Edward A. Hirsch   Alexander A. Razborov
Alexei Semenov   Anatol Slissenko (Eds.)

Computer Science – Theory and Applications

Third International Computer Science Symposium
in Russia, CSR 2008
Moscow, Russia, June 7-12, 2008
Proceedings

Springer

Volume Editors

Edward A. Hirsch
Steklov Institute of Mathematics at St. Petersburg
27 Fontanka, 191023 St. Petersburg, Russia
E-mail: hirsch@pdmi.ras.ru

Alexander A. Razborov
Institute for Advanced Study, School of Mathematics
Einstein Drive, Princeton, NJ 08540, USA
E-mail: razborov@ias.edu

Alexei Semenov
Moscow Institute of Open Education
6 Aviatsionny per., 125167 Moscow, Russia
E-mail: alsemenov@umail.ru

Anatol Slissenko
University Paris 12, Dept. of Informatics
61 Av. du Gén. de Gaulle, 94010, Créteil, France
E-mail: slissenko@univ-paris12.fr

Library of Congress Control Number: 2008926412

CR Subject Classification (1998): F.1.1-2, F.2.1-2, F.4.1, I.2.6, J.3

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

ISSN 0302-9743
ISBN-10 3-540-79708-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-79708-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12265961 06/3180 5 4 3 2 1 0

Preface

The Third International Computer Science Symposium in Russia (CSR-2008) was held during June 7–12, 2008 in Moscow, Russia, hosted by Dorodnicyn Computing Centre of Russian Academy of Sciences, Institute for System Programming of Russian Academy of Sciences, Moscow State University, Moscow Institute of Open Education, and Institute of New Technologies. It was the third event in the series of regular international meetings following CSR-2006 in St. Petersburg and CSR-2007 in Ekaterinburg.

The symposium was composed of two tracks: Theory and Applications/Technology. The opening lecture was given by Avi Wigderson and eight other invited plenary lectures were given by Eric Allender, Zurab Khasidashvili, Leonid Levin, Pavel Pudlák, Florin Spanachi, Limsoon Wong, Yuri Zhuravlev and Konstantin Rudakov, and Uri Zwick.

This volume contains the accepted papers of both tracks and also some of the abstracts of the invited speakers. The scope of the proposed topics for the symposium was quite broad and covered basically all areas of computer science and its applications. We received 103 papers in total. The Program Committee of the Theory Track selected 27 papers out of 62 submissions. The Program Committee of the Applications/Technology Track selected 6 papers out of 41 submissions.

Yandex provided Best Paper Awards; the recepients of these awards were selected by the Program Committee:

– Marius Zimand, "Two Sources Are Better Than One for Increasing the Kolmogorov Complexity of Infinite Sequences" — Best Paper Award for the Theory Track
– Laura Kovács, "Invariant Generation for P-solvable Loops with Assignments" — Best Paper Award for the Application/Technology Track
– Vladimir Podolskii, "A Uniform Lower Bound on Weights of Perceptrons" — Best Student Paper Award

The symposium featured two special sessions, "Andrei Muchnik. In Memoriam" and "Teaching Computer Science," and one pre-conference workshop ACC 2008 (Russian-Indian Workshop on Algebra, Combinatorics and Complexity) organized by Meena Mahajan and Mikhail Volkov.

The reviewing process was organized using the EasyChair conference system, thanks to Andrei Voronkov.

We are grateful to our sponsors:

– Russian Foundation for Basic Research
– NIX Computer Company (hardware and software selling, system integration)
– Institute of New Technologies (INT, Russian leader of ICT in school education)

– Yandex (the largest Russian Internet portal providing key Web services)
– Microsoft Russia

We also thank a group of young Moscow computer scientists, headed by Yuri Pritykin, who significantly helped with local organizational issues.

March 2008

Edward A. Hirsch
Alexander Razborov
Alexei Semenov
Anatol Slissenko

Organization

Program Committee

Theory Track

Sergei Artemov	CUNY Graduate Center, USA
Matthias Baaz	University of Technology, Vienna, Austria
Boaz Barak	Princeton University, USA
Lev Beklemishev	Steklov Institute/Moscow, Russia
Harry Buhrman	CWI / University of Amsterdam, Netherlands
Andrei Bulatov	Simon Fraser University, Canada
Evgeny Dantsin	Roosevelt University, USA
Volker Diekert	University of Stuttgart, Germany
Anna Frid	Sobolev Institute/Novosibirsk, Russia
Andreas Goerdt	TU Chemnitz, Germany
Andrew V. Goldberg	Microsoft Research — SVC, USA
Dima Grigoriev	University of Rennes, France
Yuri Gurevich	Microsoft Research, USA
Edward A. Hirsch	Steklov Institute/St.Petersburg, Russia
Nicole Immorlica	CWI, Netherlands
Pascal Koiran	ENS Lyon, France
Michal Koucký	Institute of Mathematics, Prague, Czech Republic
Yury Makarychev	Microsoft Research, USA
Yuri Matiyasevich	Steklov Institute/St.Petersburg, Russia
Alexander Razborov (Chair)	IAS, USA and Steklov Institute/Moscow, Russia
Victor Selivanov	Novosibirsk Pedagogical University, Russia
Alexander Shen	LIF CNRS, France and IPPI, Russia
Helmut Veith	TU Darmstadt, Germany
Nikolay Vereshchagin	Moscow State University, Russia
Sergey Yekhanin	IAS, USA

Application and Technology Track

Robert T. Bauer	IBM, USA
Egon Börger	University of Pisa, Italy
Stephane Bressan	National University of Singapore, Singapore
Gabriel Ciobanu	Cuza University, Iasi, Romania
Edward A. Hirsch	Steklov Institute/St.Petersburg, Russia
Michael Kishinevsky	Intel, USA

Gregory Kucherov	CNRS / LIFL / INRIA, France
Alexandre Petrenko	CRIM, Canada
Andreas Reuter	European Media Laboratory, Germany
Anatol Slissenko (Chair)	University of Paris-12, France
Elena Troubitsyna	Åbo Akademi University, Finland
Andrei Voronkov	University of Manchester, UK
Sergey Zhukov	Transas, Russia

Conference Chair

Alexei Semenov	Moscow Institute of Open Education, Russia

Steering Committee for CSR Conferences

Volker Diekert	University of Stuttgart, Germany
Anna Frid	Sobolev Institute/Novosibirsk, Russia
Edward A. Hirsch	Steklov Institute/St.Petersburg, Russia
Juhani Karhumäki	University of Turku, Finland
Mikhail Volkov	Ural State University, Russia

External Reviewers

Omran Ahmadi
Krzysztof R. Apt
Sergey Avgustinovich
Maxim A. Babenko
Marie-Pierre Beal
Emmanuel Beffara
Anne Benoit
Dario Andrea Bini
Eduardo Bonelli
Sergiy Boroday
Octav Brudaru
Olivier Carton
Nadia Creignou
Michel Dayde
Fredrik Degerlund
Walter Didimo
Alexandre Dikovsky
Gillian Dobbie
Arnaud Dury
Lance Fortnow
Frederic Gava
Alexander Gotmanov
Alexandre Guitton
Vesa Halava
Hesham Hallal
Kristoffer Arnsfelt Hansen
Johan Håstad
Reiko Heckel
Tom Hirschowitz
Thomas Hofmeister
Dmitry Itsykson
Cornel Izbasa
Sham Kakade
Panagiotis Kalnis
Juhani Karhumäki
Jarkko Kari
Dmitry Karpov
Manuel Kauers
Alexander Khodyrev
Boris Konev
Oleg Kudinov
Alexander Kulikov
Dietrich Kuske
Linas Laibinis
Hak-Keung Lam
Klaus-Jörn Lange
Jürn Laun
Martin Leucker
Nutan Limaye
Thomas Lukasiewicz
Konstantin Makarychev
Larisa Maksimova
Qaisar Ahmad Malik
Maurice Margenstern
Frédéric Mazoit
Klaus Meer
Ilya Mezhirov
Ilya Mironov
Petr Mitrichev
Michael Moortgat
Daniil Musatov
Nanjangud Narendra
Sergey Nikolenko
Dirk Nowotka
Michael Nuesken
Alexander Okhotin

Dmitry Pasechnik
Alexei Pastor
Holger Petersen
Martha Plaska
Alexander Pljasunov
Vladimir Podolskii
Ilia Ponomarenko
Alexey Pospelov
Ulrike Prange
Steven Prestwich
Yuri Pritykin
Svetlana Puzynina
Michael Raskin
Gaetan Richard
Lorenzo Robbiano
Andrei Romashchenko
Andrei Rumyantsev
Sattanathan Subramanian
Azadeh Saffarian
Martin Sauerhoff
Hans Juergen Schneider
Stefan Schwoon
Phil Scott
Natasha Sharygina
Amir Shpilka
Vladimir Shpilrain
Arseny M. Shur
Henny Sipma
Evgeny Skvortsov
Ludwig Staiger
Lev Stankevich
Rainer Steinwandt
Kazushige Terui
Pascal Tesson
Alexander Tiskin
Elpida Tzafestas
Vinodchandrad Variyam
Oleg Verbitsky
Konstantin Verchinine
Rakesh Vohra
Mikhail Volkov
Nicolai Vorobjov
Fabian Wagner
Derry Tanti Wijaya
Alexander Wolpert
Tatiana Yavorskaya

Table of Contents

Opening Lecture

Randomness – A Computational Complexity Perspective 1
Avi Wigderson

Invited Papers

Cracks in the Defenses: Scouting Out Approaches on Circuit Lower Bounds 3
Eric Allender

On Formal Equivalence Verification of Hardware 11
Zurab Khasidashvili

Twelve Problems in Proof Complexity 13
Pavel Pudlák

Manifestation and Exploitation of Invariants in Bioinformatics 28
Limsoon Wong

Simple Stochastic Games, Mean Payoff Games, Parity Games 29
Uri Zwick

Theory Track

Topological Semantics of Justification Logic 30
Sergei Artemov and Elena Nogina

A Logspace Algorithm for Partial 2-Tree Canonization 40
Vikraman Arvind, Bireswar Das, and Johannes Köbler

A Triple Correspondence in Canonical Calculi: Strong Cut-Elimination, Coherence, and Non-deterministic Semantics 52
Arnon Avron and Anna Zamansky

Computing Longest Common Substrings Via Suffix Arrays 64
Maxim A. Babenko and Tatiana A. Starikovskaya

Logic and Rational Languages of Words Indexed by Linear Orderings . . . 76
Nicolas Bedon, Alexis Bès, Olivier Carton, and Chloé Rispal

Complexity of the Bollobás-Riordan Polynomial: Exceptional Points and Uniform Reductions 86
Markus Bläser, Holger Dell, and Johann A. Makowsky

A Complete Characterization of Nash-Solvability of Bimatrix Games in Terms of the Exclusion of Certain 2×2 Subgames 99
Endre Boros, Khaled Elbassioni, Vladimir Gurvich, Kazuhisa Makino, and Vladimir Oudalov

Synchronization of Grammars 110
Didier Caucal and Stéphane Hassen

Lower Bounds for Depth-2 and Depth-3 Boolean Circuits with Arbitrary Gates 122
Dmitriy Yu. Cherukhin

A Semantic Proof of Polytime Soundness of Light Affine Logic 134
Ugo Dal Lago and Martin Hofmann

On Subword Complexity of Morphic Sequences 146
Rostislav Deviatov

Comparing Universal Covers in Polynomial Time 158
Jiří Fiala and Daniël Paulusma

S4LP and Local Realizability 168
Melvin Fitting

On the Expressive Power of Permanents and Perfect Matchings of Matrices of Bounded Pathwidth/Cliquewidth (Extended Abstract) 180
Uffe Flarup and Laurent Lyaudet

The Most General Conservation Law for a Cellular Automaton 194
Enrico Formenti, Jarkko Kari, and Siamak Taati

Lower Bounds on Frequency Estimation of Data Streams (Extended Abstract) 204
Sumit Ganguly

From Invariants to Canonization in Parallel 216
Johannes Köbler and Oleg Verbitsky

Self-referentiality of Justified Knowledge 228
Roman Kuznets

On the Complexity of Membership and Counting in Height-Deterministic Pushdown Automata 240
Nutan Limaye, Meena Mahajan, and Antoine Meyer

Public Key Encryption and Encryption Emulation Attacks 252
Denis Osin and Vladimir Shpilrain

A Uniform Lower Bound on Weights of Perceptrons 261
Vladimir V. Podolskii

Lambek Grammars with One Division Are Decidable in Polynomial Time 273
Yury Savateev

Cryptanalysis of Stickel's Key Exchange Scheme 283
Vladimir Shpilrain

Combinatorial Complexity of Regular Languages 289
Arseny M. Shur

On Sequences with Non-learnable Subsequences 302
Vladimir V. V'yugin

Algorithms for Multiterminal Cuts 314
Mingyu Xiao

Two Sources Are Better Than One for Increasing the Kolmogorov Complexity of Infinite Sequences 326
Marius Zimand

Applications and Technology Track

Multilayer Neuro-fuzzy Network for Short Term Electric Load Forecasting 339
Yevgeniy Bodyanskiy, Sergiy Popov, and Taras Rybalchenko

Invariant Generation for P-Solvable Loops with Assignments 349
Laura Kovács

Using Coloured Petri Nets to Model and Verify Telecommunications Systems 360
Valery Nepomniaschy, Dmitry Beloglazov, Tatiana Churina, and Mikhail Mashukov

Additive Preconditioning for Matrix Computations 372
Victor Y. Pan, Dmitriy Ivolgin, Brian Murphy, Rhys Eric Rosholt, Yuqing Tang, and Xiaodong Yan

Network as a Computer: Ranking Paths to Find Flows 384
Dusko Pavlovic

A Unified Categorical Approach for Attributed Graph Rewriting 398
Maxime Rebout, Louis Féraud, and Sergei Soloviev

Author Index 411

Randomness – A Computational Complexity Perspective

Avi Wigderson

Institute for Advanced Study,
School of Mathematics,
1 Einstein Drive, Princeton, NJ 08540, USA

Abstract. Man has grappled with the meaning and utility of randomness for centuries. Research in the Theory of Computation in the last thirty years has enriched this study considerably. This lecture will describe two main aspects of this research on randomness, demonstrating its power and weakness respectively.

Randomness is paramount to computational efficiency: The use of randomness seems to dramatically enhance computation (and do other wonders) for a variety of problems and settings. In particular, examples will be given of probabilistic algorithms (with tiny error) for natural tasks in different areas, which are exponentially faster than their (best known) deterministic counterparts.

Computational efficiency is paramount to understanding randomness: We will explain the computationally-motivated definition of "pseudorandom" distributions, namely ones which cannot be distinguished from the uniform distribution by any efficient procedure from a given class. Using this definition, we show how such pseudorandomness may be generated deterministically, from (appropriate) computationally difficult problems. Consequently, randomness is probably not as powerful as it seems above.

We conclude with the power of randomness in other computational settings, such as space complexity and probabilistic proof systems. In particular we'll discuss the remarkable properties of Zero-Knowledge proofs and of Probabilistically Checkable proofs.

The bibliography contains several useful books and surveys in which material pertaining to the computational randomness may be found. In particular, we include surveys on topics not covered in the lecture, including *extractors* (designed to purify weak random sources) and *expander graphs* (perhaps the most useful "pseudorandom" object).

Keywords: randomness, complexity, pseudorandom, derandomnization.

References

1. Goldreich, O.: Modern Cryptography, Probabilistic Proofs and Pseudorandomness. Algorithms and Combinatorics, vol. 17. Springer, Heidelberg (1998)
2. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
3. Shaltiel, R.: Recent Developments in Explicit Constructions of Extractors. Bull. EATCS 77, 67–95 (2002)
4. Wigderson, A.: P, NP and Mathematics — A computational complexity perspective. In: Proceedings of the ICM 2006, Madrid, vol. I, pp. 665–712. EMS Publishing House, Zurich (2007), http://www.icm2006.org/proceedings/Vol_I/29.pdf

Cracks in the Defenses: Scouting Out Approaches on Circuit Lower Bounds

Eric Allender

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

Abstract. Razborov and Rudich identified an imposing barrier that stands in the way of progress toward the goal of proving superpolynomial lower bounds on circuit size. Their work on "natural proofs" applies to a large class of arguments that have been used in complexity theory, and shows that no such argument can prove that a problem requires circuits of superpolynomial size, even for some very restricted classes of circuits (under reasonable cryptographic assumptions).

This barrier is so daunting, that some researchers have decided to focus their attentions elsewhere. Yet the goal of proving circuit lower bounds is of such importance, that some in the community have proposed concrete strategies for surmounting the obstacle. This lecture will discuss some of these strategies, and will dwell at length on a recent approach proposed by Michal Koucký and the author.

1 Introduction and Ancient History

More than a decade ago, the author wrote a survey of results in circuit complexity [7]. That survey is still depressingly up-to-date. Despite some interesting recent progress on circuit lower bounds (see, for example [16,22,40,21,17]), it is fairly accurate to say that only modest progress has been made in the field of circuit lower bounds since the dramatic results of the 1980s [5,20,42,25,34,41].

Already in the mid-1990s, Razborov and Rudich identified a reason that explained this lack of progress [33]. They examined the known lower bound arguments showing that some function f is not computed by a class $\mathcal{C}$ of circuits, and showed that such arguments all implied the existence of a combinatorial property Q such that

- Q *is large*: That is, most of the Boolean functions f on n input variables have property Q.
- Q *is constructive*: That is, given a truth-table of size $N = 2^n$ representing a Boolean function t on n input variables, determining whether t has property Q takes time only polynomial in N. (The main conclusions of [33] carry over unchanged if this is weakened to time $N^{\log^{O(1)} N}$.)
- Q *is useful* for proving that f is not computed by circuits in $\mathcal{C}$, in the sense that f has property Q, but no function computed by circuits in the class $\mathcal{C}$ has property Q.

Razborov and Rudich call any such argument a *natural proof*.

Razborov and Rudich showed in [33] that if there is a secure pseudorandom function generator computable in the class $\mathcal{C}$, then there can be no natural proof showing that f

is not in $\mathcal{C}$. Since Naor and Reingold [31] show that there are pseudorandom function generators computable in TC^0 (assuming that factoring Blum integers requires circuits of size 2^{n^ϵ} for some $\epsilon > 0$) this means that, in the likely case that factoring Blum integers is hard, no natural proof can show that any function lies outside of TC^0.

2 Tried and True Techniques

Of course, we do know that there *are* some problems that lie outside of TC^0. Indeed, the only arguments currently known showing that certain functions do not lie in TC^0 make use of *diagonalization*, which was already identified by Razborov and Rudich as a "non-natural" proof technique. Usually diagonalization applies only to *uniform* complexity classes (such as the arguments of [8,15] showing that dlogtime-uniform TC^0 is properly contained in C$_=$P and PP). However, even non-uniform classes such as P/poly can be separated from large enough classes by means of diagonalization. Diagonalization combined with "arithmetization[1]" yields the best-known result along these lines: the result of Buhrman, Fortnow and Thierauf [14] that $\mathrm{MA}_{\mathrm{EXP}}$ is not contained in P/poly (and hence is also not contained in (non-uniform) TC^0). Is there hope that these techniques might lead to better lower bounds for TC^0?

Perhaps there is hope – but it is tempered by the recognition that diagonalization and arithmetization also have severe limitations when it comes to proving circuit lower bounds. Diagonalization is the canonical example of a "relativizing" proof technique, and even when when combined with arithmetization techniques the known separation results "algebrize" (using the terminology introduced by Aaronson and Wigderson [1]). Aaronson and Wigderson show that algebrizing proof techniques are not strong enough to prove that NEXP is not in P/poly (so that the lower bound of [14] mentioned above is close to the best that can be obtained using these techniques).

It is true that this does not directly address the question of using diagonalization and arithmetization to prove lower bounds for "small" subclasses of P/poly (such as TC^0), and indeed it is debatable whether it is even relevant to talk about "relativized" or "algebrized" subclasses of P. This issue has been discussed in several papers [10,19,24,23]; see also Section 9 of [1]. Thus there is (as yet) no strong argument why diagonalization and arithmetization cannot prove separations from TC^0 – but there is also scant reason for optimism that this will be a promising avenue of attack. After all, there is no evidence that these techniques can provide alternative proofs of known separations (such as the result that PARITY is not in AC^0 [5,20,42,25]).

We are left to wonder what other approaches have been proposed, for obtaining circuit lower bounds.

3 The Mulmuley-Sohoni Approach

TC^0 is a class defined by Boolean circuits, but it also has appealing characterizations in terms of arithmetic circuits [2]. Specifically, TC^0 is the class of Boolean functions that can be represented as the sign of a function $\{0,1\}^n \rightarrow \mathbb{Z}$ that is computed by

[1] For more information on what terms such as "arithmetization mean, consult [1].

polynomial-size constant-depth unbounded-fan-in arithmetic circuits with $+$ and $\times$ gates, and constants from $\{0, 1, -1\}$. This class of arithmetic circuits (arithmetic AC^0 circuits) has enough restrictions so that existing lower bound techniques suffice to show that several functions cannot be computed by such circuits [2,9] (although they do not suffice to say much about what can be represented as the *sign* of such functions.) The natural proofs framework of Razborov and Rudich is not known to extend in a direct way to *arithmetic* circuits. Might this not offer an avenue of attack?

As it turns out, there is a fairly sophisticated plan of attack that is based on (a somewhat different model of) arithmetic circuits. Mulmuley and Sohoni proposed a program for using the techniques of algebraic geometry in order to prove lower bounds on the size of arithmetic formulae computing the permanent (and eventually for addressing the P vs NP question) [30]. The question of whether their approach might circumvent the natural proofs barrier was discussed briefly by Mulmuley and Sohoni [30] and subsequently was discussed at more length by Regan [37]. Regan reaches the conclusion that the approach proposed by Mulmuley and Sohoni holds the promise of being an "un-natural" proof technique, by violating the requirement of *constructivity*. That is, it seems that the proof might give rise to a useful and large combinatorial property Q with the property that, given a truth table t determining if the function represented by t has property Q might be very complex.

3.1 Other Nonconstructive Approaches

The call for lower bound arguments that violate the "constructivity" requirement of Razborov and Rudich is echoed in the current draft of the textbook by Arora and Barak [12]. Arora and Barak describe how improved circuit lower bounds could conceivably be based on the combinatorial property Q consisting of those functions that have high discrepancy. They observe that computing the discrepancy, given the truth table of a function, is hard for coNP, and thus this is a good candidate for violating the "constructivity" condition. It also might suggest that this is too complicated a notion to hope to analyze usefully in the context of a lower bound proof – but Arora and Barak go on to give examples of elegant and understandable proofs in the literature that rely on computing values that are NP-hard to compute in general, but which yield to analysis in such a way that does not yield an efficient algorithm. Quoting from Arora and Barak:

This suggests we should not blindly trust the intuition
that "nonconstructive $\equiv$ difficult."

4 Lower Bounds Via Derandomization

Let us turn again to the topic of arithmetic circuits. The *Identity Testing* problem is to determine, given an arithmetic circuit C, if the polynomial represented by the circuit is the identically zero polynomial. It is well known that this problem has an efficient probabilistic algorithm (see, e.g., [27,18,39,43]), and thus it is a tempting target for those seeking to derandomize probabilistic algorithms. Note that there have been some very impressive successes in the last decade in the campaign to transform probabilistic algorithms to efficient deterministic algorithms [3,38].

Unfortunately, anyone seeking to derandomize the Identity Testing problem will need to contend with the results of Kabanets and Impagliazzo [29], who showed that Identity Testing is in P only if one of the following two conditions hold:

- NEXP $\not\subseteq$ P/poly
- The Permanent does not have arithmetic circuits of polynomial size.

Conversely, sufficiently strong circuit lower bounds imply that Identity Testing can be derandomized. Thus derandomizing the Identity Testing problem is in some sense *equivalent* to proving circuit lower bounds.

There are two ways to view this state of affairs. The pessimist might conclude that derandomizing Identity Testing is hopeless. The optimist might conclude that this is *exactly* the problem to work on, in order to prove circuit lower bounds. (The optimist might take additional inspiration from the observation that it suffices to deal with arithmetic circuits that have *no input variables*; simply *evaluating* an arithmetic circuit to determine if it evaluates to zero is already as hard as the general problem [6].)

In fact, Agrawal has proposed a multi-step program to separate P from NP that proceeds by building progressively better pseudorandom generators, with the goal of proving lower bounds via derandomization [4]. Agrawal does cite the work of Razborov and Rudich, but he does not explicitly state how his program would circumvent the obstacle of Natural Proofs. I would characterize his approach to Natural Proofs as saying, in essence: “First, let’s prove the lower bound, and afterward we can figure out why Natural Proofs posed no obstacle.”

This is a reasonable stance to take, because, in Agrawal’s own words, “In the sequence of steps proposed to prove arithmetic and Boolean circuit lower bounds, perhaps the most important one is step 1” – and step 1 in Agrawal’s program does *not* seem to involve proving anything that Razborov and Rudich say should be hard to prove.

Step 1 in Agrawal’s program involves improving the Nisan-Wigderson pseudorandom generator for probabilistic AC^0 circuits [32]. The Nisan-Wigderson generator shows that any problem solvable by probabilistic AC^0 circuits can be solved in time $2^{\log^{O(1)} n}$. Agrawal proposes improving the parameters, in a way that would yield a polynomial-time algorithm. As he observes, such a construction would also show that there is a problem in DTIME($2^{O(n)}$) that requires AC^0 circuits of size $2^{\epsilon n}$ for some $\epsilon > 0$. This would be a significant advance beyond what is currently known; it is not even known if there is any problem in DTIME($2^{O(n)}$) that requires *depth three* circuits of this size. Perhaps there are significant barriers that prevent us from proving such lower bounds – but there seems to be nothing in the Natural Proofs framework that explains why this should be difficult.

5 Amplifying Modest Lower Bounds

This section describes work performed jointly by Michal Koucký and the author [11].

The work of Razborov and Rudich highlights a significant obstacle to proving *superpolynomial* lower bounds – but there is nothing in their framework that prevents “natural” proofs of quadratic or cubic lower bounds. Indeed, there are examples of proofs of this sort. Håstad showed that a certain function requires formulae of size nearly n^3

[26], and it is known that certain problems in P require branching programs of size nearly $n \log \log n$ [13]. Impagliazzo, Paturi, and Saks showed that any depth d TC^0 circuit for PARITY must have $n^{1+\Omega(1/(2.5)^d)}$ wires [28].

Thus we know of no reason why a natural proof cannot show that, say, the Boolean Formula Evaluation problem (a standard complete problem for NC^1) requires TC^0 circuits of size $n^{1.01}$.

It turns out that this would have significant consequences. It is shown in [11] that, if NC^1 $=$ TC^0, then for every $\epsilon > 0$, the Boolean Formula Evaluation problem has TC^0 circuits of size $n^{1+\epsilon}$. That is, proving even a size lower bound of $n^{1.01}$ would separate TC^0 from NC^1.

The reason for this "amplification" effect is that many of the well-studied problems in NC^1 (such as the Boolean Formula Evaluation problem) have a very strong self-reducibility property. Namely, there are very efficient reductions that reduce the problem for instances of length n to instances of length n^ϵ.

How does this relate to the Natural Proofs framework? Consider the combinatorial property Q consisting of all truth-tables of n-variate Boolean functions that are *not* computed by threshold circuits of depth $\log^* n$ and size $n^{1.01}$. This property certainly satisfies the largeness criterion. It is also easy to see that, given a truth table of size $N = 2^n$, it can be determined in time $N^{O \log^{.01} N}$ whether property Q holds. Thus, although this does not seem to be recognizable in polynomial time, it certainly is recognizable in quasipolynomial time, and thus is "constructive" enough to qualify as "natural" in most of the theorems presented by Razborov and Rudich. Thus we have an example of a large and constructive combinatorial property that is useful against TC^0 circuits of size $n^{1.01}$. Yet we know of no way to conclude from this (using the machinery of [33] or using any other argumentation) that factoring Blum integers is computable by circuits of size $2^{n^{o(1)}}$ – although this *would* be the case if we had a large constructive combinatorial property Q' that is useful not only against TC^0 circuits of size $n^{1.01}$ but against TC^0 circuits of polynomial size.

Of course, we don't know that the Boolean Formula Evaluation problem satisfies this property Q. But we do know that it satisfies the strong self-reducibility property mentioned earlier, which in turn implies that it satisfies property Q only if it does not lie in TC^0. However, only a *tiny fraction* of all functions on n variables satisfy this self-reducibility property. Thus if one were able to establish that the Boolean Formula Evaluation problem satisfies property Q, we see no obvious way that this would give rise to a "large" combinatorial property useful against TC^0, and thus this could provide a way to perform an end-run around the Natural Proofs barrier.

Although this provides a rough plan of attack for separating NC^1 from TC^0, it is interesting (or frustrating) to note that it does not provide a similar plan of attack for separating TC^0 from NP or NEXP. That is, if SAT is in TC^0, we do not know how to conclude that SAT has TC^0 circuits of size $n^{1.01}$; indeed, we do not know how to find any fixed k such that SAT has TC^0 circuits of size n^k, assuming only that NP = TC^0.

5.1 Other Evidence That Lower Bounds Are Hard

There is more than one way to explain our inability to prove lower bounds in circuit complexity. Razborov [36] has shown, under cryptographic assumptions, that certain circuit lower bounds are independent of certain theories of bounded arithmetic. (He also argues in [35] that these same theories capture the types of reasoning that have been used in lower bound arguments thus far.) It would be interesting to determine if these same logics are unable to prove that the Boolean Formula Evaluation problem requires TC^0 circuits of size $n^{1.00001}$, under similar cryptographic assumptions.

Acknowledgments

I thank Nikolay Vereshchagin and Michal Koucký for encouraging me to choose this topic for my CSR lecture. I also thank Michal Koucký for numerous discussions that have refined and clarified my view of this material, and I thank Alexander Razborov for raising some thought-provoking points. The research of the author is supported in part by NSF Grants CCF-0514155 and DMS-0652582.

References

1. Aaronson, S., Wigderson, A.: Algebrization: A new barrier in complexity theory. In: STOC (to appear, 2008)
2. Agrawal, M., Allender, E., Datta, S.: On TC^0, AC^0, and arithmetic circuits. Journal of Computer and System Sciences 60, 395–421 (2000)
3. Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Annals of Mathematics 160, 781–793 (2004)
4. Agrawal, M.: Proving lower bounds via pseudo-random generators. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
5. Ajtai, M.: Σ_1^1-formulae on finite structures. Annals of Pure and Applied Logic 24, 1–48 (1983)
6. Allender, E., Bürgisser, P., Kjeldgaard-Pedersen, J., Miltersen, P.B.: On the complexity of numerical analysis. In: Proc. 21st Ann. IEEE Conf. on Computational Complexity (CCC 2006), pp. 331–339 (2006)
7. Allender, E.: Circuit complexity before the dawn of the new millennium. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 1–18. Springer, Heidelberg (1996)
8. Allender, E.: The permanent requires large uniform threshold circuits. Chicago J. Theor. Comput. Sci. (1999)
9. Allender, E., Ambainis, A., Barrington, D.A.M., Datta, S., LêThanh, H.: Bounded depth arithmetic circuits: Counting and closure. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 149–158. Springer, Heidelberg (1999)
10. Allender, E., Gore, V.: On strong separations from AC^0. In: Cai, J.-Y. (ed.) Advances in Computational Complexity Theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13, pp. 21–37. AMS Press (1993)
11. Allender, E., Koucký, M.: Amplifying lower bounds by means of self-reducibility. In: IEEE Conference on Computational Complexity (to appear, 2008)
12. Arora, S., Barak, B.: Complexity theory: A modern approach. Draft currently available at `http:// www.cs.princeton.edu/theory/complexity/`
13. Beame, P., Saks, M., Sun, X., Vee, E.: Super-linear time-space tradeoff lower bounds for randomized computation. Journal of the ACM 50, 154–195 (2003)
14. Buhrman, H., Fortnow, L., Thierauf, T.: Nonrelativizing separations. In: IEEE Conference on Computational Complexity, pp. 8–12 (1998)

15. Caussinus, H., McKenzie, P., Thérien, D., Vollmer, H.: Nondeterministic NC^1 computation. Journal of Computer and System Sciences 57, 200–212 (1998)
16. Chattopadhyay, A., Hansen, K.A.: Lower bounds for circuits with few modular and symmetric gates. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 994–1005. Springer, Heidelberg (2005)
17. Cherukhin, D.: Lower bounds of complexity for depth-2 and depth-3 Boolean circuits with arbitrary gates. In: CSR. LNCS, vol. 5010, pp. 121–132. Springer, Heidelberg (2008)
18. DeMillo, R., Lipton, R.: A probabilistic remark on algebraic program testing. Information Processing Letters 7, 193–195 (1978)
19. Fortnow, L.: The role of relativization in complexity theory. Bulletin of the EATCS 52, 229–243 (1994)
20. Furst, M., Saxe, J., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. Mathematical Systems Theory 17, 13–27 (1984)
21. Green, F.: The correlation between parity and quadratic polynomials mod 3. J. Comput. Syst. Sci. 69(1), 28–44 (2004)
22. Hansen, K.A., Miltersen, P.B.: Some meet-in-the-middle circuit lower bounds. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 334–345. Springer, Heidelberg (2004)
23. Hartmanis, J., Chang, R., Chari, S., Ranjan, D., Rohatgi, P.: Relativization: A revisionistic perspective. In: Rozenberg, G., Salomaa, A. (eds.) Current Trends in Theoretical Computer Science. World Scientific Series in Computer Science, vol. 40, pp. 537–548. World Scientific, Singapore (1993)
24. Hartmanis, J., Chang, R., Kadin, J., Mitchell, S.: Some observations about relativizations of space bounded computations. In: Rozenberg, G., Salomaa, A. (eds.) Current Trends in Theoretical Computer Science. World Scientific Series in Computer Science, vol. 40, pp. 423–433. World Scientific, Singapore (1993)
25. Håstad, J.: Computational Limitations for Small Depth Circuits. MIT Press, Cambridge, MA (1987)
26. Håstad, J.: The shrinkage exponent of de Morgan formulas is 2. SIAM J. Comput. 27(1), 48–64 (1998)
27. Ibarra, O.H., Moran, S.: Equivalence of straight-line programs. JACM 30, 217–228 (1983)
28. Impagliazzo, R., Paturi, R., Saks, M.E.: Size-depth tradeoffs for threshold circuits. SIAM J. Comput. 26, 693–707 (1997)
29. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1-2), 1–46 (2004)
30. Mulmuley, K., Sohoni, M.A.: Geometric complexity theory I: An approach to the P vs. NP and related problems. SIAM J. Comput. 31(2), 496–526 (2001)
31. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM 51(2), 231–262 (2004)
32. Nisan, N., Wigderson, A.: Hardness vs. randomness. Journal of Computer and System Sciences 49, 149–167 (1994)
33. Razborov, A., Rudich, S.: Natural proofs. Journal of Computer and System Sciences 55, 24–35 (1997)
34. Razborov, A.A.: Lower bounds on the size of bounded depth networks over a complete basis with logical addition. Mathematicheskie Zametki 41, 598–607 (1987); English translation in: Mathematical Notes of the Academy of Sciences of the USSR 41, 333–338 (1987)
35. Razborov, A.A.: Bounded arithmetic and lower bounds. In: Clote, P., Remmel, J. (eds.) Feasible Mathematics II, Progress in Computer Science and Applied Logic, vol. 13, pp. 344–386. Birkhäuser, Basel (1995)
36. Razborov, A.A.: Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. Izvestiya Math. 59, 205–227 (1995)

37. Regan, K.W.: Understanding the Mulmuley-Sohoni approach to P vs. NP. Bulletin of the EATCS 78, 86–99 (2002)
38. Reingold, O.: Undirected st-connectivity in log-space. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 376–385. IEEE Computer Society Press, Los Alamitos (2005)
39. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM 27, 701–717 (1980)
40. Sherstov, A.A.: Separating AC^0 from depth-2 majority circuits. In: ACM Symposium on Theory of Computing (STOC), pp. 294–301 (2007)
41. Smolensky, R.: Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In: ACM Symposium on Theory of Computing (STOC), pp. 77–82 (1987)
42. Yao, A.: Separating the polynomial-time hierarchy by oracles. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 1–10 (1985)
43. Zippel, R.E.B.: Simplification of radicals with applications to solving polynomial equations. Master's thesis, M.I.T. (1977)

On Formal Equivalence Verification of Hardware

Zurab Khasidashvili

Formal Technology and Logic Group
Intel Corporation, Haifa, Israel

When modeling the logic functionality, hardware can be viewed as a Finite State Machine (FSM) [7]. The power-up state of hardware cannot be determined uniquely, therefore the FSM modeling the hardware does not have an initial state (or a set of initial states). Instead, it has a set of legal *operation states*, and it must be brought into this set of operation states from any power-up state by a *reboot sequence.*

Hardware is specified with a Hardware Description Language. In contemporary design, the *specification model* description is very close to its logic description. On the other hand, hardware is manufactured based on an *implementation model* originated from a transistor-level description. To make sure that the specification model has the intended logic functionality, *assertions* are written for the specification model in a temporal logic language, such as Linear Temporal Logic [1]. For the correct operation of hardware it is therefore necessary to

- Make sure that the reboot sequence brings the specification and implementation models into the intended set of *operation states*;
- Make sure that the specification model *satisfies the temporal assertions*, in the operation states;
- Make sure that the specification and implementation models are *equivalent*, in the operation states.

Several concepts of hardware equivalence have been proposed in the literature. They can be divided into two categories:

1. *combinational equivalence*, whose application is limited to comparing FSMs that have the same amount of state elements. The corresponding state elements in the two combinatoionally equivalent FSMs must have the same next-state functions. Thus, for combinational equivalence verification, *propositional satisfiability checking* is enough;
2. *sequential equivalence*, where FSMs with or without a set of given initial states can be compared using *model checking techniques* [1]. Sequential equivalence concepts include several forms of *replaceability equivalence*, 3*-valued equivalence*, *steady-state equivalence*, *alignability equivalence*, and *post-reboot equivalence* [7,2,9,4,6].

In this talk, we discuss recent advances in sequential equivalence verification of hardware. We will focus on post-reboot equivalence, since it preserves the validity of temporal properties between equivalent FSMs. In particular, we will discuss compositional methods for proving post-reboot equivalence in a divide-and-conquer fashion [2,5,6,8]. Further, we will briefly describe the underlying algorithms and model-checking techniques used in Intel's equivalence verification tool [3]. Finally, we will present some experimental data on equivalence verification of Intel designs.

References

1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
2. Huang, S.-Y., Cheng, K.-T.: Formal Equivalence Checking and Design Debugging. Kluwer Academic Publishers, Dordrecht (1998)
3. Kaiss, D., Goldenberg, S., Hanna, Z., Khasidashvili, Z.: Seqver: A Sequential Equivalence Verifier for Hardware Designs. In: ICCD (2006)
4. Khasidashvili, Z., Hanna, Z.: TRANS: Efficient sequential verification of loop-free circuits. In: HLDVT (2002)
5. Khasidashvili, Z., Skaba, M., Kaiss, D.: Theoretical Framework for Compositional Sequential Hardware Equivalence Verification in Presence of Design Constraints. In: ICCAD (2004)
6. Khasidashvili, Z., Skaba, M., Kaiss, D.: Post-reboot equivalence and compositional verification of hardware. In: FMCAD (2006)
7. Kohavi, Z.: Switching and Finite Automata Theory. McGraw-Hill, New York (1978)
8. Moon, I.-H., Bjesse, P., Pixley, C.: A compositional approach to the combination of combinational and sequential equivalence checking of circuits without known reset states. In: DATE (2007)
9. Pixley, C.: A theory and implementation of sequential hardware equivalence. IEEE transactions on CAD (1992)

Twelve Problems in Proof Complexity

Pavel Pudlák*

Mathematical Institute, Prague

1 Introduction

Proof complexity is a research area that studies the concept of complexity from the point of view of logic. Although it is very much connected with computational complexity, the goals are different. In proof complexity we are studying the question *how difficult is to prove a theorem?* There are various ways how one can measure the *"complexity"* of a theorem. We may ask what is *the length of the shortest proof* of the theorem in a given formal system. Thus the complexity is the size of proofs. This corresponds to questions in computational complexity about the size of circuits, the number of steps of Turing machines etc. needed to compute a given function. But we may also ask how strong theory is needed to prove the theorem. This also has a counterpart in computational complexity—the questions about the smallest complexity class to which a given set or function belongs.

