit{val}(\sigma) := \sup\{\mathit{val}(\pi) : \pi \in \sigma\}$ and the value of the game is $\mathit{val}(\mathcal{G}) := \inf\{\mathit{val}(\sigma) : \sigma$ is a strategy for Eve in $\mathcal{G}\}$. In other words, Eve seeks to minimize over all strategies the maximum value of all plays compatible with the strategy. Dually, if *goal* is max, then $\mathit{val}(\sigma) := \inf\{\mathit{val}(\pi) : \pi \in \sigma\}$ and $\mathit{val}(\mathcal{G}) := \sup\{\mathit{val}(\sigma) : \sigma$ is a strategy for Eve in $\mathcal{G}\}$. We will refer to games by their objective (e.g. B-parity games).

The *dual* $\overline{\mathcal{G}}$ of a game $\mathcal{G}$ is obtained by switching disjunctions and conjunctions in the control function and using the dual objective (i.e. replacing min with max, and vice versa). This switches the roles of Adam and Eve.

3.3 Cost Automata

An *alternating cost automaton* $\mathcal{A} = \langle Q, \mathbb{A}, q_0, O, \delta\rangle$ has a finite set of states Q, a ranked alphabet $\mathbb{A}$, an initial state $q_0 \in Q$, an objective $O = \langle \mathbb{C}, f, \mathit{goal}\rangle$, and a transition function δ which maps $(q, a) \in Q \times \mathbb{A}_i$ to $\mathcal{B}^+([1, i] \times \mathbb{C} \times Q)$.

Given $t \in \mathcal{T}_\mathbb{A}$, we represent $\mathcal{A}$ acting on t via the cost game $\mathcal{A} \times t = \langle Q \times \mathit{pos}(t), (q_0, \epsilon), \delta', O\rangle$ where $\delta'((p, x)) = \delta(p, t(x))[(c, (q, xk))/(k, c, q)]$ and $\phi[s'/s]$ represents the formula ϕ with s' substituted for all occurrences of s. That is, a position in the game corresponds to a state of the automaton and a location in the input tree; the control function δ' modifies the transition function δ of the automaton to map to the appropriate positions in the game. We set $[\![\mathcal{A}]\!](t) := \mathit{val}(\mathcal{A} \times t)$, so $\mathcal{A}$ defines a function $[\![\mathcal{A}]\!] : \mathcal{T}_\mathbb{A} \to \mathbb{N}_\infty$. If there is a cost function g such that $[\![\mathcal{A}]\!] \approx g$ then we say that $\mathcal{A}$ *recognizes* g.

Notice that counter and priority actions occur on transitions. It is straightforward to translate between transition-labelled automata and the more common state-labelled automata (at the price of increasing the number of states).

In the simpler case that $\delta(q, a)$ for $a \in \mathbb{A}_i$ is a disjunction of statements of the form $\bigwedge_{j \in [1,i]}(j, c_j, q_j)$, then we say that the cost automaton is *non-deterministic*.

As with cost games, we will describe a cost automaton by its objective. A fundamental result, however, is that the B-, hB-, and S-objectives are equivalent.

Theorem 1. *It is effectively equivalent for a cost function f to be recognizable by a B-parity automaton, an hB-parity automaton, and an S-parity automaton.*

The proof requires results based on composing cost games with "history-deterministic" cost automata which translate between objectives (in analogy to the deterministic automata used in the translation between Muller and parity conditions). This technique was used for finite-duration cost games [7] and can be adapted to infinite cost games, but we will not develop this idea further here.

If the cost-parity automata are given as non-deterministic S-parity and B-parity automata, then it is also possible to decide $\preccurlyeq$.

Lemma 1. *The relation $f_1 \preccurlyeq f_2$ is decidable for cost functions f_1 and f_2 over infinite trees if f_1 is given by a non-deterministic S-parity automaton and f_2 is given by a non-deterministic B-parity automaton.*

The decision procedure is an adaptation of the proof in [7]. It uses ideas from the standard algorithm for deciding language inclusion for regular languages of infinite trees. We construct a product automaton combining f_1 and f_2 and allow Eve to "guess" a tree witnessing $f_1 \not\preccurlyeq f_2$ (this is possible since we are working with non-deterministic automata). In fact, through a series of translations, the decision of $f_1 \not\preccurlyeq f_2$ is reduced to solving a classical parity game (without costs).

4 Weak Cost Automata

We have defined the general framework for cost games and cost automata over infinite trees. In this section, we restrict our attention to a subclass of cost games which are derived from weak cost automata. We are able to show that this subclass defines the same cost functions as cost WMSO, and that weak B-games admit finite-memory strategies. Using these results, we show that it is decidable whether $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$ for cost WMSO sentences φ and ψ.

4.1 Weak Cost Automata

A *weak B-automaton* $\mathcal{A} = \langle Q, \mathbb{A}, q_0, Cost_B^{\Gamma,[1,2]}, \delta \rangle$ is an alternating B-Büchi automaton such that there is no cycle in δ using both priority 1 and 2. This corresponds to the standard definition (see e.g., [11]) but adapted to the case when priorities label transitions rather than states. A B-Büchi game such that every play stabilizes to moves using only a single priority is called a *weak B-game*. If $\mathcal{A}$ is a weak B-automaton, then $\mathcal{A} \times t$ is a weak B-game for all $t \in \mathcal{T}_{\mathbb{A}}$. Weak hB- and weak S-automata and games are defined by changing the objective.

Example 2. Let $\mathbb{A} = \mathbb{A}_2 = \{a, b, c\}$ and consider a 1-counter weak B-automaton $\mathcal{A} = \langle \{q_0, q_a, q_b, q_\top\}, \mathbb{A}, q_0, Cost_B^{\{1\},[1,2]}, \delta \rangle$.

We describe informally δ such that $\mathcal{A}$ computes the function from Example 1. Adam can either count the number of b's on some branch (incrementing and checking the counter while in state q_b), or prove there are only finitely many a's in the tree. If there are infinitely many a's then there is some branch τ such that an a-labelled node is reachable from each position on τ (but this a-labelled node does not need to be on τ itself). Eve picks out such a branch (marking it with q_0). At any point on this branch, Adam can move to state q_a and force Eve to witness a reachable a-labelled node. If she can, then the play stabilizes in $q_\top$.

The only transitions with priority 1 occur when in state q_a. The automaton can reach q_a only from q_0; once in q_a it can only move to $q_\top$. Thus, there is no cycle in the transition function which visits both priorities, so $\mathcal{A}$ is weak.

A useful notion from [12] is an *alternating chain*, a sequence of states $q_0 \dots q_n$ such that there is some $p \in [1, 2]$ with q_1 reachable from q_0 using transitions of priority p, and for all $i \in [1, n-1]$, q_{i+1} reachable from q_i using transitions of priority p (respectively, $\overline{p}$) if i is even (respectively, odd) ($\overline{1} = 2$ and $\overline{2} = 1$). We say the *length* of $q_0 q_1 \dots q_n$ is n. In the example, the automaton has alternating chains of length at most 2 ($q_0 q_a q_\top$). Since the length of these chains is bounded in weak cost automata, it can serve as an induction parameter in proofs.

4.2 Closure Properties

Instead of closure under union and intersection, weak cost automata are closed under min and max (the proof requires disjoint-union and product constructions as in the classical case). More interesting is closure under *weak* inf-*projection* and *weak* sup-*projection*. These operations correspond to finite projection in the classical setting. Let $h : \mathbb{A} \to \mathbb{B}$ be a map from ranked alphabets $\mathbb{A}$ and $\mathbb{B}$ such that $\mathbb{A} \supseteq \mathbb{B}$ and $h(b) = b$ for all $b \in \mathbb{B}$. We write $h(t') = t$ for the natural extension to trees which relabels each $\mathbb{A}$-labelled vertex of t' according to h. If t' contains only finitely many vertices labelled from $\mathbb{A} \setminus \mathbb{B}$, then we write $h_f(t') = t$. *Weak op-projection* of some cost function g over the alphabet $\mathbb{A}$ is the function $g_{op,h_f}(t) = op\,\{g(t') : h_f(t') = t\}$ over the alphabet $\mathbb{B}$ where op is inf or sup.

Generalizing [11, Lemma 1] and using results from [7], we can show weak B (respectively, weak S) automata are closed under weak inf-projection (respectively, weak sup-projection). Given a weak cost automaton for g and a tree t, we simulate it in "non-deterministic mode" on a finite prefix, then switch to "alternating mode" and run the original weak automaton on the remainder of t. While in non-deterministic mode, nodes labelled $b \in \mathbb{B}$ can be relabelled with some $a \in h^{-1}(b)$. By [7], this non-deterministic version on finite trees yields a value $\approx$-equivalent to the original alternating automaton. By the semantics of B (respectively, S) automata, the non-determinism is resolved into taking the infimum (respectively, supremum) of the values of $g(t')$ for t' satisfying $h_f(t') = t$.

Lemma 2. *Weak B-automata are closed under* min, max, *weak* inf-*projection. Weak S-automata are closed under* min, max, *weak* sup-*projection.*

4.3 Equivalence with Logic

It is no coincidence that we could express the cost WMSO sentence from Example 1 using the weak B-automaton in Example 2.

Theorem 2. *It is effectively equivalent for a cost function f to be definable in cost WMSO and recognizable by a weak cost automaton.*

Proof. (Sketch) To move from logic to automata, we use the standard technique of showing that each atomic formula is recognizable using a weak cost automaton, and then proving that these automata are closed under operations corresponding to other logical constructs (using Theorem 1 and Lemma 2).

To move from a weak cost automaton to a cost WMSO sentence, we do induction on the maximum length m of alternating chain. If $m = 0$, we can write a formula which assigns a value based strictly on the counters (it describes the existence of finite partial runs of a non-deterministic version of the automaton over finite trees, given by [7]). Otherwise, if $m > 0$, we find a finite partial run such that on each path, the automaton started with a transition of priority 1 but has passed through a transition of priority 2 and hence has reached a position with alternating chains strictly less than m (this may require converting between B- and S-versions of the automaton using Theorem 1). The inductive hypothesis yields formulas which correctly capture the value of the automaton on the continuations of the run, so we take the conjunction of these formulas and a formula describing the value on the initial partial run.

4.4 Shape of Strategies

A well-known result in the theory of infinite games is that parity games are positionally determined (see [14] for an introduction to this area). This means that from each position either Adam or Eve can win, and if Eve, say, has a winning strategy, then Eve has a positional strategy which is also winning.

In order to prove results like the decidability of $\preccurlyeq$, it becomes essential to have corresponding results about the strategies needed in cost games. Martin's theorem immediately implies that cost games are determined; in the cost setting, this means that the value of the original game and the dual game is the same.

Proposition 1. *For all cost games, $val(\mathcal{G}) = val(\overline{\mathcal{G}})$.*

There is no bound on the amount of memory Eve would need in order to achieve the optimal value in a cost game. However, in the cost setting, we just need to ensure that there is a positional strategy σ for Eve in $\mathcal{G}$ such that $val(\mathcal{G}) \approx_\alpha val(\sigma)$. If σ satisfies this condition, it means that by playing positionally according to σ, the error committed by Eve is bounded by α. It was shown that positional strategies suffice for finite-duration hB-games [5,7]. We now prove a similar result for weak hB-games in which the underlying game graph has no cycles. This is a reasonable restriction since the cost games $\mathcal{A} \times t$ where $\mathcal{A}$ is a weak hB-automaton and t is an infinite tree satisfy this requirement. (The result no longer holds for infinite game graphs with cycles.)