Often the best way to find out what is going in some field of research is to look at open problems. Therefore my aim in this paper is to compile a list of problems in proof complexity that I consider to be important, but which also seem to be within the reach of our methods. With each problem, I shall define the necessary concepts and mention some related results.

The paper is intended for researchers in computational complexity who want to know what is going on in proof complexity and, perhaps, want to try some open problem there. Essentially all problems have already been stated before, sometimes in different forms. The reader interested in problems should consult monographs [6,20], survey articles [11,36] and other lists of problems [11,21].

2 Frege Systems

Most of my problems will be about propositional proof systems. I shall consider classical propositional logic, but I shall also mention some results about nonclassical propositional calculi. The general definition of a propositional proof system [12] is based on the following three conditions:

1. soundness;
2. completeness;
3. polynomial time decidability of the relation: *D is a proof of proposition* ϕ.

* Supported by grants A1019401 and 1M002162080.

Since the set of propositional tautologies is **coNP**-complete, there exists a proof system P such that every tautology has a proof of polynomial length in P if and only if $\mathbf{NP} = \mathbf{coNP}$. If P has that property, I shall say that P *is polynomially bounded.*

Although $\mathbf{NP} \neq \mathbf{coNP}$ is considered to be very likely true, we are not able to prove that some very basic proof systems are not polynomially bounded. These proof systems are called *Frege systems.* They are the well-known systems used in most textbooks and, in fact, fairly close to natural reasoning of mathematicians. A Frege system P is based on a finite number of axiom schemas and deduction rules. A proof in P is a string of propositions which are either instances of the axiom schemas or follow from previous ones by deduction rules.

There are two basic measures of complexity of Frege proofs. First, we may count the number of propositions in the proof; second, we may count the total length of an encoding of the proof as a binary string. This determines two measures of proof complexity of a tautology—the least *number of steps* in a proof and the *length* of the shortest proof. It has been shown that for every two Frege systems the the numbers of steps differ by at most a polynomial; the same holds for the length, [12]. In fact, when both systems use the same language (the same basis of connectives), then the lengths, resp. the numbers of steps, differ by at most a linear factor and the proof is trivial.

Problem 1. Prove a superpolynomial lower bound on the length of proofs for a Frege system (or prove that it is polynomially bounded).[1]

Essentially the only lower bound on the lengths of proofs in a Frege system is based on the simple observation that in a proof of an irreducible tautology τ [2] all subformulas must occur. Thus if the depth of τ is n the size of every proof of τ is $\Omega(n^2)$. No lower bounds are known for the number of steps!

This is the most difficult of all problems I am going to state in this paper. As a matter of fact, I doubt that it is within the reach of the current methods, but it is worth mentioning it, before talking about its weaker versions and other related problems. The number of steps in a proof is always at most its length. It seems possible that there are tautologies with proofs that have only polynomial number of steps while they have only proofs of exponential length. So the problem to prove superpolynomial lower bounds on the number of steps for Frege proofs is even harder.

One possible weakening is to prove superpolynomial lower bounds using some complexity-theoretical assumptions. Of course, the assumptions must not imply $\mathbf{NP} \neq \mathbf{coNP}$, as that assumption implies that there is no polynomially bounded propositional proof system.

Problem 2. Prove a superpolynomial lower bound on the length of proofs for a Frege system *using a conjecture that does not imply* $\mathbf{NP} \neq \mathbf{coNP}$.

[1] I put the alternative into parenthesis, because I believe it is very unlikely. In the rest of the paper I shall omit such alternatives.

[2] This means that all subformulas of τ are essential for τ being a tautology.

One natural place to look for such conjectures is in theoretical cryptography. Several conjectures in that field are stronger than $\mathbf{P} \neq \mathbf{NP}$ but they are not known to imply $\mathbf{NP} \neq \mathbf{coNP}$. Solving this problem requires finding some computational consequences of the existence of short Frege proofs. I shall say more about it in the next section.

The difficulty of proving lower bounds for Frege proofs is caused not only by the lack of suitable methods, but also by the lack of suitable candidates for hard tautologies. Most of the tautologies based on simple combinatorial principles and theorems, such as the Pigeon-Hole Principle and the finite Ramsey theorem, have been shown to have polynomial size proofs. On the opposite end of the spectrum of various tautologies, there are tautologies that are almost surely hard. These are tautologies expressing the consistency of strong theories.[3] But for this kind of tautologies our combinatorial methods do not work. The method of diagonalization, which is so useful in predicate logic, completely fails in propositional logic. Therefore we need tautologies that are based on natural and sufficiently hard combinatorial principles.

A class of candidates for hard tautologies was proposed in [24,2]. Let F_n be a mapping from binary strings of length n to binary strings of length $n+1$. Then there is a string b of length $n+1$ which is not in the range of F_n. If F_n is computable by a polynomial size circuit, we can define a polynomial size tautology $\tau_{F_n,b}$, for every $b \notin Rng(F_n)$, that expresses this property of b. The hope is that if F is sufficiently pseudorandom, then $\tau_{F_n,b}$ is hard for every $b \notin Rng(F_n)$. We know that it does not suffice to assume that F_n be a pseudorandom generator. In [45] Razborov stated specific properties of F and conjectured that the tatuologies based on such functions are hard.

Furthermore, the hardness of these formulas, for some proof systems, has been conjectured for the following specific function T. Let $s(x)$ be a numeric functions such that $s(x) = o(x)$. Given a number k, interpret binary strings of length $2^{s(k)}$ as codes of boolean circuits defining functions of k variables; interpret binary strings of length 2^k as truth tables of boolean functions of k variables. Then T is the mapping that, given a string c of length $2^{s(k)}$, maps c onto the truth table of the function computed by the circuit encoded by c. So, roughly speaking, the conjecture is that it is hard to prove lower bounds on circuit complexity.

Another specific function was studied in [26].

The connection to Frege proofs is not quite clear to me. In fact, it is conceivable that for suitable sequences of functions $\{F_n\}$ and strings $\{b_n\}$, the tautologies τ_{F_n,b_n} are hard for *all* proof systems. This conjecture is buttressed by the fact that for all proof systems for which superpolynomial lower bounds have been obtained, also superpolynomial lower bounds have been proved for formulas of this type.

The problem of proving lower bounds on the size and the number of steps for Frege proofs has been studied for nonclassical logics too. Quite recently Pavel Hrubeš proved exponential lower bounds on the number of steps in propositional intuitionistic logic and in several modal logics [16,17].

[3] For a theory T it is a sequence of tautologies $\{\tau_n\}$, where τ_n expresses that no string of length n is a proof of contradiction from the axioms of T.

Hrubeš's results cover a lot of modal logics, still there are other nonclassical logics for which no lower bounds on the lengths of proofs are known. One that I find particularly interesting is *orthomodular logic.*

Problem 3. Prove lower bounds on proofs in a Frege system for orthomodular logic.

Orthomodular logic is, roughly speaking, classical logic with distributivity replaced by the weaker law of modularity [13]. This is one of the logics studied in the field of quantum logic, but quantum physics and quantum computation are not my motivations. The reason why I think this problem should be studied is its connection to proofs of lower bounds on classical Frege systems. Connections with structures studied in quantum mechanics were mentioned already in [23]. One can show that if a Frege proof system for orthomodular logic is polynomially bounded, then so is every Frege proof system for classical logic.[4] I am proposing Problem 3 as a weaker version of the central Problem 1, but it may turn out that they are equivalent.

The weaker *orthologic* is also interesting in connection with lower bounds on classical Frege systems, however notice that it has polynomially bounded proof systems [15].

3 Feasible Interpolation

Feasible interpolation, a.k.a. effective interpolation, was invented by Jan Krajíček [19]. It is a way to obtain, from a short proof, some effective computation. In particular, to obtain a polynomial size circuit computing a function related to a suitable tautology from its polynomial size proof. This enables one to reduce the problem of proving lower bounds on the size of proofs to proving lower bounds on the size of circuits.

Let $\alpha(\bar{p}) \vee \beta(\bar{q})$ be a tautology where $\bar{p}$ and $\bar{q}$ are disjoint sets of propositional variables. Then either $\alpha(\bar{p})$, or $\beta(\bar{q})$, or both are tautologies. More generally, if $\alpha(\bar{r},\bar{p}) \vee \beta(\bar{r},\bar{q})$ is a tautology, then for every assignment of truth values $\bar{a}$ to $\bar{r}$, either $\alpha(\bar{a},\bar{p})$ or $\beta(\bar{a},\bar{q})$ or both are tautologies.

Definition 1. *A proof system P has the* feasible interpolation property*, if there exists a polynomial time algorithm A which outputs either* 0 *or* 1 *and such that given a proof D of $\alpha(\bar{r},\bar{p}) \vee \beta(\bar{r},\bar{q})$ and a truth assignment $\bar{a}$,*

1. *if A outputs 0, then $\alpha(\bar{a},\bar{p})$ is a tautology;*
2. *if A outputs 1, then $\beta(\bar{a},\bar{q})$ is a tautology.*

It should be noted that the feasible interpolation property is also equivalent to the following property. *There exists a polynomial time algorithm which from a proof D of $\alpha(\bar{r},\bar{p}) \vee \beta(\bar{r},\bar{q})$ constructs a circuit $C(\bar{r})$ such that for every truth assignment $\bar{a}$*

1. *if $C(\bar{a}) = 0$, then $\alpha(\bar{a},\bar{p})$ is a tautology;*
2. *if $C(\bar{a}) = 1$, then $\beta(\bar{a},\bar{q})$ is a tautology.*

[4] Thomas Vetterlein, personal communication.

A number of propositional proof systems posses this property. The first one for which this was established was the cut-free sequent calculus, soon after for the propositional Resolution system and others. This property was also shown for Frege systems for some nonclassical logics, including intuitionistic logic. For nonclassical logics, however, one has to modify it a little. For instance, in case of intuitionistic logic one has to consider tautologies of the form

$$(r_1 \vee \neg r_1) \wedge \ldots \wedge (r_n \vee \neg r_n) \rightarrow \alpha(\bar{r}, \bar{p}) \vee \beta(\bar{r}, \bar{q}).$$

For Frege systems for classical logic it has been shown that the property fails, assuming some likely conjectures, eg., that factoring integers is hard (ie., not solvable in polynomial time) [29,5].

It is not difficult to prove that the feasible interpolation property is equivalent to separation of some disjoint **NP** sets in the following sense (which I state in a bit informal way).

Proposition 1. *A proof system P has the feasible interpolation property if and only if whenever P proves that two* **NP** *sets A and B are disjoint using a sequence of polynomial size proofs, then A and B can be separated by a set in* **P/poly**, *(ie., $\exists C \in \mathbf{P/poly}(A \subseteq C \wedge B \cap C = \emptyset)$.*

Consequently, if $\mathbf{NP} \cap \mathbf{coNP} \nsubseteq \mathbf{P/poly}$, then the feasible interpolation property implies that the proof system is not polynomially bounded. This assumption is not known to imply $\mathbf{NP} \neq \mathbf{coNP}$. Thus we get conditional superpolynomial lower bounds using a condition different from $\mathbf{NP} \neq \mathbf{coNP}$. In many cases, however, after proving the feasible interpolation property also unconditional exponential lower bounds have been proved.

Since this method proved to be extremely useful for proving lower bounds, my questions concern the possibility of extending it to stronger systems. I shall state the problems only for Frege systems, but they are meaningful for every system for which we do not have the feasible interpolation property. The first problem is about the possibility to replace the separation using sets in **P/poly** by separation using more complex sets.

Problem 4. Prove that Frege systems have the feasible interpolation property in a more general sense, namely, with the separation using sets in **P/poly** replaced by separation using sets in a larger complexity class.

As shown in [37], it suffices to determine how difficult is to separate the canonical pair of **NP** sets associated with a Frege system. Recall that the canonical pair of a proof system P is the pair of the following two **NP** sets [41]:

$$\begin{aligned} &Prov(P) := \{(\phi, 0^n) \ ; \ \phi \text{ has a proof of length at most } n \text{ in } P\} \\ &NegSat := \{(\phi, 0^n) \ ; \ \neg\phi \text{ is satisfiable}\}. \end{aligned}$$

(The string 0^n only serves for padding the input to make it of length at least n.) Thus the Problem 4 asks for a nontrivial upper bound on the complexity of sets that separate the canonical pair of a Frege system. This connection also shows an important fact about the problem: *we do not not have to consider general proofs, but only certain very concrete ones.* In Frege systems the propositional translations of the sentence

$$Prov(\mathbf{F}) \cap NegSat = \emptyset$$

have polynomial size proofs, where **F** denotes some fixed Frege system.[5] These are the proofs that we only need to consider.

Notice that a solution of Problem 4 would also be a solution of Problem 2. There may be other ways one can generalize feasible interpolation so that it also holds for Frege systems, the form of which may eventually have little to do with the original concept of interpolation. Therefore I shall state a version of the previous problems in a very general form.

Problem 5. Derive any nontrivial computational consequences from the existence of a small Frege proof.

There is another reason for posing the problem in this way. It is well known that intuitionistic logic and some other related logics are *constructive.* This means, roughly speaking, that one can interpret proofs as algorithms. This mainly concerns predicate logic, but there are results of this kind also for propositional logic [8]. So the above problem can be paraphrased: *Does classical propositional logic have any constructive properties?*

Problems 2,4 and 5 are very much related, one should view them as possible ways of attacking the central Problem 1.

Since all problems about Frege systems seem to be very hard, one should start with some special cases. In case of the problems about the feasible interpolation, one should start with bounded depth Frege systems. Assuming likely conjectures, they do not have the feasible interpolation property already for small depths [5]. It would be interesting to find some generalized interpolation for as weak a system as $Res(log)$, which is a generalization of Resolution, based on disjunctions (clauses) of conjunctions of logarithmic lengths.

4 The Bounded Arithmetic Hierarchy

I shall start with a topic that at first will seem completely unrelated to previous problems. For $n \geq 0$, T_2^n denotes the theory axiomatized by induction axioms restricted to Σ_n^b formulas. The class of Σ_n^b defines precisely the class of sets Σ_n^p of the *Polynomial Hierarchy.* The *Bounded Arithmetic Hierarchy* is the sequence of theories $\{T_2^n\}$. Roughly speaking, T_2^n formalizes reasoning that uses only concepts from the n-th level of the Polynomial Hierarchy. (See [6,20] for definitions.)

We proved that if the Polynomial Hierarchy is strictly increasing, then so is the Bounded Arithmetic Hierarchy [27]. Furthermore, we proved that the relativized

[5] These sentences are also known as the *Reflection Principle for the Frege System,* see [37].

Bounded Arithmetic Hierarchy is strictly increasing. That results, however, only show that there are Σ^b_{n+2} sentences that separate T^{n+1}_2 from T^n_2, resp. the same for the relativized case.[6] It is an open problem whether one can prove similar results for sentences of fixed complexity.

Problem 6. (I) Assuming some reasonable conjecture in computational complexity theory, for some k, prove or disprove that for all $n \geq 0$, T^{n+1}_2 proves more Σ^b_k sentences than T^n_2.

(II) The same for relativized theories $T^n_2[R]$, without using unproven assumptions.

The relativized theories $T^n_2[R]$ are extensions of T^n_2 obtained by adding a new predicate R and extending the induction axioms to $\Sigma^b_n[R]$; there are no specific axioms for R. The predicate R plays a similar role as oracles in relativized complexity classes and this connection is actually used for separation results.

Recently, most research activities focused on the Σ^b_1 sentences provable in theories T^n_2. These sentences are related to a very natural concept in computational complexity theory.

Definition 2. *A* total **NP** search problem *is determined by a relation $R \in$* **P** *and a polynomial p such that*

$$\forall x \exists y(|y| \leq p(|x|) \wedge R(x, y)). \tag{1}$$

The sentence (1), which expresses that the search problem is total, is a Σ^b_1 sentence. For T^0_2, the total search problems corresponding to the provable Σ^b_1 sentences, are solvable in polynomial time. For T^1_2 the search problems belong to the well-known class *Polynomial Local Search,* **PLS**. Characterizations of the search problems of higher levels of the Bounded Arithmetic Hierarchy were obtained quite recently [38,31,46]. I shall describe the simplest characterization, which is due to Skelley and Thapen [46].

An *n-game* is an n-ary relation $G(x_1, \ldots, x_n)$. We think of it as played by two players, A is starting and B playing as the second. The players alternate in picking x_i's; B wins if $G(x_1, \ldots, x_n)$ holds true, otherwise A wins. The concept of a winning strategy is well-known. Further, we need the concept of a *reduction of an n-game G to an n-game H*. It is a string of functions $f_1, \ldots, f_n$ such that for every $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ such that $y_i = f_i(x_1, x_3, \ldots, x_i)$ for i odd, and $x_i = f(y_2, y_4, \ldots, y_i)$ for i even, if $H(y_1, \ldots, y_n)$, then $G(x_1, \ldots, x_n)$.

If we have a wining strategy for B in H and a reduction of G to H, then we obtain a winning strategy for B in G by simply composing the strategy with the reduction.

The principle GI_n says that the following is *impossible:*

> *There are games $G_0, \ldots, G_a$, a winning strategy α for A in game G_0, reductions ρ_i of G_{i+1} to G_i for $i = 0, \ldots, a-1$ and a winning strategy β for B in G_a.*

[6] More precisely, we should denote these sentences by $\forall\Sigma_{n+2}$, as we are talking about the universal closures of Σ_{n+2} sentences.

Indeed, if we compose $\beta, \rho_{a-1}, \rho_{i-2}, \dots, \rho_0$, we obtain a winning strategy for B in G_0 contradicting to the existence of a winning strategy for A in G_0.

The total **NP** search problem associated with GI_n is defined using circuits. The games $G_0, \dots, G_a$ are given by a circuit $C(z, x_1, \dots, x_n)$, where for the binary string $\bar{i}$ representing index i, $0 \leq i \leq a$, $C(\bar{i}, x_1, \dots, x_n)$ defines game G_i. Similarly, the reductions ρ_i, $0 \leq i \leq a$, are given by one circuit. Further, we have a circuit defining strategy α and a circuit defining strategy β. Notice that the number of games a is, in general, exponential in the size of input.

The task of the search problem is, given the circuits, to find out what is wrong. Namely, we should find

1. either $x_1, \dots, x_n$ that show that α is not a winning strategy for A in G_0,
2. or i, $0 \leq i < a$, $x_1, \dots, x_n$ and $y_1, \dots, y_n$ that show that ρ_i is not a reduction of G_{i+1} to G_i,
3. or $y_1, \dots, y_n$ that show that β is not a winning strategy for B in G_a.

This is also the way in which GI_n is formalized as a Σ_1^b sentence. I shall use the same notation for the principles, their formalizations and the associated search problems.

Theorem 1 ([46]). *For $n \geq 1$, $GI_n[R]$ characterizes $\Sigma_1^b[R]$ consequences of $T_2^n[R]$ (hence also GI_n characterizes Σ_1^b consequences of T_2^n).*

A search problem S is *polynomially reducible* to a search problem S', if we can solve S in polynomial time using queries to an oracle that produces solutions of the queried instances S'. If Σ_1^b theorems of $T_2^{n+1}[R]$ are the same as Σ_1^b theorems of $T_2^n[R]$, then for every oracle A, GI_{n+1}^A is polynomially reducible to GI_n^A (using also the oracle A). In other words, that assumption implies that GI_{n+1} is polynomial reducible to GI_n and the proof relativizes. This enables us to reduce the Problem 6 to a purely computational one.

Problem 7. For $n = 1, 2, \dots$, find an oracle A such that the search problem GI_{n+1}^A is not reducible to the search problem GI_n^A.

I have stated this problem for a specific characterization, but one can try other characterizations of Σ_1^b theorems of theories T_2^n. This is only a matter of convenience, all these problems are equivalent.

One should not forget about the unrelativized case.

Problem 8. For $n = 1, 2, \dots$, find a reasonable conjecture in complexity theory which implies that the search problem GI_{n+1} is not reducible to the search problem GI_n.

A solution of this problem may be a clue for solving the previous problem. Specifically, if the conjecture used in a solution to Problem 8 can be proved when relativized by an oracle, then we get a solution to Problem 7.

5 Bounded Depth Frege Systems

Bounded depth circuits are an important class of circuits studied in computational complexity, and exponential lower bounds on the size of such circuits computing explicitly defined functions have been proved. A related concept has been studied in proof complexity. A *depth d Frege system* is a Frege system in which only formulas of depth d are allowed. As in case of bounded depth circuits, the depth of Frege proofs is the the maximal number of alternations of $\wedge, \vee$ and $\neg$ in a formula of the proof (we assume that no other connectives are used).

Let $\mathbf{F}$ denote some Frege systems and $\mathbf{F}_d$ its depth d restrictions. Following the brake-through superpolynomial lower bound of Ajtai [1], exponential lower bounds on bounded depth Frege proof have been proved [19,28,34]. Specifically, for every fixed d, the tautology expressing the Pigeon-Hole Principle has only exponentially long proofs in $\mathbf{F}_d$. Superpolynomial separations of $\mathbf{F}_d$ from $\mathbf{F}_{d+1}$ was proved in [30] using padded Pigeon-Hole tautologies, which are of depth 2. Also exponential separation of $\mathbf{F}_d$ from $\mathbf{F}_{d+1}$ is known [19], but it uses tautologies of maximal depth that is possible in these systems.[7] The following is still an open problem.

Problem 9. Does there exist a k such that for every $d \geq k$ there exists a sequence of tautologies of depth k that have polynomial size proofs in $\mathbf{F}_{d+1}$, but which do not have proofs of size $2^{(\log n)^{O(1)}}$ in $\mathbf{F}_d$?

We believe that the answer to this problem is positive with $k = 2$ and with a lower bound $2^{n^{\Omega(1)}}$. But why do we need a lower bound $2^{(\log n)^{\omega(1)}}$? It is because such a lower bound would help us solve Problem 7. The statement of Problem 9 does not formally imply the statement of Problem 7. To get such a relationship we would have to insist that the sequences of tautologies are uniform in a certain well defined sense. Namely, the tautologies should be propositional translations of Σ^b_k sentences. However, it seems unlikely that the use of nonuniform sequences of families could help.

The candidate tautologies are the translations of the Σ^b_1 sentences that characterize sentences provable in T^d_2. But even the simplest ones, the GI_d are fairly complicated which is the reason why researcher have not studied their propositional translations. For a few small depths we have simpler candidates. In particular, the minimal depth in which one can prove the finite Ramsey theorem is an open problem.

The only general lower bound technique for bounded depth Frege proofs that we have is based on Switching Lemmas [3]. Formally they look very much like the classical Switching Lemma of Yao and Håstad and the proof techniques are similar, but there are additional technical complications. Every tautology

[7] The depth of these tautologies is $d + 1$, therefore one has to use $\mathbf{F}_d$ as a refutation system, in order to be able to prove such tautologies, or one can formalize $\mathbf{F}_d$ as a sequent system with arbitrary formulas and the cut rule restricted to depth d formulas.

requires a lemma of a specific form, thus the more complicated the tautology is, the more complicated the lemma is. Another problem is that these lemmas do not have interpretations in finite domains, thus we have to treat them purely syntactically, or use nonstandard models.

In boolean complexity theory exponential lower bounds have been proved for a larger class of circuit. A MOD_q^n gate is a boolean function of n variables, whose value is 1 if and only if the number of ones in the input string is divisible by q. Razborov and Smolensky considered bounded depth circuits with ANDs, ORs, NOTs and gates MOD_p^n with p prime, and proved exponential lower bounds on the circuit size of some explicitly defined functions [40,47]. After the method of random restrictions had been adapted for bounded depth Frege proofs and exponential lower bounds had been proved, researchers in proof complexity attempted to prove lower bounds on the more general type of bounded depth Frege system in which MOD_p^n gates, for p prime, were allowed. But an adaptation of the approximation method turned out to be much harder, if not impossible. So the following is still an open problem. Let $\mathbf{F}_d[p]$ denote a suitable depth d Frege system that uses gates AND, OR, NOT and MOD_p.

Problem 10. Prove superpolynomial lower bounds on $\mathbf{F}_d[p]$ proofs for p prime.

I am not considering $\mathbf{F}_d[q]$ systems with q composite, although such systems can be defined, because superpolynomial lower bounds for bounded depth circuits with MOD_q, q composite, is a widely open problem, and we expect that the corresponding problem in proof complexity will be even harder to solve.

I will now explain what is the obstacle to adapting the method of approximation to $\mathbf{F}_d[p]$. Let us first recall how the lower bounds on bounded depth circuits with modular gates are proved. The basic idea is to approximate functions computed at the gates of the circuit by low degree polynomials. Then one shows that the precision of the approximation deteriorates slowly, thus the output function should be approximated well, assuming the circuit is small. Finally, one proves that such an approximation does not exist for the given function.

Given an $\mathbf{F}_d[p]$ proof, we would like to mimic the above reasoning. So we would like to associate low degree polynomials with formulas in the proof and show that the polynomials approximate axioms very well and the precision of the approximation decreases slowly in the course of the proof. But what does it mean to approximate a formula in a proof? If we count truth assignments as in the proof for bounded depth circuits we get nowhere. Each formula in the proof is a tautology, hence it is trivially approximated by the constant 1 (which is a zero degree polynomial). According to our experience from the proofs for $\mathbf{F}_d$, we have to consider "imaginary" truth assignments that falsify the tautology ϕ for which we want to prove a lower bound. Such assignments do not exist in real world (since ϕ is a tautology), we can only imagine them, or we have to use nonstandard models. If we use a nonstandard model $\mathcal{M}$, then the imaginary assignments are represented by real objects, but they have to be *external* to $\mathcal{M}$. Then, for a polynomial p, we need to express in $\mathcal{M}$ on how many of these external assignments p vanishes. That is the problem, because there is no natural way how to count external objects inside of $\mathcal{M}$.

This is the main stumbling block when one tries to translate the approximation method in a straightforward way. Researchers tried several other ways, but they only obtained partial results. The *Polynomial Calculus* was proposed as the most rudimentary special case of $\mathbf{F}_d[p]$ proofs [10]. Exponential lower bounds have been proved for this system [44] and a reduction to lower bounds for the Polynomial Calculus extended by certain axioms has been found [7].[8] Lower bounds for a system that combines $\mathbf{F}_d$ with the Polynomial Calculus were proved in [22].

6 Integer Linear Programing

The general form of an Integer Linear Programing problem is: *for a given set of inequalities with rational coefficients, find solutions in the domain of integers.* If we want to study the complexity of Integer Linear Programing, we can simplify it by considering only the decision problem: *does the system of inequalities have an integral solution?* It is well-known that this problem is **NP**-complete.

From the point of view of proof complexity, the most interesting problem is: *how difficult is to prove that a given set of inequalities does not have an integral solution?* Since it is a **coNP**-complete problem, we believe that proofs in any proof system must be exponentially large. Since we are not able to prove this conjecture in general, we would like to prove it at least in some special cases, ie., for particular proof systems.

Exponential lower bounds have been proved for two systems [35,14]. Furthermore, exponential lower bounds have been obtained for several other systems for *tree-like proofs,* see [18], and [4] combined with the recent bounds on multiparty communication complexity of disjointness [32,9]

I shall describe in more detail one proof system which seems within the reach of our methods; it is the *Lovász-Schrijver system* [33]. We want to prove the unsatisfiability of a system of linear inequalities $\{L_i \geq 0\}_{i=1}^m$ by integers. The initial inequalities are:

1. $L_i \geq 0$, $i = 1, \dots, m$;
2. $x_j^2 - x_j \geq 0$, for any variable x_j used in $\{L_i \geq 0\}_{i=1}^m$.

A proof is a sequence of inequalities derived from the initial inequalities by the rules of the system, ending with the contradictory inequality $-1 \geq 0$. The inequalities in the proof are of degree at most 2. The rules are:

1. we can derive any positive linear combination of established inequalities;
2. from a linear inequality $L \geq 0$, we can derive $x_j L \geq 0$, for any variable x_j used in $\{L_i \geq 0\}_{i=1}^m$;

[8] More precisely, it is a reduction to an extension of a weaker system called the *Nullstellensatz System.*

3. from a linear inequality $L \geq 0$, we can derive $(1 - x_j)L \geq 0$, for any variable x_j used in $\{L_i \geq 0\}_{i=1}^m$.

Exponential lower bounds on tree-like proofs follow from the aforementioned results, for general proofs (DAG-like), however, it is an open problem.

Problem 11. Prove superpolynomial lower bounds on Lovász-Schrijver proofs.

For the Lovász-Schrijver system the feasible interpolation property has been proved [35], thus we know that the system is weak. As I have mentioned, in many cases unconditional lower bounds were found after the feasible interpolation property had been established. These lower bounds are based on monotone versions of the feasible interpolation property, in which monotonic computational models are used instead of boolean circuits. For the Lovász-Schrijver proof system the following monotonic model is needed. I call it *monotone linear programs for computing boolean functions.* Such a program P is given by a set of inequalities of the form:

$$\sum_j a_{ij} z_j \leq \sum_k b_{ik} x_k + c_i$$

where $a_{ij}, b_{i,k}, c_i \in \mathbf{Q}$ are constants, $b_{i,k} \geq 0$, and z_j, x_k are variables. Variables x_k are used for 0–1 inputs. P computes the boolean function $f(\bar{x})$ that for every string of zeros and ones $\bar{d}$ satisfies:

$$P_{\bar{x}:=\bar{d}} \text{ has a solution, iff } f(\bar{d}) = 1.$$

The solution is for the variables z_j and we require $z_j \geq 0$. Notice that P computes a monotone boolean function because of the condition $b_{i,k} \geq 0$. Without this condition the model would be as efficient as general boolean circuits.

Solving the following problem positively would be a major step towards proving superpolynomial lower bounds on Lovász-Schrijver proofs.

Problem 12. Prove a superpolynomial lower bound on the size of a monotone linear program computing an explicitly defined monotone boolean function.

The problem is important also for computational complexity, since monotone linear programs are the strongest monotonic computational model that has been defined.

Acknowledgment

I thank Jan Krajíček for discussing the problems and his remarks to the draft of this paper. It should be noted, however, that we do not fully agree on which problems are the most important in proof complexity.

References

1. Ajtai, M.: The complexity of the pigeonhole principle. In: Proc. IEEE 29th Annual Symp. on Foundation of Computer Science, pp. 346–355 (1988)
2. Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Pseudorandom Generators in Propositional Proof Complexity. SIAM Journal on Computing 34(1), 67–88 (2004)
3. Beame, P.: A switching lemma primer. Technical Report UW-CSE-95-07-01, Department of Computer Science and Engineering, University of Washington (November 1994)
4. Beame, P., Pitassi, T., Segerlind, N.: Lower Bounds for Lovász-Schrijver Systems and Beyond Follow from Multiparty Communication Complexity. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1176–1188. Springer, Heidelberg (2005)
5. Bonet, M., Pitassi, T., Raz, R.: No feasible interpolation for TC^0-Frege proofs. In: Proc. 38-th FOCS, pp. 254–263 (1997)
6. Buss, S.R.: Bounded Arithmetic. Bibliopolis (1986)
7. Buss, S., Impagliazzo, R., Krajíček, J., Pudlák, P., Razborov, A.A., Sgall, J.: Proof complexity in algebraic systems and constant depth Frege systems with modular counting. Computational Complexity 6, 256–298 (1996(/1997)
8. Buss, S., Pudlák, P.: On the computational content of intuitionistic propositional proofs. Annals of Pure and Applied Logic 109, 49–64 (2001)
9. Chattopadhyay, A., Ada, A.: Multiparty Communication Complexity of Disjointness. arXiv e-print (arXiv:0801.3624)
10. Clegg, M., Edmonds, J., Impagliazzo, R.: Using the Groebner basis algorithm to find proofs of unsatisfiability. In: Proc. 28-th ACM STOC, pp. 174–183 (1996)
11. Clote, P., Krajíček, J.: Open Problems. In: Clote, P., Krajíček, J. (eds.) Arithmetic, Proof Theory and Computational Complexity, pp. 1–19. Oxford Press (1993)
12. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. Journ. Symbolic Logic 44, 25–38 (1987)
13. Dalla Chiara, M.L., Giuntini, R.: Quantum Logics. In: Gabbay, Guenthner (eds.) Handbook of Philosophical Logic, pp. 129–228. Kluwer Academic Publishers, Dordrecht (2002)
14. Dash, S.: An Exponential Lower Bound on the Length of Some Classes of Branch-and-Cut Proofs. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 145–160. Springer, Heidelberg (2002)
15. Egly, U., Tompits, H.: On different proof-search strategies for orthologic. Stud. Log. 73, 131–152 (2003)
16. Hrubeš, P.: A lower bound for intuitionistic logic. Ann. Pure Appl. Logic 146(1), 72–90 (2007)
17. Hrubeš, P.: Lower bounds for modal logics. Journ. Symbolic Logic (to appear)
18. Kojevnikov, A., Itsykson, D.: Lower Bounds of Static Lovász-Schrijver Calculus Proofs for Tseitin Tautologies. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 323–334. Springer, Heidelberg (2006)
19. Krajíček, J.: Lower Bounds to the Size of Constant-Depth Propositional Proofs. J. of Symbolic Logic 59(1), 73–86 (1994)
20. Krajıček, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. In: Encyclopedia of Mathematics and its Applications 60, Cambridge Univ. Press, Cambridge (1995)

21. Krajíček, J.: A fundamental problem of mathematical logic. Collegium Logicum, Annals of Kurt Gödel Society 2, 56–64 (1996)
22. Krajıček, J.: Lower bounds for a proof system with an exponential speed-up over constant-depth Frege systems and over polynomial calculus. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 85–90. Springer, Heidelberg (1997)
23. Krajíček, J.: On methods for proving lower bounds in propositional logic. In: Dalla Chiara, M.L., et al. (eds.) Logic and Scientific Methods, pp. 69–83. Kluwer Acad. Publ., Dordrecht
24. Krajíček, J.: On the weak pigeonhole principle. Fundamenta Mathematicae 170(1-3), 123–140 (2001)
25. Krajíček, J.: Proof complexity. In: Laptev, A. (ed.) European congress of mathematics (ECM), Stockholm, Sweden, June 27–July 2, 2004, pp. 221–231. European Mathematical Society (2005)
26. Krajíček, J.: A proof complexity generator. In: Glymour, C., Wang, W., Westerstahl, D. (eds.) Proc. 13th Int. Congress of Logic, Methodology and Philosophy of Science, Beijing. ser. Studies in Logic and the Foundations of Mathematics. King's College Publications, London (to appear, 2007)
27. Krajíček, J., Pudlák, P., Takeuti, G.: Bounded Arithmetic and the Polynomial Hierarchy. Annals of Pure and Applied Logic 52, 143–153 (1991)
28. Krajíček, J., Pudlák, P., Woods, A.: Exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle. Random Structures and Algorithms 7(1), 15–39 (1995)
29. Krajíček, J., Pudlák, P.: Some consequences of cryptographical conjectures for S^1_2 and EF. Information and Computation 140, 82–94 (1998)
30. Krajíček, J., Impagliazzo, R.: A note on conservativity relations among bounded arithmetic theories. Mathematical Logic Quarterly 48(3), 375–377 (2002)
31. Krajíček, J., Skelley, A., Thapen, N.: NP search problems in low fragments of bounded arithmetic. J. of Symbolic Logic 72(2), 649–672 (2007)
32. Lee, T., Schraibman, A.: Disjointness is hard in the multi-party number on the forehead model. arXiv e-print (arXiv:0712.4279)
33. Lovász, L., Schrijver, A.: Cones of matrices and set-functions and 0-1 optimization. SIAM J. Optimization 1(2), 166–190 (1991)
34. Pitassi, T., Beame, P., Impagliazzo, R.: Exponential Lower Bounds for the Pigeonhole Principle. Computational Complexity 3, 97–140 (1993)
35. Pudlák, P.: Lower bounds for resolution and cutting planes proofs and monotone computations. J. Symbolic Logic 62(3), 981–998 (1997)
36. Pudlák, P.: The lengths of proofs. In: Handbook of Proof Theory, pp. 547–637. Elsevier, Amsterdam (1998)
37. Pudlák, P.: On reducibility and symmetry of disjoint NP-pairs. Theor. Comput. Science 295, 323–339 (2003)
38. Pudlák, P.: Consistency and games—in search of new combinatorial principles. In: Helsinki, Stoltenberg-Hansen, V., Vaananen, J. (eds.) Proc. Logic Colloquium 2003. Assoc. for Symbolic Logic, pp. 244–281 (2006)
39. Pudlák, P., Sgall, J.: Algebraic models of computation and interpolation for algebraic proof systems. In: Beame, P.W., Buss, S.R. (eds.) Proof Complexity and Feasible Arithmetics. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 39, pp. 279–295

40. Razborov, A.A.: Lower bounds on the size of bounded-depth networks over a complete basis with logical addition (Russian). Matematicheskie Zametki 41(4), 598–607 (1987); English translation in: Mathematical Notes of the Academy of Sci. of the USSR 41(4), 333–338 (1987)
41. Razborov, A.A.: On provably disjoint NP-pairs. BRICS Report Series RS-94-36 (1994), `http://www.brics.dk/RS/94/36/index.html`
42. Razborov, A.A.: Unprovability of lower bounds on the circuit size in certain fragments of Bounded Arithmetic. Izvestiya of the R.A.N. 59(1), 201–222 (1995); see also Izvestiya: Mathematics 59(1), 205–227
43. Razborov, A.A.: Bounded Arithmetic and Lower Bounds in Boolean Complexity. In: Feasible Mathematics II, pp. 344–386. Birkhäuser Verlag (1995)
44. Razborov, A.A.: Lower bound for the polynomial calculus. Computational Complexity 7(4), 291–324 (1998)
45. Razborov, A.A.: Pseudorandom Generators Hard for k-DNF Resolution and Polynomial Calculus Resolution (2002-2003), `http://www.mi.ras.ru/r̃azborov/resk.ps`
46. Skelley, A., Thapen, N.: The provably total search problems of Bounded Arithmetic (preprint, 2007)
47. Smolensky, R.: Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In: STOC, pp. 77–82 (1987)

Manifestation and Exploitation of Invariants in Bioinformatics

Limsoon Wong

School of Computing, National University of Singapore
Blk COM1, 13 Law Link, Singapore 117590
wongls@comp.nus.edu.sg

Abstract. The accumulation of huge amount of biomedical data and the need to turn such data into useful knowledge lead to many challenging bioinformatics problems. Many techniques have been developed for the bioinformatics problems that have emerged, and more are being proposed everyday. I present here a selection of these problems and techniques, highlighting a fundamental property that is common to all of them. Specifically, I observe that these problems are characterized by invariants that emerge naturally from the causes and/or effects of these problems, and show that the techniques for their solutions are essentially exploitation of these invariants. In the process, we learn several major paradigms (invariants, emerging patterns, guilt by association), some important applications (active sites, key mutations, origin of species, protein functions, disease diagnosis), some interesting technologies (sequence comparison, multiple alignment, machine learning, signal processing, microarrays), and the economics of bioinformatics.

Simple Stochastic Games, Mean Payoff Games, Parity Games

Uri Zwick

Tel Aviv University, Israel
zwick@cs.tau.ac.il

Abstract. Simple Stochastic Games (SSGs), Mean Payoff Games (MPGs), and Parity Games (PGs) are three closely related families of infinite-duration games played on finite graphs. The best known algorithms for the solution of these games run in subexponential time and it is a major open problem whether they can be solved in polynomial time. In the talk, I plan to define these games, describe what is known about them and present many intriguing open problems.

Topological Semantics of Justification Logic

Sergei Artemov[1] and Elena Nogina[2,*]

[1] CUNY Graduate Center, 365 Fifth Ave., New York City, NY 10016, USA
SArtemov@gc.cuny.edu
[2] BMCC CUNY, Dept. of Math, 199 Chambers Street, New York, NY 10007, U.S.A.
E.Nogina@Gmail.com

Abstract. The Justification Logic is a family of logical systems obtained from epistemic logics by adding new type of formulas $t{:}F$ which reads as *t is a justification for F.* The major epistemic modal logic $\mathsf{S4}$ has a well-known Tarski topological interpretation which interprets $\Box F$ as the interior of F (a topological equivalent of the '*knowable part of F*'). In this paper we extend the Tarski topological interpretation from epistemic modal logics to justification logics which have both: knowledge assertions $\Box F$ and justification assertions $t{:}F$. This topological semantics interprets modality as the interior, terms t represent tests, and a justification assertion $t{:}F$ represents a *part of F which is accessible for test t*. We establish a number of soundness and completeness results with respect to Kripke topology and the real line topology for $\mathsf{S4}$-based systems of Justification Logic.

Keywords: Justification Logic, Logic of Proofs, modal logic, topological semantics, Tarski.

1 Introduction

The Justification Logic is a family of logical systems originated from the Logic of Proofs LP (cf. [3,5,9,10]). These systems are obtained from epistemic modal logics by adding new type of formulas $t{:}F$ which read as

t is a justification for F.

Justification Logic overlaps mathematical logic, epistemology, λ-calculi, etc. The standard arithmetical provability semantics for LP was given in [3]. The epistemic Krike-style semantics for LP was offered in [16,17] and later extended to Justification Logic systems containing both epistemic modalities for "*F is known*" and justification assertions "*t is a justification for F*" ([9,10]). The major epistemic modal logic $\mathsf{S4}$ which in the context of the Logic of Proofs may be regarded as a logic of explicit provability has a well-known Tarski's topological interpretation. Such a connection between topology and modal logic proved to be very fruitful

* The author is supported in part by a Research Grant from the CUNY Research Foundation.

for both domains. In particular, topology was used in [25] to prove Gödel's conjecture about a fair embedding of Intuitionistic Logic to Modal Logics. On the other hand, Modal Logic was used to describe the behavior of dynamic systems in real topology ([14]).

2 Background

The application of modal logic to topology dates back to Kuratowski [21] and Riesz in [32]. Let

$$\mathcal{T} = \langle \boldsymbol{X}, \mathbb{I} \rangle$$

be a topological space, where $\boldsymbol{X}$ is a set and $\mathbb{I}$ the interior operation. The following principles hold for all subsets Y and Z of $\boldsymbol{X}$:

1. $\mathbb{I}(Y \cap Z) = \mathbb{I}Y \cap \mathbb{I}Z$;
2. $\mathbb{I}Y = \mathbb{I}\mathbb{I}Y$;
3. $\mathbb{I}Y \subseteq Y$;
4. $\mathbb{I}\boldsymbol{X} = \boldsymbol{X}$.

These principles can be written as propositional modal formulas: Boolean operations are represented by the corresponding Boolean connectives, and the interior operator $\mathbb{I}$ by the modality $\Box$:

1. $\Box(A \wedge B) = \Box A \wedge \Box B$;
2. $\Box A \rightarrow \Box\Box A$;
3. $\Box A \rightarrow A$;
4. $\Box\top$.

These are the well-known postulates of the modal logic S4. This corellation was noticed in the late 1930s by Tarski, Stone, and Tang. Neither Lewis' original motivation of modal logic ([22,23]), nor Gödel's provability interpretation of S4 ([18]) were related to topology.

The **Tarski topological interpretation** of a propositional modal language naturally extends the set-theoretical interpretation of classical propositional logic. Given a topological space $\mathcal{T} = \langle \boldsymbol{X}, \mathbb{I} \rangle$ and a valuation (mapping) $*$ of propositional letters to subsets of $\boldsymbol{X}$, we can extend it to all modal formulas as follows:

$$\begin{gathered} \neg A = \boldsymbol{X} \setminus A^*; \\ (A \wedge B)^* = A^* \cap B^*; \\ (A \vee B)^* = A^* \cup B^*; \\ (\Box A)^* = \mathbb{I}A^*. \end{gathered} \tag{1}$$

A formula A is called valid in $\mathcal{T}$ (notation: $\mathcal{T} \Vdash A$) if

$$A^* = \boldsymbol{X}$$

for any valuation $*$. The set

$$\mathbf{L}(\mathcal{T}) := \{A \mid \mathcal{F} \Vdash A\}$$

is called the modal logic of $\mathcal{T}$.

The following classical result in this area is due to McKinsey and Tarski:

Theorem. ([24]) *Let $\mathcal{S}$ be a separable dense-in-itself metric space. Then* $\mathbf{L}(\mathcal{S}) = \mathsf{S4}$.

In particular, this yields that for each $n = 1, 2, 3, \ldots$,

$$\mathbf{L}(\mathbb{R}^n) = \mathsf{S4}.$$

Simplified proofs of this theorem were obtained in [12,26,34].

Kripke semantics can be regarded a special case of topological semantics. Indeed, given a Kripke frame (W, R), one can construct the topological space $(W, \mathbb{I})$ where

$$\mathbb{I}U := \{x \mid R(x) \subseteq U\},$$

so that validities in these two entities are the same. Hence, Kripke-completeness yields the topological completeness.

As we have mentioned above, the Justification Logic grew from the Logic of Proofs LP. A first incomplete sketch of the Logic of Proofs was made in Gödel's lecture of 1938 [19], which was not published until 1995 when the full Logic of Proofs was rediscovered independently in [2]. The Logic of Proofs LP ([2,3,4,6,15]) introduces the notion of *proof polynomials*, i.e., terms built from proof variables and constants by means of three operations:

- *application* "$\cdot$", which given a proof s of an implication $F \rightarrow G$ and a proof t of its antecedent F provides a proof $s \cdot t$ of the succedent G;
- *sum* "+", which given proofs s and t returns a proof $s + t$ of everything proven by s or t;
- *proof checker* "!", which given a proof t of F verifies it and provides a proof $!t$ of the fact that t is indeed a proof of F.

LP is the classical logic with additional atoms

$$p{:}F$$

where p is a proof polynomial and F is a formula, with the intended reading

p is a proof of F .

As it was shown in [2,3], LP describes all valid principles of proof operators $t{:}F$

t is a proof of F in Peano Arithmetic

in its language. LP is able to realize the whole Gödel's S4 by recovering proof polynomials for provability assertions in any S4-derivation (realization theorem); this result provides a mathematical formalization of the Brouwer-Heyting-Kolmogorov semantics for intuitionistic logic IPC via well-known Gödel's translation of IPC into S4 [2,3,4,6,15]. The papers [1,8,28,29,30,31,33,35] studied joint logics of proofs and provability in a format that includes both provability assertions $\Box F$ and proof assertions $t{:}F$.

In [5,8,9,10] this approach has been extended to epistemic logic and applied for building mathematical models of justification, knowledge and belief. In particular, [9] introduced and studied the basic epistemic logic with justifications,

$$\mathsf{S4LP} = \mathsf{S4} \quad + \quad \mathsf{LP} \quad + \quad (t{:}F \to \Box F)\ .$$

Epistemic models for Justification Logics has been developed in [5,8,9,10, 16,17,27]. A Fitting model for S4LP is $(W, R, \mathcal{A}, \Vdash)$, where

- (W, R) is an S4-frame;
- $\mathcal{A}$ is an admissible evidence function: for each term t and formula F, $\mathcal{A}(t, F)$ is a subset of W. Informally, $\mathcal{A}(t, F)$ specifies a set of worlds where t is an admissible evidence for F. An evidence function is assumed to be monotonic:

  $$u \in \mathcal{A}(t, F) \text{ and } uRv \text{ yield } v \in \mathcal{A}(t, F)$$

  and has natural closure properties that agree with operations of S4LP;
- $\Vdash$ behaves in the standard Kripke style on Boolean connectives and $\Box$:
  - $u \Vdash P$ or $u \not\Vdash P$ is specified for each world u and each propositional variable P;
  - $u \Vdash F \wedge G$ iff $u \Vdash F$ and $u \Vdash G$, $u \Vdash F \vee G$ iff $u \Vdash F$ or $u \Vdash G$, $u \Vdash \neg F$ iff $u \neg \Vdash F$;
  - $u \Vdash \Box F$ iff $v \Vdash F$ for all v such that uRv;
- $u \Vdash t{:}F$ iff $u \Vdash \Box F$ and $u \in \mathcal{A}(t, F)$.

In [8,17], S4LP is shown to be sound and complete with respect to this epistemic semantics.

3 Topological Semantics for Justifications

We start with offering a topological semantics for operation-free single-modality Justification Logics. It means we will work with the usual language of propositional modal logic enriched by a new construction $t : F$ where t is a **proof variable** and F is a formula.

An interpretation is specified for a topological space $\mathcal{T} = \langle \boldsymbol{X}, \mathbb{I} \rangle$ supplied with a *test function* $\mathcal{M}$ which maps a term t and a formula F to $\mathcal{M}(t, F) \subseteq \boldsymbol{X}$. The informal meaning of $\mathcal{M}$ is that $\mathcal{M}(t, F)$ represents a 'potentially accessible' region of $\boldsymbol{X}$ associated with F and t.

We assume that an evaluation $*$ works on propositional variables, Boolean connectives and modality $\Box$ according to the usual aforementioned Tarski interpretation (1). We will study several natural ways to extend $*$ on formulas $t{:}F$ and corresponding subsystems of S4LP. This approach was first discussed in [11].

We build our topological semantics for the Justification Logic language on the following formal and informal assumptions.

1. Our semantics is based on Tarski's topological semantics (1), e.g.,

$$(\Box F)^* = \mathbb{I}(F^*).$$

2. Justification terms are symbolic representations of *tests*. We postulate existence of a test function $\mathcal{M}$ which for each t and F specifies a set of points $\mathcal{M}(t, F)$ which we call

   the set of possible outcomes of a test t of a property F.

3. The $t{:}F$ will return a set of points where a test t confirms F. This reading will be supported by definitions (for different subsystems of $\mathsf{S4LP}$):

$$(t{:}F)^* \;=\; F^* \cap \mathcal{M}(t, F) \tag{2}$$

   or

$$(t{:}F)^* \;=\; \mathbb{I}(F^*) \cap \mathcal{M}(t, F). \tag{3}$$

   In case (2), test t supports F at all points where the possible outcome of t lies inside F. Case (3) corresponds to the "robust" understanding of testing: test t supports F at all points of the possible outcome of t which lie in the interior of F.
4. We first consider systems without operations on tests.

Now we introduce several systems of Justification Logic and simultaneously define their topological semantics in format $(\mathcal{T}, \mathcal{M})$ where $\mathcal{T} = \langle \boldsymbol{X}, \mathbb{I}\rangle$ is a topological space and $\mathcal{M}$ is a test function.

3.1 Basic Testing System $\mathsf{S4B_0}$

The most basic system in our list is

$$\mathsf{S4B}_0 \;=\; \mathsf{S4} \;+\; (t{:}F \rightarrow F).$$

In this system, there are no any assumptions about tests; they don't necessarily produce open sets of outcomes. The topological interpretation of $\mathsf{S4B}_0$ combines Tarski topological interpretation (1) for Booleans and modality $\Box$ with the interpretation of the justification assertions like in (2), i.e.,

$$(t{:}F)^* \;=\; F^* \cap \mathcal{M}(t, F).$$

3.2 Robust Testing System $\mathsf{S4B_1}$

The next system under consideration is

$$\mathsf{S4B}_1 \;=\; \mathsf{S4} \;+\; (t{:}F \rightarrow \Box F).$$

In $\mathsf{S4B}_1$, test sets are not necessarily open; however, the justification assertions are interpreted as "robust inclusion", i.e., case (3) :

$$(t{:}F)^* \;=\; \mathbb{I}(F^*) \cap \mathcal{M}(t, F)$$

3.3 Robust Open Testing System $\mathsf{S4B_2}$

Finally, we consider

$$\mathsf{S4B_2} \;=\; \mathsf{S4} \;+\; (t{:}F \rightarrow F) \;+\; (t{:}F \rightarrow \Box t{:}F).$$

This system corresponds to the full operation-free version of $\mathsf{S4LP}$. The test sets are assumed to be open, the justification assertions are interpreted in the robust sense (3):

$$(t{:}F)^* \;=\; \mathbb{I}(F^*) \cap \mathcal{M}(t,F)$$

3.4 Topological Soundness and Completeness

Theorem 1. All three systems $\mathsf{S4B_0}$, $\mathsf{S4B_1}$, and $\mathsf{S4B_2}$ are sound and complete with respect to the corresponding classes of topological models.

Proof. The soundness proofs are straightforward. In view of the Tarski topological interpretation of $\mathsf{S4}$ ([24]), it suffices to establish validity of non-$\mathsf{S4}$ principles of $\mathsf{S4B_0}$, $\mathsf{S4B_1}$, and $\mathsf{S4B_2}$ in the corresponding cases.

Principle $t{:}F \rightarrow F$ is valid in both (2) and (3) since in each case $(t{:}F)^*$ is a subset of F^*. Principle $t{:}F \rightarrow \Box F$ is valid in (3) since $(t{:}F)^*$ is a subset of $(\Box F)^*$, which is the interior of F^*. Finally, $t{:}F \rightarrow \Box t{:}F$ is valid in (3) since the test sets are open hence $(t{:}F)^*$ are all open and coincide with their interiors.

Completeness proofs go via epistemic models which are then converted into topological spaces with topology induced by the Kripke accessibility relation.

We consider the case of $\mathsf{S4B_2}$, the remaining cases are receiving a similar treatment. Let us first establish the completeness of $\mathsf{S4B_2}$ with respect to the class of Fitting models $(W, R, \mathcal{A}, \Vdash)$ without operations on justifications.

We follow the standard canonical model construction.

– W is the set of all maximal consistent sets in $\mathsf{S4B_2}$. We denote elements of W as Γ, Δ, etc.;
– $\Gamma R \Delta$ iff $\Gamma^\sharp \subseteq \Delta$, where $\Gamma^\sharp = \{\Box F \mid \Box F \in \Gamma\}$;
– $\mathcal{A}(s,F) = \{\Gamma \in W \mid s{:}F \in \Gamma\}$;
– $\Gamma \Vdash p$ iff $p \in \Gamma$.

Let us check that $(W, R, \mathcal{A}, \Vdash)$ is indeed an $\mathsf{S4B_2}$-model. It is immediate from the definitions that the accessibility relation R is reflexive and transitive. The admissible evidence function $\mathcal{A}$ is monotonic. Indeed, suppose $\Gamma \in \mathcal{A}(t,F)$ and $\Gamma R \Delta$. Then $t{:}F \in \Gamma$, $\Box t{:}F \in \Gamma$, $\Box t{:}F \in \Delta$, and $t{:}F \in \Delta$, i.e. $\Delta \in \mathcal{A}(t,F)$.

Lemma 1 (Truth Lemma). For every formula F, $\Gamma \Vdash F$ iff $F \in \Gamma$.

Proof. Induction on F. The base case in given in the definition of the canonical model. The Boolean and modality cases are standard. Let us consider the case when F is $t{:}G$.

Let $t{:}G \in \Gamma$ and $\Gamma R \Delta$. Then $\Box t{:}G \in \Gamma$, $\Box t{:}G \in \Delta$, $t{:}G \in \Delta$ (since $\Box t{:}G \rightarrow t{:}G$), and $G \in \Delta$ (since $t{:}G \rightarrow F$). By the Induction Hypothesis, $\Delta \Vdash G$.

Furthermore, by the definition of the admissible evidence function, $\Gamma \in \mathcal{A}(t, G)$, hence $\Gamma \Vdash t{:}G$.

If $t{:}G \notin \Gamma$, then $\Gamma \notin \mathcal{A}(t, G)$, hence $\Gamma \nVdash t{:}G$.

Let us now finish the proof of completeness of $\mathsf{S4B}_2$ with respect to $\mathsf{S4B}_2$-models. Suppose $\mathsf{S4B}_2 \nvdash F$. Then the set $\{\neg F\}$ is consistent, and hence included into some maximal consistent set Γ. Naturally, $F \notin \Gamma$. By the Truth Lemma, $\Gamma \nVdash F$.

Now we convert a given countermodel $\mathcal{K} = (W, R, \mathcal{A}, \Vdash)$ for F into an appropriate topological space and find an interpretation under which F does not hold. A Kripke topological space $\mathcal{T}_{\mathcal{K}}$ associated with $\mathcal{K}$ is a topological space with the carrier W and open sets which are all subsets of W closed upward under R:

$$Y \text{ is open iff for all } u \in Y\text{, if } uRv \text{ then } v \in Y.$$

To make $\mathcal{T}_{\mathcal{K}}$ a topological $\mathsf{S4B}_2$-model it remains to define a test function

$$\mathcal{M}(t, F) = \mathcal{A}(t, F).$$

Given a Fitting model $\mathcal{K} = (W, R, \mathcal{A}, \Vdash)$ for $\mathsf{S4B}_2$ we can also define a topological interpretation $*$ of $\mathsf{S4B}_2$-language in $\mathcal{T}_{\mathcal{K}}$:

$$p^* = \{u \in W \mid u \Vdash p\} \text{ for a propositional letter } p.$$

Any interpretation $*$ is extended to all $\mathsf{S4B}_2$-formulas in the standard way:

- $(A \vee B)^* = A^* \cup B^*$;
- $(\neg A)^* = W \setminus A^*$;
- $(\Box A)^* = \mathbb{I}(A^*)$;
- $(t{:}A)^* = \mathbb{I}(A^*) \cap \mathcal{M}(t, A)$.

From the definitions it is immediate that $t{:}G \to G$ holds at this model. Note that due to monotonicity of the admissible evidence function $\mathcal{A}$, for each t and F the test sets $\mathcal{M}(t, G)$ are open in $\mathcal{T}_{\mathcal{K}}$. Therefore $t{:}G \to \Box t{:}G$ also holds at the model.

Lemma 2 (The Main Lemma)

$$u \Vdash G \quad \Leftrightarrow \quad u \in G^*$$

Proof. Induction on G. The base case when G is atomic is covered by the definition. The Boolean connective case is straightforward.

Let G be $\Box B$. Suppose $u \Vdash \Box B$, then for all $v \in W$ such that uRv, $v \Vdash A$ as well. By the Induction Hypothesis, $v \in B^*$ for all $v \in W$ such that uRv. This yields that the whole open cone $O_u = \{v \mid uRv\}$ is a subset of B^*. Therefore, $u \in \mathbb{I}(B^*) = (\Box B)^*$.

Suppose $u \in (\Box B)^* = \mathbb{I}(B^*)$. Since $\mathbb{I}(B^*)$ is open, $v \in \mathbb{I}(B^*)$ hence $v \in B^*$ for all v such that uRv. By the Induction Hypothesis, $v \Vdash B$ for all v such that uRv. Therefore, $u \Vdash \Box B$.

Let G be $t{:}B$. Suppose $u \Vdash t{:}B$. Then, by definition, $u \in \mathcal{A}(t, B)$ and $v \Vdash B$ for all v such that uRv. By the definition of a test function, $u \in \mathcal{M}(t, B)$. By the Induction Hypothesis, $v \in B^*$ for all v such that uRv, which means that $u \in \mathbb{I}(B^*)$. Hence $u \in \mathbb{I}(B^*) \cap \mathcal{M}(t, B)$, i.e., $u \in (t{:}B)^*$.

Suppose $u \in \mathbb{I}(B^*) \cap \mathcal{M}(t, B)$. Then $u \in \mathcal{M}(t, B)$ hence $u \in \mathcal{A}(t, B)$. Furthermore, $u \in \mathbb{I}(B^*)$. Like in the case $G = \Box B$, we conclude that $u \Vdash \Box B$. Altogether this yields $u \Vdash t{:}B$.

To conclude the proof of Theorem 1 consider a Fitting $\mathsf{S4B}_2$-model, where $u \nVdash F$. By the Main Lemma, $u \notin F^*$, hence F is not valid in the topological $\mathsf{S4B}_2$-model $\mathcal{T}_{\mathcal{K}}$.

3.5 Completeness with Respect to Real Topology

Theorem 2. $\mathsf{S4B}_0$, $\mathsf{S4B}_1$, $\mathsf{S4B}_2$ are complete with respect to the real topology $\mathbb{R}^n$.

Proof. We will use the following main lemma from recent refinements of the Tarski Theorem from [12,26,34] :

Lemma 3. There is an open and continuous map π from $(0, 1)$ onto the Kripke topological space corresponding to a finite rooted Kripke frame.

Such a map π preserves truth values of modal formulas at the corresponding points. It suffices now to refine the proof of Theorem 1 to produce a finite rooted Fitting counter-model for F and to define the test function $\mathcal{M}'(t, G)$ on $(0, 1)$ as

$$\mathcal{M}'(t, G) = \pi^{-1}\mathcal{M}(t, G).$$

The resulted topological model is a $(0, 1)$-countermodel for F. This construction yields completeness with respect to the real topology $\mathbb{R}^n$, for each $n = 1, 2, 3, \ldots.$

4 Future Work

The next natural steps in this direction could be introducing operations on tests. It also looks promising to introduce tests in systems of topological reasoning about knowledge [13] and Dynamic Topological Systems [7,20].

References

1. Artemov, S.: Logic of proofs. Annals of Pure and Applied Logic 67(1), 29–59 (1994)
2. Artemov, S.: Operational modal logic. Technical Report MSI 95-29, Cornell University (1995)
3. Artemov, S.: Explicit provability and constructive semantics. Bulletin of Symbolic Logic 7(1), 1–36 (2001)
4. Artemov, S.: Kolmogorov and Gödel's approach to intuitionistic logic: current developments. Russian Mathematical Surveys 59(2), 203–229 (2004)
5. Artemov, S.: Justified common knowledge. Theoretical Computer Science 357(1-3), 4–22 (2006)

6. Artemov, S., Beklemishev, L.: Provability logic. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, 2nd edn., vol. 13, pp. 229–403. Kluwer, Dordrecht (2004)
7. Artemov, S., Davoren, J., Nerode, A.: Modal logics and topological semantics for hybrid systems. Technical Report MSI 97-05, Cornell University (1997)
8. Artemov, S., Nogina, E.: Logic of knowledge with justifications from the provability perspective. Technical Report TR-2004011, CUNY Ph.D. Program in Computer Science (2004)
9. Artemov, S., Nogina, E.: Introducing justification into epistemic logic. J. of Logic and Computation 15(6), 1059–1073 (2005)
10. Artemov, S., Nogina, E.: On epistemic logic with justification. In: van der Meyden, R. (ed.) Theoretical Aspects of Rationality and Knowledge. The Tenth Conference (TARK 2005), Singapore, June 10-12, 2005, pp. 279–294. National University of Singapore (2005)
11. Artemov, S., Nogina, E.: On topological semantics of justification logic. In: Algebraic and Topological Methods in Non-Classical Logics (TANCL 2007), Oxford, England (2007), `http://www2.maths.ox.ac.uk/notices/events/special/tancl07/`
12. Bezhanishvili, G., Gehrke, M.: Completeness of S4 with respect to the real line: revisited. Annals of Pure and Applied Logic 131, 287–301 (2005)
13. Dabrowski, A., Moss, L., Parikh, R.: Topological Reasoning and the Logic of Knowledge. Annals of Pure and Applied Logic 78(1-3), 73–110 (1996)
14. Davoren, J., Nerode, A.: Logics for Hybrid Systems (invited paper). Proceedings of the IEEE 88(7), 985–1010 (2000)
15. de Jongh, D., Japaridze, G.: Logic of provability. In: Buss (ed.) Handbook of Proof Theory, pp. 475–546. Elsevier, Amsterdam (1998)
16. Fitting, M.: A semantics for the logic of proofs. Technical Report TR-2003012, CUNY Ph.D. Program in Computer Science (2003)
17. Fitting, M.: The logic of proofs, semantically. Annals of Pure and Applied Logic 132(1), 1–25 (2005)
18. Gödel, K.: Eine Interpretation des intuitionistischen Aussagenkalkuls. Ergebnisse Math. Kolloq. 4, 39–40 (1933); English translation in: Feferman, S., et al., (eds) Kurt Gödel Collected Works, vol. I, pp. 301–303. Oxford University Press, Oxford, Clarendon Press, New York (1986)
19. Gödel, K.: Vortrag bei Zilsel, 1938. In: Feferman, S. (ed.) Kurt Gödel Collected Works, vol. III, pp. 86–113. Oxford University Press, Oxford (1995)
20. Kremer, P., Mints, G.: Dynamic topological logic. Annals of Pure and Applied Logic 131, 133–158 (2005)
21. Kuratowski, C.: Sur l'operation a de l'analysis situs. Fundamenta Mathematicae 3, 181–199 (1922)
22. Lewis, C.I.: A Survey of Symbolic Logic. University of California Press (1918)
23. Lewis, C.I., Langford, C.H.: Symbolic logic. Dover New York (1932)
24. McKinsey, J.C.C., Tarski, A.: The algebra of topology. Annals of Mathematics 45, 141–191 (1944)
25. McKinsey, J.C.C., Tarski, A.: Some theorems about the sentential calculi of Lewis and Heyting. J. of Symbolic Logic 13, 1–15 (1948)
26. Mints, G., Zhang, T.: A proof of topological completeness for S4 in $(0,1)$. Annals of Pure and Applied Logic 133(1-3), 231–245 (2005)
27. Mkrtychev, A.: Models for the logic of proofs. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 266–275. Springer, Heidelberg (1997)

28. Nogina, E.: Logic of proofs with the strong provability operator. Technical Report ILLC Prepublication Series ML-94-10, Institute for Logic, Language and Computation, University of Amsterdam (1994)
29. Nogina, E.: Grzegorczyk logic with arithmetical proof operators. Fundamental and Applied Mathematics 2(2), 483–499 (1996) (in Russian)
30. Nogina, E.: On logic of proofs and provability. Bull. of Symbolic Logic 12(2), 356 (2006)
31. Nogina, E.: Epistemic completeness of GLA. Bull. of Symbolic Logic 13(3), 407 (2007)
32. Riesz, F.: Stetigkeitsbegriff und abstrakte mengenlehre. In: Atti del IV Congr. Internat. d. Mat., vol. II, Roma (1909)
33. Sidon, T.: Provability logic with operations on proofs. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 342–353. Springer, Heidelberg (1997)
34. Slavnov, S.: On completeness of dynamic topological logic. Moscow Math J. 5(2), 477–492 (2005)
35. Yavorskaya (Sidon), T.: Logic of proofs and provability. Annals of Pure and Applied Logic 113(1-3), 345–372 (2001)

A Logspace Algorithm for Partial 2-Tree Canonization

Vikraman Arvind[1], Bireswar Das[1], and Johannes Köbler[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113, India
{arvind,bireswar}@imsc.res.in
[2] Institut für Informatik, Humboldt Universität zu Berlin, Germany
koebler@informatik.hu-berlin.de

Abstract. We show that partial 2-tree canonization, and hence isomorphism testing for partial 2-trees, is in deterministic logspace. Our algorithm involves two steps: (a) We exploit the "tree of cycles" property of biconnected partial 2-trees to canonize them in logspace. (b) We analyze Lindell's tree canonization algorithm and show that canonizing general partial 2-trees is also in logspace, using the algorithm to canonize biconnected partial 2-trees.

1 Introduction

Computing canonical forms for graphs is a fundamental algorithmic problem. The problem is closely related to the graph isomorphism problem GI. Let $\mathcal{G}$ be a class of (encodings of) graphs closed under isomorphism. We say that f computes a *complete invariant* for $\mathcal{G}$, if $\forall G, H \in \mathcal{G} : G \cong H \Leftrightarrow f(G) = f(H)$. A complete invariant f for $\mathcal{G}$ that computes for any graph $G \in \mathcal{G}$ a graph $f(G)$ that is isomorphic to G is called a *canonization* for $\mathcal{G}$. We call the graph $f(G)$ the *canonical form* of G (w.r.t. f) and denote it by *canon*(G). E.g. we could define $f(G)$ as the lexicographically least graph isomorphic to G. This canonizing function is computable in $\mathsf{FP}^{\mathsf{NP}}$ by prefix search, but it is NP-hard [8,18]. Whether there is *some* canonizing function for graphs that is polynomial-time computable is a long-standing open question. No better bound than $\mathsf{FP}^{\mathsf{NP}}$ is known for general graphs (for any canonizing function). Clearly, GI is polynomial-time reducible to graph canonization. It is an open question if the converse reduction holds.

The seminal paper of Babai and Luks [8] takes a general group-theoretic approach to graph canonization. However, combinatorial methods have worked well in various special cases. For example, for random graphs the Weisfeiler-Lehman method [7,6] produces a canonical form with high probability. From a complexity-theoretical viewpoint, a key result [17,14] we recall is that tree canonization is complete for deterministic logspace.[1] Indeed, Lindell's logspace upper bound for tree canonization is our main motivation for the present paper

[1] Provided that the tree is given in the pointer notation; using the parenthesis notation the problem is NC^1-complete (see [14]).

and, more generally, our motivation for studying the space bounded complexity of Graph Isomorphism for partial k-trees.

The recent TC^1 upper bound for isomorphism of partial k-trees by Grohe and Verbitsky [10] raises the question about a tight complexity-theoretic classification of the problem. In the present paper, we give a deterministic logspace algorithm for canonizing partial 2-trees. The algorithm is based on ideas from Lindell's tree canonization algorithm and uses the combinatorial characterizations of partial 2-trees. This tightly classifies partial 2-tree isomorphism. The class of partial 2-trees coincides with the class of series-parallel graphs and contains all outerplanar graphs. Thus, we obtain logspace canonization algorithms for these graph classes. Furthermore, partial 2-trees are planar graphs. However, we do not know if planar graph isomorphism is in logspace (or in NL). In that direction, recently Thierauf and Wagner gave a $\mathsf{UL} \cap \mathsf{co}\text{-}\mathsf{UL}$ upper bound for planar *3-connected* graph isomorphism [20]. They also provide an NL algorithm for oriented graphs.

We note that Arnborg and Proskurowski gave a linear time and quadratic space[2] canonization algorithm for both partial 2-trees and partial 3-trees [5]. Their algorithm is based on a graph reduction technique and uses labels to canonically record the reductions. However, this technique does not appear useful for showing a logspace upper bound.

For partial k-trees in general, we don't know of any better bound than TC^1. We do not know of any hardness result that would indicate that partial k-tree isomorphism is not in deterministic logspace. However, the TC^1 upper bound suggests that we can put perhaps the problem in a natural complexity class contained in TC^1 like LOGCFL or DET, or in the logspace counting hierarchy [3,2]. Recently, in [1] we have shown for *full* k-trees that isomorphism testing is in the strongly unambiguous logspace class $\mathsf{StUL} \subseteq \mathsf{UL}$. The bound follows from an $\mathsf{FL}^{\mathsf{StUL}}$ canonizing algorithm for full k-trees.

Due to lack of space, some proofs are omitted from this extended abstract.

1.1 Preliminaries

The class L consists of all languages accepted by a deterministic $O(\log n)$ space bounded Turing machine. NL is defined in the same way by using nondeterministic machines. FL contains all functions computable by deterministic $O(\log n)$ space bounded Turing machines.

By graphs we mean finite simple graphs. For a graph $G = (V, E)$, let $V(G)$ denote its vertex set V and $E(G)$ denote its edge set E. For a vertex $v \in V(G)$, the set $\{w \in V(G) \mid \{v, w\} \in E(G)\}$ of all *neighbors* of v is denoted by $N(v)$. For a subset $U \subseteq V(G)$, we use $G[U]$ to denote the *induced subgraph* of G, where $V(G[U]) = U$ and $E(G[U]) = \{e \in E(G) \mid e \subseteq U\}$. For the graph $G[V - U]$ we also use the notation $G - U$. We say that U *separates* two vertices v and w, if $v \neq w$ and there is no path in $G - U$ from v to w.

Two graphs G and H are *isomorphic* (in symbols $G \cong H$) if there is a bijection $\tau : V(G) \to V(H)$, such that for all $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and

[2] They also have an $O(n \log n)$ time algorithm which they refer to as log-linear time and space in their paper.

only if $\{\tau(u), \tau(v)\} \in E(H)$. In case the vertices of G and H are labeled (or colored), then the isomorphism τ must also preserve the labels (resp. colors). An *automorphism* of a (possibly vertex/edge colored) graph $G = (V, E)$ is a bijection $\xi : V \longrightarrow V$ that is an isomorphism from G to itself. The set $Aut(G)$ of all automorphisms of a graph G is a group under permutation composition. The Graph Isomorphism problem, denoted GI, is to decide if two input graphs G and H are isomorphic. Further, we consider the following problems:

– Aut (automorphism group): On input a graph G, compute a generating set for $Aut(G)$.
– pGA (partially specified graph automorphism): Given a graph G and a list of pairs of vertices $(u_1, v_1), \ldots, (u_l, v_l)$, does G have an automorphism ξ such that $\xi(u_i) = v_i$ for $i = 1, \ldots, l$? Let sGA denote the search version of pGA.

An isomorphism $\phi : V(G) \longrightarrow V(canon(G))$ from a graph G to its canonical form $canon(G)$ is called a *canonical labeling*. We assume $V(canon(G)) = \{1, 2, \ldots, |V(G)|\}$. Hence, the canonical labeling actually gives an ordering of the vertices of G. For a canonical form, the set of isomorphisms from G to $canon(G)$ is the *canonical labeling coset* $CL(G)$. Clearly, $CL(G) = Aut(G)\xi$, for some isomorphism ξ from G to $canon(G)$. Thus, $CL(G)$ can be represented by some $\xi \in CL(G)$ together with a generating set for $Aut(G)$. As shown by Gurevich, canonization of general graphs is polynomial-time equivalent to computing a complete invariant [9]. We define two closely related problems on canonization.

– CL-Coset: Given a graph G compute the canonical labeling coset $CL(G)$ of G.
– Color-CL: Given a colored graph G, compute some isomorphism from G to $canon(G)$, i.e., compute a member of the canonical labeling coset $CL(G)$ of the colored graph G.

Notice that Color-CL is a search problem (i.e. there might exist more than one solution for a given graph G). We assume that an oracle for Color-CL is actually a *functional* oracle that returns for any query G some canonical labeling in $CL(G)$.

2 Relative Complexity of Computing Canonical Forms, Canonical Labelings, and Labeling Cosets

It is well-known (see, e.g., [11,16]) that the problems Aut, pGA, and sGA are all polynomial-time equivalent to Graph Isomorphism. Similarly, the problem of computing canonical forms is easily seen to be polynomial-time equivalent to the problem CL-Coset of computing the corresponding canonical labeling coset as well as to Color-CL [8]. However, it is not clear whether these reductions can also be performed in logspace. In the following we compare the different problems w.r.t. logspace Turing reductions (denoted by $\leq_T^L$) and show that all these problems reduce to the canonical labeling problem for colored graphs.

Lemma 1. $\text{pGA} \leq_T^L \text{Aut} \leq_T^L \text{sGA} \leq_T^L \text{Color-CL}$.

As $\text{Aut} \leq_T^L \text{Color-CL}$ we easily get the following consequence.

Corollary 2. $\text{CL-Coset} \leq_T^L \text{Color-CL}$.

In order to canonize a given partial 2-tree G, we decompose G into its biconnected components $G_1, \ldots, G_r$. Since G is a tree T of its biconnected components, we will essentially canonize T using the biconnected component canonization as a subroutine. Now, for this entire procedure to work in logspace, we design a deterministic logspace base machine that makes calls to subroutines that canonize the biconnected partial 2-trees $G_1, \ldots, G_r$ and T. As suggested by the reductions in this section, it suffices to solve the Color-CL problem for $G_1, \ldots, G_r$ and T in order to be able to compute a canonical labeling of G.

3 Canonizing Biconnected Partial 2-Trees

First we recall the definition of (partial) k-trees (cf. [12]).

Definition 3. *The class of* k-trees *is inductively defined as follows.*

- *A clique with k vertices (k-clique for short) is a k-tree.*
- *Given a k-tree G' with n vertices, a k-tree G with $n+1$ vertices can be constructed by introducing a new vertex v and picking a k-clique C in G' and then joining v to each vertex u in C. Thus, $V(G) = V(G') \cup \{v\}$, $E(G) = E(G') \cup \{\{u, v\} \mid u \in C\}$.*

A partial k-tree *is a subgraph of a k-tree.*

In this section we give a logspace algorithm for biconnected partial 2-tree canonization. But first we state a useful characterization of partial 2-trees [15].

Definition 4. *Let $G = (V, E)$ be a graph. A vertex $v \in V$ is an* articulation point *if $G - v$ has more connected components than G. G is* biconnected *if it has no articulation point. A* cell *of G is a set of edges in a chordless cycle of G. The* cell-completion *of $G = (V, E)$ is the graph $\overline{G} = (V, E')$ where E' is obtained from E by adding all edges $\{x, y\} \subseteq V$ for which $G - \{x, y\}$ has at least three connected components.*

Definition 5. *A* tree of cycles *is a member of the class $\mathcal{TC}$ of graphs defined inductively as follows:*

- *Every chordless cycle is an element of $\mathcal{TC}$.*
- *Given a graph G in $\mathcal{TC}$ and a chordless cycle C, the graph obtained by identifying an edge and its two end vertices of C with an edge and its two end-vertices of G is also in $\mathcal{TC}$.*

Theorem 6. [15] *A biconnected graph G is a partial 2-tree if and only if its cell-completion $\overline{G}$ is a tree of cycles.*

We first consider COLOR-CL for colored biconnected partial 2-trees G. For this, we exploit the special structure of G as explained in Definition 5 and Theorem 6. More precisely, we now give logspace algorithms for computing the tree completion $\overline{G}$ of G and its decomposition into the "tree of cycles". This will allow us to again use ideas from Lindell's tree canonization algorithm to solve the problem.

A logspace algorithm can check for each pair $\{x, y\} \subseteq V$ if $G - \{x, y\}$ has three connected components. If this is the case and if $\{x, y\}$ is not an edge in G then the algorithm outputs $\{x, y\}$ as a "red" edge in the cell-completion $\overline{G}$. We color $\{x, y\}$ red to indicate that the edge is not present in G.

Lemma 7. *Let $G = (V, E)$ be a tree of cycles. Two distinct edges $e_1 = \{x, y\}$ and $e_2 = \{a, b\}$ are in the same cell if and only if either of the following conditions holds true:*

1. *The set $\{x, b\}$ separates y from a and the set $\{y, a\}$ separates x from b but $\{y, b\}$ does not separate x from a and $\{x, a\}$ does not separate y from b.*
2. *The set $\{x, a\}$ separates y from b and the set $\{y, b\}$ separates x from a but $\{y, a\}$ does not separate x from b and $\{x, b\}$ does not separate y from a.*

Notice that the conditions stated in Lemma 7 can be verified by querying an oracle for s-t connectivity and hence is in logspace [19]. As a consequence, for any tree of cycles G we can compute the cells $C_1, C_2, \ldots, C_m$ of G in logspace.

Definition 8. *Let $G = (V, E)$ be a tree of cycles with cells $C_1, \ldots, C_m$. The* skeleton *of G is the tree $S(G) = (V', E')$ with vertex set $V' = \{C_1, \ldots, C_m\} \cup E$ and $\{C_i, e\} \in E'$ if and only if $e \in C_i$.*

Notice that the bipartite graph $S(G) = (V', E')$ is a tree when G is a tree of cycles. For two cells C_i and C_j in a tree of cycles G we have $C_i \cap C_j = \{e\}$ precisely when $\{C_i, e\}$ and $\{C_j, e\}$ are edges in the skeleton $S(G)$. Since we can find the cells $C_1, C_2, \ldots, C_m$ of G in deterministic logspace, it follows that $S(G)$ can be computed in deterministic logspace.

Definition 9. *Let $e = \{a, b\}$ be an edge of a tree of cycles G. Suppose we orient e as the ordered pair (a, b). This orientation naturally induces an orientation of every edge on any cycle C containing e, by walking along C in the (a, b) direction. Applying this step repeatedly yields a unique orientation of all edges. We call this the* orientation *of G induced by (a, b).*

The following symmetry property clearly holds for orientations: Let $e = \{a, b\}$ and $e' = \{a', b'\}$ be two edges in a tree of cycles G. Then the orientation (a, b) induces the orientation (a', b') if and only if the orientation (a', b') induces the orientation (a, b).