We start with an arbitrary strategy τ that witnesses a bounded cost n in a weak hB-game $\mathcal{G}$ with an acyclic game graph, alternating chains of length at most m, and k hierarchical counters. Without loss of generality, we assume that all edges from a position v in the game have the same priority p (denoted $\Omega(v) = p$). We consider the corresponding *strategy tree* T, the tree of all plays consistent with τ. Let V (respectively, S) denote the set of positions in $\mathcal{G}$ (respectively, T). Let $h : S \to V$ denote the homomorphism which maps a position in the strategy tree to the corresponding game position. Using an optimal strategy τ, Eve's choice at a particular $v \in V$ may depend on the history of the play leading to v. Thus, there may be $s, s' \in h^{-1}(v)$ such that the moves possible from s are different than from s'; however, because the game graph is acyclic, s, s', and v must be at the same depth. A positional strategy σ can be viewed as a mapping from V to S which for each v selects a single element of $h^{-1}(v)$ for Eve to use (regardless of the history). To build this map, we use the notion of "signatures".[2]

We first define the components of the signature. For the B-condition, we let $\beta_j(s)$ for $1 \leq j \leq k$ be the number of times a path from s can increment counter j before it is reset. Note that $\beta_j(s) \leq n$ for all $s \in S$ since T has a bounded cost n. The strategy should try to minimize β_j in order to minimize the cost.

For the weak acceptance condition, let $\beta_{\text{alt}}(s)$ be the maximum length of alternating chain on a path from s in T. Just minimizing β_{alt} would not guarantee that the resulting positional strategy satisfies the weak acceptance condition. Thus, we also define inductively a strictly increasing sequence of depths $(d_i)_{i \in \mathbb{N}}$ and a function $\beta : S \to \{0, 1\}$. The depths $(d_i)_{i \in \mathbb{N}}$ "slice" T based on reachability of transitions of priority 2. Let $d_0 := 0$. Given d_i, the depth $d_{i+1} > d_i$ is chosen such that every path from every position in the set $S_i = \{s \in S : s \text{ is at depth } d_i \text{ and } \Omega(s) = 1\}$ visits priority 2 by depth d_{i+1}. This uniform choice is possible since (i) there are only finitely many nodes of priority 1 at a particular depth (because cost games have finite branching), and (ii) from a particular node s with $\Omega(s) = 1$, there is a bound on the length of paths of priority 1 from s (if not, König's Lemma would imply that there is an infinite path of priority 1 in T, which is impossible).

Let s be a node between d_i and d_{i+1}. We set $\beta(s) := 0$ if every path from s can reach priority 2 by d_{i+1}. Otherwise, if $\Omega(s) = 1$ and some path from s cannot reach a node s' with $\Omega(s') = 2$ by d_{i+1}, then $\beta(s) := 1$. The strategy should minimize β_{alt} and β in order to ensure that plays stabilize in priority 2.

The *signature* for $s \in S$ is $sig(s) := \langle \beta_{\text{alt}}(s), \beta(s), \beta_k(s), \beta_{k-1}(s), \ldots, \beta_1(s) \rangle$. Let $\sigma : V \to S$ be the positional strategy which maps v to the element in $h^{-1}(v)$ with the lexicographically-least signature. It turns out that minimizing this signature ensures the weak acceptance condition is satisfied and the value of the play is still bounded.

Theorem 3. *If $\mathcal{G}$ is a weak hB-game with an acyclic game graph, alternating chains of length at most m, and k hierarchical counters, then there is a positional strategy σ (defined above) such that $val(\mathcal{G}) \approx_\alpha val(\sigma)$ for $\alpha(n) = 2m(n+1)^k$.*

[2] We do not use infinite ordinals in the definition of these signatures so, in that sense, it is simpler than many of the classical proofs which use this approach.

It is possible to generalize this technique to show that weak B-games and and B-Büchi games admit finite-memory strategies.

Theorem 4. *For all $k \in \mathbb{N}$, there exists α_k such that for any $\mathcal{G}$ which is a weak B-game (or its dual) or a B-Büchi game, if $\mathcal{G}$ has an acyclic game graph and k counters then there is a finite-memory strategy σ such that $val(\mathcal{G}) \approx_{\alpha_k} val(\sigma)$.*

4.5 Results

Taking advantage of Theorems 3 and 4, we can show that it is possible to simulate a weak cost automaton with a non-deterministic automaton. The idea is that the non-deterministic version guesses a finite-memory strategy in the weak cost game corresponding to the original weak automaton, and then computes its value.

Theorem 5. *If a cost function f is recognizable by a weak cost automaton $\mathcal{A}$ then a non-deterministic B-Büchi automaton $\mathcal{B}$ and non-deterministic S-Büchi automaton $\mathcal{S}$ can be effectively constructed from $\mathcal{A}$ such that $f \approx [\![\mathcal{B}]\!] \approx [\![\mathcal{S}]\!]$.*

The decidability for cost WMSO follows from Theorems 2, 5, and Lemma 1.

Corollary 1. *The relation $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$ is decidable for any cost WMSO sentences φ and ψ over infinite trees.*

Another nice consequence of Theorem 5 is a separation result. Rabin [13] showed there is a language of infinite trees definable in MSO not definable in WMSO. The separating language L consists of infinite trees over the alphabet $\{a, b\}$ on which every branch has finitely many b's. A similar result holds in the cost setting (and the separating function is the characteristic function of L).

Proposition 2. *There is a cost function over infinite trees definable in cost MSO (in fact, in pure MSO) which is not definable in cost WMSO.*

5 Conclusion

We have extended the framework of cost games and cost automata to infinite trees, building on the work from the finite-tree case [7]. In doing so, we were able to prove that the relations $\preccurlyeq$ and $\approx$ are decidable for cost WMSO, an expressive fragment of cost MSO. The proof relies crucially on the fact that finite-memory strategies suffice in the cost games derived from weak cost automata.

The natural extension of this work would be to show that full cost MSO is decidable over infinite trees. This paper and the work by Colcombet and Löding in [7] have already set the stage for such a result. The missing link is a proof that finite-memory strategies suffice in cost-parity games. As explained in [6], this is a challenging open problem because of the complex interplay between an arbitrary parity condition and the actions of the counters in the cost game.

Another interesting direction would be to compare cost WMSO with the extension of WMSO with the unboundedness operator $\mathbb{U}$, where $\mathbb{U}X.\varphi(X)$ expresses "there is no bound on the size of sets X satisfying φ". This logic has been

studied over infinite words in [1] where an equivalent automaton model called deterministic max automata are introduced. These automata lack non-determinism but allow an explicit max operation on counters. It would be interesting to find a similar deterministic model for cost automata over infinite words.

Acknowledgments. I would like to thank Thomas Colcombet, Denis Kuperberg, and the referees for their helpful comments and support.

References

1. Bojanczyk, M.: Weak MSO with the unbounding quantifier. In: Albers, S., Marion, J.Y. (eds.) STACS. LIPIcs, vol. 3, pp. 159–170. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)
2. Bojanczyk, M., Colcombet, T.: Bounds in ω-regularity. In: LICS, pp. 285–296. IEEE Computer Society, Los Alamitos (2006)
3. Colcombet, T.: Regular cost functions over words (2009), manuscript online
4. Colcombet, T.: The theory of stabilisation monoids and regular cost functions. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 139–150. Springer, Heidelberg (2009)
5. Colcombet, T., Löding, C.: The nesting-depth of disjunctive mu-calculus. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 416–430. Springer, Heidelberg (2008)
6. Colcombet, T., Löding, C.: The non-deterministic Mostowski hierarchy and distance-parity automata. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 398–409. Springer, Heidelberg (2008)
7. Colcombet, T., Löding, C.: Regular cost functions over finite trees. In: LICS, pp. 70–79. IEEE Computer Society, Los Alamitos (2010)
8. Hashiguchi, K.: Relative star height, star height and finite automata with distance functions. In: Pin, J.E. (ed.) LITP 1988. LNCS, vol. 386, pp. 74–88. Springer, Heidelberg (1989)
9. Kirsten, D.: Distance desert automata and the star height problem. RAIRO - Theoretical Informatics and Applications 39(3), 455–509 (2005)
10. Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. International Journal of Algebra and Computation 4, 405–425 (1994)
11. Muller, D.E., Saoudi, A., Schupp, P.E.: Alternating automata. the weak monadic theory of the tree, and its complexity. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 275–283. Springer, Heidelberg (1986)
12. Parigot, M.: Automata, games, and positive monadic theories of trees. In: Nori, K.V. (ed.) FSTTCS 1987. LNCS, vol. 287, pp. 44–57. Springer, Heidelberg (1987)
13. Rabin, M.O.: Weakly definable relations and special automata. In: Mathematical Logic and Foundations of Set Theory (Proc. Internat. Colloq., Jerusalem, 1968), pp. 1–23. North-Holland, Amsterdam (1970)
14. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, vol. 3, pp. 389–455. Springer, Heidelberg (1997)

Linear Problem Kernels for Planar Graph Problems with Small Distance Property

Jianxin Wang[1,⋆], Yongjie Yang[1], Jiong Guo[2,⋆⋆], and Jianer Chen[3]

[1] School of Information Science and Engineering
Central South University
Changsha 410083, P.R. China
{jxwang,yyjcsu}@mail.csu.edu.cn
[2] Universität des Saarlandes
Campus E 1.7, D-66123 Saarbrücken, Germany
jguo@mmci.uni-saarland.de
[3] Department of Computer Science and Engineering
Texas A&M University
College Station, Texas 77843-3112, USA

Abstract. Recently, various linear problem kernels for NP-hard planar graph problems have been achieved, finally resulting in a meta-theorem for classification of problems admitting linear kernels. Almost all of these results are based on a so-called region decomposition technique. In this paper, we introduce a simple partition of the vertex set to analyze kernels for planar graph problems which admit the distance property with small constants. Without introducing new reduction rules, this vertex partition directly leads to improved kernel sizes for several problems. Moreover, we derive new kernelization algorithms for Connected Vertex Cover, Edge Dominating Set, and Maximum Triangle Packing problems, further improving the kernel size upper bounds for these problems.

1 Introduction

Planar graph problems have been extensively studied in parameterized complexity theory. It is well-known that many problems being W[1]-hard on general graphs can be shown fixed-parameter tractable when restricted to planar graphs. Recently, deriving linear *problem kernels* for NP-hard planar graph problems received considerable attention. A *kernelization* algorithm has as input an instance (G, k) of a parameterized problem P and applies a set of polynomial-time executable data reduction rules to shrink the size of G, resulting in an equivalent instance (G', k'). Hereby, (G, k) is a yes-instance of P if and only if (G', k') is a yes-instance of P, $k' \leq k$, and the size of G' is bounded by a function $f(k)$. The new instance (G', k') is called the problem kernel, while the function $f(k)$

⋆ Supported by National Natural Science Foundation of China (60873265, 61073036); the Program for New Century Excellent Talents in University (NCET-10-0798).
⋆⋆ Supported by the DFG Cluster of Excellence MMCI.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 592–603, 2011.

is the kernel size. If f is a linear function, we have then a linear kernel. For more background on data reduction and problem kernels, we refer to [7,2].