Lemma 10. *Let G be a tree of cycles and let $e_0 = \{a, b\}$ be an edge in G. Then the orientation of any edge $e' = \{x, y\}$ of G induced by (a, b) can be computed in deterministic logspace.*

Proof. Let $C_1, \ldots, C_m$ be the cells of G and suppose that $e_0 \in C$ and $e' \in C'$. Let $2d$ be the distance of C' from C in the skeleton $S(G)$ of G and let $C = C_{i_0}, e_1, C_{i_1}, e_2, C_{i_2}, \ldots, e_d, C_{i_d} = C'$ be the path form C to C' in $S(G)$, where $C_{i_{t-1}} \cap C_{i_t} = \{e_t\}$ for $t = 1, \ldots, d$. If $d = 0$, then the orientation of e' induced by (a, b) is computable in logspace by traversing the nodes in the cycle $C = C'$. If $d \geq 1$, the algorithm determines the orientations of $e_1, \ldots, e_d$ one after another. Once it knows the orientation of e_d, the orientation of e' can be determined in the same way as in the case $C = C'$. Inductively suppose the algorithm knows C_{i_t} (i.e., the index i_t), e_t and the orientation of e_t induced by (a, b). In order to find $C_{i_{t+1}}$, e_{t+1} and the orientation of e_{t+1} induced by (a, b) it finds (in logspace) a vertex C'' in the skeleton $S(G)$ such that for some edge e'' of G, $C_{i_t} \cap C'' = \{e''\}$ and the unique path in $S(G)$ from C_{i_t} to C' passes through C''. Since $S(G)$ is a tree, it is clear that the unique choice for C'' is $C_{i_{t+1}}$ and that $e'' = e_{t+1}$. Also, the orientation of e_{t+1} induced by (a, b) can be determined from the orientation of e_t by traversing the nodes in the cycle C_{i_t}. □

3.1 The Tree Representation for the Tree of Cycles

Let G be a colored tree of cycles, C be a given cell in G and let $e = (a, b)$ be an oriented edge in C. Let $C_1, \ldots, C_m$ be the cells of G and let $e_1, \ldots, e_l$ be the oriented edges of G induced by (a, b). The *tree representation* of G with respect to the cell C and the oriented edge (a, b) is a colored ordered tree $T(G, C, a, b) = (V, E)$ with root C, where $V = \{C_1, \ldots C_m\} \cup \{e_1, \ldots, e_l\}$ and $E = \{\{e_i, C_j\} \mid e_i \in C_j\}$. We call the nodes $C_1, \ldots, C_m$ "cell-nodes" and the nodes $e_1, \ldots, e_l$ are referred to as "edge-nodes". Thus, we have cell-nodes and edge-nodes alternating along any root to leaf path, starting with the cell-node C as root. The coloring of $T(G, C, a, b)$ and the ordering of the children of each cell-node C_i will be described below (the children of the edge-nodes e_i are unordered). Our goal is to describe an appropriate canonical labeling (computable in logspace) of $T(G, C, a, b)$ from which we can extract (in logspace) a canonical labeling of the colored tree of cycles G (for a given cell C and oriented edge (a, b)). Once we do this, we can determine the overall canonical labeling as the one yielding the lexicographically smallest tree of cycles over all root cells C as well as all oriented edges (a, b) on C.

First, we notice certain restrictions on this tree representation enforced by the cell structure and the orientation of edges. We claim that as soon as a root cell C and an oriented edge (a, b) on C are fixed, the children of any cell-node C_i can be totally ordered in a canonical way. To see how, we start with the root C. Its children are the edge-nodes corresponding to the edges in the cycle C. We designate (a, b) as the first child of C and the remaining children are ordered by walking along C in the (a, b) direction. For any other cell-node C_i let (a_i, b_i) be its parent edge-node in $T(G, C, a, b)$. The children of C_i are the edge-nodes corresponding to the remaining edges in C_i and they are ordered by walking from node a_i along C in the (a_i, b_i) direction.

Since some edges and vertices of G may be colored, we need to add this information to the nodes of $T(G, C, a, b)$. If $\{x, y\}$ is an edge of G that is colored

c, and (x, y) is the edge-node of $T(G, C, a, b)$ corresponding to $\{x, y\}$ then we include the color c to the set of colors for edge-node (x, y). Further, for each vertex v of G having color c, the color $(1, c)$ is included in the color set of each edge-node in $T(G, C, a, b)$ of the form (v, u) and the color $(c, 1)$ is added to the color set of each edge-node of the form (u, v). This completes the description of the tree representation $T(G, C, a, b)$ of a tree of cycles G.

It is easy to see that any tree T isomorphic to $T(G, C, a, b)$ contains enough information to reconstruct the original tree of cycles G (up to isomorphism) from T. In fact, as we will show next, G is even computable in logspace from T. We first list a set of conditions that are necessary and sufficient for the existence of a tree of cycles G having T as its tree representation. For a node v in T whose children are totally ordered as $u_1, \ldots, u_{l-1}$, the *orientation-order* of the neighbors of v is defined as follows. If v is the root node r of T, then the orientation-order is $(u_1, \ldots, u_{l-1})$. If $v \neq r$, then the orientation-order is $(u_0, u_1, \ldots, u_{l-1})$, where u_0 is the parent of v. In either case we say that the neighbors of v are *color-consistent*, if u_i has a color $(c, 1)$ in its color set if and only if $succ(u_i)$ has the color $(1, c)$ in its color set, where $succ(u_i)$ denotes the cyclic successor to u_i in the orientation-order. Now it is not hard to prove the following lemma.

Lemma 11. *Let G be a tree of cycles, C be a cell of G and let (a, b) be an oriented edge on C. Then the tree representation $T = T(G, C, a, b)$ fulfills the following properties:*

1. *Every node at even distance from the root r of T has degree at least three.*
2. *The children of each node v at even distance from r are totally ordered and the neighbors of v are color-consistent with respect to their orientation-order.*
3. *The nodes at odd distance from r are colored by a set of colors of the form c, $(1, c)$ or $(c, 1)$ containing at most one color of the form c.*

Further, for each tree T fulfilling these properties, a tree of cycles G', a cell C in G' and an oriented edge (a, b) on C fulfilling $T(G', C, a, b) \cong T$ is computable in FL.

3.2 Isomorphism Ordering

We use colored tree canonization to canonize the tree representation T of a tree of cycles. For that we encode the orientation-order using special colors $c_1, c_2, \ldots$ where we assume that $c_i < c_{i+1}$. Let $(u_0, u_1, \ldots, u_{l-1})$ be the orientation-order of the neighbors of a node v in T. If v is the root r of T then we color u_i with color c_{i+1} for $i = 0, \ldots, l-1$. If $v \neq r$, then we color u_i with color c_i for $i = 1, \ldots, l-1$. Notice that in the latter case the parent u_0 of v does not get any color due to the orientation-order of the neighbors of v (though u_0 may get some color due to the orientation-order of the neighbors of some other node). We denote the (unordered) tree obtained from T by adding these special colors by $\hat{T}$.

For an ordered set C of colors let $\mathcal{T}_C$ denote the class of all rooted trees whose nodes are colored with colors from C. We denote the color of a node v by $col(v)$

and the number of its children by $\#v$. The number of nodes in a graph G is denoted as $|G|$. First we inductively define an ordering $\preceq$ on $\mathcal{T}_C$.

Definition 12. *Let $T, T' \in \mathcal{T}_C$ with roots r and r', respectively. We say that $T \preceq T'$ if either of the following conditions is fulfilled:*

1. *$col(r) < col(r')$.*
2. *$col(r) = col(r')$ and $|T| < |T'|$.*
3. *$col(r) = col(r')$, $|T| = |T'|$ and $\#r < \#r'$.*
4. *$col(r) = col(r')$, $|T| = |T'|$, $\#r = \#r'$ and $(T_{i_1}, \ldots, T_{i_k}) \preceq (T'_{j_1}, \ldots, T'_{j_k})$ in the lexicographic sense, where $T_1, \ldots, T_k$ are the subtrees rooted at the children of r, $T'_1, \ldots, T'_k$ are the subtrees rooted at the children of r', and inductively $T_{i_1} \preceq \cdots \preceq T_{i_k}$ and $T'_{j_1} \preceq \cdots \preceq T'_{j_k}$.*

For any two trees $T, T' \in \mathcal{T}_C$ at least one of $T \preceq T'$ and $T' \preceq T$ holds. Further, T' and T are isomorphic if and only if both $T \preceq T'$ and $T' \preceq T$ hold. The ordering defined in Definition 12 is similar to the one defined in [17] except that we give highest priority to colors of nodes to distinguish the subtrees rooted at them. This isomorphism ordering of colored trees gives an isomorphism ordering of the tree representations of the trees of cycles when we encode the orientation-orders by colors. We can easily modify Lindell's algorithm to take into account the priority due to coloring and get a logspace algorithm for computing the isomorphism order of colored trees in $\mathcal{T}_C$. By applying this algorithm we get a deterministic logspace algorithm for computing the isomorphism order of the tree representations of the trees of cycles.

3.3 Canonical Labeling of Biconnected Partial 2-Trees

We first explain how a colored tree $T \in \mathcal{T}_C$ is traversed using the isomorphism order defined above. The traversal starts at the root of T. Suppose we arrive at a node v. Then the algorithm tries to move to the first child in the isomorphism order, i.e., the child with minimal isomorphism order. Ties are broken using the input order. If it fails to move to the first child (if v is a leaf node), it tries to go back to the parent of v. When it comes back to the parent from a child it tries to move to the next sibling in the isomorphism order. Again ties are broken using the input order. If it fails to move to the next sibling (if it comes back from the last child) then it tries to move to the parent (if the node has no parent, the traversal stops).

To determine the canonical labeling of a tree $T \in \mathcal{T}_C$, the algorithm traverses T and when it discovers a *new* node it outputs the node. Note that a node is *new* if it comes through a successful first-child or next-sibling move. The list of nodes in the traversal order serves as the canonical labeling of T. To find the canonical labeling of the tree representation T of a tree of cycles G we first encode the orientation-order using colors and then find the canonical labeling of the colored tree as just described.

Now we can compute the canonical labeling of a tree of cycles G as follows. The algorithm computes for any cell C in G and each oriented edge (a, b) of C

the tree representation $T(G, C, a, b)$. Then it determines the canonical labeling for $\hat{T}(G, C, a, b)$ and reconstructs from the relabeled tree representation in deterministic logspace a tree of cycles G' as stated in Lemma 11. G' is our canonical form $canon(G, C, a, b)$ of G with respect to C and (a, b). The canonical form of G is the lexicographically least such canon.Notice that also the canonical labeling can be recovered by keeping track of the original vertices from the canonical labeling of $\hat{T}(G, C, a, b)$.

Clearly, the canonical labeling of the cell-completion $\overline{G}$ of a biconnected partial 2-tree G can serve as a canonical labeling of G because in $\overline{G}$ the edges that are not present in G are colored "red".

Theorem 13. *For colored biconnected partial* 2*-trees a canonical labeling can be computed in logspace.*

4 Canonizing Partial 2-Trees

In this section we show that the problem Color-CL for partial 2-trees is in logspace. We show this by designing a logspace canonical labeling algorithm for partial 2-trees that is a combination of Lindell's tree canonization algorithm with the logspace canonization algorithm of Section 3 for the class of biconnected partial 2-trees.

Theorem 14. *The problem* Color-CL *for partial* 2*-trees is in logspace. I.e. colored partial* 2*-trees are canonizable in logspace.*

Proof (sketch). Let $G = (V, E)$ be a colored input partial 2-tree for our canonization algorithm. Using Reingold's logspace s-t connectivity algorithm for undirected graphs [19] we can compute its biconnected components $G_1, \ldots, G_r$ and the set A of its articulation points. Consider the tree $T(G) = (V', E')$ of biconnected components and articulation points defined as follows: $V' = \{v_1, \ldots, v_r\} \cup A$, where v_i corresponds to G_i for each i and $E' = \{\{a, v_i\} \mid a \text{ is an articulation point in } G_i\}$. We can assume that $G, G_1, \ldots, G_r$ and $T(G)$ are given as input.

By distinguishing some articulation point $a \in A$ as the root, the tree $T(G)$ becomes a rooted tree $T(G, a)$ with root a, defined as follows. Based on Lindell's canonization method [17], given G with a vertex $v \in V$ specially marked as root, in the following steps we describe an inductive definition of an ordering on such rooted trees $T(G)$. Using this inductive definition we will be able to compute a canonical labeling (and hence a canonical form) of G. We use $\#r(v)$ to denote the number of children of the root $r(v)$ in $T(G, v)$. Further, for a node $v' \in V'$ we denote the subgraph of G corresponding to the subtree of $T(G, v)$ originating from node v' by $G(v')$. In case $v' = v_i$ corresponds to some biconnected component G_i, we assume that the parent $a_i \in A$ of v_i in $T(G, v)$ (if it exists) is colored with the special color "red" in $G(v_i)$.

Let $G, G' \in \mathcal{G}$ with vertices v and v' marked as root, respectively. We define $G \preceq G'$ if one of the following conditions holds:

1. $size(G) < size(G')$.
2. $size(G) = size(G')$ and $\#r(v) < \#r(v')$.
3. $size(G) = size(G')$, $\#r(v) = \#r(v')$ and v is an articulation point in G but v' is not an articulation point in G'.
4. $size(G) = size(G')$, $\#r(v) = \#r(v')$, either both or neither of v and v' are articulation points in G and G', respectively, and $(G(v_1), \ldots, G(v_s)) \prec (G'(v'_1), \ldots, G'(v'_s))$ in the lexicographic sense ($G \prec G'$ means that $G \preceq G'$ holds but not $G' \preceq G$), where $v_1, \ldots, v_s$ and $v'_1, \ldots, v'_s$ are the children of $r(v)$ and $r(v')$ in $T(G, v)$ and $T(G', v')$, respectively, and the corresponding graphs are inductively ordered as $G(v_1) \preceq \cdots \preceq G(v_s)$ and $G'(v'_1) \preceq \cdots \preceq G'(v'_s)$.
5. If in item 4 we have $G(v_i) \approx G'(v'_i)$ for $i = 1, \ldots, s$ (where $G \approx G'$ means that both $G \preceq G'$ and $G' \preceq G$ hold) and if $r(v)$ and $r(v')$ correspond to biconnected components G_i and G'_j of G and G', respectively, then let $a_1, \ldots, a_s$ and $a'_1, \ldots, a'_s$ be the children of $r(v)$ and $r(v')$ in $T(G, v)$ and $T(G', v')$, respectively. Inductively find the indices $0 = i_0 < i_1 < \cdots < i_{k-1} < i_k = s$ such that $G(a_1) \approx \cdots \approx G(a_{i_1}) \prec G(a_{i_1+1}) \approx \cdots \approx G(a_{i_2}) \prec G(a_{i_2+1}) \approx \cdots \approx G(a_{i_k}) \prec G(a_{i_k+1}) \approx \cdots \approx G(a_s)$. In the biconnected component G_i color v with the special color "red" and color the articulation point a_i with color r if and only if $i_{r-1} + 1 \leq i \leq i_r$. Similarly color the vertices $v', a'_1, \ldots, a'_s$ in G'_j. Now canonize the colored biconnected graphs G_i and G'_j (using the oracle for canonizing graphs in $\mathcal{B}(\mathcal{G})$) and if $G_i \preceq G'_j$ then order $G \preceq G'$.

We claim that this inductive ordering defines a canonical form for the graph G. It remains to argue that the canonical form can be computed in logspace, basically using Lindell's algorithm. As shown in Section 3, we can compute a tree representation for biconnected partial 2-trees. If G is a partial 2-tree with more than one biconnected component, then in the rooted tree defined above each biconnected component B of G will have a fixed red node r (which is an articulation point of G). Using this property, we can give a modified tree representation $T(B, r)$ for B in which r is the root, its children are the cells of B containing r, for each such cell C its children will be an ordered list of *vertices* (where the ordering is either forward or backward determined by orienting a fixed edge incident on r), and so on. The vertices of B that are other articulation points of G will be initially colored blue. In the overall tree structure, these blue nodes will have as children the red nodes which are roots of the tree representation of other biconnected components. An overview of the algorithm is now as follows: We start Lindell's algorithm on the tree $T(G, a)$ rooted at articulation point a. Every time we need to compare two trees rooted at two biconnected components as in Step 5 of the inductive definition, we will essentially have to compare two trees of the form $T(B, r)$ and $T(B', r')$. We will again use Lindell's method here. We classify the blue nodes into blocks, using the level numbers and the sizes of the subgraphs rooted at them. We first recusively compare nodes that are identical blocks of size 1 (crucially, we do not need any storage to do this recursion similar to Lindell) and order $T(B,r)$ and $T(B',r')$ as per this outcome. If all single blocks turn out to have same isomorphism order, as recursively computed, we proceed to run Lindell on the trees $T(B,r)$ and $T(B',r')$, where we need to orient an edge on roots r and r' and remember the orientation (in all 2 bits of space). The rest of the computation proceeds exactly as in Lindell, where blocks of larger size are recursively compared and the space is managed to be logarithmic. Finally, notice that to canonize G we can run the above procedure to canonize $T(G,a)$ for each articulation point a and choose the lexicographically smallest amongst them. Details of the algorithm and correctness proof will be given in the full version. □

5 Recognizing Partial 2-Trees

Jakoby and Liskiewicz [13] proved that recognition of partial 2-trees is in SL. Hence, it is in L by Reingold's result [19]. Here we give a simple logspace algorithm for recognizing partial 2-trees is based on Theorem 6 and Lemma 7. By Theorem 6 it suffices to check that the cell-completion of each biconnected component B of G is a tree of cycles. Clearly, the cell-completion $\overline{B}$ of B can be computed in logspace as described in Section 3. In order to check that $\overline{B} = (V,E)$ is a tree of cycles, the algorithm computes for each edge $e = \{x,y\}$ in $\overline{B}$ a set C_e consisting of all edges $e' = \{a,b\}$ for which either of the two conditions stated in Lemma 7 is satisfied. Next it removes all duplicate occurrences of the sets C_e. Let $C_1, \dots, C_m$ be the remaining sets. If $|C_i \cap C_j| > 1$ for some $1 \le i < j \le m$, then we know that G cannot be a tree of cycles. Otherwise the algorithm checks for $i = 1, 2, \dots, m$ that the graph $G[V_i]$ induced by the vertex set V_i of C_i is actually a cycle by verifying that each node has degree 2 and the graph $G[V_i]$ is connected. Finally, the algorithm checks that the bipartite graph $S(\overline{B}) = (V', E')$ is a tree, where $V' = \{C_1, \dots, C_m\} \cup E$ and $E' = \{\{C_i, e\} \mid e \in C_i\}$. Recall from Definition 8 that $S(\overline{B})$ is just the *skeleton* of $\overline{B}$ in case $\overline{B}$ is a tree of cycles. Now it is easy to see that G is a partial 2-tree if and only if the cell-completion $\overline{B}$ of each biconnected component B of G passes all the tests described above.

Theorem 15. *Given a graph G there is a deterministic logspace algorithm to decide if G is a partial 2-tree.*

References

1. Arvind, V., Das, B., Köbler, J.: The space complexity of k-tree isomorphism. In: Proc. 18th International Symposium on Algorithms and Computation. LNCS, pp. 822–833. Springer, Heidelberg (2007)
2. Allender, E.: Arithmetic circuits and counting complexity classes. In: Krajíček, J. (ed.) Complexity of Computations and Proofs. Seconda Universita di Napoli. Quaderni di Matematica, vol. 13, pp. 33–72 (2004)
3. Allender, E., Ogihara, M.: Relationships among PL, #L and the determinant. R.A.I.R.O. Informatique Théorique et Applications 30(1), 1–21 (1996)

4. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23(2), 11–24 (1989)
5. Arnborg, S., Proskurowski, A.: Canonical representations of partial 2- and 3-trees. BIT 32(2), 197–214 (1992)
6. Babai, L., Erdős, P., Selkow, S.M.: Random graph isomorphism. SIAM Journal on Computing 9(3), 628–635 (1980)
7. Babai, L., Kučera, L.: Canonical labelling of graphs in linear average time. In: Proc. 20th IEEE Symposium on the Foundations of Computer Science, pp. 39–46 (1979)
8. Babai, L., Luks, E.: Canonical labeling of graphs. In: Proc. 15th ACM Symposium on Theory of Computing, pp. 171–183 (1983)
9. Gurevich, Y.: From invariants to canonization. Bulletin of the European Association of Theoretical Computer Science (BEATCS) 63, 115–119 (1997)
10. Grohe, M., Verbitsky, O.: Testing graph isomorphism in parallel by playing a game. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 3–14. Springer, Heidelberg (2006)
11. Hoffmann, C.M.: Group-Theoretic Algorithms and Graph Isomorphism. LNCS, vol. 136. Springer, Heidelberg (1982)
12. Harary, F., Palmer, E.M.: On acyclic simplicial complexes. Mathematica 15, 115–122 (1968)
13. Jakoby, A., Liskiewicz, M.: Paths Problems in Symmetric Logarithmic Space. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 269–280. Springer, Heidelberg (2002)
14. Jenner, B., Köbler, J., McKenzie, P., Torán, J.: Completeness results for graph isomorphism. Journal of Computer and System Sciences 66, 549–566 (2003)
15. Kloks, T.: Treewidth: Computation and Approximation. LNCS, vol. 842. Springer, Heidelberg (1994)
16. Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: Its Structural Complexity. In: Progress in Theoretical Computer Science. Birkhäuser, Boston (1993)
17. Lindell, S.: A logspace algorithm for tree canonization. In: Proc. 24th ACM Symposium on Theory of Computing, pp. 400–404. ACM Press, New York (1992)
18. Luks, E.M.: Permutation groups and polynomial time computations. In: Finkelstein, L., Kantor, W.M. (eds.) Groups and Computation. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 11, pp. 139–175. American Mathematical Society (1993)
19. Reingold, O.: Undirected ST-connectivity in log-space. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 376–385 (2005)
20. Thierauf, T., Wagner, F.: The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. Technical Report TR07-068, Electronic Colloquium on Computational Complexity (ECCC) (2007)

A Triple Correspondence in Canonical Calculi: Strong Cut-Elimination, Coherence, and Non-deterministic Semantics

Arnon Avron and Anna Zamansky

School of Computer Science, Tel-Aviv University

Abstract. An (n, k)-ary quantifier is a generalized logical connective, binding k variables and connecting n formulas. Canonical systems with (n, k)-ary quantifiers form a natural class of Gentzen-type systems which in addition to the standard axioms and structural rules have only logical rules in which exactly one occurrence of a quantifier is introduced. The semantics for these systems is provided using two-valued non-deterministic matrices, a generalization of the classical matrix. In this paper we use a constructive syntactic criterion of coherence to characterize strong cut-elimination in such systems. We show that the following properties of a canonical system G with arbitrary (n, k)-ary quantifiers are equivalent: (i) G is coherent, (ii) G admits strong cut-elimination, and (iii) G has a strongly characteristic two-valued generalized non-deterministic matrix.

1 Introduction

The possibility to eliminate cuts is a crucial property of useful sequent calculi. This property was first established by Gentzen (in his classical [10]) for sequent calculi for classical and intuitionistic first-order logic. Since then many other cut-elimination theorems, for many systems, have been proved by various methods[1]. Now showing that a given sequent calculus admits cut-elimination is a difficult task, often carried out using heavy syntactic arguments and based on many case-distinctions. It is thus important to have some useful criteria that *characterize* cut-elimination (i.e., conditions which are both necessary and sufficient for having an appropriate cut-elimination theorem).

In this paper we give a constructive characterization of a general strong form of cut-elimination for a very natural class of Gentzen-type systems called *canonical systems*. These are systems which in addition to the standard axioms and structural rules have only logical introduction rules of the ideal type, in which exactly one occurrence of a connective or quantifier is introduced, and no other connective or quantifier is mentioned in the formulation of the rule.

[1] We note that by 'cut-elimination' we mean here just the *existence* of proofs without (certain forms of) cuts, rather than an algorithm to transform a given proof to a cut-free one (for the assumptions-free case the term "cut-admissibility" is sometimes used, but this notion is too weak for our purposes).

For the propositional case canonical systems were first introduced and investigated in [2]. There semantics for such systems was provided using two-valued non-deterministic matrices (called 2Nmatrices)[2]. It was shown in [2] that for propositional canonical systems there is an exact *triple correspondence* between cut-elimination, the existence of a characteristic 2Nmatrix for them, and a constructive syntactic property called *coherence*. In [17] the same *triple correspondence* was shown to hold also for canonical systems with unary quantifiers. The next natural stage was to consider languages and systems with arbitrary (n, k)-ary quantifiers. By an (n, k)-ary quantifier ([13,15]) we mean a generalized logical connective, which binds k variables and connects n formulas. In particular, any n-ary propositional connective is an $(n, 0)$-ary quantifier, the unary quantifiers considered in [17] (including the standard first-order quantifiers $\exists$ and $\forall$) are $(1, 1)$-quantifiers, bounded universal and existential quantifiers used in syllogistic reasoning ($\forall x(p(x) \to q(x))$ and $\exists x(p(x) \wedge q(x))$) are (2,1)-ary quantifiers, while the simplest Henkin quantifier[3] $\mathcal{Q}^H$ is a (1,4)-quantifier:

$$\mathcal{Q}^H x_1x_2y_1y_2\psi(x_1, x_2, y_1, y_2) := \begin{matrix}\forall x_1 & \exists y_1 \\ \forall x_2 & \exists y_2\end{matrix}\,\psi(x_1, x_2, y_1, y_2)$$

The first steps in investigating canonical systems with (n, k)-ary quantifiers were taken in [4]. There the semantics of 2Nmatrices and the coherence criterion were extended to languages with $(n, 1)$-ary quantifiers. Then it was shown that coherence is equivalent in this case to the existence of a characteristic 2Nmatrix, and it implies (but is not equivalent to) cut-elimination. When canonical systems are generalized to languages with arbitrary (n, k)-ary quantifiers, two serious problems emerge. The first problem is that the semantics of 2Nmatrices employed in [17,4] is no longer adequate in case $k > 1$. The second problem is that even in case of $k = 1$, coherence is not a necessary condition for standard cut-elimination, and so the *triple correspondence* seems to be lost. Now the first problem was solved in [3] by introducing *generalized* two-valued Nmatrices (2GNmatrices), where a more complex approach to quantification is used. However, the problem of finding an appropriate form of cut-elimination for which coherence *is* a necessary condition, and reestablishing the *triple correspondence* was explicitly left open in [4] and then also in [3].

In this paper we provide a full solution to the second problem[4]. This is achieved in two main steps. The first is to include *substitution* as one of the

[2] Non-deteministic matrices form a natural generalization of ordinary matrices, in which the value assigned by a valuation to a complex formula can be chosen non-deterministically out of a certain nonempty set of options.

[3] It should be noted though that the canonical systems with (n, k)-ary quantifiers studied in this paper are still not sufficient for treating Henkin quantifiers, as the representation language is not expressive enough to capture dependencies between variables; one direction is extending the representation language with function symbols, which would lead to the inevitable loss of the decidability of coherence.

[4] A partial solution for the restricted case $k = 1$ is already provided in [4].

structural rules of a canonical system[5]. The second step is to extend cut-elimination to *deduction from assumptions* (following [1,16]). For historical reasons, reasoning with non-logical assumptions (in the form of sequents) is usually reduced to pure provability of sequents. However, this is not always possible (the resolution calculus, primitive recursive arithmetics, pure equality reasoning, and disjunctive databases are four cases in point). Even when such reduction is possible, it is not necessarily desirable, as the case of first-order logic with equality shows (see e.g. [16]). Thus it is in fact very natural to investigate and characterize cut-elimination in the context of deduction from assumptions.

With the aid of the above two steps we again establish an exact *triple correspondence* between coherence of canonical systems with arbitrary (n, k)-ary quantifiers, their 2GNmatrices-based semantics, and a strong form of cut-elimination for them. More specifically, we show that the following properties of a canonical system G are equivalent: (i) G is coherent, (ii) G admits strong cut-elimination, and (iii) G has a strongly[6] characteristic 2GNmatrix.

2 Preliminaries

In what follows, L is a language with (n, k)-ary quantifiers, that is with quantifiers $\mathcal{Q}_1, ..., \mathcal{Q}_m$ with arities (n_1, k_1), ..., (n_m, k_m) respectively. For any $n > 0$ and $k \geq 0$, if a quantifier $\mathcal{Q}$ in a language L is of arity (n, k), then $\mathcal{Q}x_1...x_k(\psi_1, ..., \psi_n)$ is an L-formula whenever $x_1, ..., x_k$ are distinct variables and $\psi_1, ..., \psi_n$ are formulas of L. Denote by Frm_L (Frm_L^{cl}) the set of L-formulas (closed L-formulas). Denote by Trm_L (Trm_L^{cl}) the set of L-terms (closed L-terms). We write $\mathcal{Q}\overrightarrow{x}A$ instead of $\mathcal{Q}x_1...x_kA$, and $\psi\{\overrightarrow{\mathbf{t}}/\overrightarrow{z}\}$ instead of $\psi\{\mathbf{t}_1/z_1, ..., \mathbf{t}_k/z_k\}$.

A set of sequents $\mathcal{S}$ satisfies the *free-variable condition* if the set of variables occurring bound in $\mathcal{S}$ is disjoint from the set of variables occurring free in $\mathcal{S}$.

In the following two subsections, we briefly reproduce the relevant definitions from [4,3] of canonical rules with (n, k)-ary quantifiers and of the framework of non-deterministic matrices. Note the important addition of the definition of *full canonical systems*, which include the rule of substitution.

2.1 Full Canonical Systems with (n, k)-ary Quantifiers

We use the simplified representation language from [4,3] for a schematic representation of canonical rules.

Definition 2.1. *For $k \geq 0$, $n \geq 1$, L_k^n is the language with n k-ary predicate symbols $p_1, ..., p_n$, the set of constants $Con = \{c_1, c_2, ...,\}$ and the set of variables $Var = \{v_1, v_2, ...,\}$.*

In this paper we assume for simplicity that L_k^n and L share their sets of variables and constants.

[5] See [1] for the general need for this step, e.g. for the foundations of the resolution calculus, and for reasoning from assumptions in general.

[6] See Defn. 2.15 below.

Definition 2.2. *A* canonical rule of arity (n, k) *has the form* $\{\Pi_i \Rightarrow \Sigma_i\}_{1 \leq i \leq m}/C$*, where* $m \geq 0$*,* C *is either* $\Rightarrow \mathcal{Q}v_1...v_k(p_1(v_1,...,v_k),...,p_n(v_1,...,v_k))$ *or* $\mathcal{Q}v_1...v_k(p_1(v_1,...,v_k),...,p_n(v_1,...,v_k)) \Rightarrow$ *for some* (n, k)*-ary quantifier* $\mathcal{Q}$ *of* L *and for every* $1 \leq i \leq m$*:* $\Pi_i \Rightarrow \Sigma_i$ *is a clause*[7] *over* L_k^n*.*

For a specific application of a canonical rule we need to instantiate the schematic variables by the terms and formulas of L. This is done using a mapping function:

Definition 2.3. *Let* $R = \Theta/C$ *be an* (n, k)*-ary canonical rule, where* C *is of one of the forms* $(\mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}),...,p_n(\overrightarrow{v})) \Rightarrow)$ *or* $(\Rightarrow \mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}),...,p_n(\overrightarrow{v})))$*. Let* Γ *be a set of* L*-formulas and* $z_1,...,z_k$ *- distinct variables of* L*. An* $\langle R, \Gamma, z_1,...,z_k\rangle$*-*mapping *is any function* χ *from the predicate symbols, terms and formulas of* L_k^n *to formulas and terms of* L*, satisfying the following conditions:*

- *For every* $1 \leq i \leq n$*,* $\chi[p_i]$ *is an* L*-formula.* $\chi[y]$ *is a variable of* L*, and* $\chi[x] \neq \chi[y]$ *for every two variables* $x \neq y$*.* $\chi[c]$ *is an* L*-term, such that* $\chi[x]$ *does not occur in* $\chi[c]$ *for any variable* x *occurring in* Θ*.*
- *For every* $1 \leq i \leq n$*, whenever* $p_i(\boldsymbol{t}_1,...,\boldsymbol{t}_k)$ *occurs in* Θ*, for every* $1 \leq j \leq k$*:* $\chi[\boldsymbol{t}_j]$ *is a term free for* z_j *in* $\chi[p_i]$*, and if* $\boldsymbol{t}_j$ *is a variable, then* $\chi[\boldsymbol{t}_j]$ *does not occur free in* $\Gamma \cup \{\mathcal{Q}z_1...z_k(\chi[p_1],...,\chi[p_n])\}$*.*
- $\chi[p_i(\boldsymbol{t}_1,...,\boldsymbol{t}_k)] = \chi[p_i]\{\chi[\boldsymbol{t}_1]/z_1,...,\chi[\boldsymbol{t}_k]/z_k\}$*.*

χ *is extended to sets of* L_k^n*-formulas as follows:* $\chi[\Delta] = \{\chi[\psi] \mid \psi \in \Delta\}$*.*

Definition 2.4. *Let* $R = \Theta/C$ *be an* (n, k)*-ary canonical rule, where* $\Theta = \{\Pi_i \Rightarrow \Sigma_i\}_{1 \leq i \leq m}$ *and* C *has the form* $\mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}),...,p_n(\overrightarrow{v})) \Rightarrow$*. An* application *of* R *is any inference step of the form:*

$$\frac{\{\Gamma, \chi[\Pi_i] \Rightarrow \Delta, \chi[\Sigma_i]\}_{1 \leq i \leq m}}{\Gamma, \mathcal{Q}z_1...z_k\ (\chi[p_1],...,\chi[p_n]) \Rightarrow \Delta}$$

where $z_1,...,z_k$ *are variables,* Γ, Δ *are any sets of* L*-formulas and* χ *is some* $\langle R, \Gamma \cup \Delta, z_1,...,z_k\rangle$*-mapping.*
An application of a canonical rule of the form Θ/C' *there* C' *has the form* $\Rightarrow \mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}),...,p_n(\overrightarrow{v}))$ *is defined similarly.*

Example 2.5. *The standard introduction rules for the* $(1, 1)$*-ary quantifier* $\forall$ *can be formulated as follows:* $\{p(c) \Rightarrow\}/\forall v_1\, p(v_1) \Rightarrow$ *and* $\{\Rightarrow p(v_1)\}/\Rightarrow \forall v_1\, p(v_1)$*. Applications of these rules have the forms:*

$$\frac{\Gamma, A\{\boldsymbol{t}/w\} \Rightarrow \Delta}{\Gamma, \forall w\, A \Rightarrow \Delta}\ (\forall \Rightarrow) \qquad \frac{\Gamma \Rightarrow A\{z/w\}, \Delta}{\Gamma \Rightarrow \forall w\, A, \Delta}\ (\Rightarrow \forall)$$

where z *is free for* w *in* A*,* z *is not free in* $\Gamma \cup \Delta \cup \{\forall w A\}$*, and* $\boldsymbol{t}$ *is any term free for* w *in* A*.*

[7] By a clause we mean a sequent containing only atomic formulas.

Notation 2.6 (Following [2,4]). Let $-t = f, -f = t$. Let $ite(t, A, B) = A$ and $ite(f, A, B) = B$. Let Φ, A^s (where Φ may be empty) denote $ite(s, \Phi \cup \{A\}, \Phi)$. For instance, the sequents $A \Rightarrow$ and $\Rightarrow A$ are denoted by $A^{-a} \Rightarrow A^a$ for $a = f$ and $a = t$ respectively. With this notation, an (n, k)-ary canonical rule has the form $\{\Sigma_j \Rightarrow \Pi_j\}_{1\leq j\leq m}/\mathcal{Q}\overrightarrow{z}(p_1(\overrightarrow{z}), ..., p_n(\overrightarrow{z}))^{-s} \Rightarrow \mathcal{Q}\overrightarrow{z}(p_1(\overrightarrow{z}), ..., p_n(\overrightarrow{z}))^s$ for some $s \in \{t, f\}$. For further abbreviation, we denote such rule by $\{\Sigma_j \Rightarrow \Pi_j\}_{1\leq j\leq m}/\mathcal{Q}(s)$.

Definition 2.7. *A* full canonical calculus *G is a Gentzen-type system, which consists of (i) The α-axiom $\psi \Rightarrow \psi'$ for $\psi \equiv_\alpha \psi'$, (ii) The standard structural rules with the addition of the substitution rule, and (iii) Canonical inference rules.*

The *coherence* criterion used in [3,4] can be straightforwardly extended to full canonical calculi:

Definition 2.8. *For two sets of clauses Θ_1, Θ_2 over L^n_k, $\mathsf{Rnm}(\Theta_1 \cup \Theta_2)$ is a set $\Theta_1 \cup \Theta'_2$, where Θ'_2 is obtained from Θ_2 by a fresh renaming of constants and variables which occur in Θ_1.*

Definition 2.9. (Coherence[8]) *A full canonical calculus G is* coherent *if for every two canonical rules of the form $\Theta_1/\Rightarrow A$ and $\Theta_2/A \Rightarrow$, the set of clauses $\mathsf{Rnm}(\Theta_1 \cup \Theta_2)$ is classically inconsistent.*

Example 2.10. *Consider the calculus G_1 consisting of the rules $\Theta_1/\forall v_1\, p(v_1) \Rightarrow$ and $\Theta_2/\Rightarrow \forall v_1\, p(v_1)$ where $\Theta_1 = \{p(c) \Rightarrow\}$ and $\Theta_2 = \{\Rightarrow p(v_1)\}$ (these rules are from Example 2.5). $\mathsf{Rnm}(\Theta_1 \cup \Theta_2) = \{p(c) \Rightarrow, \Rightarrow p(v_1)\}$ (note that no renaming is needed here) is clearly classically inconsistent and so G_1 is coherent.*

Proposition 2.11. (Decidability of coherence) *The coherence of a full canonical calculus is decidable.*

2.2 Generalized Non-deterministic Matrices

Our main semantic tool in this paper are *generalized non-deterministic matrices* introduced in [3], which are a generalization of non-deterministic structures used in [2,17,4].

Definition 2.12. *A* generalized non-deterministic matrix *(henceforth GNmatrix) for L is a tuple $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where: (i) $\mathcal{V}$ is a non-empty set of truth values, (ii) $\mathcal{D}$ is a non-empty proper subset of $\mathcal{V}$, and (iii) For every (n, k)-ary quantifier $\mathcal{Q}$ of L, $\mathcal{O}$[9] includes a corresponding operation $\tilde{\mathcal{Q}}_S : (D^k \to \mathcal{V}^n) \to P^+(\mathcal{V})$ for every L-structure $S = \langle D, I \rangle$.*

A 2GNmatrix is any GNmatrix with $\mathcal{V} = \{t, f\}$ and $\mathcal{D} = \{t\}$.

[8] The coherence criterion was first introduced in [2]. A related criterion also called coherence was later used in [14], where linear logic is used to specify and reason about a number of sequent systems.

[9] Strictly speaking, the tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ is not well-defined, since $\mathcal{O}$ is a proper class. Since all our results remain intact if we concentrate only on countable models, this technical problem can be overcome by assuming that the domains of all the structures are prefixes of the set of natural numbers.

The notion of an L-structure is defined standardly (see, e.g. [4]). In order to interpret quantifiers, the substitutional approach is used, which assumes that every element of the domain has a term referring to it. Thus given an L-structure $S = \langle D, I \rangle$, the language L is extended with *individual constants*: $\{\overline{a} \mid a \in D\}$. Call the extended language $L(D)$. The interpretation function I is extended to $L(D)$ as follows: $I[\overline{a}] = a$. An S-substitution σ is any function from variables to $Trm^{cl}_{L(D)}$. For an S-substitution σ and a term $\mathbf{t}$ (a formula ψ), the closed term $\sigma[\mathbf{t}]$ (the sentence $\sigma[\psi]$) is obtained from $\mathbf{t}$ (ψ) by substituting every variable x for $\sigma[x]$. We write[10] $\psi \sim^S \psi'$ if ψ' can be obtained from ψ by renamings of bound variables and by any number of substitutions of a closed term $\mathbf{t}$ for another closed term $\mathbf{s}$, so that $I[\mathbf{t}] = I[\mathbf{s}]$.

Definition 2.13. *Let $S = \langle D, I \rangle$ be an L-structure for a GNmatrix $\mathcal{M}$. An S-valuation $v : Frm^{\mathsf{cl}}_{L(D)} \to \mathcal{V}$ is* legal *in $\mathcal{M}$ if it satisfies the following conditions: $v[\psi] = v[\psi']$ for every two sentences ψ, ψ' of $L(D)$, such that $\psi \sim^S \psi'$, $v[p(\boldsymbol{t}_1, ..., \boldsymbol{t}_n)] = I[p][I[\boldsymbol{t}_1], ..., I[\boldsymbol{t}_n]]$, and $v[\mathcal{Q}x_1, ..., x_k(\psi_1, ..., \psi_n)$ is in the set $\tilde{\mathcal{Q}}_S[\lambda a_1, ..., a_k \in D.\langle v[\psi_1\{\overline{a}_1/x_1, ..., \overline{a}_k/x_k\}], ..., v[\psi_n\{\overline{a}_1/x_1, ..., \overline{a}_k/x_k\}]\rangle]$ for every (n, k)-ary quantifier $\mathcal{Q}$ of L.*

Definition 2.14. *Let $S = \langle D, I \rangle$ be an L-structure for an GNmatrix $\mathcal{M}$. An $\mathcal{M}$-legal S-valuation v is* a model of a sentence ψ in $\mathcal{M}$, *denoted by $S, v \models_{\mathcal{M}} \psi$, if $v[\psi] \in \mathcal{D}$. For an $\mathcal{M}$-legal S-valuation v, a sequent $\Gamma \Rightarrow \Delta$ is* $\mathcal{M}$-valid in $\langle S, v \rangle$ *if for every S-substitution σ: whenever $S, v \models_{\mathcal{M}} \sigma[\psi]$ for every $\psi \in \Gamma$, there is some $\varphi \in \Delta$, such that $S, v \models_{\mathcal{M}} \sigma[\varphi]$. A sequent $\Gamma \Rightarrow \Delta$ is $\mathcal{M}$-valid, denoted by $\vdash_{\mathcal{M}} \Gamma \Rightarrow \Delta$, if for every L-structure S and every $\mathcal{M}$-legal S-valuation v, $\Gamma \Rightarrow \Delta$ is $\mathcal{M}$-valid in $\langle S, v \rangle$. For a set of sequents $\mathcal{S}$, $\mathcal{S} \vdash_{\mathcal{M}} \Gamma \Rightarrow \Delta$ if for every L-structure S and every $\mathcal{M}$-legal S-valuation v: whenever the sequents of $\mathcal{S}$ are $\mathcal{M}$-valid in $\langle S, v \rangle$, $\Gamma \Rightarrow \Delta$ is also $\mathcal{M}$-valid in $\langle S, v \rangle$.*

Definition 2.15. *A system G is* strongly sound *for a GNmatrix $\mathcal{M}$ if for every set of sequents $\mathcal{S}$: $\mathcal{S} \vdash_G \Gamma \Rightarrow \Delta$ entails $\mathcal{S} \vdash_{\mathcal{M}} \Gamma \Rightarrow \Delta$. A system G is* strongly complete *for a GNmatrix $\mathcal{M}$ if for every set of sequents $\mathcal{S}$: $\mathcal{S} \vdash_{\mathcal{M}} \Gamma \Rightarrow \Delta$ entails $\mathcal{S} \vdash_G \Gamma \Rightarrow \Delta$. A GNmatrix $\mathcal{M}$ is* strongly characteristic *for G if G is strongly sound and strongly complete for $\mathcal{M}$.*

Note that strong soundness implies (weak) soundness[11]. A similar remark applies to completeness and a characteristic GNmatrix.

In addition to L-structures for languages with (n, k)-ary quantifiers, we will also use L^n_k-structures for the simplified languages L^n_k, using which the canonical rules are formulated. To make the distinction clearer, we shall use the metavariable S for the former and $\mathcal{N}$ for the latter. Since the formulas of L^n_k are always

[10] The motivation for this definition, in addition to capturing α-equivalence, is purely technical and is related to extending the language with the set of individual constants $\{\overline{a} \mid a \in D\}$. Suppose we have a closed term $\mathbf{t}$, such that $I[\mathbf{t}] = a \in D$. But a also has an individual constant $\overline{a}$ referring to it. We would like to be able to substitute $\mathbf{t}$ for $\overline{a}$ in every context.

[11] A system G is (weakly) sound for a GNmatrix $\mathcal{M}$ if $\vdash_G \Gamma \Rightarrow \Delta$ entails $\vdash_{\mathcal{M}} \Gamma \Rightarrow \Delta$.

atomic, the specific 2GNmatrix for which $\mathcal{N}$ is defined is immaterial, and can be omitted. Henceforth we may speak simply of validity of sets of sequents over L_k^n.

Definition 2.16. *Let $\mathcal{N} = \langle D, I\rangle$ be a structure for L_k^n. The* functional distribution *of $\mathcal{N}$ is a function $FDist_{\mathcal{N}} \in D^k \to \{t, f\}^n$, such that: $FDist_{\mathcal{N}} = \lambda a_1, ..., a_k \in D.\langle I[p_1][a_1, ..., a_k], ..., I[p_n][a_1, ..., a_k]\rangle$.*

3 The Triple Correspondence

In this section we establish an exact *triple correspondence* between the coherence of full canonical systems, their 2GNmatrices-based semantics and *strong cut-elimination*, a version of cut-elimination for deduction with assumptions taken from [1]:

Definition 3.1. *Let G be a full canonical calculus and let $\mathcal{S}$ be some set of sequents. A proof P of $\Gamma \Rightarrow \Delta$ from $\mathcal{S}$ in G is $\mathcal{S}$-*cut-free *if all cuts in P are on substitution instances of formulas from $\mathcal{S}$.*

Definition 3.2. *A Gentzen-type calculus G admits* strong cut-elimination *if for every set of sequents $\mathcal{S}$ and every sequent $\Gamma \Rightarrow \Delta$, such that $\mathcal{S} \cup \{\Gamma \Rightarrow \Delta\}$ satisfies the free-variable condition it holds that if $\mathcal{S} \vdash_G \Gamma \Rightarrow \Delta$, then $\Gamma \Rightarrow \Delta$ has an $\mathcal{S}$-cut-free proof in G.*

Note that strong cut-elimination implies standard cut-elimination (which corresponds to the case of an empty set $\mathcal{S}$).

Remark: At this point the importance of the substitution rule should be stressed. Consider for instance the canonical calculus with two standard $(1, 1)$-ary rules for $\forall$ from Example 2.5. Consider the following deduction:

$$\dfrac{\dfrac{\Rightarrow p(x)}{\Rightarrow \forall x p(x)}\,(\Rightarrow \forall) \qquad \dfrac{p(c) \Rightarrow}{\forall x p(x) \Rightarrow}\,(\forall \Rightarrow)}{\Rightarrow}\,(Cut)$$

The above application of Cut can only be eliminated using an explicit application of the substitution rule. An alternative, less satisfactory solution (instead of adding the substitution rule to canonical calculi explicitly,) would be considering only sets of non-logical assumptions, which are closed under substitution[12].

In [3] a strongly sound and (weakly) complete 2GNmatrix $\mathcal{M}_G$ is defined for every coherent canonical calculus G. This can be straightforwardly extended to full canonical calculi. We strengthen this result in the sequel for full canonical calculi by showing that $\mathcal{M}_G$ is also *strongly* complete for G.

Definition 3.3. *Let G be a coherent full canonical calculus. For every L-structure $S = \langle D, I\rangle$, the GNmatrix $\mathcal{M}_G$ contains the operation $\tilde{\mathcal{Q}}_S$ defined*

[12] This was done in [4] for the restricted class of canonical systems with $k = 1$.

as follows. For every (n,k)-ary quantifier $\mathcal{Q}$ of L, every $r \in \{t,f\}$ and every $g \in D^k \to \{t,f\}^n$:

$$\tilde{\mathcal{Q}}_S[g] = \begin{cases} \{r\} & \Theta/\mathcal{Q}(r) \in G \text{ and there is an } L_k^n - \text{structure } \mathcal{N} = \langle D_{\mathcal{N}}, I_{\mathcal{N}} \rangle \\ & \text{such that } D_{\mathcal{N}} = D,\ FDist_{\mathcal{N}} = g \text{ and } \Theta \text{ is valid in } \mathcal{N}. \\ \{t,f\} & \text{otherwise} \end{cases}$$

Proposition 3.4. *If a full canonical calculus G is coherent, then it is strongly sound for $\mathcal{M}_G$.*

Proof: The proof is similar to the proof of Theorem 23 in [3], with the addition of checking that the substitution rule is strongly sound for $\mathcal{M}_G$. In fact, it is easy to see that the substitution rule is strongly sound for any 2GNmatrix $\mathcal{M}$.

Now we come to the main result of this paper - establishing the correspondence:

Theorem 3.5 (The Triple Correspondence). *Let G be a full canonical calculus. Then the following statements concerning G are equivalent:*

1. *G is coherent.*
2. *G has a strongly characteristic 2GNmatrix.*
3. *G admits strong cut-elimination.*

Proof: We first prove that $\underline{(1) \Rightarrow (2)}$. Suppose that G is coherent. By proposition 3.4, G is strongly sound for $\mathcal{M}_G$. For strong completeness, let $\mathcal{S}$ be some set of sequents. Suppose that a sequent $\Gamma \Rightarrow \Delta$ has no proof from $\mathcal{S}$ in G. Then it also has no $\mathcal{S}$-cut-free proof from $\mathcal{S}$ in G. If $\mathcal{S} \cup \{\Gamma \Rightarrow \Delta\}$ does not satisfy the free-variable condition, obtain $\mathcal{S}' \cup \{\Gamma' \Rightarrow \Delta'\}$ by renaming the bound variables, so that $\mathcal{S}' \cup \{\Gamma' \Rightarrow \Delta'\}$ satisfies the condition (otherwise, take $\Gamma' \Rightarrow \Delta'$ and $\mathcal{S}'$ to be $\Gamma \Rightarrow \Delta$ and $\mathcal{S}$ respectively). Then $\Gamma' \Rightarrow \Delta'$ has no proof from $\mathcal{S}'$ in G (otherwise we could obtain a proof of $\Gamma \Rightarrow \Delta$ from $\mathcal{S}$ by using cuts on logical axioms), and so it also has no $\mathcal{S}'$-cut-free proof from $\mathcal{S}'$ in G. By proposition 3.6, $\mathcal{S}' \not\vdash_{\mathcal{M}_G} \Gamma' \Rightarrow \Delta'$. That is, there is an L-structure S and an $\mathcal{M}_G$-legal valuation v, such that the sequents in $\mathcal{S}'$ are $\mathcal{M}_G$-valid in $\langle S, v \rangle$, while $\Gamma' \Rightarrow \Delta'$ is not. Since v respects the $\equiv_\alpha$-relation, the sequents of $\mathcal{S}$ are also $\mathcal{M}_G$-valid in $\langle S, v \rangle$, while $\Gamma \Rightarrow \Delta$ is not. And so $\mathcal{S} \not\vdash_{\mathcal{M}_G} \Gamma \Rightarrow \Delta$. We have shown that G is strongly complete (and strongly sound) for $\mathcal{M}_G$. Thus $\mathcal{M}_G$ is a strongly characteristic 2GNmatrix for G.

Now we prove that $\underline{(2) \Rightarrow (1)}$. Suppose that G has a strongly characteristic 2GNmatrix $\mathcal{M}$. Assume by contradiction that G is not coherent. Then there exist two (n,k)-ary rules of the forms $R_1 = \Theta_1 / \Rightarrow A$ and $R_2 = \Theta_2 / A \Rightarrow$ in G, such that $\mathsf{Rnm}(\Theta_1 \cup \Theta_2)$ is classically consistent and $A = \mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}), ..., p_n(\overrightarrow{v}))$. Recall that $\mathsf{Rnm}(\Theta_1 \cup \Theta_2) = \Theta_1 \cup \Theta_2'$, where Θ_2' is obtained from Θ_2 by renaming constants and variables that occur also in Θ_1 (see defn. 2.8). For simplicity[13] we

[13] This assumption is not necessary and is used only for simplification of presentation, since we can instantiate the constants by any L-terms.

assume that the fresh constants used for renaming are all in L. Let $\Theta_1 = \{\Sigma^1_j \Rightarrow \Pi^1_j\}_{1\leq j\leq m}$ and $\Theta'_2 = \{\Sigma^2_j \Rightarrow \Pi^2_j\}_{1\leq j\leq r}$. Since $\Theta_1 \cup \Theta'_2$ is classically consistent, there exists an L^n_k-structure $\mathcal{N} = \langle D, I\rangle$, in which both Θ_1 and Θ'_2 are valid. Recall that we also assume that L^n_k is a subset of L[14] and so the following are applications of R_1 and R_2 respectively:

$$\frac{\{\Sigma^1_j \Rightarrow \Pi^1_j\}_{1\leq j\leq m}}{\Rightarrow \mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}), ..., p_n(\overrightarrow{v}))} \qquad \frac{\{\Sigma^2_j \ \Rightarrow \Pi^2_j\}_{1\leq j\leq m}}{\mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}), ..., p_n(\overrightarrow{v})) \Rightarrow}$$

Let S be any extension of $\mathcal{N}$ to L and v - any $\mathcal{M}$-legal S-valuation. It is easy to see that the premises of the applications above are $\mathcal{M}$-valid in $\langle S, v\rangle$ (since the premises contain atomic formulas). Since G is strongly sound for $\mathcal{M}$, both $\Rightarrow \mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}), ..., p_n(\overrightarrow{v}))$ and $\mathcal{Q}\overrightarrow{v}(p_1(\overrightarrow{v}), ..., p_n(\overrightarrow{v})) \Rightarrow$ should also be $\mathcal{M}$-valid in $\langle S, v\rangle$, which is of course impossible.

Next, we prove that $\underline{(1) \Rightarrow (3)}$. Let G be a coherent full canonical calculus. Let $\mathcal{S}$ be a set of sequents, and let $\Gamma \Rightarrow \Delta$ be a sequent, such that $\mathcal{S} \cup \{\Gamma \Rightarrow \Delta\}$ satisfies the free-variable condition. Suppose that $\mathcal{S} \vdash_G \Gamma \Rightarrow \Delta$. We have already shown above that $\mathcal{M}_G$ is a strongly characteristic 2GNmatrix for G. Thus $\mathcal{S} \vdash_{\mathcal{M}} \Gamma \Rightarrow \Delta$. Now we need the following proposition, the proof of which is given in Appendix A:

Proposition 3.6. *Let G be a coherent full canonical calculus. Let $\mathcal{S}$ be a set of sequents and $\Gamma \Rightarrow \Delta$ - a sequent such that $\mathcal{S}\cup\{\Gamma \Rightarrow \Delta\}$ satisfies the free-variable condition. If $\Gamma \Rightarrow \Delta$ has no $\mathcal{S}$-cut-free proof from $\mathcal{S}$ in G, then $\mathcal{S}\not\vdash_{\mathcal{M}_G}\Gamma \Rightarrow \Delta$.*

By this proposition, $\Gamma \Rightarrow \Delta$ has an $\mathcal{S}$-cut-free proof from $\mathcal{S}$ in G. Thus G admits strong cut-elimination.

Finally, we prove that $\underline{(3) \Rightarrow (1)}$. Suppose that G admits strong cut-elimination. Suppose by contradiction that G is not coherent. Then there are two canonical rules of G of the forms: $\Theta_1/ \ \Rightarrow \ A$ and $\Theta_2/A \ \Rightarrow$, such that $\mathsf{Rnm}(\Theta_1 \cup \Theta_2)$ is classically consistent. Let $\Theta = \mathsf{Rnm}(\Theta_1 \cup \Theta_2)$. Then $\Theta \cup \{\Rightarrow\}$ satisfy the free-variable condition, since only atomic formulas are involved and no variables are bound there. Since $\Theta \vdash_G\Rightarrow A$ and $\Theta \vdash_G A \Rightarrow$, by using cut we get: $\Theta \vdash_G\Rightarrow$. But $\Rightarrow$ has no Θ-cut-free proof in G from Θ since Θ is consistent, in contradiction to the fact that G admits strong cut-elimination.

Corollary 3.7. *For every full canonical calculus, the question whether it admits strong cut-elimination is decidable.*

Remark: The results presented above are related to [8], where a general class of sequent calculi with (n, k)-ary quantifiers and a (not necessarily standard) set of structural rules, are defined. Canonical calculi are a particular instance of such calculi which includes all of the standard structural rules. While handling a a wider class of calculi than canonical systems (different combinations of structural rules are allowed), [8] provides no semantics for them. Syntactic conditions are given, which are sufficient and under certain additional limitations also necessary

[14] This assumption is again not essential for the proof, but it simplifies the presentation.

for modular cut-elimination, a particular version of cut-elimination for deduction with non-logical assumptions containing only atomic formulas. In the context of canonical systems, these conditions can be shown to be equivalent to our coherence criterion, but (unlike coherence) they are not constructive.

Acknowledgement

This research was supported by the *Israel Science Foundation* founded by the Israel Academy of Sciences and Humanities (grant No 809/06).

References

1. Avron, A.: Gentzen-Type Systems, Resolution and Tableaux. Journal of Automated Reasoning 10, 265–281 (1993)
2. Avron, A.,, Lev, I.: Non-deterministic Multi-valued Structures. Journal of Logic and Computation 15, 241–261 (2005)
3. Avron, A., Zamansky, A.: Generalized Non-deterministic matrices and (n,k)-ary quantifiers. In: Artemov, S.N., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 26–41. Springer, Heidelberg (2007)
4. Avron, A., Zamansky, A.: Canonical calculi with (n,k)-ary quantifiers. Journal of Logical Methods in Computer Science (forthcming, 2008)
5. Baaz, M., Fermüller, C.G., Salzer, G., Zach, R.: Labeled Calculi and Finite-valued Logics. Studia Logica 61, 7–33 (1998)
6. Carnielli, W.A.: Systematization of Finite Many-valued Logics through the method of Tableaux. Journal of Symbolic Logic 52(2), 473–493 (1987)
7. Carnielli, W.A., Conglio, M.E.: Splitting Logics. In: Artemov,, Barringer,, Garcez,, Lamb (eds.) We Will Show Them!, Essays in Honour of Dov Gabbay, vol. 1, pp. 389–414. Woods College Publications (2005)
8. Ciabattoni, A., Terui, K.: Modular cut-elimination: Finding proofs or counterexamples. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 135–149. Springer, Heidelberg (2006)
9. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5, 56–68 (1940)
10. Gentzen, G.: Investigations into Logical Deduction. In: Szabo, M.E. (ed.) The collected works of Gerhard Gentzen, pp. 68–131. North Holland, Amsterdam (1969)
11. Hähnle, R.: Commodious Axiomatization of Quantifiers in Many-valued Logic. Studia Logica 61, 101–121 (1998)
12. Henkin, L.: Some remarks on infinitely long formulas. In: Infinistic Methods, pp. 167–183. Pergamon Press, Oxford (1961)
13. Kalish, D., Montague, R.: Logic, Techniques. of Formal Reasoning. Brace and World, Inc., New York, Harcourt (1964)
14. Miller, D., Pimentel, E.: Using Linear logic to reason about sequent systems. In: Egly, U., Fermüller, C. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, pp. 2–23. Springer, Heidelberg (2002)
15. Shroeder-Heister, P.: Natural deduction calculi with rules of higher levels. Journal of Symbolic Logic 50, 275–276 (1985)
16. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory, pp. 126–130. Cambridge University Press, Cambridge (2000)
17. Zamansky, A., Avron, A.: 'Cut Elimination and Quantification in Canonical Systems'. Studia Logica 82(1), 157–176 (2006)

A Appendix: Proof of Proposition 3.6

Let $\Gamma \Rightarrow \Delta$ be a sequent which satisfies the free-variable condition. Suppose that $\Gamma \Rightarrow \Delta$ has no $\mathcal{S}$-cut-free proof from $\mathcal{S}$ in G. To show that $\mathcal{S} \not\vdash_{\mathcal{M}_G} \Gamma \Rightarrow \Delta$, we will construct an L-structure S and an $\mathcal{M}_G$-legal valuation v, such that $\mathcal{S}$ is $\mathcal{M}_G$-valid in $\langle S, v\rangle$, while $\Gamma \Rightarrow \Delta$ is not.

It is easy to see that we can limit ourselves to the language L^*, which is a subset of L, consisting of all the constants and predicate and function symbols, occurring in $\Gamma \Rightarrow \Delta$. Let $\mathbf{T}$ be the set of all the terms in L^* which do not contain variables occurring bound in $\Gamma \Rightarrow \Delta$. It is a standard matter to show that Γ, Δ can be extended to two (possibly infinite) sets Γ', Δ' (where $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$), satisfying the following properties:

1. For every finite $\Gamma_1 \subseteq \Gamma'$ and $\Delta_1 \subseteq \Delta'$, $\Gamma_1 \Rightarrow \Delta_1$ has no cut-free proof in G.
2. There are no $\psi \in \Gamma'$ and $\varphi \in \Delta'$, such that $\psi \equiv_\alpha \varphi$.
3. If $\{\Pi_j \Rightarrow \Sigma_j\}_{1 \le j \le m} / \mathcal{Q}(r)$ is an (n,k)-ary rule of G and $\mathcal{Q} z_1 ... z_k\ (A_1, ..., A_n) \in ite(r, \Delta', \Gamma')$ (recall Notation 2.6), then there is some $1 \le j \le m$ satisfying the following condition. Let $\mathbf{t}_1, ..., \mathbf{t}_m$ be all the L_k^n-terms occurring in $\Pi_j \cup \Sigma_j$, where $\mathbf{t}_{j_1}, ..., \mathbf{t}_{j_l}$ are all the constants and $\mathbf{t}_{j_{l+1}}, ..., \mathbf{t}_{j_m}$ are all the variables. Then for every $\mathbf{s}_1, ..., \mathbf{s}_l \in \mathbf{T}$ there are some[15] $\mathbf{s}_{l+1}, ..., \mathbf{s}_m \in \mathbf{T}$, such that whenever $p_i(\mathbf{t}_{n_1}, ..., \mathbf{t}_{n_k}) \in ite(r, \Pi_j, \Sigma_j)$, then $A_i\{\mathbf{s}_{n_1}/z_1, ..., \mathbf{s}_{n_k}/z_k\} \in ite(r, \Gamma', \Delta')$.
4. For every formula ψ occurring in $\mathcal{S}$ and every substitution instance ψ' of ψ: $\psi' \in \Gamma' \cup \Delta'$.

Note that the last condition can be satisfied because cuts on substitution instances of formulas from $\mathcal{S}$ are allowed in an $\mathcal{S}$-cut-free proof.

Let $S = \langle D, I\rangle$ be the L^*-structure defined as follows: $D = \mathbf{T}$, $I[c] = c$ for every constant c of L^*; $I[f][\mathbf{t}_1, ..., \mathbf{t}_n] = f(\mathbf{t}_1, ..., \mathbf{t}_n)$ for every n-ary function symbol f; $I[p][\mathbf{t}_1, ..., \mathbf{t}_n] = t$ iff $p(\mathbf{t}_1, ..., \mathbf{t}_n) \in \Gamma'$ for every n-ary predicate symbol p.

It is easy to show by induction on $\mathbf{t}$ that: **(lem1)** for every $\mathbf{t} \in \mathbf{T}$: $I[\sigma^*[\mathbf{t}]] = \mathbf{t}$.

Let σ^* be any S-substitution satisfying $\sigma^*[x] = \overline{x}$ for every $x \in \mathbf{T}$. (Note that every $x \in \mathbf{T}$ is also a member of the domain and thus has an individual constant referring to it in $L^*(D)$).

For an $L(D)$-formula ψ (an $L(D)$-term $\mathbf{t}$), we will denote by $\widehat{\psi}$ ($\widehat{\mathbf{t}}$) the L-formula (L-term) obtained from ψ ($\mathbf{t}$) by replacing every individual constant of the form $\overline{\mathbf{s}}$ for some $\mathbf{s} \in \mathbf{T}$ by the term $\mathbf{s}$.

Define the S-valuation v as follows: (i) $v[p(\mathbf{t}_1, ..., \mathbf{t}_n)] = I[p][I[\mathbf{t}_1], ..., I[\mathbf{t}_n]]$, (ii) If there is some $C \in \Gamma' \cup \Delta'$, s.t. $C \equiv_\alpha \widehat{\mathcal{Q} \overrightarrow{z} (\psi_1, ..., \psi_n)}$, then $v[\mathcal{Q} \overrightarrow{z} (\psi_1, ..., \psi_n)] = t$ iff $C \in \Gamma'$. Otherwise $v[\mathcal{Q} \overrightarrow{z} (\psi_1, ..., \psi_n)] = t$ iff $\tilde{\mathcal{Q}}_S[\lambda a_1 ... a_k \in D. \{\langle v[\psi_1\{\overrightarrow{a}/\overrightarrow{z}\}], ..., v[\psi_n\{\overrightarrow{a}/\overrightarrow{z}\}]\rangle\}] = \{t\}$.

[15] Note that in contrast to $\mathbf{t}_1, ..., \mathbf{t}_m$, $\mathbf{s}_1, ..., \mathbf{s}_m$ are L-terms and not L_k^n-terms.

The proof of the following lemmas is not hard and is left to the reader:

(lem2): For every $L(D)$-formula ψ: $\psi \sim^S \sigma^*[\widehat{\psi}]$.
(lem3): For every $\psi \in \Gamma' \cup \Delta'$: $\widehat{\sigma^*[\psi]} = \psi$.
(lem4): v is legal in $\mathcal{M}_G$.

Next we prove: **(lem5)** For every $\psi \in \Gamma' \cup \Delta'$: $v[\sigma^*[\psi]] = t$ iff $\psi \in \Gamma'$. If $\psi = p(\mathbf{t}_1, ..., \mathbf{t}_n)$, then $v[\sigma^*[\psi]] = I[p][I[\sigma^*[\mathbf{t}_1]], ..., I[\sigma^*[\mathbf{t}_n]]]$. Note[16] that for every $1 \leq i \leq n$, $\mathbf{t}_i \in \mathbf{T}$. By **(lem1)**, $I[\sigma^*[\mathbf{t}_i]] = \mathbf{t}_i$, and by the definition of I, $v[\sigma^*[\psi]] = t$ iff $p(\mathbf{t}_1, ..., \mathbf{t}_n) \in \Gamma'$. Otherwise $\psi = Q\overrightarrow{z}(\psi_1, ..., \psi_n)$. If $\psi \in \Gamma'$, then by **(lem3)**: $\widehat{\sigma^*[\psi]} = \psi \in \Gamma'$ and so $v[\sigma^*[\psi]] = t$. If $\psi \in \Delta'$ then by property 2 of $\Gamma' \cup \Delta'$ it cannot be the case that there is some $C \in \Gamma'$, such that $C \equiv_\alpha \widehat{\sigma^*[\psi]} = \psi$ and so $v[\sigma^*[\psi]] = f$.

Finally, we prove that for every sequent $\Sigma \Rightarrow \Pi \in \mathcal{S}$, $\Sigma \Rightarrow \Pi$ is $\mathcal{M}_G$-valid in $\langle S, v\rangle$. Suppose by contradiction that there is some $\Sigma \Rightarrow \Pi \in \mathcal{S}$, which is not $\mathcal{M}_G$-valid in $\langle S, v\rangle$. Then there exists some S-substitution μ, such that for every $\psi \in \Sigma$: $S, v \models_{\mathcal{M}_G} \mu[\psi]$, and for every $\varphi \in \Pi$: $S, v \not\models_{\mathcal{M}_G} \mu[\varphi]$. Note that for every $\phi \in \Sigma \cup \Pi$, $\widehat{\mu[\phi]}$ is a substitution instance of ϕ. By property 5 of $\Gamma' \cup \Delta'$: $\widehat{\mu[\phi]} \in \Gamma' \cup \Delta'$. By **(lem5)**, if $\widehat{\mu[\phi]} \in \Gamma'$ then $v[\sigma^*[\widehat{\mu[\phi]}]] = t$, and if $\widehat{\mu[\phi]} \in \Delta'$ then $v[\sigma^*[\widehat{\mu[\phi]}]] = f$. By **(lem2)**, $\mu[\phi] \sim^S \sigma^*[\widehat{\mu[\phi]}]$. Since v is $\mathcal{M}_G$-legal, it respects the $\sim^S$-relation and so for every $\phi \in \Sigma \cup \Pi$: $v[\mu[\phi]] = v[\sigma^*[\widehat{\mu[\phi]}]]$. Thus $\widehat{\mu[\Sigma]} \subseteq \Gamma'$ and $\widehat{\mu[\Pi]} \subseteq \Delta'$. But $\widehat{\mu[\Sigma]} \Rightarrow \widehat{\mu[\Pi]}$ has an $\mathcal{S}$-cut-free proof from $\mathcal{S}$ in G (note that $\widehat{\mu[\Sigma]} \Rightarrow \widehat{\mu[\Pi]}$ is obtained from $\Sigma \Rightarrow \Pi$ by applying the substitution rule), in contradiction to property 1 of $\Gamma' \cup \Delta'$.

Thus, all sequents of $\mathcal{S}$ are $\mathcal{M}_G$-valid in $\langle S, v\rangle$. However, by **(lem5)**: $\Gamma \Rightarrow \Delta$ is not $\mathcal{M}_G$-valid in $\langle S, v\rangle$ (recall that $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$). Thus $\mathcal{S} \not\vdash_{\mathcal{M}_G} \Gamma \Rightarrow \Delta$.

[16] This is obvious if $\mathbf{t}_i$ does not occur in $\Gamma \Rightarrow \Delta$. If it occurs in $\Gamma \Rightarrow \Delta$, then since $\Gamma \Rightarrow \Delta$ satisfies the free-variable condition, $\mathbf{t}_i$ does not contain variables bound in this set and so $\mathbf{t}_i \in \mathbf{T}$ by definition of $\mathbf{T}$.

Computing Longest Common Substrings Via Suffix Arrays

Maxim A. Babenko and Tatiana A. Starikovskaya

Moscow State University
max@adde.math.msu.su,
tat.starikovskaya@gmail.com

Abstract. Given a set of N strings $A = \{\alpha_1, \ldots, \alpha_N\}$ of total length n over alphabet Σ one may ask to find, for each $2 \le K \le N$, the longest substring β that appears in at least K strings in A. It is known that this problem can be solved in $O(n)$ time with the help of suffix trees. However, the resulting algorithm is rather complicated (in particular, it involves answering certain least common ancestor queries in $O(1)$ time). Also, its running time and memory consumption may depend on $|\Sigma|$.

This paper presents an alternative, remarkably simple approach to the above problem, which relies on the notion of suffix arrays. Once the suffix array of some auxiliary $O(n)$-length string is computed, one needs a simple $O(n)$-time postprocessing to find the requested longest substring. Since a number of efficient and simple linear-time algorithms for constructing suffix arrays has been recently developed (with constant not depending on $|\Sigma|$), our approach seems to be quite practical.

1 Introduction

Consider the following problem:

> (LCS) *Given a collection of N strings $A = \{\alpha_1, \ldots, \alpha_N\}$ over alphabet Σ find, for each $2 \le K \le N$, the longest string β that is a substring of at least K strings in A.*

It is known as a generalized version of the *Longest Common Substring* (LCS) problem and has a plenty of practical applications, see [Gus97] for a survey.

Even in the simplest case of $N = K = 2$ a linear-time algorithm is not easy. A standard approach is to construct the so-called *generalized suffix tree* T (see [Gus97]) for $\alpha_1\$_1$ and $\alpha_2\$_2$, which is a compacted symbol trie that captures all the substrings of $\alpha_1\$_1$, $\alpha_2\$_2$. Here $\$_i$ are special symbols (called *sentinels*) that are distinct and do not appear in α_1 and α_2. Then, nodes of T are examined in a bottom-up fashion and those having sentinels of both types in their subtrees are listed. Among these nodes of T let us choose a node v with the largest *string depth* (which is the length of the string obtained by reading letters along the path from root to v). The string that corresponds to v in T is the answer. See [Gus97] for more details.

In practice, the above approach is not very efficient since it involves computing T. Several linear-time algorithms for the latter task are known (possibly, the most famous one is due to Ukkonen [Ukk95]). However, suffix trees are still not very convenient. They do have linear space bound but the hidden constants can be pretty large. Most of modern algorithms for computing suffix trees have the time bound of $O(n \log |\Sigma|)$ (where n denotes the length of a string). Hence, their running time depends on $|\Sigma|$. Moreover, achieving this time bound requires using balanced search trees to store arcs. The latter data structures further increase constants in both time- and space-bounds making these algorithms rather impractical. Other options include employing hashtables or assuming that $|\Sigma|$ is small and using direct addressing to access arcs leaving each node. These approaches have their obvious disadvantages.

If one assumes that N and K are arbitrary then additional complications arise. Now we are interested in finding the deepest (in sense of string depth) node v in T such that the tree rooted at v contains sentinels of at least K distinct kinds. Moreover, this routine should run in parallel for all possible values of K and take linear time. This seems to be an involved task. A possible solution requires answering *Least Common Ancestor* (LCA) queries on T in $O(1)$ time, e.g. using a method from [BFC00]. Reader may refer to [Gus97] for a complete outline.

In this paper we present an alternative approach that is based on the notion of *suffix arrays*. The latter were introduced by Manber and Myers [MM90] in an attempt to overcome the issues that are inherent to suffix trees. The *suffix array* (SA) of string α having length n is merely an array of n integers that indicate the lexicographic order of non-empty suffixes of α (see Section 2 for a precise definition). Its simplicity and compactness make it an extremely useful tool in modern text processing. Originally, an $O(n \log n)$-time algorithm for constructing SA was suggested [MM90]. This algorithm is not very practical. Subsequently, much simpler and faster algorithms for computing SA were developed. We particularly mention an elegant approach of Kärkkäinen and Sanders [KS03]. A comprehensive practical evaluation of different algorithms for constructing SA is given in [PST07].

We present two algorithms. The first one, which is simpler, assumes that parameter K is fixed. It first builds an auxiliary string α by concatenating strings α_i and intermixing them with sentinels $\$_i$ $(1 \le i \le N)$ and then constructs the suffix array for string α. Also, an additional *LCP array* is constructed. Finally, a sliding window technique is applied to these arrays to obtain the answer. Altogether, the running time is linear and does not depend on $|\Sigma|$.

The second algorithm deals with all possible values of K simultaneously. Its initial stage stage is similar: string α, suffix array for α, and LCP array are constructed. Then, *Cartesian tree* (CT) is constructed from LCP array. Finally, a certain postprocessing aimed to count the number of distinct types of nodes appearing in subtrees of CT is used. It should be noticed that this postprocessing does not require answering any least common ancestor queries.

The paper is organized as follows. Section 2 gives a formal background, introduces useful notation and definitions. It also explains the notion of suffix arrays and indicates how an auxiliary LCP array is constructed in linear time. Section 3 presents the algorithm for the case of fixed K. Finally, Section 4 considers the general case when the value of K is not fixed.

2 Preliminaries

We shall start with a number of definitions first. In what follows we assume that a finite non-empty set Σ (called an *alphabet*) is fixed. The elements of Σ are *letters* or *symbols*. A finite ordered sequence of letters (possibly empty) is called a *string*.

We assume the usual RAM model of computation [AUH74]. Letters are treated just as integers in range $\{1, \ldots, |\Sigma|\}$, so one can compare any pair of them in $O(1)$ time. This *lexicographic* order on Σ is linear and can be extended in a standard way to the set of strings in Σ. We write $\alpha < \beta$ to denote that α lexicographically precedes β; similarly for other relation signs.

We usually use Greek symbols to denote strings. Letters in a string are numbered starting from 1, that is, for a string α of *length* k its letters are $\alpha[1], \ldots, \alpha[k]$. The length k of α is denoted by $|\alpha|$. The *substring* of α from position i to position j (inclusively) is denoted by $\alpha[i..j]$. Also, if $i = 1$ or $j = |\alpha|$ then these indices are omitted from the notation and we write just $\alpha[..j]$ and $\alpha[i..]$. String $\beta = \alpha[..j]$ is called a *prefix* of α. Similarly, if $\beta = \alpha[i..]$ then β is called a *suffix* of α. For a set of strings S let $lcp(S)$ denote the longest common prefix of all strings in S.

Recall that A stands for the collection of the input strings α_i. We start with an almost trivial observation:

Proposition 1. *Let $B = \{\beta_1, \ldots, \beta_m\} \subseteq A$ be an arbitrary subset of A obeying $m \geq K$. Let γ_i be an arbitrary suffix of β_i for each $1 \leq i \leq m$. Solving* (LCS) *amounts to computing the longest string among*

$$lcp(\gamma_1, \ldots, \gamma_m)$$

where maximum is taken over all possible choices of subsets B and suffixes $\{\gamma_i\}$.

Let us combine the strings in A as follows:

$$\alpha = \alpha_1 \$_1 \alpha_2 \$_2 \ldots \alpha_N \$_N \tag{1}$$

Here $\$_i$ are pairwise distinct *sentinel* symbols not appearing in strings of A. We assume that these sentinels are lexicographically smaller than other (normal) symbols. The lexicographic order between sentinels is not important.

String α captures all needed information about set A. For each index i ($1 \leq i \leq N$) and a position j in α_i ($1 \leq j \leq |\alpha_i|$) one may consider the *corresponding* position $p(i, j)$ in α:

$$p(i, j) := \sum_{k=1}^{i-1} (|\alpha_k| + 1) + j$$

	suffixes	SA	sorted suffixes	lcp
1	`mississippi`	11	`i`	1
2	`ississippi`	8	`ippi`	1
3	`ssissippi`	5	`issippi`	4
4	`sissippi`	2	`ississippi`	0
5	`issippi`	1	`mississippi`	0
6	`ssippi`	10	`pi`	1
7	`sippi`	9	`ppi`	0
8	`ippi`	7	`sippi`	2
9	`ppi`	4	`sissippi`	1
10	`pi`	6	`ssippi`	3
11	`i`	3	`ssissippi`	

Fig. 1. String `mississippi`, its suffixes, and the corresponding suffix and LCP arrays

Positions in α of the form $p(i, j)$ are called *essential*; the remaining positions (those containing sentinels) are called *unessential.*

Let us employ the following metaphor: for each $1 \le i \le N$ and $1 \le j \le |\alpha_i|$ we say that position $p(i, j)$ is *of type* i (it corresponds to the i-th string). Remaining (unessential) positions k in α are said to be of type 0.

Now taking into account the properties of sentinels one can easily derive the following claim from Proposition 1:

Proposition 2. *Let* $P = \{p_1, \dots, p_m\}$ *be an arbitrary set of essential positions in* α *such that elements of* P *are of at least* K *distinct types. Solving* (LCS) *amounts to computing the longest string among*

$$lcp\big(\alpha[p_1..], \dots, \alpha[p_m..]\big)$$

where maximum is taken over all possible choices of P.

This does not seem very promising at the first glance. However, the longest common prefix computation exhibits a nice structure when it is applied to suffixes of a fixed string (in our case, string α).

To explain this structure we first introduce the notion of suffix arrays. Let ω be an arbitrary string of length n. Consider its non-empty suffixes

$$\omega[1..],\ \omega[2..],\ \dots,\ \omega[n..]$$

and order them lexicographically. Let $SA(i)$ denote the starting position of the suffix appearing on the i-th place ($1 \le i \le n$) in this order:

$$\omega[SA(1)..] < \omega[SA(2)..] < \dots < \omega[SA(n)..]$$

Clearly, SA is determined uniquely since all suffixes of ω are distinct. An example is depicted in Fig. 1.

Since SA is a permutation of $\{1, \dots, n\}$ there must be an inverse correspondence. We denote it by *rank*; that is, *rank* is also a permutation of $\{1, \dots, n\}$ and

$$SA(rank(i)) = i \qquad \text{holds for all } 1 \le i \le n.$$

Historically, the first algorithm to compute SA was due to Manber and Myers [MM90]; this algorithm takes $O(n \log n)$ time. Currently, simple linear-time algorithms for this task are known, see [PST07] for a list. The latter linear time bounds do not depend on $|\Sigma|$.