The first non-trivial result on linear kernels of planar graph problems has been achieved by Alber, Fellows, and Niedermeier [1]; they proved a linear kernel for the Dominating Set problem on planar graphs. In this seminal paper they introduced a so-called "region decomposition" of planar graphs to analyze the size of the kernel. Later, Guo and Niedermeier [8] observed that this decomposition technique can be used for various planar graph problems which admitting a "distance property" and achieved linear kernels for Connected Vertex Cover, Edge Dominating Set, Maximum Triangle Packing, and Efficient Dominating Set. After this, a series of papers used this technique to show linear kernels for a variety of problems on planar graphs, such as Induced Matching [13], Full-Degree Spanning Tree [9], and Connected Dominating Set [6,10,11]. Finally, Bodlaender et al. [3] came up with a meta-theorem for classifying problems with linear problem kernels on planar graphs, using a decomposition similar to region decompositions.

In this paper, we introduce a simple vertex partition method for analyzing kernels of problems on planar graphs, and based on this, we improve the kernels for several problems which admit the distance property with small constants, introduced by Guo and Niedermeier [8] as the precondition for applying their framework for designing linear kernelization algorithms. A problem is said to admit the distance property with constants c_V and c_E if

1. the problem asks for a set S of vertices or edges satisfying a specified property and
2. for every solution set S with the vertex set $V(S)$, it holds that,

$$\forall u{\in}V, \exists v{\in}V(S), d(v,u){\leq}c_V \quad \text{and } \forall e{\in}E, \exists v{\in}V(S), d(e,v){\leq}c_E,$$

where the distance $d(v,u)$ between two vertices v and u is the length of a shortest path between them and the distance $d(e,w)$ between an edge $e = (u,v)$ and a vertex w is the minimum of $d(u,w)$ and $d(v,w)$.

With the general framework [8], linear kernels have been achieved for Connected Vertex Cover, Edge Dominating Set, and Maximum Triangle Packing on planar graphs, with the size bounds of $14k$, $28k$[1], and $624k$, respectively. We demonstrate the power of the new vertex partition method by showing that, without introducing new data reduction rules, the new method directly improves the upper bounds on the kernels for Connected Vertex Cover, Edge Dominating Set, and Maximum Triangle Packing. These problems are defined in the following. Moreover, we derive new data reduction rules for these problems, arriving at new kernel sizes $4k$, $12k$, and $75k$.

Definition. This paper considers only undirected, loopless, simple graphs. For a given graph $G = (V,E)$, let $V(G)$ denote the vertex set of G and $E(G)$ denote

[1] The analysis in [8] for Edge Dominating Set mistakes the number of edges as the number of vertices in the solution set. Thus, the actual kernel size achieved there is $28k$ instead of $14k$.

the edge set of G. We use K_3 to denote a triangle, that is, a graph with 3 vertices and 3 edges. For a vertex v, $N(v)$ is the open neighborhood of v. The degree of v, denoted by $\deg(v)$, is the size of $N(v)$. A vertex in a graph is a *cut-vertex* if the removal of the vertex increases the number of connected components. For $V' \subseteq V$, define $N(V') := \{u | u \in V \setminus V', \exists v \in V', (v,u) \in E\}$ and we denote by $G[V']$ the subgraph of G which is induced by V'. Denote by $G - V'$ and $G - E'$ the graphs obtain from G by removing V' and E', respectively, where $V' \subseteq V$ and $E' \subseteq E$.

A *vertex cover* C is a vertex subset of a graph G such that every edge has at least one of its endpoints in C. A *connected vertex cover* C is a vertex cover with the additional requirement that $G[C]$ is connected. An *edge dominating set* D is an edge subset such that every edge is either in D or has an common endpoint with some edge in D. A *triangle packing* is a subgraph of G consisting of vertex-disjoint triangles. The size of a triangle packing is the number of triangles it contains. The problems considered here have a planar graph G (we always assume that G dose not contain any isolated vertices) and an integer $k > 0$ as input. (Connected) Vertex Cover asks for a (connected) vertex cover with size at most k. Edge Dominating Set seeks for an edge dominating set with at most k edges. Finally, Maximum Triangle Packing decides whether there is a triangle packing with at least k vertex-disjoint triangles.

2 A Simple Vertex Partition for Analysis of Kernels

We first introduce a simple partition of the vertex set of a graph and show that for some problems satisfying the distance property with $c_V = 1$, this partition leads directly to better upper bounds on the kernel sizes without introducing new data reduction rules. Let $I = (G = (V, E), k)$ be an instance of such a problem P. In the following, we always use S to denote the vertex set of a solution for P on I. Setting $J := V \setminus S$, we define the following subsets:

$J_3 := \{v \in J \mid |N(v) \cap S| \geq 3\}$,
$J_2 := \{v \in J \mid |N(v) \cap S| = 2\}$,
$J_1 := \{v \in J \mid |N(v) \cap S| = 1\}$, and
$J_0 := J \setminus (J_3 \cup J_2 \cup J_1)$.

Given a planar graph G for a problem satisfying the distance property $c_V = 1$, we can easily conclude that $J_0 = \emptyset$. Moreover, the planarity of G gives directly the bound on $|J_3|$:

Lemma 1. *Given a planar graph $G = (V, E)$ as the input instance for a problem P and the vertex set $S \subseteq V$ of a solution for P on G, we have $|J_3| \leq \max\{0, 2(|S| - 2)\}$.*

Proof. Consider the bigraph $B := G[S \cup J_3] - E(G[S]) - E(G[J_3])$. It is well-known that any triangle-free planar graph with $n \geq 3$ vertices and m edges satisfies $m \leq 2n - 4$. Therefore, $3|J_3| \leq |E(B)| \leq 2(|S| + |J_3|) - 4$, which implies that $|J_3| \leq 2(|S| - 2)$. □

While analyzing the kernel sizes for problems with $c_V = 1$ on planar graphs, we only need to bound the sizes of J_1 and J_2. To this end, we partition J_2 into the following subsets $J_2(uv) := \{w \in J_2 \mid N(w) \cap S = \{u, v\}\}$ for all $u, v \in S$. Since each such subset can be considered as an edge between two vertices in S, the planarity of G directly implies a bound on the number of non-empty subsets.

Lemma 2. *Given a planar graph $G = (V, E)$ as the input instance for a problem P and the vertex set $S \subseteq V$ of a solution for P on G, there are at most $3|S| - 6$ pairs of vertices $u, v \in S$ for which the corresponding set $J_2(uv)$ is not empty.*

To get a linear-size problem kernel, it remains to bound the size of J_1 and the maximal size of the non-empty $J_2(uv)$. We can show that with the same set of reduction rules from [8], we can use the above vertex partition to derive better kernel size bounds than the ones in [8]: For Vertex Cover, there are two rules introduced to get a kernel with at most $10k$ vertices [8], one removing the degree-1 vertices and the other removing the common degree-2 neighbors of two vertices u and v, if u and v have at least two such neighbors. Now we analyze the kernel size with the vertex partition method. If a reduced instance has a vertex cover S with at most k vertices, we have then $J_0 = J_1 = \emptyset$. Moreover, by Lemma 1 and Lemma 2, $|J_3| \leq 2k - 4$ and there are at most $3|S| - 6$ pairs of vertices in S which have non-empty $J_2(uv)$. Since each $J_2(uv)$ has at most 1 vertex, the total kernel size is bounded by $6k$. For Connected Vertex Cover, the first rule in [8] removes the degree-1 neighbors of each vertex, except for one. Then, if two vertices have more than two common degree-2 neighbors, only 2 of them are kept. Thus, $|J_2| \leq 2(3|S|-6) \leq 6k-12$ and $|J_1| \leq |S|$. Then, the kernel size is bounded by $10k$. Since the rules for Edge Dominating Set in [8] are the same as for Connected Vertex Cover, the same argument works as well. However, the vertex set of the edge dominating sets can have at most $2k$ vertices. Thus, the kernel here can have at most $20k$ vertices. We can also achieve better kernel size bounds with the vertex partition method for Maximum Triangle Packing, based on a more complicated analysis. The details of this analysis is omitted since we will present some new rules and an even better bound for this problem in Sect. 5.

In the next sections, we introduce new data reduction rules to achieve more improved kernels for Connected Vertex Cover, Edge Dominating Set, and Maximum Triangle Packing.

3 Connected Vertex Cover

The Connected Vertex Cover (CVC) problem is known to be NP-hard, even for 2-connected planar graphs with maximum degree 4 [15]. While CVC on general graphs has no kernels of polynomial size [4], Guo and Niedermeier [8] gave a kernel with at most $14k$ vertices for planar graphs. We apply the following 2 data reduction rules. The first one has been used in [8].

For a vertex v, we define $N^1(v)$ as the set of vertices which are in $N(v)$ and have degree one, that is, $N^1(v) := \{u | u \in N(v), \deg(u) = 1\}$.

Rule 1. If there is a vertex v with $|N^1(v)| \geq 2$, remove all the vertices in $N^1(v)$ except for one.
Rule 2. For a vertex v with $N(v) = \{u, w\}$, if v is not a cut-vertex, then remove v, and if $N^1(u) = \emptyset$ (resp. $N^1(w) = \emptyset$), add a new degree-1 neighbor to u (resp. w).

The correctness of Rule 2 follows from the following lemma.

Lemma 3. *Let v be a degree-2 vertex such that $N(v) = \{u, w\}$, and v is not a cut-vertex. Then, G has a connected vertex cover of size bounded by k if and only if $G - \{v\}$ has a connected vertex cover of size bounded by k that contains u and w.*

Proof. Suppose that G has a connected vertex cover C of size bounded by k. If $v \notin C$, then C must contain both u and w, and we are done; otherwise, we distinguish two cases. If v is a degree-1 vertex in $G[C]$, say $u \in C$ but $w \notin C$, then clearly all neighbors of w must be included in C. Since v is not a cut-vertex, w has at least one other neighbor than v. Therefore, $C \setminus \{v\} \cup \{w\}$ must be connected and covers all edges in $G - \{v\}$. If v is not a degree-1 vertex in $G[C]$, then both u and w are in C. If $G[C \setminus \{v\}]$ is connected, we are then done, since $C \setminus \{v\}$ remains a connected vertex cover of $G - \{v\}$ and thus, satisfies the claim; otherwise, we have exactly two connected components in $G[C \setminus \{v\}]$, one containing u and the other containing w. Since v is not a cut-vertex of G, v must be contained in a cycle in G, together with u and w. Let A be the set of other vertices on this cycle. Since C is a vertex cover, there must be a vertex $b \in A \setminus C$ which is adjacent to both components. Replacing v by b then gives a connected vertex cover for $G - \{v\}$. The other direction is clearly true. □

Obviously, all rules run in polynomial time. We arrive now at the kernel size.

Theorem 4. Connected Vertex Cover *on planar graphs admits a problem kernel with at most $4k - 4$ vertices.*

Proof. Let S denote a solution for a reduced instance $(G = (V, E), k)$. Without loss of generality, assume $|S| \geq 2$. We partition the vertices in G as introduced in Sect. 2 with respect to S. Clearly, $|J_0| = 0$ and by Lemma 1, $|J_3| \leq 2|S| - 4$. Since J is independent, each vertex in J_1 has only one neighbor and this neighbor is in S. Due to Rule 1, $|J_1| \leq |S|$. Moreover, since all cut-vertices must be contained in any connected vertex cover, Rule 2 implies that there exists no degree-2 vertex in J, and thus, $J_2 = \emptyset$. Thus, we have $|V| = |J_3| + |J_1| + |S| \leq 4|S| - 4 \leq 4k - 4$. □

4 Edge Dominating Set

The Edge Dominating Set (EDS) problem is NP-hard on planar graphs [5]. The currently best kernel of EDS is of size $O(k^2)$ for general graphs [14] and with at most $28k$ vertices for planar graphs [8].

Let v be a vertex in G, and D be an edge dominating set of G. We say D contains v, or v is contained in D if v is an endpoint of some edge of D. For two vertices v and u, we define $N^2(v,u) := \{w | w \in N(v) \cap N(u), \deg(w) = 2\}$.

Rule 1. If there is a vertex v with $|N^1(v)| \geq 2$, remove all the vertices in $N^1(v)$ except for one.

Rule 2. If there are two vertices v and u with $(v,u) \notin E(G)$ and $|N^2(v,u)| \geq 2$, remove all the vertices in $N^2(v,u)$, and if $N^1(v) = \emptyset$ (resp. $N^1(u) = \emptyset$), introduce a new degree-1 vertex adjacent to v (resp. u).

Rule 3. Suppose that G is reduced with respect to Rule 1. If there is an edge (v,u) with $|N^2(v,u) \cup N^1(v) \cup N^1(u)| \geq 2$, remove all vertices in $N^2(v,u) \cup N^1(v) \cup N^1(u)$, and introduce a new degree-2 vertex adjacent to both v and u.

The correctness of Rule 1 is obvious. The next two lemmas prove the correctness of Rules 2 and 3, respectively.

Lemma 5. *Let v and u be two vertices with $(v,u) \notin E(G)$ and $|N^2(v,u)| \geq 2$, then there exists a minimum edge dominating set that contains both v and u.*

Proof. Suppose that D is a minimum edge dominating set for G. In order to cover all the edges between $\{v,u\}$ and $N^2(v,u)$, at least two vertices from $\{v,u\} \cup N^2(v,u)$ must be contained in D. Since $N^2(v,u)$ induces an independent set and $N(N^2(v,u)) = \{v,u\}$, one of v and u must be in D. If only one of v and u is contained in D, without loss of generality, suppose that v is contained in D and u is not. Consider two vertices $x, y \in N^2(v,u)$. In order to cover (x,u) and (y,u), both (v,x) and (v,y) must be in D. Replacing (v,y) by (u,y) clearly gives a new edge dominating set D' with $|D| = |D'|$. The lemma follows. □

Lemma 6. *Let w be a vertex with $N(w) = \{v,u\}$ and $(v,u) \in E(G)$. There exists a minimum edge dominating set that contains both v and u.*

Proof. Assume there is a minimum edge dominating set D for G that contains at most one of v and u. Without loss of generality, we suppose that v is not contained in D. Then, $(u,w) \in D$. Replace (u,w) by (v,u) in D, and denote the new set as D'. Since all the edges dominated by (u,w) can also be dominated by (v,u), D' is a minimum edge dominating set of G. □

The polynomial running time of the rules is easy to see. We arrive now at the kernel size.

Theorem 7. Edge Dominating Set *on planar graphs admits a kernel with at most $12k - 10$ vertices.*

Proof. Let $(G = (V,E), k)$ be a reduced instance with respect to the three rules. Suppose that G has an edge dominating set D with $|D| \leq k$. The corresponding vertex set $S := V(D)$ contains at most $2k$ vertices. We consider again the decomposition introduced in Sect. 2. Clearly, $|J_0| = 0$ and $|J_3| \leq 2|S| - 4$. Moreover, since $V \setminus S$ induces an independent set, all vertices in J_2 are degree-2 vertices and the vertices in J_1 have degree one. Let $S_1 := S \cap N(J_1)$ and $S_2 := S \setminus S_1$.

We introduce an auxiliary graph $G_S = (S, E_{S_1} \cup E_J)$, where $E_{S_1} = \{(v,u) \mid (v,u) \in D, v \in S_1 \text{ and } u \in S_2\}$, and $E_J = \{(v,u) \mid \{v,u\} \subseteq S \text{ and } N^2(v,u) \neq \emptyset\}$.

Obviously, G_S is a planar graph and thus, $|E(G_S)| \leq 3|S| - 6$. Since S_1 induces an independent set in G (this is true, since, otherwise, Rule 3 would be applied) and $S_1 \subseteq S$, for each vertex $v \in S_1$, there exists an edge $(v, u) \in D$ with $u \in S_2$. Thus, we have $|E_{S_1}| \geq |S_1|$. By Rule 3, there exists no edge $(v, u) \in D$ with $v \in S_1$, $u \in S_2$ and $N^2(v, u) \neq \emptyset$ and thus, we have $E_{S_1} \cap E_J = \emptyset$. By Rules 2 and 3, any two vertices in S share at most one common degree-2 neighbor. Thus, $|J_2| = |E(G_S)| - |E_{S_1}| \leq |E(G_S)| - |S_1|$. It is clearly that $|J_1| \leq |S_1|$, therefore, $|J_1| + |J_2| \leq |E(G_S)| \leq 3|S| - 6$. Thus, we have $|V| = |S| + |J_1| + |J_2| + |J_3| \leq 6|S| - 10$. Together with $|S| \leq 2k$, the size bound follow. □

5 Maximum Triangle Packing

The Maximum Triangle Packing (MTP) problem is NP-hard on planar graphs [5]. A kernel of size $45k^2$ for MTP on general graphs [12] and a kernel with at most $624k$ vertices for MTP on planar graphs [8] have been known. As already observed in [8], MTP does not satisfy the distance property before applying the following cleaning rule introduced in [8]:

Cleaning rule. Remove all vertices and edges that are not in any triangle.

After removing these vertices and edges, MTP has the distance constants $c_V = 1$ and $c_E = 1$. Note that (Connected) Vertex Cover and Edge Dominating Set have the constant $c_E = 0$. With $c_E = 1$, we have some edges outside S which make the analysis more involved. The following 6 rules together with the cleaning rule are applied. The first three are from [8].

Rule 1. If a vertex v has two neighbors w_1 and w_2 that form a triangle with v but are not involved in any other triangle that does not contain v, then remove v, w_1, and w_2 and decrease the parameter k by one.

Rule 2. If v and u have two common neighbors w_1 and w_2 such that $N(w_1) = \{v, u\}$ and w_2 is only contained in triangles that also contains v or u, then remove w_1.

Rule 3. If there are six vertices v, u, w_1, w_2, w_3, w_4 such that u, w_1, w_2 form a triangle, v, w_3, w_4 form a triangle, and there is no triangle that contains one of w_1, w_2, w_3, w_4 but none of v and u, then remove v, u, w_1, w_2, w_3, w_4 and decrease the parameter k by two.

Now we introduce three new reduction rules.

Rule 4. If there are four vertices v, u, w_1, w_2 such that v, u, w_1 form a triangle, v, u, w_2 form a triangle, w_1 is only contained in triangles that also contains v, and w_2 is only contained in triangles that also contains v or u, then remove (u, w_1).

Rule 5. Consider four vertices v, u, w_1, w_2 such that $\{w_1, w_2\} \subseteq N(v) \cap N(u)$, $(w_1, w_2) \in E(G)$ and every triangle containing w_1 contains also u or v. If there is a vertex x such that $N(x) = \{v, u, w_1\}$ or there are two vertices $a, b \in N(w_1)$ such that every triangle containing a contains u and every triangle containing b contains v, then remove (w_1, w_2).

Rule 6. Consider three vertices u, w_1, w_2 such that $\{w_1, w_2\} \subseteq N(u)$, $(w_1, w_2) \in E(G)$, and every triangle containing w_2 contains u or w_1. If there is a

vertex $a \in N(w_2) \cap N(u)$ such that every triangle containing a contains u, then remove (w_1, u).

Lemma 8. *Rules 4, 5, and 6 are correct.*

Proof. Since the graph G' resulting by applying Rules 4 to 6 to a given graph G is a subgraph of G, every triangle packing of G' is a triangle packing of G. We prove now that if there is a triangle packing P of size at least k for G, then G' has a triangle packing of size at least k as well.

Rule 4. If (u, w_1) is not contained in P, then we are done; otherwise, since w_1 is only contained in triangles that also contains v and w_2 is only contained in triangles that also contains v or u, the vertices u, w_1, and v form a triangle of P and w_2 is not contained in P. Thus, replace the triangle formed by v, u, w_1 by the triangle formed by v, u, w_2, we can get another triangle packing that contains at least k triangles for $G - \{(u, w_1)\}$.

Rule 5. If (w_1, w_2) is not contained in P, then we are done; otherwise, since w_1 is only contained in triangles that also contains v or u, there is a triangle T with w_1, w_2, y in P with $y \in \{u, v\}$. Then for the both cases of this rule, there remains a vertex x or one of two vertices $a, b \in N(w_1)$ unused by P. Thus, by replacing T by the triangle formed by y, w_1, x or by y, w_1, a (or b, depending on y) in P, we can get a triangle packing that contains at least k triangles from $G - \{(w_1, w_2)\}$.

Rule 6. If (w_1, u) is not contained in P, then we are done; otherwise, let T be the triangle containing (w_1, u). Clearly, vertices a and w_2 cannot be in any other triangle in P than T. Then, replacing T by the triangle formed by a, w_2, and u, we can get a triangle packing that contains at least k triangles from $G - \{(w_1, u)\}$. □

Theorem 9. Maximum Triangle Packing *on planar graphs admits a kernel with less than* $75k$ *vertices.*

Proof. To derive the kernel size, we consider only the case that the algorithm in Fig. 1 returns a reduced instance (G, k) and a triangle packing P for this instance with less than k triangles. Let $S := V(P)$. Clearly, $|S| \leq 3k - 3$. We use again the partition of the vertex set of G with respect to S. Clearly $J_0 = \emptyset$ and $|J_3| \leq 2|S| - 4$. We derive some observations of J_1 and J_2.

Observation 1. J_1 induces an independent set.

This observation is true due to the cleaning rule and Rule 1 and implies that each vertex in J_1 has some neighbor in $J_2 \cup J_3$. Moreover, a vertex $v \in J_1$ with $N(v) \cap S = \{u\}$ must be adjacent to a vertex $w \in J_2 \cup J_3$ with $u \in N(w) \cap S$. We define $J_2(uv)$ as the set of vertices w in J_2 with $N(w) \cap S = \{u, v\}$ and $J_1(v) := \{u \in J_1 \mid N(u) \cap S = \{v\}\}$.

Observation 2. Each vertex $u \in J_2 \cup J_3$ is adjacent to at most one vertex in $J_1(v)$ for every $v \in S$.

We prove this observation by contradiction. Suppose that there are $a, b \in J_1(v)$ adjacent to $u \in J_2 \cup J_3$. Clearly, u is adjacent to v. However, Rule 4 can then be applied for vertices a, b, u, v.