However, knowing SA is not enough for our purposes. We also have to precompute the lengths of longest common prefixes for each pair of consequent suffixes (with respect to the order given by SA). More formally,

$$lcp(i) := \big|lcp\big(\omega[SA(i)..], \omega[SA(i+1)..]\big)\big| \qquad \text{for all } 1 \le i < n.$$

This gives rise to array lcp of length $n-1$; we call it the *LCP array of* ω. The latter array not only enables to answer LCP queries for consequent (w.r.t. SA) suffixes of ω but also carries enough information to answer *any* such query. Formally [MM90]:

Lemma 1. *For each pair $1 \le i < j \le n$ one has*

$$\big|lcp\big(\omega[SA(i)..], \omega[SA(j)..]\big)\big| = \min_{i \le k < j} lcp(k)$$

Knowing the suffix array, LCP array may be constructed in $O(n^2)$ time by a brute-force method. However, an elegant modification ([KLA$^+$01], see also [CR03]) allows to compute the longest common prefix for a pair of consequent suffixes in $O(1)$ amortized time. The key is to compute these values in a particular order, namely

$$lcp(rank(1)),\ lcp(rank(2)),\ \ldots,\ lcp(rank(n))$$

The efficiency of this approach relies on the following fact [KLA$^+$01]:

Lemma 2. $lcp(rank(i+1)) \ge lcp(rank(i)) - 1$ *for each* $1 \le i < n$ *such that* $rank(i) < n$ *and* $rank(i+1) < n$.

Hence, when computing the value of $lcp(rank(i+1))$ one can safely skip $lcp(rank(i)) - 1$ initial letters of $\omega[i+1..]$ and $\omega[SA(rank(i+1)+1)..]$. This easily implies the required linear time bound for the whole computation.

3 Fixed K: Sliding Window

We now proceed by describing the algorithm that solves (LCS) for a fixed value of K. There are several reasons for considering this case separately. Firstly, this problem also seems natural and the resulting approach is somewhat simpler. Secondly, the techniques that we develop for this special case turn out to be useful for the general problem.

The algorithm works are follows. It first combines the input strings into string α of length L (see (1)) and invokes the suffix array computation algorithm thus obtaining the suffix array SA for α. It also constructs array lcp in $O(L)$ time as described in Section 2.

Refer to Proposition 2 and consider an arbitrary set of essential positions $P = \{p_1, \ldots, p_m\}$ that contains positions of at least K distinct types. Replace these positions with ranks by putting $r_i := rank(p_i)$ and, thus, forming the set $R = \{r_1, \ldots, r_m\}$. Lemma 1 implies the equality

$$\left|lcp\bigl(\alpha[p_1..], \ldots, \alpha[p_m..]\bigr)\right| = \min_{R^- \le j < R^+} lcp(j)$$

where $R^- := \min R$ and $R^+ := \max R$.

Let us consider a segment $\Delta \subseteq [1, L]$ and call it *K-good* if for $i \in \Delta$ positions $SA(i)$ are of at least K distinct essential types. This enables us to restate Proposition 2 as follows:

Proposition 3. *The length of the longest common substring that appears in at least K input strings is equal to*

$$\max_{\Delta} \min_{\Delta^- \le j < \Delta^+} lcp(j)$$

where $\Delta = [\Delta^-, \Delta^+]$ ranges over all K-good segments.

This formula is already an improvement (compared to Proposition 2) since it only requires to consider a polynomial number of possibilities. Moreover, we shall indicate how maximum in Proposition 3 can be found in $O(L)$ time.

To this aim, note that if a K-good segment Δ is already considered then any $\Delta' \supset \Delta$ cannot give us a bigger value of minimum. For each i one may consider the segment $\Delta_i = [\Delta_i^-, \Delta_i^+]$ obeying the following properties:

- Δ_i starts at position i;
- Δ_i is K-good;
- Δ_i is the shortest segment obeying the above conditions.

Here index i ranges over $[1, L_0]$, where $L_0 \le L$ is the smallest integer such that segment $[L_0 + 1, L]$ is not K-good.

Note that due to our assumption that sentinels are strictly less than normal letters, the first N elements of SA correspond to unessential positions occupied by the sentinels. The algorithm does not need to consider these positions and only examines segments $\Delta_{N+1}, \ldots, \Delta_{L_0}$.

Put

$$w(i) := \min_{\Delta_i^- \le j < \Delta_i^+} lcp(j) \qquad \text{for all } N < i \le L_0.$$

and consider the sequence:

$$w(N+1),\ w(N+2),\ \ldots,\ w(L_0) \tag{2}$$

Once the maximum among (2) is found, the problem is solved. We construct a pipeline with the first stage computing the sequence of segments

$$\Delta_{N+1},\ \Delta_{N+2},\ \ldots, \Delta_{L_0}$$

and the second stage calculating the respective minima (2)

Put $\Delta_i = [\Delta_i^-, \Delta_i^+]$. The first stage works as follows. It initially sets $\Delta_{N+1}^- := N+1$ and then advances the right endpoint Δ_{N+1}^+ until getting a K-good segment. Then, on each subsequent iteration i it puts $\Delta_i^- := i$, $\Delta_i^+ := \Delta_{i-1}^+$ and again advances the right endpoint Δ_i^+ until Δ_i becomes K-good. (In case, no such segment can be extracted, it follows that $i > L_0$, so the end is reached.)

Lemma 3. *Δ_i is the shortest K-good segment starting at i.*

Proof. We claim that for any K-good segment $[i, j]$ one has $j \geq \Delta_{i-1}^+$. Indeed, suppose towards contradiction that $j < \Delta_{i-1}^+$. Since $[i, j]$ is K-good then so is $[i-1, j]$. The latter, however, contradicts the minimality of Δ_{i-1}.

To test in $O(1)$ time if the current candidate forms a K-good segment the algorithm maintains an array of counters $c(1), \ldots, c(N)$. For each $N < j \leq L$ put $t(j)$ to be the type of position $SA(j)$ in α (recall that all these positions are essential). For each index i ($1 \leq i \leq N$) the entry $c(i)$ stores the number of positions j in the current segment such that $t(j) = i$. Also, the number of non-zero entries of c (denoted by *npos*) is maintained.

Initializing c and *npos* for Δ_{N+1} is trivial. Then, when the algorithm puts $\Delta_i^- = \Delta_{i-1}^- + 1$ it decrements the entry of c that corresponds to position $i-1$ (which has just been removed from the window) and adjusts *npos*, if necessary. Similarly, when the current segment is extended to the right, certain entries of c are increased and *npos* is adjusted. To see whether the current segment is K-good one checks if $npos \geq K$. This completes the description of the first stage of the pipeline. Note that it totally takes $O(L)$ time.

The second stage aims to maintain the values (2) dynamically. This is done by the following (possibly folklore) trick. Consider a queue Q that, at any given moment i, holds the sequence of keys

$$lcp(\Delta_i^-),\ lcp(\Delta_i^- + 1),\ \ldots,\ lcp(\Delta_i^+ - 1)$$

Increasing index i the algorithm dequeues value $lcp(\Delta_i^-)$ from the head of Q (to account for the increase of Δ_i^-) and then enqueues some (possibly none) additional values to the tail of Q (to account for the increase of Δ_i^+, if any). The total number of these queue operations is $O(L)$.

We describe a method for maintaining the minimum of keys in Q under insertions and removals and serving each such request in $O(1)$ amortized time. A queue Q that holds a sequence $(q_1, \ldots, q_m)$ may be simulated by a pair of stacks S^1 and S^2. A generic configuration of these stacks during this simulation is as follows (here $0 \leq s \leq m$):

$$\begin{array}{l} S^1 = (\ q_s,\quad q_{s-1}, \ldots, q_1\ \ ) \\ S^2 = (\ q_{s+1}, q_{s+2}, \ldots, q_m\ ) \end{array} \qquad (3)$$

Here stack elements are listed from bottom to top. Initially Q is empty, hence so are S^1 and S^2. To enqueue a new key x (which becomes q_{m+1}) to Q one pushes x onto S^2. This takes $O(1)$ time. To dequeue q_1 from Q consider two cases. If $s > 0$

then S^1 is non-empty; pop the top element q_1 from S^1. Otherwise, one needs to transfer elements from S^2 to S^1. This is done by popping elements from S^2 one after another and simultaneously pushing them onto S^1 (in the same order). By (3) these operations preserve the order of keys in Q. Once they are complete, S^1 becomes non-empty and the first case applies. To estimate the running time note that any enqueued element may participate in an S^2-to-S^1 transfer at most once. Hence, an amortized bound of $O(1)$ follows.

Our algorithm simulates Q via S^1 and S^2, as explained above. Each S^i is additionally augmented to maintain the minimum among the keys it contains. This is achieved by keeping minima m^1, m^2 and a pair of auxiliary stacks M^1, M^2. When a new key x is pushed to S^i the algorithm saves the previous minimum m^i in M^i and updates m^i by $m^i := \min(m^i, x)$. When an element is popped from S^i the algorithm also pops m^i from M^i.

Altogether these manipulations with Q, S^i, M^i, and m^i take time that is proportional to the number of operations applied to Q. The latter is known to be $O(L)$. Hence, the algorithm computes the sequence of minima (2) and chooses the maximum (call it $M(K)$) among these values in $O(L)$ time, as claimed.

Let the above maximum be attained by a segment $\Delta = [\Delta^-, \Delta^+] \subseteq [N+1, L]$. Suppose that position $SA(\Delta^-)$ in string α corresponds to some position j in some input string α_i. Now the desired longest common substring is $\alpha_i[j..j+M(K)-1]$.

4 Arbitrary *K*: Cartesian Tree

Now we go back to the original problem (LCS) and no longer assume that the value of K is given in advance. Similarly to Section 3, we start by constructing string α, suffix array SA of α, and the corresponding LCP array.

The cornerstone of the algorithm is a novel postprocessing, which combines the above information. Firstly, we need to overcome the following technical issue. Recall from the previous section that $t(j) \in \{1, \ldots, N\}$ denotes the type of position $SA(j)$ in string α ($N < j \le L$). Also, a segment Δ is called K-good if $t(j)$ gives at least K distinct values while j ranges over Δ. According to Proposition 3 to estimate the length of the common substring corresponding to segment Δ one needs to compute $\min_j lcp(j)$ where j ranges over Δ *without its right endpoint.*

To simplify the matters we construct a new pair of arrays lcp' and t' (whose elements are numbered starting from 1) by intermixing elements of lcp and t with artificial values as follows:

$$\begin{array}{lllllllll} lcp' := (& \infty, & lcp(N+1), & \infty, & lcp(N+2), & \ldots, & lcp(L-1), & \infty &) \\ t' \;\; := (& t(N+1), & 0, & t(N+2), & 0, & \ldots, & 0, & t(L) &) \end{array}$$

Now for arrays lcp' and t' it is clear that the same segment Δ should be used both for calculating the number of distinct non-zero values among $t'(j)$, $j \in \Delta$ and the respective minima $\min\,(lcp'(j) : j \in \Delta)$.

We remind the reader the notion of *Cartesian trees* (see, e.g., [BFC00]). Let $A = (a_1, \ldots, a_n)$ be an arbitrary (possibly empty) sequence of items, and let each a_i be assigned a real number $key(a_i)$. We associate a tree $CT(A)$ with A by the following rules:

– if $n = 0$ then $CT(A)$ is the *null tree* (having no nodes);
– if $n > 0$ then let a_i denote an item in A with the smallest key $key(a_i)$ (the choice of i may not be unique); now $CT(A)$ is constructed by taking a_i as its root, recursively constructing trees $CT_L := CT(a_1, \ldots, a_{i-1})$ and $CT_R := CT(a_{i+1}, \ldots, a_n)$, and attaching CT_L to a_i as its left child and CT_R as its right child.

Proposition 4. *Consider a pair $1 \le i \le j \le n$ and let $lca(a_i, a_j)$ denote the least common ancestor of nodes a_i and a_j in $CT(A)$. Then*

$$\min_{i \le k \le j} key(a_k) = key(lca(a_i, a_j)).$$

Let L' denote the length of lcp' and t'. We construct a Cartesian tree CT taking positions $i \in \{1, \ldots, L'\}$ as nodes and using values $lcp'(i)$ as keys. For a node v in CT let CT_v denote the subtree rooted at v and $z(v)$ — the number of distinct non-zero values of $t'(u)$ for $u \in CT_v$.

The next reformulation of (LCS) is an immediate consequence of Proposition 4:

Proposition 5. *The length of the longest common substring that appears in at least K input strings is equal to $\max_v lcp'(v)$, where v ranges over all nodes of CT such that $z(v) \ge K$.*

This provides us with the following two problems: how to construct CT from lcp' values and how to compute z values. The first task is easily solvable in $O(L)$ time (see, e.g., [BFC00]); we shall describe this method below.

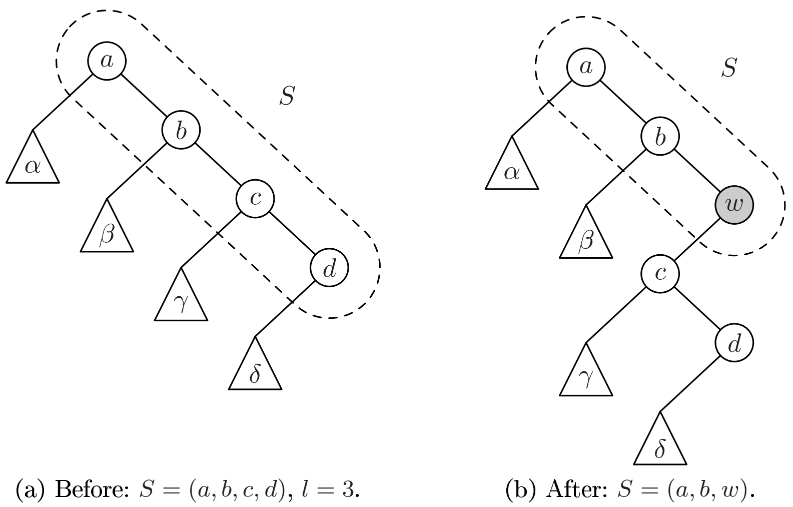
The second task is more involved. Instead of working directly with values of z consider an integer-valued function ζ on the nodes of CT such that

$$z(v) = \sum_{u \in CT_v} \zeta(u).$$

Clearly, ζ is uniquely determined by z and vice versa. We construct ζ and CT incrementally by scanning arrays lcp' and t' and adding, at the i-th step $(1 \le i \le L')$, node i with key $lcp'(i)$ and type $t'(i)$.

The algorithm maintains a stack $S = (S(1), \ldots, S(m))$ that holds nodes of CT contained on the *rightmost path* (that is, the path formed by walking from the root of CT and taking the right child on each step until reaching a null reference). Here m is the length of the rightmost path, $S(1)$ is the root of CT (the bottom of the stack), and $S(m)$ is the last node on the rightmost path (the top of the stack).

Adding the first node 1 to CT is straightforward. Suppose that nodes $1, \ldots, i$ are already added to CT and the corresponding values of ζ are computed. To insert node $i + 1$ into CT the algorithm iteratively pops a sequence of nodes $S(m), S(m-1), \ldots, S(l)$ from the stack as long as the last popped node $S(l)$



S
a
b
c
d
α
β
γ
δ
(a) Before: $S = (a, b, c, d)$, $l = 3$.
S
a
b
w
c
d
α
β
γ
δ
(b) After: $S = (a, b, w)$.

Fig. 2. Inserting a new node w into a Cartesian tree. Here $key(a) \leq key(b) < key(w) \leq key(c) \leq key(d)$.

obeys $lcp'(S(l)) \geq lcp'(i+1)$. In particular, if $lcp'(S(m)) < lcp'(i+1)$ then $l := m+1$ and no nodes are popped. Tree CT is adjusted as follows (see Fig. 2 for an example):

- the right child of $i+1$ is set to null;
- the left child of $i+1$ is set to $S(l)$ if $l \leq m$ or to null if $l > m$;
- node $i+1$ is added to the end of the rightmost path.

The total time that is necessary to perform these operations for node $i+1$ is $O(2 + \Delta m)$, where Δm denotes the decrease of m. By amortization, this easily yields a linear time bound for the total process.

Now we need to explain how the values of ζ are changed by the insertion. If $t'(i+1) = 0$ then it suffices to put $\zeta(i+1) := 0$. Otherwise, let $t'(i+1) \neq 0$. Clearly, we may only need to change values $\zeta(S(1)), \ldots, \zeta(S(l-1))$ (if any) and to initialize $\zeta(i+1)$; other values of ζ correspond to subtrees of CT that are not affected by the insertion.

At any given moment we assign *ranks* to the nodes of CT by the following rule: for each $1 \leq i \leq m$ node $S(i)$ and all nodes in its left subtree are of rank i. For each possible type $j \in \{1, \ldots, N\}$ let $max\text{-}rank(j)$ denote the maximum rank of a node in CT having type j. In case no node of type j exists, we put $max\text{-}rank(j) := -\infty$.

Two cases are possible. Firstly, suppose $l' := max\text{-}rank(t'(i+1)) \geq l$. This means that prior to insertion of node $i+1$ the first node of type $t'(i+1)$ (with regard to the in-order traversal) was occurring in $CT_{S(l)}$. Then, the step is completed by putting $\zeta(i+1) := 0$.

Secondly, let $l' < l$ (in particular, this case applies when $l' = -\infty$). The algorithm assigns $\zeta(i+1) := 1$ and decreases the value $\zeta(S(l'))$ by 1 (if $l' > -\infty$).

It is easy to see that these changes are correct and produce the required values of ζ. However, it remains to explain how the *max-rank* values are computed. To this aim, we need a data structure for maintaining an array $\mathit{max\text{-}rank} = (\mathit{max\text{-}rank}(1), \ldots, \mathit{max\text{-}rank}(N))$ of these values. In addition to the standard read and write requests, this array should also be capable of performing *trimming* as follows: given a value l put

$$\mathit{max\text{-}rank}(j) := \min(\mathit{max\text{-}rank}(j), l) \qquad \text{for all } 1 \leq j \leq N.$$

This operation is invoked to adjust the maximum ranks each time node $S(l)$ is turned into a left child of node $i + 1$.

A possible implementation is based on keeping, for each index $1 \leq j \leq N$, an integer *timestamp* $\mathit{last\text{-}write}(j)$ that keeps the moment of time when this entry was last updated. Also, instead of performing it directly, the algorithm maintains the timestamp *last-trim* of the latest trimming operation (together with the corresponding trimming parameter l). Now to get the actual value $\mathit{max\text{-}rank}(j)$ one compares $\mathit{last\text{-}write}(j)$ with *last-trim* to see if trimming applies to the currently stored value. With this implementation, each read access, write access or trimming takes $O(1)$ time.

Therefore, we can construct CT and compute the values of ζ in linear time. Then, the values of z are computed from ζ in a bottom-up fashion. Simultaneously, for each possible value of $z(i)$ we accumulate the largest value of $\mathit{lcp}'(i)$ and construct the array

$$\mathit{max\text{-}lcp}'(k) := \max\bigl(\mathit{lcp}'(i) : z(i) = k\bigr) \qquad \text{for all } 2 \leq k \leq N.$$

The length of the longest common substring corresponding to a certain value of K is

$$M(K) := \max\bigl(\mathit{max\text{-}lcp}'(k) : k \geq K\bigr) \qquad \text{for all } 2 \leq K \leq N. \tag{4}$$

Hence, all values $M(2), \ldots, M(N)$ may be computed by an obvious recurrence.

The longest substrings themselves may also be easily extracted. For each node v of CT enumerate the nodes in CT_v via an in-order traversal and put u^- (resp. v^+) to be the first (resp. the last) node in CT_v such that $t'(v^-) \neq 0$ (resp. $t'(v^+) \neq 0$). Clearly, computing all nodes v^+ and v^- takes linear time.

Now consider a fixed value of K. Let the maximum in (4) be attained by some $k \geq K$. Next, let v denote a node in CT such that $z(v) = k$. Nodes v^- and v^+ give rise to a k-good segment $\Delta = [\Delta^-, \Delta^+] \subseteq [N + 1, L]$ obeying the equality $\min\left(\mathit{lcp}(j) : \Delta^- \leq j < \Delta^+\right) = M(K)$. The corresponding longest common substring is constructed from Δ in the same way as in Section 3.

Acknowledgements

The authors are thankful to the students of Department of Mathematical Logic and Theory of Algorithms (Faculty of Mechanics and Mathematics, Moscow

State University), and also to Maxim Ushakov and Victor Khimenko (Google Moscow) for discussions.

References

[AUH74] Aho, A.V., Ullman, J.D., Hopcroft, J.E.: The design and analysis of computer algorithms. Addison-Wesley, Reading, MA (1974)

[BFC00] Bender, M., Farach-Colton, M.: The LCA problem revisited, pp. 88–94 (2000)

[CR03] Crochemore, M., Rytter, W.: Jewels of stringology. World Scientific Publishing Co. Inc., River Edge, NJ (2003)

[Gus97] Gusfield, D.: Algorithms on strings, trees, and sequences: Computer science and computational biology. Cambridge University Press, New York, NY, USA (1997)

[KLA$^+$01] Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)

[KS03] Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719. Springer, Heidelberg (2003)

[MM90] Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. In: SODA 1990: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 319–327 (1990)

[PST07] Puglisi, S.J., Smyth, W.F., Turpin, A.H.: A taxonomy of suffix array construction algorithms. ACM Comput. Surv. 39(2), 4 (2007)

[Ukk95] Ukkonen, E.: On-line construction of suffix trees. Algorithmica 14(3), 249–260 (1995)

Logic and Rational Languages of Words Indexed by Linear Orderings

Nicolas Bedon[1], Alexis Bès[2], Olivier Carton[3], and Chloé Rispal[1]

[1] Université Paris-Est and CNRS
Laboratoire d'informatique de l'Institut Gaspard Monge, CNRS UMR 8049
Nicolas.Bedon@univ-mlv.fr, Chloe.Rispal@univ-mlv.fr
[2] Université Paris-Est, LACL
bes@univ-paris12.fr
[3] Université Paris 7 and CNRS
LIAFA, CNRS UMR 7089
Olivier.Carton@liafa.jussieu.fr

Abstract. We prove that every rational language of words indexed by linear orderings is definable in monadic second-order logic. We also show that the converse is true for the class of languages indexed by countable scattered linear orderings, but false in the general case. As a corollary we prove that the inclusion problem for rational languages of words indexed by countable linear orderings is decidable.

1 Introduction

In [4,6], Bruyère and Carton introduce automata and rational expressions for words on linear orderings. These notions unify naturally previously defined notions for finite words, left- and right-infinite words, bi-infinite words, and ordinal words. They also prove that a Kleene-like theorem holds when the orderings are restricted to countable scattered linear orderings; recall that a linear ordering is scattered if it does not contain any dense sub-ordering. Since [4], the study of automata on linear orderings was carried on in several papers. The emptiness problem and the inclusion problem for rational languages is addressed in [7,11]. The papers [5,2] provide a classification of rational languages with respect to the rational operations needed to describe them. Algebraic characterizations of rational languages are presented in [2,21,20]. The paper [3] introduces a new rational operation of shuffle of languages which allows to deal with dense orderings, and extends the Kleene-like theorem proved in [4] to languages of words indexed by all linear orderings.

In this paper we are interested in connections between rational languages and languages definable in a logical formalism. The main motivations are, on one hand, to extend the classical results to the case of linear orderings, and on the other hand to get a better understanding of monadic second order (shortly: MSO) theories of linear orderings. Let us recall the state-of-the-art. In his seminal paper [8], Büchi proved that rational languages of finite words coincide with languages definable in the weak MSO theory of $(\omega, <)$, which allowed him to prove decidability of this theory. In [9] he proved that a similar equivalence holds between rational languages of infinite words of length ω and languages definable in the MSO theory of $(\omega, <)$. The result was then extended to languages of words indexed by a countable ordinal [10]. What can be said about MSO theories for linear orderings beyond ordinals ? Using the automata technique, Rabin proved decidability of the MSO theory of the binary tree [19], from which he deduces decidability of the MSO theory of $\mathbb{Q}$, which in turn implies decidability of the MSO theory of countable linear orderings. Shelah [24] (see also [12,26]) improved model-theoretical techniques that allow him to reprove almost all known decidability results about MSO theories, as well as new decidability results for the case of linear orderings. He proved in particular that the MSO theory of $\mathbb{R}$ is undecidable. Shelah's decidability method is model-theoretical, and up to now no corresponding automata techniques are known. This led Thomas to ask [26] whether there is an appropriate notion of automata for words indexed by linear orderings beyond the ordinals. As mentioned in [4], this question was an important motivation for the introduction of automata over words indexed by linear orderings.

In this paper we study rational languages in terms of definability in MSO logic. Our main result is that, assuming the axiom of choice, every rational language of words indexed by linear orderings is definable in MSO logic. The proof does not rely on the classical encoding of an accepting run of an automaton accepting the language, but on an induction on the rational expression denoting the language. As a corollary we prove that the inclusion problem for rational languages of countable linear orderings is decidable, which extends [7] where the result was proved for countable scattered linear orderings. We also study the converse problem, i.e. whether every MSO-definable language of words indexed by linear orderings is rational. A key argument in order to prove this kind of results is the closure of the class of rational languages under complementation. Carton and Rispal [21] proved (using semigroup theory) that the class of rational languages of words indexed by countable scattered orderings is closed under complementation; building on this, we prove that every MSO-definable language of words indexed by countable scattered linear orderings is rational, giving thus the equivalence between rational expressions and MSO logic in this case. On the other hand we show that for every finite alphabet A the language of words over A indexed by scattered orderings is not rational, while its complement is. This proves that the class of rational languages of words over linear orderings is not closed under complementation, and as a corollary of the previous results that this class is strictly included in the class of MSO-definable languages.

The paper is organized as follows: we recall in Section 2 some useful definitions related to linear orderings. Section 3 introduces rational expressions for words over linear orderings. Section 4 recalls useful notions related to MSO. In Section 5 we show that rational languages are MSO-definable. Section 6 deals with the converse problem. We conclude the paper with some open questions.

2 Linear Orderings

In this section we recall useful definitions and results about linear orderings. A good reference on the subject is Rosenstein's book [22].

A *linear ordering* J is an ordering $<$ which is total, that is, for any $j \neq k$ in J, either $j < k$ or $k < j$ holds. Given a linear ordering J, we denote by $-J$ the *backwards* linear ordering obtained by reversing the ordering relation. For instance, $-\omega$ is the backwards linear ordering of ω which is used to index the so-called left-infinite words.

The sum of orderings is concatenation. Let J and K_j for $j \in J$, be linear orderings. The linear ordering $\sum_{j\in J} K_j$ is obtained by concatenation of the orderings K_j with respect to J. More formally, the *sum* $\sum_{j\in J} K_j$ is the set L of all pairs (k, j) such that $k \in K_j$. The relation $(k_1, j_1) < (k_2, j_2)$ holds if and only if $j_1 < j_2$ or ($j_1 = j_2$ and $k_1 < k_2$ in K_{j_1}). The sum of two orderings K_1 and K_2 is denoted $K_1 + K_2$.

Given two elements j, k of a linear ordering J, we denote by $[j; k]$ the interval $[\min(j, k), \max(j, k)]$. The elements j and k are called *consecutive* if $j < k$ and if there is no element $i \in J$ such that $j < i < k$. An ordering is *dense* if it contains no pair of consecutive elements. More generally, a subset $K \subset J$ is *dense* in J if for any $j, j' \in J$ such that $j < j'$, there is $k \in K$ such that $j < k < j'$.

A *cut* of a linear ordering J is a pair (K, L) of intervals such that $J = K \cup L$ and such that for any $k \in K$ and $l \in L$, $k < l$. The set of all cuts of the ordering J is denoted by $\hat{J}$. This set $\hat{J}$ can be linearly ordered by the relation defined by $c_1 < c_2$ if and only if $K_1 \subsetneq K_2$ for any cuts $c_1 = (K_1, L_1)$ and $c_2 = (K_2, L_2)$. This linear ordering can be extended to $J \cup \hat{J}$ by setting $j < c_1$ whenever $j \in K_1$ for any $j \in J$.

The consecutive elements of $\hat{J}$ deserve some attention. For any element j of J, define two cuts c_j^- and c_j^+ by $c_j^- = (K, \{j\} \cup L)$ and $c_j^+ = (K \cup \{j\}, L)$ where $K = \{k \mid k < j\}$ and $L = \{k \mid j < k\}$. It can be easily checked that the pairs of consecutive elements of $\hat{J}$ are the pairs of the form (c_j^-, c_j^+).

A *gap* of an ordering J is a cut (K, L) such that $K \neq \varnothing$, $L \neq \varnothing$, K has no greatest element and L has no least element. An ordering J is *complete* if it has no gap.

3 Words and Rational Expressions

Given a finite alphabet A, a *word* $(a_j)_{j\in J}$ is a function from J to A which maps any element j of J to a letter a_j of A. We say that J is the *length* $|x|$ of the word x. For instance, the *empty word* ε is indexed by the empty linear ordering $J = \varnothing$. Usual finite words are the words indexed by finite orderings $J = \{1, 2, \ldots, n\}$, $n \geq 0$. A word of length $J = \omega$ is usually called an ω-word or an infinite word. A word of length $\zeta = -\omega + \omega$ is a sequence $\ldots a_{-2}a_{-1}a_0a_1a_2 \ldots$ of letters which is usually called a bi-infinite word.

The sum operation on linear orderings leads to a notion of product of words as follows. Let J and K_j for $j \in J$, be linear orderings. Let $x_j = (a_{k,j})_{k\in K_j}$ be a word of length K_j, for any $j \in J$. The *product* $\prod_{j \in J} x_j$ is the word z of length $L = \sum_{j \in J} K_j$ equal to $(a_{k,j})_{(k,j) \in L}$. For instance, the word $a^\zeta = b^{-\omega} a^\omega$ of length ζ is the product of the two words $b^{-\omega}$ and a^ω of length $-\omega$ and ω respectively.

We now recall the notion of rational sets of words indexed by linear orderings as defined in [4,3]. The rational operations include of course the usual Kleene operations for finite words which are the union $+$, the concatenation $\cdot$ and the star operation $*$. They also include the omega iteration ω usually used to construct ω-words and the ordinal iteration $\sharp$ introduced by Wojciechowski [28] for ordinal words. Four new operations are also needed: the backwards omega iteration $-\omega$, the backwards ordinal iteration $-\sharp$, a binary operation denoted $\diamond$ which is a kind of iteration for all orderings, and finally a shuffle operation which allows to deal with dense linear orderings.

We respectively denote by $\mathcal{N}$, $\mathcal{O}$ and $\mathcal{L}$ the classes of finite orderings, the class of all ordinals and the class of all linear orderings. For an ordering J, we denote by $\hat{J}^*$ the set $\hat{J} \setminus \{(\varnothing, J), (J, \varnothing)\}$ where $(\varnothing, J)$ and $(J, \varnothing)$ are the first and last cut. Given two sets X and Y of words, define

$$\begin{aligned}
X + Y &= \{z \mid z \in X \cup Y\},\\
X \cdot Y &= \{x \cdot y \mid x \in X, y \in Y\},\\
X^* &= \{\textstyle\prod_{j \in \{1,\ldots,n\}} x_j \mid n \in \mathcal{N}, x_j \in X\},\\
X^\omega &= \{\textstyle\prod_{j \in \omega} x_j \mid x_j \in X\},\\
X^{-\omega} &= \{\textstyle\prod_{j \in -\omega} x_j \mid x_j \in X\},\\
X^\sharp &= \{\textstyle\prod_{j \in \alpha} x_j \mid \alpha \in \mathcal{O}, x_j \in X\},\\
X^{-\sharp} &= \{\textstyle\prod_{j \in -\alpha} x_j \mid \alpha \in \mathcal{O}, x_j \in X\},\\
X \diamond Y &= \{\textstyle\prod_{j \in J \cup \hat{J}^*} z_j \mid J \in \mathcal{L}, z_j \in X \text{ if } j \in J \text{ and } z_j \in Y \text{ if } j \in \hat{J}^*\}.
\end{aligned}$$

We denote by $A^\diamond$ the set of words over A indexed by linear orderings. Note that we have $A^\diamond = (A \diamond \varepsilon) + \varepsilon$.

For every finite alphabet A, every $n \geq 1$, and all languages $L_1, \ldots, L_n \subseteq A^\diamond$, we define $\mathrm{sh}(L_1, \ldots, L_n)$ as the set of words $w \in A^\diamond$ that can be written as $w = \prod_{j \in J} w_j$, where J is a complete linear ordering without first and last element, and there exists a partition $(J_1, \ldots, J_n)$ of J such that all J_i's are dense in J, and for every $j \in J$, if $j \in J_k$ then $w_j \in L_k$.

An abstract *rational expression* is a well-formed term of the free algebra over $\{\varnothing\} \cup A$ with the symbols denoting the rational operations as function symbols. Each rational expression denotes a set of words which is inductively defined by the above definitions of the rational operations. A set of words is *rational* if it can be denoted by a rational expression. As usual, the dot denoting concatenation is omitted in rational expressions.

Automata which recognize languages of words indexed by linear orderings were introduced in [4]. In the latter paper a Kleene-like theorem was also shown for the special case of languages of words indexed by countable scattered linear orderings. Recall that a linear ordering is scattered if it does not contain any dense sub-ordering. The general case of words indexed by all linear orderings was proven in [3]. We refer e.g. to these papers for more details about automata; in this paper we shall deal only with rational expressions.

4 Monadic Second-Order Logic

In this section we recall useful elements of monadic second-order logic, and settle some notations. For more details about MSO logic we refer e.g. to Thomas' survey paper [27]. Monadic second-order logic is an extension of first-order logic that allows to quantify over elements as well as subsets of the domain of the structure.

Given a signature $\mathcal{L}$, one can define the set of *MSO-formulas* over $\mathcal{L}$ as well-formed formulas that can use first-order variable symbols $x, y, \ldots$ interpreted as elements of the domain of the structure, monadic second-order variable symbols $X, Y, \ldots$ interpreted as subsets of the domain, symbols from $\mathcal{L}$, and a new binary predicate $x \in X$ interpreted as "x belongs to X". We call *MSO sentence* any MSO formula without free variable. As usual, we will often confuse logical symbols with their interpretation. Moreover we will use freely abreviations such as $\exists x \in X\ \varphi$, $\forall X \subseteq Y\ \varphi$, $\exists! t\varphi$, and so on.

Given a signature $\mathcal{L}$ and an $\mathcal{L}-$structure M with domain D, we say that a relation $R \subseteq D^m \times (2^D)^n$ is MSO-definable in M if and only if there exists a MSO-formula over $\mathcal{L}$, say $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_n)$ which is true in M if and only if $(x_1, \ldots, x_m, X_1, \ldots, X_n)$ is interpreted by an $(m+n)-$tuple of R.

Given a finite alphabet A, let us consider the signature $\mathcal{L}_A = \{<, (P_a)_{a \in A}\}$ where $<$ is a binary relation symbol and the P_a's are unary predicates (over first-order variables). One can associate to every word $w = (a_j)_{j \in J}$ over A (where $a_j \in A$ for every j) the $\mathcal{L}_A-$structure $M_w = (J; <; (P_a)_{a \in A})$ where $<$ is interpreted as the ordering over J, and $P_a(x)$ holds if and only if $a_x = a$. In order to take into account the case $w = \varepsilon$, which leads to the structure M_ε which has an empty domain, we will allow structures to be empty. Given a MSO sentence φ over the signature $\mathcal{L}_A$, we define the language L_φ as the set of words w over A such that $M_w \models \varphi$. We will say that a language L over A is definable in MSO logic (or *MSO-definable*) if and only if there exists a MSO-sentence φ over the signature $\mathcal{L}_A$ such that $L = L_\varphi$.

5 Rational Languages are MSO-Definable

Büchi's proof [8] that every rational language L of finite words is definable in MSO logic relies on the encoding of an accepting run of an automaton $\mathcal{A}$ recognizing L. Given a word w, one expresses the existence of a successful path in $\mathcal{A}$ labeled by w, by encoding each state of the path on a position of w, which is possible because - up to a finite number of elements - the underlying ordering of the path is the same as the one of the word. This property still holds when one considers infinite words of length ω, and more generally of any ordinal length. However it does not hold anymore for words indexed by all linear orderings, since for a word of length J, the path of the automaton is defined on the set $\hat{J}$ of cuts of J (see [4]), and in general $\hat{J}$ can be quite different from J - consider e.g. the case $J = \mathbb{Q}$ for which J is countable while $\hat{J}$ is not. Thus in our situation there seems to be no natural extension of the classical Büchi's encoding technique. In order to overcome this issue, we use a proof by induction over rational expressions.

Proposition 1. *(Assuming the Axiom of Choice) For every finite alphabet A and every language $L \subseteq A^\diamond$, if L is rational then it is definable in monadic second-order logic.*

Let us give a quick outline of the proof. One proves that for every rational language L there exists a MSO formula $\varphi(X)$ over the signature $\mathcal{L}_A$ such that for every word w over A indexed by some linear ordering J, we have $w \in L$ if and only if M_w satisfies φ when X is interpreted by J. This yields Proposition 1 since every rational language L can then be defined by the MSO sentence $\exists X(\varphi(X) \wedge \forall x\ x \in X)$. The proof proceeds by induction on the rational expression denoting L; this approach is not new, see e.g. [15] where it is used in the case of finite words. The case of the empty word, as well as the one of singletons, union and product operations, are easy. For the other rational operations one has to find a way to express that the set X can be partitioned in some way in intervals. Consider for instance the case of the ω-power operation. Assume that L is definable by the MSO formula $\varphi(X)$. Then L^ω could be defined by a MSO formula which express the existence of a partition of X in a sequence $(Y_i)_{i\in\omega}$ of intervals Y_i such that $\varphi(Y_i)$ holds for every i. Since the existence of such a partition cannot be expressed directly in MSO, one reformulates this property as the existence of a partition of X in two subsets X_1, X_2 such that every Y_i corresponds to an interval which consists in elements of X_1 only, or elements of X_2 only, and which is maximal for inclusion among such intervals. These maximal intervals are definable in MSO in terms of X, X_1 and X_2, and moreover one can express that the order type of the sequence of these maximal intervals is ω. This allows to find a MSO formula which defines L^ω. The idea of interleaving finitely many subsets in order to encode some partition of X in intervals is also used for the other rational operations.

We illustrate Proposition 1 with several examples, over the alphabet $A = \{a, b\}$.

Example 1. Let L_1 be the set of words $w = (a_j)_{j\in J}$ (with $a_j \in A$) such that J has a least element j_0 with $a_{j_0} = a$, and $a_j = b$ for some $j \in J$. This language can be represented by the rational expression $aA^\diamond bA^\diamond$. It is also MSO-definable by the sentence

$$\exists x \exists y (P_a(x) \wedge P_b(y) \wedge \neg \exists z\ z < x).$$

Example 2. Let L_2 be the set of words indexed by a linear ordering J such that the set of positions $j \in J$ for which $w_j = a$ (respectively $w_j = b$) is dense in J. This language can be represented by the rational expression $\mathrm{sh}(a, b, \varepsilon)$. It is MSO-definable by the sentence

$$\forall x \forall y (x < y \Longrightarrow \exists z \exists t (x < z < y \wedge P_a(z) \wedge x < t < y \wedge P_b(t))).$$

Example 3. The language $L_3 = a^\omega a^{-\omega}$ is definable in MSO by the formula

$$\begin{aligned} &\forall x P_a(x) \wedge \exists X_1 \exists X_2 (\forall x\ (x \in X_1 \leftrightarrow x \notin X_2) \\ &\quad \wedge \forall x \forall y ((x \in X_1 \wedge y \in X_2) \to x < y) \\ &\quad \wedge \mathrm{Omega}(X_1) \wedge \mathrm{MinusOmega}(X_2)) \end{aligned}$$

where Omega(X_1) (respectively MinusOmega(X_2)) expresses that the order type of X_1 is ω (respectively $-\omega$). One can show that the predicates Omega and MinusOmega are MSO-definable.