Input: A planar graph G and a positive integer k.
Output: A triangle packing containing at least k triangles, or an instance (G', k') and a maximal triangle packing of size at most $k'-1$ for G'.
1 reduce (G, k) with respect to the rules;
2 find a maximal triangle packing P of G;
3 if P contains at least k triangles, return P;
4 let $G' := G, k' := k$ and return G', k', and P;

Fig. 1. The kernelization algorithm for MTP

From this observation, we know that each vertex $v \in J_3$ can have at most $|N(v) \cap S|$ many neighbors in J_1. By $|J_3| \leq 2|S| - 4$, the vertices in J_3 altogether can have at most $2(|S| + |J_3|) - 4 \leq 6|S| - 12$ neighbors in J_1.

Observation 3. If $|J_2(uv)| \geq 2$, there is no degree-2 vertex in $J_2(uv)$ for every $u, v \in S$.

This observation is true due to Rule 2. In the following, we will assume that $|J_2(uv)| \geq 2$ for all $u, v \in S$ and thus, no degree-2 vertices in J_2.

Observation 4. For every pair of vertices $u, v \in S$ with $J_2(uv) \neq \emptyset$, all connected components of the subgraph of G induced by $J_2(uv)$, except for one which can have at most 3 vertices and 2 edges, are singleton components, that is, components with a single vertex.

If there are two connected components with at least two vertices, then Rule 3 would apply. Consider that the non-singleton component C. If $|C| > 3$, then either there exists a path of length three or $G[J_2(uv)]$ is a star. In former case, we have a matching of size 2 in $G[J_2(uv)]$ and Rule 3 would be applied. If $G[J_2(uv)]$ is a star, then there is a vertex adjacent to all other vertices. Since all vertices in $J_2(uv)$ are adjacent to both u and v, we have then a $K_{3,3}$, contradicting the planarity of G. Clearly, there cannot be a triangle in $G[J_2(uv)]$, since P is maximal. There are at most two edges in C.

Observation 5. If there are at least three vertices from $J_1(u) \cup J_1(v)$ adjacent to the vertices in $J_2(uv)$, then they are all from $J_1(u)$ or all from $J_1(v)$.

Due to Observation 2, if there are three vertices from $J_1(u) \cup J_1(v)$ adjacent to the vertices in $J_2(uv)$, then they cannot be adjacent to the same vertex in $J_2(uv)$. If the observation is not true, there are at least two vertices in $J_2(uv)$ adjacent to $J_1(u) \cup J_1(v)$, one having a neighbor in $J_1(u)$ and the other having a neighbor in $J_1(v)$. Then Rule 3 can be applied.

Next we will consider first the case that each $J_2(uv)$ for vertices $u, v \in S$ induces an independent set. Later, we show that in the case that there are edges in $G[J_2(uv)]$, we can achieve an even smaller kernel size bound.

Observation 6. There is at most one vertex x in $J_2(uv)$ such that $N(N(x) \setminus \{u, v\}) \cap S \subseteq \{u, v\}$.

If there is a vertex $x \in J_2(uv)$ with $N(N(x) \setminus \{u, v\}) \cap S \subseteq \{u, v\}$, then all its neighbors are from $J_2(uv) \cup J_1(v) \cup J_1(u)$. By the assumption that $J_2(uv)$

induces an independent set, x must be adjacent to $J_1(v) \cup J_1(u)$. By Rule 1, we know that x is adjacent to both $J_1(v)$ and $J_1(u)$. If there were two such vertices in $J_2(uv)$, then Rule 3 would be applied.

Observation 7. If there is a vertex $x \in J_2(uv)$ with $N(N(x) \setminus \{u,v\}) \cap S \subseteq \{u,v\}$, then no vertex $y \in J_2(uv) \setminus \{x\}$ can have neighbor from $J_1(u) \cup J_1(v)$.

This follows from the proof of Observation 6 and Rule 3.

By Observation 7 and the assumption that $|J_2(uv)| \geq 2$, we can assume that there is no vertex in $J_2(uv)$ with $N(N(x) \setminus \{u,v\}) \cap S \subseteq \{u,v\}$ and every vertex in $J_2(uv)$ has exactly one neighbor from $J_1(u) \cup J_1(v)$. This assumption is sound due to the following reason: Although, in the case that there is a vertex $x \in J_2(uv)$ with $N(N(x) \setminus \{u,v\}) \cap S \subseteq \{u,v\}$, the vertex x can have two neighbors from $J_1(u) \cup J_1(v)$, one from $J_1(u)$ and one from $J_1(v)$, no other vertex $y \in J_2(uv)$ with $y \neq x$ can have neighbor from $J_1(u) \cup J_1(v)$, due to Observation 7. Moreover, by the assumption $|J_2(uv)| \geq 2$, there is at least one other vertex $y \in J_2(uv)$ and, thus, this case will not give a greater upper bound on $|J_2 \cup J_1|$. Now, we use this assumption that every vertex in $J_2(uv)$ for every pair of vertices $u, v \in S$ has a neighbor which is adjacent to a vertex $w \in S$ with $w \notin \{u,v\}$ to analyze the size of J_2.

We consider an arbitrary but fixed planar embedding of G. If for vertices $u, v \in S$, there are at least two vertices in $J_2(uv)$, then we can find a closed area $R(uv)$ of the embedding containing all vertices in $J_2(uv)$, enclosed by two paths between u and v, each path having length 2. Let x and y be the middle vertices of these paths. If $|J_2(uv)| > 2$, then there must be some vertex in S contained in $R(uv)$. The reason for this is that each vertex from $J_2(uv)$ has a neighbor b with $(N(b) \setminus \{u,v\}) \cap S \neq \emptyset$.

In the following, we call a vertex $w \in S \setminus \{u,v\}$ "supports" a vertex $a \in J_2(uv)$, if $(N(w) \cap N(a)) \setminus \{u,v\} \neq \emptyset$. Every vertex $w \in S \setminus \{u,v\}$ can support at most 2 vertices in $J_2(uv)$, since, otherwise, we would have $K_{3,3}$ as a minor of G, contradicting the planarity of G. Let $S(uv)$ denote the set of vertices from $S \setminus \{u,v\}$ that lie inside of $R(uv)$ and support some vertex $b \in J_2(uv) \setminus \{x,y\}$. We can get $|J_2(uv)| \leq 2 + 2|S(uv)|$.

Next we show that no vertex $w \in S(uv)$ can be in $S(u'v')$ for $\{u,v\} \neq \{u',v'\}$. Suppose that this is not true. Since w lies inside of $R(uv)$, the closed area $R(u'v')$ between u' and v' must be completely contained in $R(uv)$. Let a be a vertex in $J_2(uv)$ supported by w and b be the common neighbor of w and a. By $a \in J_2(uv)$, a is not in $R(u'v')$ and then, b must be on one of the paths enclosing $R(u'v')$. By $(w,b) \in E$ and $b \in J_2(u'v')$, we have $w \in \{u',v'\}$ and $w \notin S(u'v')$.

Now we can conclude that $|J_2| \leq \sum_{u,v \in S, J_2(uv) \neq \emptyset} (2 + 2|S(uv)|)$. Since $S(uv) \cap S(u'v') = \emptyset$ for $\{u,v\} \neq \{u',v'\}$ and by Lemma 2, there are at most $3|S| - 6$ non-empty $J_2(uv)$'s, we have $|J_2| \leq 8|S| - 12$. By Observations 2 and 6, there are at most $|J_2|$ many vertices from J_1 being adjacent to J_2. Altogether, $|J_1| \leq |J_2| + 6|S| - 12 \leq 14|S| - 24$. The kernel has then at most $|J_1| + |J_2| + |J_3| + |S| \leq 25|S| - 40 \leq 75k - 115$.

Next, we consider the case that there are some edges in $G[J_2(uv)]$ for some $u, v \in S$. By Observation 4, there is exactly one component in $G[J_2(uv)]$ having some edges. The following observation follows from Rule 3.

Observation 8. If there is a connected component C in $G[J_2(uv)]$ for $u, v \in S$ with $|C| \geq 2$, then no vertex in J_1 can be adjacent to the vertices in $J_2(uv) \setminus V(C)$.

First we consider the case that this only non-singleton component C contains three vertices and two edges. Let w_1, w_2, w_3 be these vertices with $w_1, w_3 \in N(w_2)$. Due to Rule 5, both w_1 and w_3 must have some neighbors other than w_2, u, v. They cannot be adjacent to $J_2(uv) \setminus V(C)$. They cannot be adjacent to vertices in $J_1(u) \cup J_1(v)$ either, since, otherwise, Rule 3 would be applied. So, each of them must have some neighbor in J_3 or $J_2(u'v')$ for $\{u', v'\} \neq \{u, v\}$, that is, it has a neighbor which is adjacent to a vertex $w \in S$ with $w \notin \{u, v\}$. Then, these two non-adjacent vertices can be considered together with the independent set vertices from other components in $G[J_2(u, v)]$, as discussed above. Moreover, these two vertices and all vertices in $J_2(uv) \setminus V(C)$, due to Observation 8, cannot be adjacent to vertices in J_1. Finally, by Rules 4 and 5, we know that, if w_2 has no neighbor which is adjacent to a vertex $w \in S$ with $w \notin \{u, v\}$, then w_2 can have at most one neighbor from J_1. Thus, with this additional component C, we will not get a greater upper bound on the kernel size: We have vertex w_2 and its only neighbor from J_1 in addition, but at least two vertices—w_1, w_3, and all vertices in $J_2(uv) \setminus V(C)$—lose their neighbors in J_1.

Now consider the case that C consists of only two vertices w_1 and w_2 and one edge. If w_1 and w_2 both have some neighbor which is adjacent to a vertex $w \in S$ with $w \notin \{u, v\}$, then we can consider them together with the independent set vertices together, as discussed above. Note that Observation 7 holds also for these two vertices due to Rules 4 and 5. If only one of them has no neighbor which is adjacent to a vertex $w \in S$ with $w \notin \{u, v\}$, say, w_1, then w_1 can only have neighbors in $J_1(u) \cup J_1(v)$. However, by Rule 5, w_1 cannot have neighbors from both $J_1(u)$ and $J_1(v)$. If w_1 has some neighbor from $J_1(u)$, then Rule 6 would be applied. Thus, although we have in this case one vertex from C which cannot be considered together with the independent set vertices, we lose the neighbor of w_2 from J_1. Finally, if both w_1 and w_2 have no neighbor which is adjacent to a vertex $w \in S$ with $w \notin \{u, v\}$, then w_1 and w_2 together with $J_2(uv) \setminus C$ cannot have neighbors from $J_1(v) \cup J_1(u)$; otherwise, Rule 4 or Rule 5 would be applied. Clearly, this case does not give a greater upper bound on $|J_2(uv)|$. □

6 Conclusion

In this paper, we introduced a simple vertex partition method to analyze the kernel size for planar graph problems. This method, compared to the region decomposition [1], is much simple and leads to better kernel sizes for the same set of reduction rules, as shown with (Connected) Vertex Cover, Edge Dominating Set, and Maximum Triangle Packing. Furthermore, we introduced new reduction rules for Connected Vertex Cover, Edge Dominating Set, and Maximum Triangle

Packing, further improving the kernel sizes for these problems. We also mention here that the analysis for Connected Vertex Cover and Edge Dominating Set can be lifted up to graphs with bounded genus.