Example 4. The language L_4 of words whose length is a complete ordering can be represented by the rational expression $(\varepsilon + \mathrm{sh}(a+b)) \diamond (a+b)$. It is also MSO-definable by the sentence

$$\forall Y ((\exists x \varphi(x, Y)) \Longrightarrow (\exists x (\varphi(x, Y) \wedge \forall z (\varphi(z, Y) \Longrightarrow x \leq z))))$$

where $\varphi(x, Y)$ is an abbreviation for $\forall y \in Y\ y \leq x$.

Example 5. Consider the language L_5 of words over A whose length is a non-scattered ordering. It follows from [22, chap. 4] that L_5 consists in words w which can be written as $w = \prod_{k \in K} w_k$ where K is a dense ordering, and $w_k \neq \varepsilon$ for every $k \in K$. From this decomposition one can deduce that a convenient rational expression for L_5 is $\mathrm{sh}(A^\diamond(a+b)A^\diamond, \varepsilon)$. The language L_5 can also be defined by the following MSO formula

$$\begin{aligned} &\exists X (\exists x_1 \in X\ \exists x_2 \in X\ x_1 < x_2 \\ &\quad \wedge \forall y_1 \in X\ \forall y_2 \in X (y_1 < y_2 \Longrightarrow \exists z \in X (y_1 < z \wedge z < y_2))). \end{aligned}$$

Combining Proposition 1 and Rabin's result [19] about the decidability of the MSO theory of countable linear orderings, yields the following result.

Corollary 1. *The inclusion problem for rational languages of words over countable linear orderings is decidable.*

This improves [7] where the authors prove the result for words over *scattered* countable linear orderings.

6 MSO-Definable Languages vs Rational Languages

In this section we consider the problem whether MSO-definable languages are rational. The answer is positive if we consider words indexed by countable scattered linear orderings. Indeed we can prove the following result.

Proposition 2. *For every finite alphabet A and every language L of words over A indexed by countable scattered linear orderings, L is rational if and only if it is MSO-definable.*

As for the finite words case, the proof that $L = L_\phi$ for some MSO sentence ϕ implies that L is rational relies on the construction of an automaton accepting L, by induction on the structure of ϕ. The effectiveness of this construction, together with the decidability of the emptiness problem for automata on words indexed by countable scattered linear orderings [11], yield the following corollary.

Corollary 2. *The monadic second order theory of countable scattered linear orderings is decidable.*

Note that the latter result is also a direct consequence of Rabin's result [19] about the decidability of the MSO theory of countable linear orderings (the property "to be scattered" is expressible in the latter theory).

Proposition 2 does not hold anymore if we consider languages of words indexed by all linear orderings. Indeed consider, for every finite alphabet A, the language S_A of words over A indexed by scattered linear orderings, i.e. the complement of the language L_5 of Example 5. Since L_5 is definable in MSO, the same holds for S_A. However one can show by a pumping argument that no automaton can recognize S_A. This fact together with the equivalence between rational languages and languages recognizable by automata [3] yield the following result.

Proposition 3. *For every finite alphabet A, the language S_A of words over A indexed by scattered linear orderings is not rational.*

On the other hand the language L_5 was shown to be rational. Thus we can deduce the following result from Propositions 1 and 3.

Corollary 3. *For every finite alphabet A, the class of rational languages over A is not closed under complementation, and is strictly included in the class of MSO-definable languages.*

7 Open Questions

Let us mention some related problems. It would be interesting to determine which syntactic fragment of the monadic second-order theory captures rational languages. The proof of Proposition 1, which uses an induction on the rational expression, gives rise to defining formulas where the alternation of (second-order) quantifiers is unbounded. However if we consider the special form of formulas used in the proof, together with classical techniques of re-using variables we can show that every rational language can be defined by MSO formulas of the form $\forall X_1 \ldots \forall X_m \exists Y_1 \ldots \exists Y_n \forall Z_1 \ldots \forall Z_p\ \varphi$, where φ has no monadic second-order quantifier. We already know that the $\forall\exists\forall$-fragment of MSO contains non-rational languages, since by Proposition 3 the language of words indexed by scattered orderings, which can be defined by a $\forall$-formula, is not rational. Thus it would be interesting to know the expressive power of smaller syntactic fragments with respect to rational languages, and in particular the existential fragment. Recall that for the MSO theory of ω (and more generally any countable ordinal) the existential fragment is equivalent in terms of expressive power to the full theory. This comes from the fact that the formula encoding a successful run of an automaton is existential (for second-order variables). In our context the existential fragment does not capture all rational languages, as one can prove e.g. that the language a^ω is not existentially definable. We conjecture that the class of languages definable by existential formulas is strictly included in the class of rational languages.

Another related problem is the expressive power of first-order logic. For finite words the McNaughton-Papert Theorem [14] shows that sets of finite words defined by first-order sentences coincide with star-free languages. Schützenberger gave another characterization of star-free sets, based on the equivalence of automata and an algebraic formalism, the finite monoids, for the definition of sets of finite words. He proved that the star-free sets are exactly those definable by a finite group-free monoid [23]. This double equivalence of Schützenberger, McNaughton and Papert was already extended to the infinite words by Ladner [13], Thomas [25] and Perrin [17], to words whose letters are indexed by all the relative integers by Perrin and Pin [17,16,18], and to the countable ordinals case by Bedon [1]. We already know [2] that a language of countable scattered linear orderings is star-free if and only if its syntactic $\diamond$-semigroup is finite and aperiodic. However, one can show that first-order definable languages of countable scattered linear orderings do not coincide any more with star-free and aperiodic ones [2,22]. It would be interesting to characterize languages which are first-order definable.

References

1. Bedon, N.: Logic over words on denumerable ordinals. Journal of Computer and System Science 63(3), 394–431 (2001)
2. Bedon, N., Rispal, C.: Schützenberger and Eilenberg theorems for words on linear orderings. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 134–145. Springer, Heidelberg (2005)
3. Bès, A., Carton, O.: A Kleene theorem for languages of words indexed by linear orderings. Int. J. Found. Comput. Sci. 17(3), 519–542 (2006)
4. Bruyère, V., Carton, O.: Automata on linear orderings. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 236–247. Springer, Heidelberg (2001)
5. Bruyère, V., Carton, O.: Hierarchy among automata on linear orderings. In: Baeza-Yate, R., Montanari, U., Santoro, N. (eds.) Foundation of Information technology in the era of network and mobile computing, pp. 107–118. Kluwer Academic Publishers, Dordrecht (2002)
6. Bruyère, V., Carton, O.: Automata on linear orderings. J. Comput. System Sci. 73(1), 1–24 (2007)
7. Bruyère, V., Carton, O., Sénizergues, G.: Tree automata and automata on linear orderings. In: Harju, T., Karhumäki, J. (eds.) WORDS 2003, pp. 222–231. Turku Center for Computer Science (2003)
8. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik und grundl. Math. 6, 66–92 (1960)

9. Büchi, J.R.: On a decision method in the restricted second-order arithmetic. In: Proc. Int. Congress Logic, Methodology and Philosophy of science, Berkeley 1960, pp. 1–11. Stanford University Press (1962)
10. Büchi, J.R.: Transfinite automata recursions and weak second order theory of ordinals. In: Proc. Int. Congress Logic, Methodology, and Philosophy of Science, Jerusalem 1964, pp. 2–23. North Holland, Amsterdam (1965)
11. Carton, O.: Accessibility in automata on scattered linear orderings. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 155–164. Springer, Heidelberg (2002)
12. Gurevich, Y.: Monadic second-order theories. In: Barwise, J., Feferman, S. (eds.) Model-Theoretic Logics. Perspectives in Mathematical Logic, pp. 479–506. Springer, Heidelberg (1985)
13. Ladner, R.E.: Application of model theoretic games to discrete linear orders and finite automata. Inform. Control 33 (1977)
14. McNaughton, R., Papert, S.: Counter free automata. MIT Press, Cambridge, MA (1971)
15. Michaux, C., Point, F.: Les ensembles k-reconnaissables sont définissables dans $< N, +, V_k >$ (the k-recognizable sets are definable in $< N, +, V_k >$). C. R. Acad. Sci. Paris Sér. I (303), 939–942 (1986)
16. Perrin, D.: An introduction to automata on infinite words. In: Perrin, D., Nivat, M. (eds.) Automata on Infinite Words. LNCS, vol. 192, pp. 2–17. Springer, Heidelberg (1984)
17. Perrin, D.: Recent results on automata and infinite words. In: Chytil, M.P., Koubek, V. (eds.) MFCS 1984. LNCS, vol. 176, pp. 134–148. Springer, Heidelberg (1984)
18. Perrin, D., Pin, J.E.: First order logic and star-free sets. J. Comput. System Sci. 32, 393–406 (1986)
19. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 141, 1–35 (1969)
20. Rispal, C.: Automates sur les ordres linéaires: Complémentation. PhD thesis, University of Marne-la-Vallée, France (2004)
21. Rispal, C., Carton, O.: Complementation of rational sets on countable scattered linear orderings. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 381–392. Springer, Heidelberg (2004)
22. Rosenstein, J.G.: Linear orderings. Academic Press, New York (1982)
23. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Inform. Control 8, 190–194 (1965)
24. Shelah, S.: The monadic theory of order. Annals of Mathematics 102, 379–419 (1975)
25. Thomas, W.: Star free regular sets of ω-sequences. Inform. Control 42, 148–156 (1979)
26. Thomas, W.: Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) Structures in Logic and Computer Science. LNCS, vol. 1261, pp. 118–143. Springer, Heidelberg (1997)
27. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. III, pp. 389–455. Springer, Heidelberg (1997)
28. Wojciechowski, J.: Finite automata on transfinite sequences and regular expressions. Fundamenta informaticæ 8(3-4), 379–396 (1985)

Complexity of the Bollobás-Riordan Polynomial

Exceptional Points and Uniform Reductions

Markus Bläser[1], Holger Dell[2,⋆], and Johann A. Makowsky[3,⋆⋆]

[1] Saarland University, Saarbrücken, Germany
mblaeser@cs.uni-sb.de
[2] Humboldt University of Berlin, Berlin, Germany
dell@informatik.hu-berlin.de
[3] Technion-Israel Institute of Technology, Haifa, Israel
janos@cs.technion.ac.il

Abstract. The coloured Tutte polynomial by Bollobás and Riordan is, as a generalization of the Tutte polynomial, the most general graph polynomial for coloured graphs that satisfies certain contraction-deletion identities. Jaeger, Vertigan, and Welsh showed that the classical Tutte polynomial is #**P**-hard to evaluate almost everywhere by establishing reductions along curves and lines.

We establish a similar result for the coloured Tutte polynomial on integral domains. To capture the algebraic flavour and the uniformity inherent in this type of result, we introduce a new kind of reductions, *uniform algebraic reductions*, that are well-suited to investigate the evaluation complexity of graph polynomials. Our main result identifies a small, algebraic set of exceptional points and says that the evaluation problem of the coloured Tutte is equivalent for all non-exceptional points, under polynomial-time uniform algebraic reductions.

1 Introduction

Graph polynomials map directed or undirected graphs to polynomials in one or more variables, such that this mapping is invariant under graph isomorphisms. Their purpose is to study the combinatorial properties of graphs using algebraic and analytic properties of the associated polynomials. Probably the most famous graph polynomials are the *chromatic polynomial* $\chi(G; x)$ and its generalization, the *Tutte polynomial* $T(G; x, y)$. The chromatic polynomial is the polynomial in the variable x that counts the number of proper x-colourings of a given undirected graph (cf. [8] for an extensive modern exposition). Surprisingly, $\chi(G; -1)$ has a combinatorial interpretation, too: It counts the number of acyclic orientations [17].

⋆ Partially supported by the Deutsche Forschungsgemeinschaft within the research training group “Methods for Discrete Structures” (GRK 1408).

⋆⋆ Partially supported by a grant of the Israel Science Foundation (2007-2010) and the Fund for the Promotion of Research of the Technion - Israel Institute of Technology.

Certain evaluations of the Tutte polynomial T in two variables x and y have interpretations in different fields of combinatorics. For example, $T(G;1,1)$ counts the number of spanning trees, $T(G;1,2)$ counts the number of spanning subgraphs of an undirected graph G. Similarly the number of nowhere-zero flows, acyclic and Eulerian orientations can be obtained. Furthermore, the chromatic polynomial and the Jones polynomial of an alternating link can be derived from the Tutte polynomial via suitable substitutions and very simple algebraic transformations [5,18].

Due to its rich combinatorial content, it is natural to analyze variations and generalization of the Tutte polynomial. Bollobás and Riordan [6] introduce the *coloured Tutte polynomial* and prove that it is the *most general* graph invariant that satisfies certain contraction-deletion identities. Related are the polynomials by Kauffman [11] and Sokal [16]. While the classical Tutte polynomial is in two variables, the coloured Tutte polynomial is defined on edge-coloured graphs, introducing four variables for every colour, and also some additional variables for initial conditions.

The purpose of this paper is the complexity analysis of and the reducibilities between evaluations of the coloured Tutte polynomial. For this, we propose a new kind of reductions, *(uniform) algebraic reductions*, that seems to be more suited for the complexity analysis of graph polynomials: Graph polynomials have two components, a combinatorial one, the graph, and an algebraic one, the values. So far, only the usual polynomial-time many-one or Turing reductions have been used for the complexity analysis. If one wants to talk about evaluations at irrational points like $\sqrt{2}$ or some root of unity –points that have meaningful interpretations for certain graph polynomials– there is no natural way of representing these points in a discrete setting. Jaeger, Vertigan, and Welsh [10] just adjoin the value they are interested in to $\mathbb{Q}$, but also admit that this is an ad hoc solution. Our reductions take care of this issue by also having two parts.

The combinatorial part of our reductions transforms the graph using a usual polynomial-time computable function from $\Sigma^* \to \Sigma^*$, mapping encodings of graphs to encodings of graphs. The algebraic part transforms the evaluation points and values in polynomial time, but these transformations are now restricted to be *rational* mappings and can be naturally extended to $\mathbb{C}$.

Previous Results. Jaeger, Vertigan, and Welsh [10] have shown that, except along one hyperbola and at four special rational and five special complex points, computing the Tutte polynomial is #**P**-hard. To show this, they construct for each non-exceptional evaluation point (a,b) a reduction to a point where evaluating the Tutte polynomial is already known to be #**P**-hard. All their reductions are very similar and depend only, and in some sense uniformly, on the point (a,b). However, this uniformity is not spelled out in their paper.

Recently, Lotz and Makowsky [13] proved that the coloured Tutte polynomial is complete for Valiant's algebraic complexity class **VNP**, and Goldberg and Jerrum [9] showed that the classical Tutte polynomial is inapproximable for large parts of the Tutte plane. Although not spelling out the inherent algebraicity and uniformity, many reductions in graph polynomials are of that type, among them are some reductions in matching polynomials [1], in the interlace polynomial [3], and in the cover polynomial [2].

Our Contribution. In the first main part, we introduce the notion of uniform and non-uniform algebraic reductions which spell out what Jaeger, Vertigan, and Welsh [10] had in mind in capturing combinatorial and algebraic aspects of graph polynomials (Sec. 3). For these reduction types, we prove that "#**P**-complete" graph polynomials can be *uniformly* reduced to any #**P**-hard numerical graph invariant (Sec. 4).

In the second main part, we establish the #**P**-hardness of the coloured Tutte polynomial under *uniform algebraic* reductions on all but a few exceptional evaluation points (Sec. 7 to 9). We also show in Sec. 7 how to carry over the inapproximability results of Goldberg and Jerrum [9] to the coloured Tutte polynomial using a simple approximation-preserving reduction from the classical Tutte polynomial.

The situation at the exceptional points for the coloured Tutte polynomial is less clear than for the classical Tutte polynomial, because of the larger number of possible colours involved. It seems that evaluating the coloured Tutte polynomial at the exceptional points is computable in polynomial time, as is the case in the classical Tutte polynomial. However, in [10] this is proven with sometimes very different proofs, which do not exhibit a common feature.

Our results also confirm the Uniform Difficult Point Conjecture [15] for the coloured Tutte polynomial, and our reductions can be used to analyze graph polynomials in a more general context, as described in [14,15].

2 Preliminaries

Coloured graphs. Let $\mathbb{N} = \{0, 1, \dots\}$. The graphs in this paper are undirected *multigraphs* $G = (V, E)$ with parallel edges and loops allowed. By Λ, we denote a fixed finite set of colours, and $c : E \to \Lambda$ is called *colouring*. We denote by $\mathcal{G}$ the set of all graphs G and by $\mathcal{G}_c$ the set of all coloured graphs (G, c). We write $n(G)$ for the number of vertices, $m(G)$ for the number of edges, and $k(G)$ for the number of connected components of G. Two coloured graphs are called *isomorphic* if there is a bijective mapping on the vertices that transforms one graph into the other, thereby maintaining the colours.

Polynomials. Polynomials $p(x_1, \dots, x_v)$ are elements of a polynomial ring $\mathbb{Q}[x_1, \dots, x_v]$. We write $\mathbb{Q}(x_1, \dots, x_v)$ for its field of fractions.

Any univariate polynomial can be interpolated if sufficiently many point-value pairs are known. For multivariate polynomials, this is not always true, since the points must also be positioned nicely, e.g. in a grid. If, for a bivariate polynomial $p(x, y)$ of maximal degree d, the values at the points (x_α, y_α) for $\alpha = 1, \dots, n$

with $n = (d+1)^2$ are known, p can be interpolated if, in addition, the bivariate Vandermonde matrix V_2 is non-singular:

$$V_2 = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 & \cdots & x_1^i y_1^j & \cdots & x_1^d y_1^d \\ 1 & x_2 & y_2 & x_2y_2 & \cdots & x_2^i y_2^j & \cdots & x_2^d y_2^d \\ \vdots & & & & & & & \vdots \\ 1 & x_n & y_n & x_ny_n & \cdots & x_n^i y_n^j & \cdots & x_n^d y_n^d \end{bmatrix}.$$

Graph invariants. A *graph invariant* is a function $f : \mathcal{G} \to F$, mapping elements from $\mathcal{G}$ to some set F, such that all pairs of isomorphic graphs G and G' have the same image under f. If $F \subseteq \mathbb{Q}$, then f is called a *numeric graph invariant.* A *parameterized numeric graph invariant (PNGI)* is a function $f : \mathcal{G} \times \mathbb{N}^v \to \mathbb{N}$ which is invariant under graph isomorphisms. If, for each $G \in \mathcal{G}$, the function $f(G; \_)$ is a polynomial, then f is called *graph polynomial.* In this case, f has a natural extension to $\mathbb{C}$, which we use sometimes. Graph invariants and PNGI's for coloured graphs are defined in an analogous manner.

The *chromatic polynomial* $\chi(G; x)$ is the polynomial in x with the property that $\chi(G; Q)$, for $Q \in \mathbb{N}$, is the number of ways to colour the vertices of a graph with Q colours such that adjacent vertices have different colours.

3 Uniform Algebraic Reductions for Graph Polynomials

We want to study reducibilities between evaluations of PNGI's, that is, between the parameter-free numeric graph invariants $f(\boldsymbol{k}) := f(\_; \boldsymbol{k})$ for fixed $\boldsymbol{k} \in \mathbb{C}^v$.

It is widely accepted that the Tutte polynomial is combinatorially speaking strictly more expressive than the chromatic polynomial. Under usual polynomial-time Turing reductions, however, both graph polynomial are equivalent since they are both #**P**-complete (if restricted to positive integers) and can thus be reduced to each other. We now introduce a notion of uniform reducibility which comes closer to capture the intuitively accepted hierarchy between graph polynomials and PNGI's.

Definition 1. *Let f and g be two PNGI's. Denote by v and w the numbers of variables of f and g, respectively. Let $\boldsymbol{x} = x_1, \dots, x_v$ and $\boldsymbol{y} = y_1, y_2, \dots$ be distinct variable symbols.*

(i) We say f algebraically reduces to g uniformly*, or $f \preccurlyeq^{\mathrm{P}}_{AU} g$, if there exist*
  (a) a parameterized rational function $A : \mathcal{G} \to \mathbb{Q}(\boldsymbol{x}, \boldsymbol{y})$,
  (b) functions $r : \mathbb{N} \times \mathcal{G} \to \mathcal{G}$, and
  (c) parameterized rational substitutions *$\sigma : \mathbb{N} \to (\mathbb{Q}(\boldsymbol{x}))^w$,*
  all polynomial-time computable, such that, for every $G \in \mathcal{G}$, the following identity holds for all possible values of $\boldsymbol{x}$:

$$f(G; \boldsymbol{x}) = A(G)\Big[y_i := g\big(G_i; \boldsymbol{x}_i\big)\Big],$$

where $G_i = r(i, G)$ and $\boldsymbol{x}_i = \sigma(i)$. The brackets indicate that the variables y_i of the preceding polynomial are substituted by $g(G_i; \boldsymbol{x}_i)$. So basically, for given G, we express $f(G; \boldsymbol{x})$ in terms of a rational expression in $\boldsymbol{x}$ and $y_i = g(G_i; \boldsymbol{x}_i)$, where the $\boldsymbol{x}_i$ are again rational in $\boldsymbol{x}$.

(ii) If all the graph transductions $r(i, \_)$ are the identity we say, that f is a substitution instance *of g and we write $f \preccurlyeq^{\mathrm{p}}_{SUB} g$.*

(iii) We say f (algebraically) parsimoniously many-one reduces *to g in polynomial time, or $f \preccurlyeq^{\mathrm{p}} g$, if $A(G) = y_1$ for all $G \in \mathcal{G}$.*

(iv) We say that f (non-uniformly) algebraically reduces *to g if, for all* fixed *$\boldsymbol{k} \in \mathbb{Q}^v$, the parameter-free graph invariant $f(\boldsymbol{k}) = f(\_; \boldsymbol{k})$ is uniformly reducible to g, that is, $f(\boldsymbol{k}) \preccurlyeq^{\mathrm{p}}_{\mathrm{AU}} g$ for all $\boldsymbol{k}$. So basically, every $\boldsymbol{k}$ has its own $A_{\boldsymbol{k}}$, $r_{\boldsymbol{k}}$, and $\sigma_{\boldsymbol{k}}$.*

(v) A meaningful way of algebraically reducing any *function $h : \Sigma^* \to \mathbb{N}$ to a parameter-free numeric graph invariant g is by mapping the input $x \in \Sigma^*$ to graphs $G_i = r(i, x)$ and then write $h(x)$ as a rational function $A(x)$ in the oracle queries $y_i = g(G_i)$.*

The function r transforms the given graph G into graphs G_i, the function σ transforms the given point $\boldsymbol{x}$ into new points $\boldsymbol{x}_i$. Since the function A runs in polynomial time, only a polynomial number of the variables y_i can be introduced, that is, only a polynomial number of oracle queries $g(G_i; \boldsymbol{x}_i)$ take place. The outputs of these queries are combined using a rational expression in the helper variables y_i, which get later substituted by the $g(G_i; \boldsymbol{x}_i)$.

Our reductions are *uniform* because they are independent of $\boldsymbol{x}$. Our reductions are *algebraic* because the input substitutions before calling oracle g and the processing of the oracle outputs are bound to be rational transformations.

In general, the numbers of variables v and w do not have to be equal. One particularly important case is $w = 0$. Uniform algebraic reductions are similar to straight-line programs with oracle g (cf. [13], e.g.).

It is clear what it means that the functions r are polynomial time computable, and, as long as we work over $\mathbb{Q}$, this is also clear for σ or A, using a binary representation of the inputs. For ease of presentation, we restrict ourselves to $\mathbb{Q}$, but we can easily extend all our results to fields like $\mathbb{R}$ or $\mathbb{C}$ since rational functions over $\mathbb{Q}$ naturally extend to $\mathbb{R}$ or $\mathbb{C}$. Over $\mathbb{R}$ or $\mathbb{C}$, we can use the BSS-model [4] or a uniform variant of Valiant's model (cf. [7], e.g.) to define polynomial-time computable rational functions. Since these are unit cost models, the reductions become more powerful. However, the reductions in this work have the nice feature that their restrictions to $\mathbb{Q}$ are polynomial-time computable in ordinary bit models.

4 Hardness vs. Uniform Reductions

One way to state results about the complexity of a graph polynomial is to give a dichotomy theorem as in [10]: the points are partitioned completely into #**P**-hard points and easy points. But #**P**-hardness alone does not tell us, whether or not we can reduce the evaluation at one point to the evaluation at another point. Furthermore, it is not clear whether hard points capture the whole graph polynomial in a uniform way.

Recall that graph polynomials are functions $f : \mathcal{G} \times \mathbb{N}^v \to \mathbb{N}$. Thus, encoding the input of f in binary, the statement $f \in \#\mathbf{P}$ makes sense. Furthermore, for typical f one often knows some particular point $\boldsymbol{k}_0$ such that the numeric graph invariant $f(\boldsymbol{k}_0) = f(\_;\boldsymbol{k}_0)$ is $\#\mathbf{P}$-hard (in the Tutte polynomial, this might a-priori be the number of three-colourings). For such f, we prove that uniform and non-uniform reducibility to a parameter-free numeric graph invariant and the hardness of that invariant are equivalent.

Theorem 1. *Let $f \in \#\mathbf{P}$ be a graph polynomial in v variables such that $f(\boldsymbol{k}_0)$ is $\#\mathbf{P}$-hard under polynomial-time algebraic reductions, for some $\boldsymbol{k}_0$. Let g be a parameter-free numeric graph invariant. The following statements are equivalent:*

(i) It holds $f \preccurlyeq^{\mathrm{p}}_{\mathrm{A}U} g$.
(ii) For every $\boldsymbol{k}$, it holds $f(\boldsymbol{k}) \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} g$.
(iii) The function g is $\#\mathbf{P}$-hard under polynomial-time algebraic reductions.

Proof. The implications (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) are trivial.

Now we prove (iii) $\Rightarrow$ (i). From the assumptions, we get $f \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} g$, where we see f as a $\#\mathbf{P}$-function and the reduction in the sense of part (v) in Definition 1. Thus, there exist a function $r' : \mathbb{N} \times (\mathcal{G} \times \mathbb{N}^v) \to \mathcal{G}$, mapping some parameter from $\mathbb{N}$ and instances of f to instances of g in polynomial time, and a parameterized rational function $A' : \mathcal{G} \times \mathbb{N}^v \to \mathbb{Q}(y_1, y_2, \ldots)$ such that

$$f(G;\boldsymbol{j}) = A'(G,\boldsymbol{j})\Big[y_i := g\big(G_{i,\boldsymbol{j}}\big)\Big],$$

with $G_{i,\boldsymbol{j}} := r'(i,(G,\boldsymbol{j}))$ holds for all $(G,\boldsymbol{j}) \in \mathcal{G} \times \mathbb{N}^v$. We assume that the reduction is such that, for all G, the non-zero variables y_i are distinct for different $\boldsymbol{j}$, so we can write $y_{i,\boldsymbol{j}}$ for the variable to be replaced by $g(G_{i,\boldsymbol{j}})$.

Let d_G be the maximal degree of the polynomial $f(G;\boldsymbol{x})$. Given $G \in \mathcal{G}$ as an input, we compute $f(G;\boldsymbol{x})$ in a uniform way from g as follows. First we express the values $f(G;\boldsymbol{j})$ in terms of algebraic expressions $A'(G,\boldsymbol{j})$ in the variables $y_{i,\boldsymbol{j}} = g(G_{i,\boldsymbol{j}})$ using the reduction from above, and we do that for all $\boldsymbol{j}$ in the grid $\{0, 1, \ldots, d_G\}^v$. Formally, the graph transductions $r : \mathbb{N} \times \mathcal{G} \to \mathcal{G}$ interpret the parameter as a pair $\langle i, \boldsymbol{j} \rangle \in \mathbb{N}$ encoded as a nonnegative integer: We define the graph transductions to be $r(\langle i,\boldsymbol{j}\rangle, G) := G_{i,\boldsymbol{j}} = r'(i,(G,\boldsymbol{j}))$.

Next we apply multivariate interpolation to the point-value pairs $(\boldsymbol{j}, f(G;\boldsymbol{j}))$ of the grid, that is, we choose $A(G)$ to be an interpolation polynomial. Such a polynomial can be derived from the v-variate Vandermonde matrix V_v with evaluation points $\boldsymbol{j}$ and from the solution vector $\boldsymbol{b}$ with entries $f(G;\boldsymbol{j})$. Using the entries of $V_v^{-1}\boldsymbol{b}$ as the coefficients of a polynomial in $\boldsymbol{x}$, we get an explicit representation of $f(G;\boldsymbol{x})$ in terms of rational expressions in $\boldsymbol{x}$ and g. □

For PNGI's f that are not graph polynomials, the proof above does not work. Furthermore, it is enough to assume that f is in the closure of $\#\mathbf{P}$ under polynomial-time algebraic reductions. Any natural graph polynomial seems to have this weaker property.

If we look at the special case $g = f(\_; \boldsymbol{k'})$ for fixed $\boldsymbol{k'}$, we can answer the question raised in the beginning of this section. Statement (iii) is the kind of hardness result that is widely used for graph polynomials. At first glance, one might conjecture (i) to be a much stronger property. But our theorem says that this intuition is not true if we have #**P**-hardness under algebraic reductions.

5 The Bollobás-Riordan Polynomial

The coloured Tutte polynomial [6] or *Bollobás-Riordan polynomial* is the most general graph polynomial which can be defined by a spanning tree expansion or a contraction-deletion identity.

The Bollobás-Riordan polynomial of a coloured graph G is a polynomial in the variables γ_k, X_λ, Y_λ, x_λ, and y_λ for $k \in \mathbb{N}$ and $\lambda \in \Lambda$. For a graph G, a colouring $c : E \to \Lambda$, and a (bijective) ordering of the edges $\Phi : E \to \{1, \cdots, m\}$, we define the weight of an edge e of colour λ with respect to a spanning forest $F \subset E$ to be

$$w(G, c, \Phi, F, e) = \begin{cases} X_\lambda & \text{if } e \text{ is internally active,} \\ Y_\lambda & \text{if } e \text{ is externally active,} \\ x_\lambda & \text{if } e \text{ is internally inactive,} \\ y_\lambda & \text{if } e \text{ is externally inactive.} \end{cases}$$

Here we say that an edge $\{u, v\} = e \in F$ is internally active if it is the first edge of E (with respect to Φ) that touches the connected components of u and v in $F - e$, and internally inactive, otherwise. An edge $e \in E - F$ is said to be externally active if it is the first edge of the unique cycle in $F \cup e$, and externally inactive, otherwise. The Bollobás-Riordan polynomial is defined as

$$T_{\mathrm{col}}(G, c, \Phi) = \gamma_{k(G)} \sum_{F \subset E(G)} \prod_{e \in E} w(G, c, \Phi, F, e), \tag{1}$$

where the sum is over all spanning forests of G. In order to remove the dependence on the order, computation must be modulo some ideal I'_0. For an arbitrary ideal $I \supset I'_0$, we define the quotient ring in which the coloured Tutte polynomial lives as $\mathcal{R} = \mathbb{Z}\left[\gamma_k, X_\lambda, Y_\lambda, x_\lambda, y_\lambda : k \in \mathbb{N}, \lambda \in \Lambda\right] \big/ I$. In the case that $\mathcal{R}$ is an integral domain, Bollobás and Riordan [6] prove that T_{col} on $\mathcal{R}$ is a graph invariant, i.e. independent from the order Φ if and only if, in the polynomial ring $\mathcal{R}$, it holds either

$$X_\lambda y_\mu - y_\lambda X_\mu = x_\lambda Y_\mu - Y_\lambda x_\mu = x_\lambda y_\mu - y_\lambda x_\mu \tag{2}$$

for all colours λ and μ, or $\gamma_k = 0$ for all k, or $X_\lambda = Y_\lambda = 0$ for all colours λ. (The ideal I'_0 establishes exactly this situation). Let us write $T_{\mathrm{col}}(G, c) = T_{\mathrm{col}}(G, c, \Phi)$ in $\mathcal{R}$ for an arbitrary ordering Φ since the ordering is now negligible. We abbreviate $T_{\mathrm{col}}(G) = T_{\mathrm{col}}(G, c)$.

In what follows, we assume that I is chosen in such a way that $\mathcal{R}$ is an integral domain, that is, if $pq = 0$ then $p = 0$ or $q = 0$ in $\mathcal{R}$, for all $p, q \in \mathcal{R}$. Furthermore, we assume that we are *not* in the second case, that is, we assume $\gamma_k \neq 0$ and either $X_\lambda \neq 0$ or $Y_\lambda \neq 0$ for some k and λ. In the second case we would have $T_{\text{col}}(G) = 0$ for all graphs G which is not very interesting.

The *Tutte polynomial* $T(G; x, y)$ of a graph G is the instance of the Bollobás-Riordan polynomial with $X_\lambda = x$, $Y_\lambda = y$, $x_\lambda = y_\lambda = \gamma_k = 1$ for all λ and k. It is known that $\chi(G; x) = (-1)^{k(G)} T(G; 0, 1 - x)$ holds. The standard #**P**-hardness proof [12] for the chromatic polynomial χ actually uses *algebraic* reductions:

Lemma 1. *The numerical graph invariants $\chi(\_; 0), \chi(\_; 1), \chi(\_; 2)$ are polynomial-time computable, and all other $\chi(\_; Q)$ are #**P**-hard under polynomial-time* algebraic *reductions.*

6 Evaluations

An *evaluation point* $\sigma : \mathcal{R} \to \mathbb{Q}$ is an arbitrary ring homomorphism mapping the variables that may occur in the coloured Tutte polynomial to rationals, such that $\sigma(x_\lambda y_\mu \gamma_k) \neq 0$ for all λ, μ, and k. Note that this restriction excludes only computationally trivial cases that can be eliminated in polynomial time by applying the contraction-deletion identity [6] recursively.

For the values of the variables of colour λ, we write

$$(a, b, c, d) = \sigma|_\lambda := (\sigma(X_\lambda), \sigma(Y_\lambda), \sigma(x_\lambda), \sigma(y_\lambda)).$$

We also write $\sigma|_{\neq\lambda}$ for the tuple of all other values (including $\sigma(\gamma_k)$ for all k).

Let us define three important invariants in the fraction field of $\mathcal{R}$:

$$q_\lambda := (X_\lambda - x_\lambda)/y_\lambda \ , \quad r_\lambda := (Y_\lambda - y_\lambda)/x_\lambda \ , \text{ and } \quad Q_\lambda := q_\lambda r_\lambda \ .$$

For an evaluation point σ, we define $q_\sigma = \sigma(q_\lambda)$, $r_\sigma = \sigma(r_\lambda)$, and $Q_\sigma = \sigma(Q_\lambda)$ for an arbitrary colour λ. Rather surprisingly, the following holds.

Lemma 2. *The values q_σ, r_σ, and Q_σ are well-defined, i.e., independent from the choice of λ.*

Proof. We prove the claim only for q_σ. Using (2), we compute in $\mathcal{R}$: $y_\lambda y_\mu q_\lambda = y_\lambda y_\mu (X_\lambda - x_\lambda)/y_\lambda = X_\lambda y_\mu - x_\lambda y_\mu = y_\lambda X_\mu - y_\lambda x_\mu = y_\lambda y_\mu (X_\mu - x_\mu)/y_\mu = y_\lambda y_\mu q_\mu$. The fraction field of $\mathcal{R}$ is an integral domain, and the claim follows. □

This lemma says that, for evaluation points from $\mathcal{R} \to \mathbb{Q}$, not all value combinations are allowed for the variables. To make this structure more concrete, we define the sets

$$L_{q,r} := \Big\{ (a, b, c, d) \in \mathbb{Q}^4 : qd = a - c \ \text{ and } \ rc = b - d \Big\},$$
$$P_{q,r} := \Big\{ \sigma : \sigma|_\lambda \in L_{q,r} \text{ for all } \lambda \in \Lambda \Big\}.$$

From Lemma 2, we immediately get the following observation.

Lemma 3. *The set $\bigcup_{q,r\in\mathbb{Q}} P_{q,r}$ and the set of all evaluations $\mathcal{R} \to \mathbb{Q}$ are equal.*

We define the counting problem of evaluating the Bollobás-Riordan polynomial as

$$\sigma T_{\mathrm{col}} : \mathcal{G}_c \to \mathbb{Q} \quad \text{with} \quad G \mapsto \sigma T_{\mathrm{col}}(G) := \sigma\big(T_{\mathrm{col}}(G)\big).$$

The numerical graph invariant σT_{col} evaluates the Bollobás-Riordan polynomial of a given coloured graph G over $\mathcal{R}$ at the point σ.

Choosing an appropriate variable substitution in the coloured Tutte polynomial yields the classical Tutte polynomial: We define a ring homomorphism $\varphi : \mathcal{R} \to \mathbb{Q}[x, y]$ with $\varphi(\gamma_k) = 1$ and $\varphi|_\lambda = (x, y, 1, 1)$ for all k and λ. Bollobás and Riordan [6] prove that $\varphi T_{\mathrm{col}}(G) = T(G; x, y)$ holds, where $T(G; x, y)$ is the (classical) Tutte polynomial of G.

7 Simple Reductions

Our first simple reduction shows that we can ignore the choice of $\sigma(\gamma_k)$ for the rest of this paper.

Lemma 4. *Let σ and σ' be two evaluation points with $\sigma|_\lambda = \sigma'|_\lambda$ for all colour λ. It holds $\sigma' T_{col} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{col}$.*

Proof. Using (1) or alternatively (3.12) from [6], we drag γ_k out of the coloured Tutte polynomial, $\sigma' \left(T_{\mathrm{col}}(G)/\gamma_{k(G)}\right) = \sigma \left(T_{\mathrm{col}}(G)/\gamma_{k(G)}\right)$. We get $\sigma' T_{\mathrm{col}}(G) = \big(\sigma'(\gamma_{k(G)})/\sigma(\gamma_{k(G)})\big) \cdot \sigma T_{\mathrm{col}}(G)$. □

Our second simple reduction relies on the homogeneity of variables. From (1) one can see that $T_{\mathrm{col}}(G)$ is homogeneous in the X, Y, x, y-variables, with a summed degree of m in these variables. Furthermore, the variables X and x always have together a summed degree of $n-k$, and Y and y have a summed degree of $m-n+k$, correspondingly. Thus, for every s from the fraction field of $\mathcal{R}$,

$$\eta_s T_{\mathrm{col}}(G) = s^{m-n+k} T_{\mathrm{col}}(G),$$

where η_s is the ring homomorphism with $\eta_s(Y) = s \cdot Y$, $\eta_s(y) = s \cdot y$, and that leaves everything else identical. This gives us the following many-one reduction.

Lemma 5. *For an arbitrary point σ, we have $(\sigma \circ \eta_s) T_{col} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{col}$ for all $s \in \mathbb{Q}$.*

Note that $q_{\sigma\circ\eta_s} = q_\sigma/s$ and $r_{\sigma\circ\eta_s} = r_\sigma \cdot s$ holds, meaning that q_σ and r_σ are in general not invariant under η_s. However, this is true for $Q_\sigma = Q_{\sigma\circ\eta_s}$.

Using the homogeneity reduction, we can easily see the following reduction.

Proposition 1. *Let σ be an evaluation point with $\sigma_\lambda = (a, b, c, d)$ for some colour λ. Then $T(a/c, b/d) \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{col}$.*

Proof. Given input G, we compute $T(G; a/c, b/d)$ from $\sigma T_{\mathrm{col}}(G)$: We assign colour λ to every edge, and we use Lemma 5 and its analogue for the X, x-variables to get $T(G; a/c, b/d) = \sigma T_{\mathrm{col}}(G)/\big(c^{n-k} d^{m-n+k}\big)$. □

This reduction is an approximation-preserving many-one reduction. Therefore, the inapproximability results by Goldberg and Jerrum [9] transfer immediately to the coloured Tutte polynomial. Although it also gives hardness immediately, as well, we prove it independently in the following, because the proof gives some insights into the structure of the Bollobás-Riordan Polynomial.

8 Interpolation Using Parallel and Series Reduction

We use the parallel and series identities from Theorems 7 and 9 in [6] to obtain algebraic reductions for the coloured Tutte polynomial.

When using r_λ or q_λ in the following, we actually always stay in the ring $\mathcal{R}$ since the denominators cancel out.