References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating set. J. ACM 51(3), 363–384 (2004)
2. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
3. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) Kernelization. In: FOCS 2009, pp. 629–638. IEEE Computer Society, Los Alamitos (2009)
4. 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)
5. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
6. Gu, Q., Imani, N.: Connectivity is not a limit for kernelization: Planar connected dominating set. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 26–37. Springer, Heidelberg (2010)
7. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
8. Guo, J., Niedermeier, R.: Linear problem kernels for NP-hard problems on planar graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 375–386. Springer, Heidelberg (2007)
9. Guo, J., Niedermeier, R., Wernicke, S.: Fixed-parameter tractability results for full-degree spanning tree and its dual. Networks 56(2), 116–130 (2010)
10. Lokshtanov, D., Mnich, M., Saurabh, S.: Linear kernel for planar connected dominating set. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 281–290. Springer, Heidelberg (2009)
11. Luo, W., Wang, J., Feng, Q., Guo, J., Chen, J.: An improved kernel for planar connected dominating set. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 70–81. Springer, Heidelberg (2011)
12. 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)
13. Moser, H., Sikdar, S.: The parameterized complexity of the induced matching problem. Discrete Applied Mathematics 157(4), 715–727 (2009)
14. Prieto, E.: Systematic Kernelization in FPT Algorithm Design. Ph.D. thesis, Department of Computer Science, University of Newcastle, Australia (2005)
15. Privadarsini, P., Hemalatha, T.: Connected vertex cover in 2-connected planar graph with maximum degree 4 is NP-complete. International Journal of Mathematical, Physical and Engineering Sciences 2(1), 51–54 (2008)

New Parameterized Algorithms for the Edge Dominating Set Problem

Mingyu Xiao[1,⋆], Ton Kloks[2,⋆⋆], and Sheung-Hung Poon[2,⋆⋆⋆]

[1] School of Computer Science and Engineering
University of Electronic Science and Technology of China, China
myxiao@gmail.com
[2] Department of Computer Science
National Tsing Hua University, Taiwan
{kloks,spoon}@cs.nthu.edu.tw

Abstract. An edge dominating set of a graph $G = (V, E)$ is a subset $M \subseteq E$ of edges in the graph such that each edge in $E - M$ is incident with at least one edge in M. In an instance of the parameterized edge dominating set problem we are given a graph $G = (V, E)$ and an integer k and we are asked to decide whether G has an edge dominating set of size at most k. In this paper we show that the parameterized edge dominating set problem can be solved in $O^*(2.3147^k)$ time and polynomial space. We also show that this problem can be reduced to a quadratic kernel with $O(k^3)$ edges.

1 Introduction

The *edge dominating set problem* (EDS), to find an edge dominating set of minimum size in a graph, is one of the basic problems highlighted by Garey and Johnson in their work on NP-completeness [6]. It is known that the problem is NP-hard even when the graph is restricted to planar or bipartite graphs of maximal degree three [16]. The problem in general graphs and in sparse graphs has been extensively studied in approximation algorithms [16,5,2]. Note that a maximum matching is a 2-approximation for EDS. The 2-approximation algorithm for the weighted version of EDS is considerably more complicated [5].

Recently, EDS also draws much attention from the exact and parameterized algorithms community. Randerath and Schiermeyer [10] designed an $O^*(1.4423^m)$ algorithm for EDS which was improved to $O^*(1.4423^n)$ by Raman *et al.* [11]. Here n and m are the number of vertices and edges in the graph and the O^*-notation suppresses polynomial factors. Fomin *et al.*, [4] further improved this result to $O^*(1.4082^n)$ by considering the treewidth of the graph. Rooij and Bodlaender [12] designed an $O^*(1.3226^n)$ algorithm by using the 'measure and conquer method.'

⋆ Supported in part by Grant 60903007 of NSFC, China.
⋆⋆ Supported in part by Grant 99-2218-E-007-016 of NSC, Taiwan.
⋆⋆⋆ Supported in part by Grant 97-2221-E-007-054-MY3 of NSC, Taiwan.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 604–615, 2011.

For *parameterized edge dominating set* (PEDS) with the parameter k being the size of the edge dominating set, Fernau [3] gives an $O^*(2.6181^k)$ algorithm. Fomin *et al.* [4] obtain an $O^*(2.4181^k)$-time and exponential-space algorithm based on dynamic programming on bounded treewidth graphs. Binkele-Raible and Fernau [1] further improve the running time to $O^*(2.3819^k)$.

Faster algorithms are known for graphs that have maximal degree three. The EDS and PEDS problems in degree-3 graphs can be solved in $O^*(1.2721^n)$ [14] and $O^*(2.1479^k)$ [15].

In this paper, we present two new algorithms for PEDS. The first one is a simple and elegant algorithm that runs in $O^*(2.3715^k)$ time and polynomial space. We improve the running-time bound to $O^*(2.3147^k)$ by using a technique that deals with remaining graphs of maximal degree three. We also design a linear-time algorithm that obtains a quadratic kernel which is smaller than previously-known kernels.

Our algorithms for PEDS are based on the technique of enumerating minimal vertex covers. We introduce the idea of the algorithms in Section 2 and introduce some basic techniques in Section 3. We present a simple algorithm for PEDS in Section 4 and an improved algorithm in Section 5. In Section 6 we discuss the problem kernel.

2 Enumeration-Based Algorithms

As in many previous algorithms for the edge dominating set problem [3,4,12,14] our algorithms are based on the enumeration of minimal vertex covers. Note that the vertex set of an edge dominating set is a vertex cover. Conversely, let C be a minimal vertex cover and M be a minimum edge dominating set containing C in the set of its endpoints. Given C, M can be computed in polynomial time by computing a maximum matching in induced graph $G[C]$ and adding an edge for each unmatched vertex in C. This observation reduces the problem to that of finding the right minimal vertex cover C. Now, the idea is to enumerate all minimal vertex covers. Moon and Moser showed that the number of minimal vertex covers is bounded by $3^{n/3}$ and this shows that one can solve EDS in $O(1.4423^n)$ time [7,8].

For PEDS, we want to find an edge dominating set of size bounded by k. It follows that we need to enumerate minimal vertex covers of size only up to $2k$. We use a branch-and-reduce method to find vertex covers. We fix some part of a minimal vertex cover and then we try to extend it with at most p vertices. Initially $p = 2k$.

For a subset $C \subseteq V$ and an independent set $I \subseteq V - C$ in G, an edge dominating set M is called a (C, I)-*eds* if

$$C \subseteq V(M) \quad \text{and} \quad I \cap V(M) = \varnothing.$$

In the search for the vertex cover $V(M)$ of a minimum (C, I)-eds M, we keep track of a partition of the vertices of G in four sets: C, I, U_1 and U_2. Initially $C = I = U_1 = \varnothing$ and $U_2 = V$. The following conditions are kept invariant.

1. I is an independent set in G, and
2. each component of $G[U_1]$ is a clique component of $G[V \setminus (C \cup I)]$.

The vertices in $U_1 \cup U_2$ are called *undecided* vertices. We use a five-tuple

$$(G, C, I, U_1, U_2)$$

to denote the state described above. We let $q_i = |Q_i|$ denote the number of vertices of a clique component Q_i of $G[U_1]$. Rooij and Bodlaender proved the following lemma in [12].

Lemma 1. *If $U_2 = \varnothing$ then a minimum (C, I)-eds M of G can be found in polynomial time.*

When there are no undecided vertices in the graph we can easily find a minimum (C, I)-eds. Lemma 1 tells us that clique components in the *undecided graph* $G[V \setminus (C \cup I)]$ do not cause trouble. We use some branching rules to deal with vertices in U_2.

Consider the following simple branching rule. For any vertex $v \in U_2$ consider two branches that either include v into the vertex cover or exclude v from the vertex cover. In the first branch we move v into C. In the second branch we move v into I and move the set $N(v)$ of neighbors of v into C.

When we include a number of vertices into the vertex cover, we reduce the parameter p by the same value. Furthermore, in each branch we move any newly-found clique component Q in $G[U_2]$ into U_1 and reduce p by $|V(Q)| - 1$. The reason is that each clique has at most one vertex that is not in the vertex cover.

Let $C(p)$ denote the worst-case running time to enumerate vertex covers up to size p. Then we have the following inequality:

$$C(p) \leq C(p - 1 - q_v) + C(p - |N(v)| - q_{N(v)}), \tag{1}$$

where q_v (resp., $q_{N(v)}$) denotes the sum of $|V(Q)| - 1$ over all cliques Q in $G[U_2]$ that appear after removing v (resp., $N(v)$) from U_2.

At worst, both q_v and $q_{N(v)}$ are 0. Then we end up with the recurrence

$$C(p) \leq C(p - 1) + C(p - |N(v)|).$$

Note that one can always branch on vertices of degree at least 2 in $G[U_2]$. In this manner Fernau [3] solves the edge dominating set problem in $O^*(1.6181^p) = O^*(2.6181^k)$ time which stems from the solution of the Fibonacci recurrence

$$C(p) \leq C(p - 1) + C(p - 2).$$

Fomin *et.al.*, [4] refine this as follows. Their algorithm first branches on vertices in $G[U_2]$ of degree at least 3 and then it considers the treewidth of the graph when all the vertices in $G[U_2]$ have degree one or two. If the treewidth is small the algorithm solves the problem by dynamic programming and if the treewidth

is large the algorithm branches further on vertices of degree two in $G[U_2]$. This algorithm uses exponential space and its running time depends on the running time of the dynamic programming algorithms.

The method of iteratively branching on vertices of maximum degree d is powerful when $d > 2$. Unfortunately, it seems that we cannot avoid some branchings on vertices of degree 2, especially when each component of $G[U_2]$ is a 2-path, *i.e.*, a path that consists of two edges. We say that we are in the *worst case* when every component of $G[U_2]$ is a 2-path.

Our algorithms branch on vertices of maximum degree and on some other local structures in $G[U_2]$ until $G[U_2]$ has only 2-path components. When we are in the worst case our algorithms deal with the graph in the following way. Let $P = v_0v_1v_2$ be a 2-path in $G[U_2]$. We say P is *signed* if $v_1 \in V(M)$, and *unsigned* if $v_1 \notin V(M)$. We use an efficient way to enumerate all signed 2-paths in $G[U_2]$.

In the next section we introduce our branching rules.

3 Branching Rules

Besides the simple technique of branching on a vertex, we also use the following branching rules. Recall that in our algorithm, once a clique component Q appears in $G[U_2]$, we move $V(Q)$ into U_1 and reduce p by $|V(Q)| - 1$.

Tails. Let the vertex v_1 have degree two. Assume that v_1 has one neighbor v_0 of degree one and that the other neighbor v_2 has degree greater than one. Then we call the path $v_0v_1v_2$ a *tail*.

In this paper, when we use the notation $v_0v_1v_2$ for a tail, we implicitly mean that the first vertex v_0 is the degree-1 vertex of the tail. *Branching on a tail* $v_0v_1v_2$ means that we branch by either including v_2 into the vertex cover or excluding v_2 from the vertex cover.

Lemma 2. *If $G[U_2]$ has a tail then we can branch with the recurrence*

$$C(p) \leq 2C(p-2) \quad \Rightarrow \quad C(p) = O(1.4143^p). \tag{2}$$

Proof. Let the tail be $v_0v_1v_2$. In the branch where v_2 is included into C, $\{v_0, v_1\}$ becomes a clique component and is moved into U_1. Then p reduces by 1 from v_2 and by 1 from $\{v_0, v_1\}$. In the branch where v_2 is included into I, $N(v_2)$ is included into C. Since $|N(v_2)| \geq 2$, p also reduces by 2 in this branch. □

4-Cycles. We say that $abcd$ is a 4-*cycle* if there exist the four edges ab, bc, cd and da in the graph. Xiao [13] used the following lemma to obtain a branching rule for the maximum independent set problem. In this paper we use it for the edge dominating set problem.

Lemma 3. *Let $abcd$ be a 4-cycle in graph G, then any vertex cover in G contains either a and c or b and d.*

As our algorithm aims at finding a vertex cover, it branches on a 4-cycle $abcd$ in $G[U_2]$ by including a and c into C or including b and d into C. Notice that we obtain the same recurrence as in Lemma 2.

4 A Simple Algorithm

Our first algorithm is described in Fig. 1. The search tree consists of two parts. First, we branch on vertices of maximum degree, tails and 4-cycles in Lines 3-4 until every component in $G[U_2]$ is a 2-path. Second, we enumerate the unsigned 2-paths in $G[U_2]$. In each leaf of the search tree we find an edge dominating set in polynomial time by Lemma 1. We return a smallest one.

Algorithm $EDS(G, C, I, U_1, U_2, p)$

Input: A graph $G = (V, E)$, and a partition of V into sets C, I, U_1 and U_2. Initially $C = I = U_1 = \varnothing$, $U_2 = V$. Integer p; initially $p = 2k$.
Output: An edge dominating set of size $\leq k$ in G if it exists.

1. **While** there is a clique component Q in $G[U_2]$ **do** move it into U_1 and reduce p by $|Q| - 1$.
2. **If** $p < 0$ **then halt**.
3. **While** there is a tail or 4-cycle in $G[U_2]$ **do** branch on it.
4. **If** there is component Q of $G[U_2]$ that is not a 2-path **then** pick a vertex v of maximum degree in Q and branch on it.
5. **Else** let P be the set of 2-paths in $G[U_2]$ and $y = |P|$.
6. **If** $y > \min(p, k)$ **then halt**.
7. Let $z = \min(p - y, k - y)$.
8. **For** each subset $P' \subseteq P$ of size $0 \leq |P'| \leq z$ **do**
   For each $v_0v_1v_2 \in P'$ **do** move $\{v_0, v_2\}$ into C and move v_1 into U_1;
   For each $v_0v_1v_2 \in P - P'$ **do** move v_1 into C and move $\{v_0, v_2\}$ into U_1.
9. Compute the candidate edge dominating set M and return the smallest one. (Here $U_2 = \varnothing$, $C \cup I \cup U_1 = V$)

Fig. 1. Algorithm $EDS(G, C, I, U_1, U_2, p)$

4.1 Analysis

In order to show the correctness of the algorithm we need to explain Line 6 and Line 8.

For each 2-path in $G[U_2]$ we need at least one edge to dominate it. So, we must have that $y \leq p$ and $y \leq k$. This explains the condition in Line 6.

It is also easy to see that for each unsigned 2-path we need at least two different edges to dominate it. Let p' be the number of unsigned 2-paths. In Line 8, we enumerate the possible sets $P' \subseteq P$ of unsigned 2-paths. Notice that

$$(y + p' \leq k \text{ and } y + p' \leq p) \quad \Leftrightarrow \quad p' \leq z.$$

Now we analyze the running time of this algorithm. Lemma 1 guarantees that the subroutine in Line 9 runs in polynomial time. We focus on the exponential part of the running time. We prove a bound of the size of the search tree in our algorithm with respect to measure p.

First, we consider the running time of Lines 3-4.

Lemma 4. *If the graph has a vertex of degree* ≥ 3 *then Algorithm EDS branches with*

$$C(p) \leq C(p-1) + C(p-3) \quad \Rightarrow \quad C(p) = O(1.4656^p). \tag{3}$$

Proof. If the algorithm branches on a tail or a 4-cycle we have the upper bound given by (2). Otherwise the algorithm branches on a vertex of maximum degree and generates a recurrence covered by (3). Notice that (3) covers (2). This proves the lemma. □

Lemma 5. *If all components of the graph are paths and cycles then the branchings of Algorithm EDS before Line 5 satisfy (3).*

Proof. If there is a path component of length greater than 2, then there is a tail and the algorithm branches on it with (2).

If there is a component C_l which is an l-cycle in $G[U_2]$, the algorithm deals with it in this way: If the cycle is a 3-cycle, the algorithm moves it into U_1 without branching since it is a clique.

If the cycle is a 4-cycle then, according to Lemma 3, our algorithm branches on it with (2).

If the cycle has length at least 5, our algorithm selects an arbitrary vertex v_0 and branches on it. Subsequently it branches on the path that is created as long as the length of the path is greater than 2. When the cycle is a 5-cycle we obtain the recurrence

$$C(p) \leq 3C(p-3) \quad \Rightarrow \quad C(p) = O(1.4423^p).$$

When the cycle is a 6-cycle we obtain the recurrence

$$C(p) \leq C(p-2) + C(p-3) + C(p-4) \quad \Rightarrow \quad C(p) = O(1.4656^p).$$

The two recurrences above are covered by (3). Straightforward calculations show that when the cycle has length ≥ 7, we also get a recurrence covered by (3). For brevity we omit the details of this analysis. □

By Lemma 4 and Lemma 5 we know that the running time of the algorithm, before it enters Line 5 is $O^*(1.4656^x)$, where x is the size of C upon entering the loop in Line 8. We now consider the time that is taken by the loop in Line 8 and then analyze the overall running time.

First we derive a useful inequality.

Lemma 6. *Let* r *be a positive integer. Then for any integer* $0 \leq i \leq \lfloor \frac{r}{2} \rfloor$

$$\binom{r-i}{i} = O(1.6181^r). \tag{4}$$

Proof. Notice that

$$\binom{r-i}{i} \leq \sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} \binom{r-i}{i} = F(r+1),$$

where $F(r)$ is the r^{th} Fibonacci number. We have $F(r) = O(1.6181^r)$. □

Now we are ready to analyze the running time of the algorithm. It is clear that the loop in Line 8 takes less than $y\binom{y}{z}$ basic computations. First assume that $x \leq k$. We have that $z \leq k - y$ thus $y + z \leq k$. If we apply Lemma 6 with $r = y + z$ we find that the running time of the loop in Line 8 is $O^*(1.6181^k)$. By Lemmas 4 and 5 the running time of the algorithm is therefore bounded by $O^*(1.4656^x \cdot 1.6181^k) = O^*(2.3715^k)$.

Assume that $x > k$. We now use that $z \leq p - y$ thus $y + z \leq p$. By Lemma 6 the running time of Step 8 is $O^*(1.6181^p)$. Now $p \leq 2k - x$ and $x > k$. The running time of the algorithm is therefore bounded by

$$O^*(1.4656^x \cdot 1.6181^p) = O^*(2.3715^k).$$

We summarize the result in the following theorem.

Theorem 1. *Algorithm EDS solves the parameterized edge dominating set problem in $O^*(2.3715^k)$ time and polynomial space.*

5 An Improvement

In this section we present an improvement on Algorithm *EDS*. The improved algorithm is described in Fig. 2.

The search tree of this algorithm consists of three parts. First, we iteratively branch on vertices of degree ≥ 4 until $G[U_2]$ has no such vertices anymore (Line 3). Then we partition the vertices in U_2 into two parts: $V(P)$ and U_2', where P is the set of 2-path components in $G[U_2]$ and $U_2' = U_2 \setminus V(P)$. Then the algorithm branches on vertices in U_2' until U_2' becomes empty (Lines 4-5). Finally, we enumerate the number of unsigned 2-paths in P (Line 9) and continue as in Algorithm *EDS*.

In Algorithm *EDS*1 a subroutine *Branch*3 deals with some components of maximum degree 3. It is called in Line 5. This is the major difference with Algorithm *EDS*. Algorithm *Branch*3 is described in Fig. 3. The algorithm contains several simple branching cases. They could be described in a shorter way but we avoided doing that for analytic purposes.

We show the correctness of the condition in Line 7 of Algorithm *EDS*1. The variable p_0 in Algorithm *PEDS*1 marks the decrease of p by subroutine *Branch*3. Note that no vertices in $V(P)$ are adjacent to vertices in U_2'. Let M_1 be the set of edges in the solution with at least one endpoint in U_2' and let M_2 be the set of edges in the solution with at least one endpoint in $V(P)$. Then

$$M_1 \cap M_2 = \emptyset \quad \text{and} \quad |M_1| + |M_2| \leq k \quad \text{and} \quad |M_1| \geq \frac{p_0}{2}.$$

Thus $|M_2| \leq k - \frac{p_0}{2}$. The correctness of Algorithm *EDS*1 now follows since the only difference is the subroutine *Branch*3.

Algorithm $EDS1(G, C, I, U_1, U_2, p)$

Input: A graph $G = (V, E)$ and a partition of V into sets C, I, U_1 and U_2. Initially $C = I = U_1 = \varnothing$, $U_2 = V$. Integer p; initially $p = 2k$.
Output: An edge dominating set of size $\leq k$ in G if it exists.

1. **While** there is a clique component Q in $G[U_2]$ **do** move it into U_1 and decrease p by $|Q| - 1$.
2. **If** $p < 0$ **halt**.
3. **While** there is a vertex v of degree ≥ 4 in $G[U_2]$ **do** branch on it.
4. Let P denote the set of 2-path components in $G[U_2]$ and $U_2' = U_2 \setminus V(P)$. Let $y = |P|$ and $p' = p$.
5. **While** $U_2' \neq \varnothing$ and $p \geq 0$ **do** $(G, C, I, U_1, U_2, p) = Branch3(G, C, I, U_1, U_2 = U_2' \cup V(P), p)$.
6. Let $p_0 = p' - p$.
7. **If** $y > \min(p, k - p_0/2)$ **halt**.
8. Let $z = \min(p - y, k - p_0/2 - y)$.
9. **For** each subset $P' \subseteq P$ of size $0 \leq |P'| \leq z$ **do**
   For each $v_0v_1v_2 \in P'$ **do** move $\{v_0, v_2\}$ into C and move v_1 into U_1;
   For each $v_0v_1v_2 \in P - P'$ **do** move v_1 into C and move $\{v_0, v_2\}$ into U_1.
10. Compute the candidate edge dominating set M and return the smallest one. (Here $U_2 = \varnothing$, $C \cup I \cup U_1 = V$.)

Fig. 2. Algorithm $EDS1(G, C, I, U_1, U_2, p)$

5.1 Analysis of Algorithm *EDS1*

The proof of the following lemma is omitted here due to lack of space. You can access this proof in our current draft located at the ArXiv.

Lemma 7. *The branchings of Algorithm Branch3 satisfy the recurrence*

$$C(p) \leq C(p-2) + 2C(p-3) \quad \Rightarrow \quad C(p) = O(1.5214^p). \tag{5}$$

Algorithm $EDS1$ first branches on vertices of degree at least 4. These branchings of the algorithm satisfy

$$C(p) \leq C(p-1) + C(p-4) \quad \Rightarrow \quad C(p) = O(1.3803^p). \tag{6}$$

Recall that the subroutine $Branch3$ reduces p by p_0. The analysis without the subroutine is similar to the analysis of Algorithm EDS in Section 4.1 except that k is replaced by $k - \frac{p_0}{2}$ and that Formula (3) is replaced by Formula (6). Thus without the subroutine $Branch3$ the algorithm has running time proportional to

$$(1.3803 \cdot 1.6181)^{k-\frac{p_0}{2}} = 2.2335^{k-\frac{p_0}{2}}.$$

By Lemma 7 the running time of the algorithm is therefore bounded by

$$O^*(2.2335^{k-p_0/2} \cdot 1.5214^{p_0}) = O^*(2.2335^{k-p_0/2} \cdot 2.3147^{p_0/2}) = O^*(2.3147^k).$$

This proves the following theorem.

Algorithm $Branch3(G, C, I, U_1, U_2 = U_2' \cup V(P), p)$

1. **If** there is a clique component in $G[U_2']$ **then** move it to U_1.
2. **If** there is a 2-path component $v_0v_1v_2$ in $G[U_2']$ **then** branch on v_1.
3. **If** $U_2' \neq \varnothing$ **then**
   - 3.1 **If** there is a degree-3 vertex v adjacent to two degree-1 vertices in $G[U_2']$ **then** branch on v.
   - 3.2 **If** there is a tail $v_0v_1v_2$ such that v_2 is a degree-2 vertex in $G[U_2']$ **then** branch on the tail.
   - 3.3 **If** there is a tail $v_0v_1v_2$ such that v_2 is a degree-3 vertex in $G[U_2']$ **then** branch on the tail.
   - 3.4 **If** there is a degree-3 vertex v adjacent to one degree-1 vertex in $G[U_2']$ **then** branch on v.
   - 3.5 **If** there is a 4-cycle in $G[U_2']$ **then** branch on it.
   - 3.6 **If** there is a degree-3 vertex v adjacent to any degree-2 vertex in $G[U_2']$ **then** branch on v.
   - 3.7 Pick a maximum vertex v in $G[U_2']$ and branch on it.

   In addition to 3.1–3.7:
* **If** some 2-path component $v_0v_1v_2$ is created in 3.1 – 3.7 **then** branch on v_1.

Fig. 3. Algorithm $Branch3(G, C, I, U_1, U_2, p)$

Theorem 2. *Algorithm EDS*1 *solves the parameterized edge dominating set problem in* $O^*(2.3147^k)$ *time and polynomial space.*

6 Kernelization

A *kernelization algorithm* takes an instance of a parameterized problem and transforms it into an equivalent parameterized instance (called the *kernel*), such that the new parameter is at most the old parameter and the size of the new instance is a function of the new parameter.

For the parameterized edge dominating set problem Prieto [9] presented a quadratic-time algorithm that finds a kernel with at most $4k^2 + 8k$ vertices by adapting 'crown reduction techniques.' Fernau [3] obtained a kernel with at most $8k^2$ vertices.

We present a new linear-time kernelization that reduces a parameterized edge dominating set instance (G, k) to another instance (G', k') such that

$$|V(G')| \leq 2k'^2 + 2k' \quad \text{and} \quad |E(G')| = O(k'^3) \quad \text{and} \quad k' \leq k.$$

In our kernelization algorithm we first find an arbitrary maximal matching M_0 in the graph in linear time. Let $m = |M_0|$, then we may assume that $m \geq k + 1$ otherwise M_0 solves the problem directly. Let

$$V_m = V(M_0) \quad \text{and} \quad V^* = V - V_m.$$

Since M_0 is a maximal matching, we know that V^* is an independent set. For a vertex $v_i \in V_m$, let $x_i = |V^* \cap N(v_i)|$. We call vertex $v_i \in V_m$ *overloaded*, if $m + x_i > 2k$. Let $A \subseteq V_m$ be the set of overloaded vertices.

Lemma 8. *Let M be an edge dominating set M of size at most k. Then*

$$A \subseteq V(M).$$

Proof. If an overloaded vertex $v_i \notin V(M)$ then all neighbors of v_i are in $V(M)$. Note that at least one endpoint of each edge in M_0 must be in $V(M)$ and that $V^* \cap N(v_i)$ and $V(M_0)$ are disjoint. Therefore, $|V(M)| \geq x_i + m$. Since v_i is an overloaded vertex we have that $|V(M)| > 2k$. This implies that $|M| > k$ which is a contradiction. □

Lemma 8 implies that all overloaded vertices must be in the vertex set of the edge dominating set. We *label* these vertices to indicate that these vertices are in the vertex set of the edge dominating set.

We also label a vertex v which is adjacent to a vertex of degree one.

Our kernelization algorithm is presented in Fig. 4. In the algorithm the set A' denotes the set of labeled vertices. The correctness of the algorithm follows from the following observations. Assume that there is a vertex u only adjacent to labeled vertices. Then we can delete it from the graph without increasing the size of the solution. The reason is this. Let ua be an edge that is in the edge dominating set of the original graph where a is a labeled vertex. Then we can replace ua with another edge that is incident with a to get an edge dominating set of the new graph. This is formulated in the reduction rule in Line 4 of the algorithm. We add a new edge for each labeled vertex in Line 5 to enforce that the labeled vertices are selected in the vertex set of the edge dominating set.

Algorithm $Kernel(G, k)$

1. Find a maximal matching M_0 in G.
2. Find the set A of overloaded vertices and let $A' = A$.
3. If there is a vertex $v \in V_m$ that has a degree-1 neighbor then delete v's degree-1 neighbors from the graph and let $A' \leftarrow A' \cup \{v\}$.
4. If there is a vertex $u \in V^*$ such that $N(u) \subseteq A'$ then delete u from G.
5. For each vertex $w \in A'$ add a new vertex w' and a new edge $w'w$ (In the analysis we assume that the new vertex w' is in V^*).
6. Return $(G', k' = k)$, where G' is the new graph.

Fig. 4. Algorithm $Kernel(G, k)$

It is easy to see that each step of the algorithm can be implemented in linear time. Therefore, the algorithm takes linear time.

We analyze the number of vertices in the new graph G' returned by Algorithm $Kernel(G, k)$. Note that A' is a subset of V_m. Let $B = V_m - A'$. Let q be the number of edges between V^* and B. Then

$$q = \sum_{v_i \in B} x_i \leq \sum_{v_i \in B} (2k - m) = |B|(2k - m).$$

Let

$$V_1^* = \bigcup_{v \in B} N(v) \cap V^* \quad \text{and} \quad V_2^* = V^* - V_1^*.$$

Each vertex in V_1^* is adjacent to a vertex in B. Since there are at most q edges between V_1^* and B we have

$$|V_1^*| \leq q.$$

Notice that all vertices of V_2^* have only neighbors in A'. In Line 4 the algorithm deletes all vertices that have only neighbors in A'. In Line 5 the algorithm adds a new vertex w' and a new edge $w'w$ for each vertex w in A'. Thus V_2^* is the set of new vertices that are added in Line 5. This proves

$$|V_2^*| = |A'| = 2m - |B|.$$

The total number of vertices in the graph is

$$\begin{aligned} |V_m| + |V_1^*| + |V_2^*| &\leq 2m + |B|(2k - m) + (2m - |B|) \\ &= 4m + |B|(2k - m - 1) \\ &\leq 4m + 2m(2k - m - 1) && \text{since } |B| \leq 2m \\ &= 2m(2k - m + 1) \\ &\leq 2k(k + 1) && \text{since } m \geq k + 1. \end{aligned}$$

Note that the maximal value of $2m(2k - m + 1)$ as a function of m is attained for $m = k + \frac{1}{2}$. Thus the function $2m(2k - m + 1)$ is decreasing for $m \geq k + 1$.

To obtain a bound for the number of edges we partition the edge set into three disjoint sets.

1. Let E_1 be the set of edges with two endpoints in V_m;
2. let E_2 be the set of edges between A' and V^*; and
3. let E_3 be the set of edges between B and V^*.

It is easy to see that

$$|E_1| = O(m^2) = O(k^2) \quad \text{and} \quad |E_3| = q = |B|(2k - m) = O(k^2).$$

By the analysis above

$$|E_2| \leq |A'| \times |V_1^*| + |V_2^*| \leq |A'|q + |V_2^*| \quad \Rightarrow \quad |E_2| = O(k^3).$$

Thus we obtain the following theorem.

Theorem 3. *Algorithm Kernel runs in linear time and linear space and it returns a kernel with at most $2k^2 + 2k$ vertices and $O(k^3)$ edges.*

References

1. Binkele-Raible, D., Fernau, H.: Enumerate and measure: Improving parameter budget management. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 38–49. Springer, Heidelberg (2010)
2. Cardinal, J., Langerman, S., Levy, E.: Improved approximation bounds for edge dominating set in dense graphs. Theor. Comput. Sci. 410, 949–957 (2009)
3. Fernau, H.: Edge dominating set: Efficient Enumeration-Based Exact Algorithms. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 142–153. Springer, Heidelberg (2006)
4. Fomin, F., Gaspers, S., Saurabh, S., Stepanov, A.: On two techniques of combining branching and treewidth. Algorithmica 54(2), 181–207 (2009)
5. Fujito, T., Nagamochi, H.: A 2-approximation algorithm for the minimum weight edge dominating set problem. Discrete Applied Mathematics 118(3), 19–207 (2002)
6. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. Freeman, San Francisco (1979)
7. Johnson, D., Yannakakis, M., Papadimitriou, C.: On generating all maximal independent sets. Information Processing Letters 27(3), 119–123 (1988)
8. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
9. Prieto, E.: Systematic kernelization in FPT algorithm design. PhD-thesis, The University of Newcastle, Australia (2005)
10. Randerath, B., Schiermeyer, I.: Exact algorithms for minimum dominating set. Technical Report zaik 2005-501, Universität zu Köln, Germany (2005)
11. Raman, V., Saurabh, S., Sikdar, S.: Efficient exact algorithms through enumerating maximal independent sets and other techniques. Theory of Computing Systems 42(3), 563–587 (2007)
12. van Rooij, J.M.M., Bodlaender, H.L.: Exact Algorithms for Edge Domination. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 214–225. Springer, Heidelberg (2008)
13. Xiao, M.: A Simple and Fast Algorithm for Maximum Independent Set in 3-Degree Graphs. In: Rahman, M. S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 281–292. Springer, Heidelberg (2010)
14. Xiao, M., Nagamochi, H.: Exact algorithms for annotated edge dominating set in cubic graphs. TR 2011-009. Kyoto University (2011); A preliminary version appeared as: Xiao, M.: Exact and parameterized algorithms for edge dominating set in 3-degree graphs. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 387–400. Springer, Heidelberg (2010)
15. Xiao, M., Nagamochi, H.: Parameterized edge dominating set in cubic graphs. In: Proceedings FAW-AAIM. LNCS, vol. 6681, pp. 100–112. Springer, Heidelberg (2011)
16. Yannakakis, M., Gavril, F.: Edge dominating sets in graphs. SIAM J. Appl. Math. 38(3), 364–372 (1980)

Author Index