Lemma 6. *Let G be a coloured graph, let λ be a colour, and let $G^{\alpha\text{-}fat\text{-}\lambda}$ be the graph obtained from G by replacing each edge of colour λ by α parallel edges of the same colour λ.*

Then $f_{\alpha,\lambda}\big(T_{col}(G)\big) = T_{col}\big(G^{\alpha\text{-}fat\text{-}\lambda}\big)$ where $f_{\alpha,\lambda}$ is the unique ring homomorphism:

$$\begin{aligned} f_{\alpha,\lambda}(X_\lambda) &= r_\lambda^{-1}(Y_\lambda^\alpha - y_\lambda^\alpha) + q_\lambda y_\lambda^\alpha, & f_{\alpha,\lambda}(Y_\lambda) &= Y_\lambda^\alpha, \\ f_{\alpha,\lambda}(x_\lambda) &= r_\lambda^{-1}(Y_\lambda^\alpha - y_\lambda^\alpha), & f_{\alpha,\lambda}(y_\lambda) &= y_\lambda^\alpha, \quad \textit{and} \\ f_{\alpha,\lambda}|_{\neq\lambda} &= id|_{\neq\lambda}. \end{aligned}$$

Proof (sketch). The proof is by induction on α. The induction step uses Theorem 7 from [6] which basically talks about the case $\alpha = 2$. □

Similarly, using Theorem 9 from [6], one can prove an analoguous lemma for series reduction. Let $g_{\beta,\lambda}$ be the ring homomorphism that moves points using the β-stretching of a graph. The two lemma together establish parsimonious reductions in the Bollobás-Riordan polynomial.

Lemma 7. *Let $\alpha, \beta \in \mathbb{N}_{>0}$, $\lambda \in \Lambda$, and $\sigma : \mathcal{R} \to \mathbb{Q}$ be an evaluation point.*
It holds $\big(\sigma \circ g_{\beta,\lambda} \circ f_{\alpha,\lambda}\big)T_{col} \preccurlyeq^{\mathrm{p}} \sigma T_{col}$.

In the remainder of this section we interpolate the coloured Tutte polynomial from the given data points $(\sigma \circ g_{\beta,\lambda} \circ f_{\alpha,\lambda})$, for different α and β (but fixed λ).

The main problem of the interpolation process is that, in contrast to the situation in the classical Tutte polynomial, the parallel and series reductions do *not* move evaluation points along a relatively simple one-dimensional variety. Instead, all we can say is that the points produced by these reductions lie in the variety $L_{q,r}$, which has *two* dimension.

We call an evaluation σ *stuck* in λ if, for $\sigma|_\lambda = (a, b, c, d)$, we have $|a| \in \{0, |c|\}$ and $|b| \in \{0, |d|\}$. For every fixed λ and $c, d \neq 0$, there are exactly nine stuck points.

Theorem 2. *For $q, r \in \mathbb{Q}$, let $\sigma \in P_{q,r}$ be an evaluation point that is not stuck in λ. Then, for all $\sigma' \in P_{q,r}$ with $\sigma'|_{\neq\lambda} = \sigma|_{\neq\lambda}$, it holds $\sigma' T_{col} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{col}$.*

Proof (sketch). The rather technical proof works by showing that the Vandermonde matrix V_2 is non-singular for the data points given by the stretching or fattening reductions. □

For $c = d = 1$, this theorem specializes to the line interpolation theorem for the classical Tutte polynomial [10]. In a suitable model of computation, the theorem also works for $\mathbb{R}$ and $\mathbb{C}$.

9 Complexity of the Bollobás-Riordan Polynomial

The dichotomy theorem from the classical paper [10] says that evaluating the Tutte polynomial over $\mathbb{R}$ is #**P**-hard, except for the *special* points $(1, 1)$, $(0, -1)$, $(-1, 0)$, $(-1, -1)$ and one hyperbola $(x - 1)(y - 1) = 1$ in the Tutte plane, where this is easy. For the Bollobás-Riordan polynomial, one expects a similar dichotomy. The hyperbola becomes the variety $Q_\sigma = 1$, and we extend the notion of special points in a natural way: We say that σ is *special* if, for all colours λ and for $(a, b, c, d) = \sigma|_\lambda$, it holds

$$(a, b) \in \{(c, d), (0, -d), (-c, 0), (-c, -d)\} .$$

We classify the complexity of evaluating the Bollobás-Riordan polynomial in the following way.

Main Theorem. *Let σ be an evaluation point. It holds:*

*(i) If $Q_\sigma \neq 1$ and σ is not special, σT_{col} is #**P**-hard under Turing reductions.*
*(ii) If $Q_\sigma \neq 1$ and σ is not special, σT_{col} is #**P**-hard under algebraic reductions.*
(iii) If $Q_\sigma = 1$, then σT_{col} is polynomial-time computable.

Proof. (i) Let λ be a colour in which σ is non-special and write $(a, b, c, d) = \sigma|_\lambda$. By Proposition 1, $T(a/c, b/d) \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{\mathrm{col}}$. Since $(a/c, b/d)$ neither lies on the hyperbola $(x - 1)(y - 1) = 1$ nor is one of the special points $(1, 1)$, $(0, -1)$, $(-1, 0)$, $(-1, -1)$ in the Tutte plane, it is #**P**-hard to evaluate under Turing reductions [10].

(ii) There exists a colour λ in which σ is not stuck. This is because, for $Q_\sigma \notin \{0, 1, 2\}$, σ can only be stuck if it is of the form $(-c, -d, c, d)$ and therefore special. Let φ be the evaluation point with $\varphi|_\lambda = (1 - Q_\sigma, 0, 1, 1)$ and $\varphi|_{\neq\lambda} = \sigma|_{\neq\lambda}$. Both σ and φ are evaluation points in $P_{q,r}$, so we can apply Theorem 2 to get $\varphi T_{\mathrm{col}} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{\mathrm{col}}$. The claim follows for $Q_\sigma \notin \{0, 1, 2\}$ because $\chi(Q_\sigma) = \varphi T_{\mathrm{col}}$ holds (cf. Sec. 5) and $\chi(Q_\sigma)$ is #**P**-hard by Lemma 1. For $Q_\sigma = 0, 2$, we can use the reduction from (i): Hardness of such points is proven in [10] by an *algebraic* reduction from reliability computations or from the permanent, both of which are problems that can be proven to be hard under *algebraic* reductions.

(iii) From [10], we know that $\varphi T_{\mathrm{col}} = T(2, 0)$ is polynomial-time computable, where $\varphi|_\mu := (2, 0, 1, 1)$ for all μ. Furthermore, $\varphi|_\mu$ is not stuck in any colour, which allows us to apply Theorem 2 in series to each colour to get the reduction $(\sigma \circ \eta_s)T_{\mathrm{col}} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \varphi T_{\mathrm{col}}$. Here we choose $s := q_\varphi/q_\sigma = r_\sigma/r_\varphi$ in such a way, that $(\sigma \circ \eta_s) \in P_{q_\varphi, r_\varphi}$. To finish the proof, we apply Lemma 5 and obtain $\sigma T_{\mathrm{col}} = (\sigma \circ \eta_s \circ \eta_{1/s})T_{\mathrm{col}} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} (\sigma \circ \eta_s)T_{\mathrm{col}}$. □

For the proof of (ii), we could have used (i) also for the cases where $Q_\sigma \neq 0, 2$ since the reductions in [10] are actually algebraic. Instead, we decided to give an independent proof.

By Theorem 1, any non-special point with $Q_\sigma \neq 1$ is as hard as the whole graph polynomial, even under polynomial-time uniform and algebraic reductions.

Corollary 1. *For any non-special evaluation point σ with $Q_\sigma \neq 1$, we have $T_{col} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}U} \sigma T_{col}$. In particular, we have $\sigma' T_{col} \preccurlyeq^{\mathrm{p}}_{\mathrm{A}} \sigma T_{col}$ for all points σ'.*

Open Problem. *Is σT_{col} polynomial-time computable if σ is special?*

References

1. Averbouch, I., Makowsky, J.A.: The complexity of multivariate matching polynomials (January 2007) (preprint)
2. Bläser, M., Dell, H.: Complexity of the cover polynomial. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 801–812. Springer, Heidelberg (2007)
3. Bläser, M., Hoffmann, C.: On the complexity of the interlace polynomial. arXiv:0707.4565 (2007)
4. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and real computation. Springer, New York (1998)
5. Bollobás, B.: Modern Graph Theory. Springer, Heidelberg (1999)
6. Bollobás, B., Riordan, O.: A Tutte polynomial for coloured graphs. Combinatorics, Probability and Computing 8(1-2), 45–93 (1999)
7. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. In: Grundlehren der mathematischen Wissenschaften, February 1997. Springer, Heidelberg (1997)
8. Dong, F.M., Koh, K.M., Teo, K.L.: Chromatic Polynomials and Chromaticity of Graphs. World Scientific, Singapore (2005)
9. Goldberg, L.A., Jerrum, M.: Inapproximability of the Tutte polynomial. In: Johnson, D.S., Feige, U. (eds.) STOC, pp. 459–468. ACM, New York (2007)
10. Jaeger, F., Vertigan, D.L., Welsh, D.J.A.: On the computational complexity of the Jones and Tutte polynomials. Mathematical Proceedings of the Cambridge Philosophical Society 108(1), 35–53 (1990)
11. Kauffman, L.H.: A Tutte polynomial for signed graphs. Discrete Applied Mathematics 25, 105–127 (1989)
12. Linial, N.: Hard enumeration problems in geometry and combinatorics. SIAM J. Algebraic Discrete Methods 7(2), 331–335 (1986)
13. 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)
14. Makowsky, J.A.: Algorithmic uses of the Feferman-Vaught theorem. Annals of Pure and Applied Logic 126, 1–3 (2004)
15. Makowsky, J.A.: From a zoo to a zoology: Towards a general theory of graph polynomials. Theory of Computing Systems (2008), ISSN 1432-4350

16. Sokal, A.D.: The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In: Webb, B.S. (ed.) Surveys in Combinatorics. London Mathematical Society Lecture Note Series, vol. 327, pp. 173–226. Cambridge University Press, Cambridge (2005)
17. Stanley, R.P.: Acyclic orientations of graphs. Discrete Mathematics 306(10-11), 905–909 (2006)
18. Welsh, D.J.A.: Matroid Theory. London Mathematical Society Monographs, vol. 8. Academic Press, London (1976)

A Complete Characterization of Nash-Solvability of Bimatrix Games in Terms of the Exclusion of Certain 2×2 Subgames[⋆]

Endre Boros[1], Khaled Elbassioni[2], Vladimir Gurvich[1], Kazuhisa Makino[3], and Vladimir Oudalov[1]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003
{boros,gurvich,voudalov}@rutcor.rutgers.edu
[2] Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
elbassio@mpi-sb.mpg.de
[3] Graduate School of Information Science and Technology, University of Tokyo, Tokyo, 113-8656, Japan
makino@mist.i.u-tokyo.ac.jp

Abstract. In 1964 Shapley observed that a matrix has a saddle point whenever every 2×2 submatrix of it has one. In contrast, a bimatrix game may have no Nash equilibrium (NE) even when every 2×2 subgame of it has one. Nevertheless, Shapley's claim can be generalized for bimatrix games in many ways as follows. We partition all 2×2 bimatrix games into fifteen classes $C = \{c_1, \dots, c_{15}\}$ depending on the preference pre-orders of the two players. A subset $t \subseteq C$ is called a NE-theorem if a bimatrix game has a NE whenever it contains no subgame from t. We suggest a general method for getting all minimal (that is, strongest) NE-theorems based on the procedure of joint generation of transversal hypergraphs given by a special oracle. By this method we obtain all (six) minimal NE-theorems.

1 Introduction, Main Concepts and Results

1.1 Bimatrix Games and Nash Equilibria

Let X_1 and X_2 be finite sets of strategies of players 1 and 2. Pairs of strategies $x = (x_1, x_2) \in X_1 \times X_2 = X$ are called *situations*. A *bimatrix game* $U = (U_1, U_2)$ is a pair of real-valued matrices $U_i : X \to \mathbb{R},\ i = 1, 2$, with common set of entries X. Value $U_i(x)$ is interpreted as utility function (also called profit or payoff) of player $i \in \{1, 2\}$ in the situation x. A situation $x = (x_1, x_2) \in X_1 \times X_2 = X$ is called a Nash equilibrium (NE) if

$$U_1(x_1', x_2) \le U_1(x_1, x_2)\ \forall x_1' \in X_1 \text{ and } U_2(x_1, x_2') \le U_2(x_1, x_2)\ \forall x_2' \in X_2;$$

[⋆] The second author gratefully acknowledges the partial support of DIMACS, the NSF Center for Discrete Mathematics and Theoretical Computer Science.

in other words, if no player can make a profit by choosing a new strategy if the opponent keeps the old one. A bimatrix game U is called a *zero sum or matrix* game if $U_1(x) + U_2(x) = 0$ for every $x \in X$. In this case the game is well-defined by one of two matrices, say, by U_1, and a NE is called a *saddle point* (SP).

1.2 Locally Minimal SP-Free Matrix and NE-Free Bimatrix Games

Standardly, we define a subgame as the restriction of U to a subset $X' = X_1' \times X_2' \subseteq X_1 \times X_2 = X$, where $X_1' \subseteq X_1$ and $X_2' \subseteq X_2$. In 1964 Shapley [8] noticed that a matrix has a saddle point whenever each of its 2×2 submatrices has one. Obviously, in this case, every submatrix has a SP, too. In other words, all minimal SP-free matrices are of size 2×2. Moreover, all locally minimal SP-free matrices are of size 2×2, too; in other words, every SP-free matrix of larger size has a row or column whose elimination still results in an SP-free submatrix; see [1]. Other generalizations of Shapley's theorem can be found, for example, in [6,7]. Let us also notice that a 2×2 matrix has no SP if and only if one of its diagonals is strictly larger than the other.

The "naive generalization" of Shapley's claim to bimatrix games fails: a 3×3 game might have no NE even if each its 2×2 subgame has one; moreover, for each $n \geq 3$ a $n \times n$ bimatrix game might have no NE even if every its subgame has one; see Example 1 in [6] or [1] and also examples given below. However, all locally minimal NE-free games admit the following explicit characterization [1].

For the sake of brevity, let us denote situation (x_1^i, x_2^j) by $x_{i,j}$, where $X_1 = \{x_1^1, x_1^2, \ldots\}$ and $X_2 = \{x_2^1, x_2^2, \ldots\}$.

Given an integer $n \geq 2$ and a bimatrix game U with $|X_1| \geq n$ and $|X_2| \geq n$, let us say that U has the *canonical strong improvement n-cycle C_n^0* if each situation $x_{1,1}, x_{2,2}, \ldots, x_{n-1,n-1}, x_{n,n}$ (respectively, $x_{1,2}, x_{2,3}, \ldots, x_{n-1,n}, x_{n,1}$) is a unique largest in its row with respect to U_2 (in its column with respect to U_1) and is the second largest, not necessarily, unique, in its column with respect to U_1 (in its row with respect to U_2). Any other strong improvement n-cycle C_n is obtained from the canonical one C_0 by arbitrary permutations of the rows of X_1 and columns of X_2.

It is easy to see that if an $n \times n$ bimatrix game U has a strong improvement cycle then U has no NE, yet, every proper subgame obtained from U by elimination of either one row or one column has a NE. In other words, U is a *locally minimal* NE-free bimatrix game. Moreover, the inverse holds, too.

Theorem 1. *([1]). A bimatrix game U is a locally minimal NE-free game if and only if U is of size $n \times n$ for some $n \geq 2$ and it contains a strong improvement n-cycle.* $\square$

Thus, locally minimal NE-free games can be arbitrary large. Several examples are given in Figures 2 - 6, where each game has the canonical strong improvement cycle. Although it seems difficult to characterize or recognize the minimal NE-free games (see [1]), yet, the above characterization of the locally-minimal ones will be sufficient for us.

1.3 Pre-orders

Given a set Y and a mapping $P : Y^2 \to \{<, >, =\}$ that assigns one of these three symbols to every ordered pair $y, y' \in Y$, we say that y is *less or worse* than y' if $y < y'$, respectively, y is *more or better* than y' if $y > y'$, and finally, y and y' are *equivalent or they make a tie* if $y = y'$. Furthermore, P is called a *pre-order* if the following standard properties (axioms) hold for all $y, y', y'' \in Y$:

symmetry: $y < y' \Leftrightarrow y' > y, \quad y = y' \Leftrightarrow y' = y,$ and $y = y;$
transitivity: $y < y' \;\; \& \;\; y' < y'' \Rightarrow y < y'', \quad y < y' \;\; \& \;\; y' = y'' \Rightarrow y < y'',$
$y = y' \;\; \& \;\; y' < y'' \Rightarrow y < y'', \quad y = y' \;\; \& \;\; y' = y'' \Rightarrow y = y'',$

A pre-order without ties is called a *(linear or complete) order.*

We use standard notation: $y \leq y'$ if $y < y'$ or $y = y'$ and $y \geq y'$ if $y > y'$ or $y = y'$. Obviously, transitivity and symmetry still hold:
$y \leq y' \;\; \& \;\; y' < y'' \Rightarrow y < y'', \quad y < y' \;\; \& \;\; y' \leq y'' \Rightarrow y < y'',$
$y \leq y' \;\; \& \;\; y' \leq y'' \Rightarrow y \leq y'',$ and $y \leq y' \Leftrightarrow y' \geq y.$

In Figures 1-6 we use the following notation: an arrow from y to y' for $y > y'$, a line with two dashes for $y = y'$, and an arrow with two dashes for $y \geq y'$.

1.4 Configurations; Fifteen 2-Squares

Let us notice that to decide whether a situation $x = (x_1, x_2) \in X_1 \times X_2 = X$ is a NE in U, it is sufficient to know only two pre-orders: in the row x_1 with respect to U_2 and in column x_2 with respect to U_1.

Given X_1 and X_2, let us assign a pre-order P_{x_i} over X_{3-i} to each $x_i \in X_i$; $i = 1, 2$, and call the obtained preference profile $P = \{P_{x_1}, P_{x_2} \mid x_1 \in X_1, \; x_2 \in X_2\}$ a *configuration or bi-pre-order.*

Naturally, every bimatrix game $U = (U_1, U_2)$ defines a unique configuration $P = P(U)$, where P_{x_i} is the pre-order over X_{3-i} defined by U_i; $i = 1, 2$. Clearly, each configuration is realized by infinitely many bimatrix games. Yet, it is also clear that to get all NE in game U it is enough to know its configuration $P(U)$.

For brevity, we will refer to a 2×2 configuration as a 2-*square.* Up to permutations and transpositions, there exist only fifteen different types of 2-squares. They are listed in Figure 1 together with the corresponding bimatrix games (for the first 6 squares). Four 2-squares c_1, c_2, c_3, c_4 have no ties; another four, c_5, c_6, c_7, c_8 and the next five, c_9, c_{10}, c_{11}, c_{12}, c_{13}, have, respectively, one and two ties each; finally, c_{14} and c_{15} have 3 and 4 ties.

Fifteen 2-squares have $0, 2, 1, 1, 1, 2, 1, 2, 3, 2, 2, 2, 2, 3,$ and 4 NE, respectively. Thus, only c_1 has no NE. Shapley's theorem asserts that each c_1-free zero-sum game (or configuration) has a NE. Let us note that 2-squares c_1 - c_6 are frequent in the literature. For example, the non-zero-sum bimatrix games realizing c_2 and c_4 may represent classical "family dispute" and "prisoner's dilemma"; respectively, c_5 and c_6 illustrate the concepts of the "promise" and "threat".

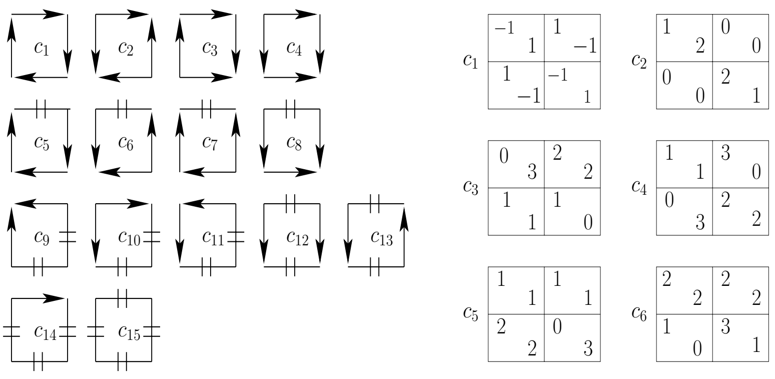

c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_8
c_9
c_{10}
c_{11}
c_{12}
c_{13}
c_{14}
c_{15}
c_1: −1, 1 | 1, −1 / 1, −1 | −1, 1
c_2: 1, 2 | 0, 0 / 0, 0 | 2, 1
c_3: 0, 3 | 2, 2 / 1, 1 | 1, 0
c_4: 1, 1 | 3, 0 / 0, 3 | 2, 2
c_5: 1, 1 | 1, 1 / 2, 2 | 0, 3
c_6: 2, 2 | 2, 2 / 1, 0 | 3, 1

Fig. 1. Fifteen 2-squares

1.5 Dual or Transversal Hypergraphs

Let C be a finite set whose elements we denote by $c \in C$. A *hypergraph* H (*on the ground set* C) is a family of subsets $h \subseteq C$ that are called the *edges* of H. A hypergraph H is called Sperner if containment $h \subseteq h'$ holds for no two distinct edges of H. Given two hypergraphs T and E on the common ground set C, they are called *transversal* or *dual* if the following properties hold:

(i) $t \cap e \neq \emptyset$ for every $t \in T$ and $e \in E$;
(ii) for every subset $t' \subseteq C$ such that $t' \cap e \neq \emptyset$ for each $e \in E$ there exists an edge $t \in T$ such that $t \subseteq t'$;
(iii) for every subset $e' \subseteq C$ such that $e' \cap t \neq \emptyset$ for each $t \in T$ there exists an edge $e \in E$ such that $e \subseteq e'$.

Property (i) means that edges of E and T are transversal, while (ii) and (iii) mean that T contains all minimal transversals to E and E contains all minimal transversals to T, respectively. It is well-known, and not difficult to see, that (ii) and (iii) are equivalent whenever (i) holds. Although for a given hypergraph $\mathcal{H}$ there exist infinitely many dual hypergraphs, yet, only one of them, which we will denote by $\mathcal{H}^d$, is Sperner. Thus, within the family of Sperner hypergraphs duality is well-defined; moreover, it is an involution, that is, equations $T = E^d$ and $E = T^d$ are equivalent. It is also easy to see that dual Sperner hypergraphs have the same set of elements. For example, the following two hypergraphs are dual:

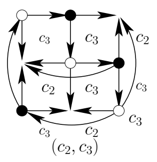
$$E' = \{(c_1), (c_2, c_3), (c_5, c_9), (c_3, c_5, c_6)\}, \tag{1}$$

$$T' = \{(c_1, c_2, c_5), (c_1, c_3, c_5), (c_1, c_2, c_6, c_9), (c_1, c_3, c_9)\}; \tag{2}$$

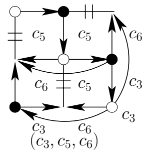
as well as the following two:

$$E = \{(c_1), (c_2, c_3), (c_5, c_9), (c_3, c_5, c_6), (c_2, c_4, c_5, c_6)\}, \tag{3}$$

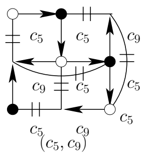
$$T = \{(c_1, c_2, c_5), (c_1, c_3, c_5), (c_1, c_2, c_3, c_9), (c_1, c_2, c_6, c_9), (c_1, c_3, c_4, c_9), (c_1, c_3, c_6, c_9)\}. \tag{4}$$



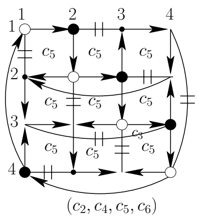
1, 2	2, 1	0, 0
0, 0	1, 2	2, 1
2, 1	0, 0	1, 2



0, 1	1, 0	0, 0
0, 0	0, 2	2, 1
1, 1	0, 0	1, 2



0, 1	1, 0	0, 0
0, 0	0, 1	1, 0
1, 0	0, 0	0, 1



	12	13	14	23	24	34
12	c_5	c_6	c_5	c_5	c_4	c_5
13	c_4	c_2	c_6	c_6	c_2	c_4
14	c_5	c_4	c_5	c_5	c_6	c_5
23	c_5	c_4	c_5	c_5	c_6	c_5
24	c_6	c_2	c_4	c_4	c_2	c_6
34	c_5	c_6	c_5	c_5	c_4	c_5

1, 2	2, 1	0, 1	1, 0
1, 0	1, 2	2, 1	0, 1
0, 1	1, 0	1, 2	2, 1
2, 1	0, 1	1, 0	1, 2

Fig. 2. NE-examples

1.6 Hypergraphs of Examples and Theorems

Let $C = \{c_1, \dots, c_{15}\}$. We call a subset $e \subseteq C$ a *NE-example* if there is a NE-free configuration P such that e is the set of types of 2-squares in P; respectively, a subset $t \subseteq C$ is called a *NE-theorem* if a configuration has a NE whenever it contains no 2-squares from t. Obviously, $e \cap t \neq \emptyset$ for any NE-example e and NE-theorem t, since otherwise e is a counterexample to t. Moreover, it is well-known and easy to see that the hypergraphs of all inclusion-minimal (that is, strongest) NE-examples E_{NE} and NE-theorems T_{NE} are transversal. Let us consider c_1 and four configurations in Figure 2. It is easy to verify that all five contain canonical strong cycles and hence, they are locally minimal (in fact, minimal) NE-free configurations. These five configurations are chosen because they contain few types of 2-squares; the corresponding sets are given in Figure 2; they form the hypergraph E defined by (3). Figure 2 shows that each edge of E is a NE-example.

Let us consider the dual hypergraph T given by (4). We will prove that every edge $t \in T$ is a NE-theorem, thus, showing that the "research is complete", that is, $E = E_{NE}$ and $T = T_{NE}$ are the hypergraphs of all strongest NE-examples and theorems.

Remark 1. Given a family of NE-examples E', the dual hypergraph T' should be viewed as a hypergraph of conjectures rather than theorems. Indeed, some inclusion-minimal examples might be missing in E'; moreover, some examples of E' might be reducible. In this case some conjectures from the dual hypergraph $T' = E'^d$ will fail, being too strong. For instance, let us consider E' given by (1) in which the NE-example (c_2, c_4, c_5, c_6) is missing. (In fact, it is not that easy to obtain a minimal 4×4 example without computer.) Respectively, conjecture (c_1, c_3, c_9) appears in $T' = E'^d$. This conjecture is too strong, so it fails. In $T = T_{NE}$ we substitute for it three weaker (but correct) NE-theorems (c_1, c_3, c_9, c_2), (c_1, c_3, c_9, c_4), and (c_1, c_3, c_9, c_6). Thus, if it seems too difficult to prove a conjecture, one should look for new examples.

1.7 Joint Generation of Examples and Theorems

Of course, this approach can be applied not only to NE-free bimatrix games.

In general, given a set of objects Q (in our case, configurations), list C of subsets (properties) $Q_c \subseteq Q$, $c \in C$ (in our case, c-free configurations), the target subset $Q_0 \subseteq Q$ (configurations that have a NE), we introduce a pair of hypergraphs $E = E(Q, Q_0, C)$ and $T = T(Q, Q_0, C)$ (examples and theorems) defined on the ground set C as follows:

(i) every set of properties assigned to an edge $t \in T$ (a theorem) implies Q_0, that is, $q \in Q_0$ whenever q satisfies all properties of t, or in other words, $\cap_{c \in t} Q_c \subseteq Q_0$; in contrast,

(ii) each set of properties corresponding to the complement $C \setminus e$ of an edge $e \in E$ (an example) does not imply Q_0, i.e., there is an object $q \in Q \setminus Q_0$ satisfying all properties of $C \setminus e$, or in other words, $\cap_{c \notin e} Q_c \not\subseteq Q_0$.

If hypergraphs E and T are dual then we can say that "our understanding of Q_0 in terms of C is perfect", that is, every new example $e' \subset C$ (theorem $t' \subseteq C$) is a superset of some old example $e \in E$ (theorem $t \in T$).

Without loss of generality we can assume that examples of $e \in E$ and theorems $t \in T$) are inclusion-wise minimal in C; or in other words both hypergraphs E and T are Sperner.

Given Q, Q_0 and C, we try to generate hypergraphs E and T jointly [5]. Of course, the oracle may be a problem: Given a subset $C' \subseteq C$, it may be difficult to decide whether C' is a theorem (i.e., if $q \in Q_0$ whenever q satisfies all properties of C') or an example (i.e., if there is a $q \in Q \setminus Q_0$ satisfying all properties of $C \setminus C'$). However, the stopping criterion, $E^d = T$, is well-defined and, moreover, it can be verified in quasi-polynomial time [3].

Remark 2. Let us notice that containment $\cap_{c \in t} Q_c \subseteq Q_0$ might be strict. In other words, theorem t gives sufficient but not always necessary conditions for $q \in Q_0$. We can also say that theorems $t \in T$ give all optimal "inscribed approximations" of $Q_0 \subseteq Q$ in terms of C.

Remark 3. In [4], this approach was illustrated by a simple model problem in which Q is the set of 4-gons, Q_0 is the set of squares, C is a set of six properties of a 4-gon. Two dual hypergraphs of all minimal theorems T and examples E were constructed. In [2], the same approach was applied to a more serious problem related to families of Berge graphs.

1.8 Strengthening NE-Theorems; Main Results

We will prove all six NE-theorems $t \in T_{NE}$. Formally, they cannot be strengthened, since t' is not a NE-theorem whenever $t' \subset t \in T_{NE}$ and the containment $t' \subset t$ is strict. Still, we can get stronger claims in slightly different terms.

Let us notice that for any t the class of t-free configurations (games) is hereditary. Indeed, if a configuration (game) is t-free then every subconfiguration (subgame) of it is t-free, too. Hence, we can restrict ourselves by the locally minimal NE-free examples, which are characterized by Theorem 1.

Now, let us consider NE-theorems (c_1, c_2, c_5), (c_1, c_3, c_5), and (c_1, c_2, c_6, c_9). Formally, since 2-square c_1 has no NE, it must be eliminated. Yet, in a sense, it is the only exception. More precisely, we can strengthen the above three NE-theorems as follows.

Theorem 2. *The 2-square c_1 is a unique locally minimal NE-free configuration that is also (c_2, c_5)- or (c_3, c_5)-, or (c_2, c_6, c_9)-free.*

Furthermore, theorems (c_1, c_3, c_9, c_2), (c_1, c_3, c_9, c_4), (c_1, c_3, c_9, c_6) can be strengthened, too. In fact, we will characterize explicitly the configurations that are locally minimal NE-free and also (c_3, c_9)-free. This family is sparse but still infinite. In particular, we obtain the following result. Let $C(P)$ denote the set of all types of 2-squares of configuration P; furthermore, let $C' = \{c_2, c_4, c_5, c_6, c_7, c_8, c_{13}, c_1\}$ and $C'' = C' \cup \{c_{12}\}$.

Theorem 3. *Let P be a locally minimal NE-free $n \times n$ configuration that is also (c_3, c_9)-free. Then*

(i) n is even unless $n = 1$; (ii) if $n = 2$ then P is c_1;

(iii) if $n = 4$ then P is a unique (c_2, c_4, c_5, c_6)-configuration in Figure 2;

(iv) if $n = 6$ then $C(P) = C'$;

(v) if $n = 8$ then $C' \subseteq C(P) \subseteq C''$ and there exist P with $C(P) = C'$;

(vi) finally, if $n \geq 10$ is even then $C(P) = C''$.

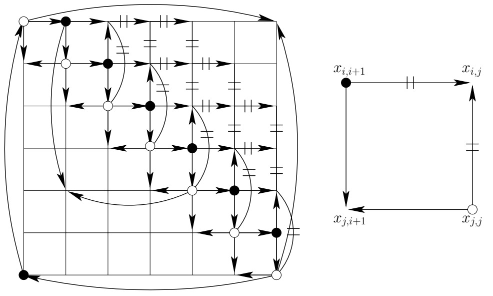
It is clear that this statement implies the remaining three NE-theorems: (c_1, c_3, c_9, c_2), (c_1, c_3, c_9, c_4), and (c_1, c_3, c_9, c_6).

2 Proof of Theorems 2 and 3

As we already mentioned, we can restrict ourselves to the locally minimal NE-free configurations. By Theorem 1, each such configuration P is of size $n \times n$ for some $n \geq 2$ and P contains a strong improvement cycle C_n. Without loss of generality we can assume that $C_n = C_n^0$ is canonical. In particular,

$$x_{i,i+1} \geq x_{i,j},\ x_{i,i+1} > x_{j,i+1},\ \text{for } j \neq i,\ x_{j,j} \geq x_{i,j},\ x_{j,j} > x_{j,i+1},\ \text{for } j \neq i+1. \tag{5}$$

Furthermore, if $n = 2$ then 2-square c_1 is a unique NE-free configuration (in fact, c_1 is a strong 2-cycle). Hence, we will assume that $n \geq 3$. Additionally, we assume that P is t-free and consider successively the following subsets t : (c_2, c_5), (c_3, c_5), (c_2, c_6, c_9), and (c_3, c_9). Theorem 2 will follow, since in the first three cases we get a contradiction. For $t = (c_3, c_9)$ we will characterize the corresponding configurations explicitly, thus proving Theorem 3.



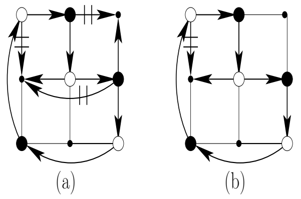


Fig. 3. Locally minimal NE-free and (c_2, c_5)-free configurations do not exist, except c_1

Fig. 4. Locally minimal NE-free and (c_2, c_6, c_9)- or (c_3, c_5)-free configurations do not exist, except c_1

2.1 Locally Minimal NE-Free and (c_2, c_5)-Free Configurations

Let us consider C_n^0 in Figure 3 (where $n = 7$). By (5), $x_{i,i} > x_{i,j}$ (with respect to U_2) whenever $j \neq i$; in particular, $x_{i,i} > x_{i,i-1}$ for $i \in [n] = \{1, \dots, n\}$, where standardly, $0 \equiv n$. Similarly, $x_{i,i} \geq x_{j,i}$ whenever $j \neq i - 1$ (with respect to U_1); in particular, $x_{i,i} \geq x_{i+1,i}$ for $i \in [n] = \{1, \dots, n\}$, where standardly, $n + 1 \equiv 1$. Moreover, the latter n inequalities are also strict, since otherwise c_5 would appear.

By similar arguments we show that $x_{i,i+1} > x_{i,i+2}$ and $x_{i,i+1} > x_{i-1,i+1}$ for $i = 1, \dots, n - 1$; see Figure 3.

Next, let us notice that $x_{i,i} = x_{i-2,i}$ for $i = 2, \dots n$. Indeed, $x_{i,i} \geq x_{i-2,i}$, since C_n is a strong cycle, and c_2 would appear in case $x_{i,i} > x_{i-2,i}$.

Furthermore, $x_{i,i+2} \geq x_{i,i+3}$ for $i = 1, \dots, n-3$, since otherwise $x_{i,i+2}$, $x_{i,i+3}$, $x_{i+2,i+2}$, $x_{i+2,i+3}$ would form a c_5.

Next, let us notice that $x_{i,i+3} = x_{i+1,i+3}$ for $i = 1, \dots, n-3$. Indeed, $x_{i,i+3} \leq x_{i+3,i+3} = x_{i+1,i+3}$, and if $x_{i,i+3} < x_{i+1,i+3}$ then $x_{i,i+1}$, $x_{i,i+3}$, $x_{i+3,i+1}$, $x_{i+3,i+3}$ would form a c_2, by (5).

Similarly, by induction on j, we show that $x_{i,i+j} \geq x_{i,i+j+1}$ and $x_{i,i+j} = x_{i+1,i+j}$ for $1 \leq i \leq n - 3$ and $2 \leq i + j \leq n - 1$.

In particular, $x_{n,n} = x_{n-2,n} = x_{n-3,n} = \ldots = x_{2,n} = x_{1,n}$ in contradiction with the strict inequality $x_{n,n} > x_{1,n}$ obtained before. □

2.2 Locally Minimal NE-Free and (c_2, c_6, c_9)- or (c_3, c_5)-Free Configurations

These two cases are easy. Let us consider C_n^0 in Figures 4 (a) and (b) (where $n = 3$), corresponding respectively to the two cases. By definition, in both cases $x_{2,2} > x_{2,1}$ $x_{1,1} \geq x_{2,1}$. In case (b) we already got a contradiction, since four above situations form c_3 or c_5.

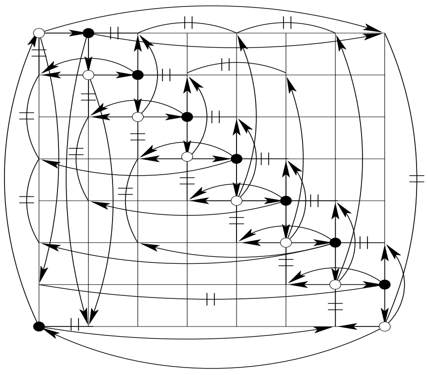


Fig. 5. Locally minimal NE-free and (c_3, c_9)-free configurations

In case (a) we have to proceed a little further. Clearly, $x_{2,3} \geq x_{2,1}$, $x_{1,2} \geq x_{1,3}$, $x_{2,3} > x_{1,3}$, and again we get a contradiction, since situations $x_{1,1}$, $x_{1,3}$, $x_{2,1}$, $x_{2,3}$ form c_9 if two equalities hold, c_6 if exactly one, and c_2 if none. □

2.3 Locally Minimal NE-Free and (c_3, c_9)-Free Configurations

Let us consider C_n^0 in Figure 5 (where $n = 8$). By (5), for all i we have:

$$x_{i,i} > x_{i,i+1}, x_{i,i} > x_{i,i-1}, x_{i,i} \geq x_{i+1,i}, x_{i,i} \geq x_{i-2,i};$$

$$x_{i,i+1} > x_{i+1,i+1}, x_{i,i+1} > x_{i-1,i+1}, x_{i,i+1} \geq x_{i,i+2}, x_{i,i+1} \geq x_{i,i-1}.$$

Furthermore, it is not difficult to show that

$$x_{i,i} = x_{i+1,i} \text{ and } x_{i,i+1} = x_{i,i+2}, \tag{6}$$

since otherwise c_3 appears, while

$$x_{i,i} > x_{i-2,i} \text{ and } x_{i,i+1} > x_{i,i-1}, \tag{7}$$

since otherwise c_9 appears; see Figure 5.

Standardly, we prove all four claims in (6) and (7) by induction introducing situations in the following (alternating diagonal) order:

$x_{2,1}$, $x_{1,3}$, $\ldots$, $x_{i,i-1}$, $x_{i-1,i+1}$, $\ldots$, $x_{n,n-1}$, $x_{n-1,1}$, $x_{1,n}$, $x_{n,2}$.

Furthermore, $x_{1,1} = x_{2,1} \geq x_{4,1}$ unless $n < 5$; moreover, $x_{2,1} = x_{4,1}$, since otherwise situations $x_{2,1}$, $x_{4,1}$, $x_{2,4}$, and $x_{4,4}$ form c_3.

Similarly, we prove that $x_{1,3} = x_{1,5}$ unless $n < 5$.

Then let us recall that $x_{4,5} \geq x_{4,1}$ and conclude that $x_{4,5} > x_{4,1}$, since otherwise situations $x_{1,1}$, $x_{4,1}$, $x_{1,5}$, and $x_{4,5}$ form c_9.

In general, it is not difficult to prove by induction that

$$x_{i,i} = x_{i+1,i} = x_{i+3,i} = \ldots = x_{i+2j-1,i}, \text{ while } x_{i-1,i} > x_{i,i} > x_{i+2j,i}; \tag{8}$$

$$x_{i,i+1} = x_{i,i+2} = x_{i,i+4} = \ldots = x_{i,i+2j}, \text{ while } x_{i,i} > x_{i,i+1} > x_{i,i+2j+1}. \tag{9}$$

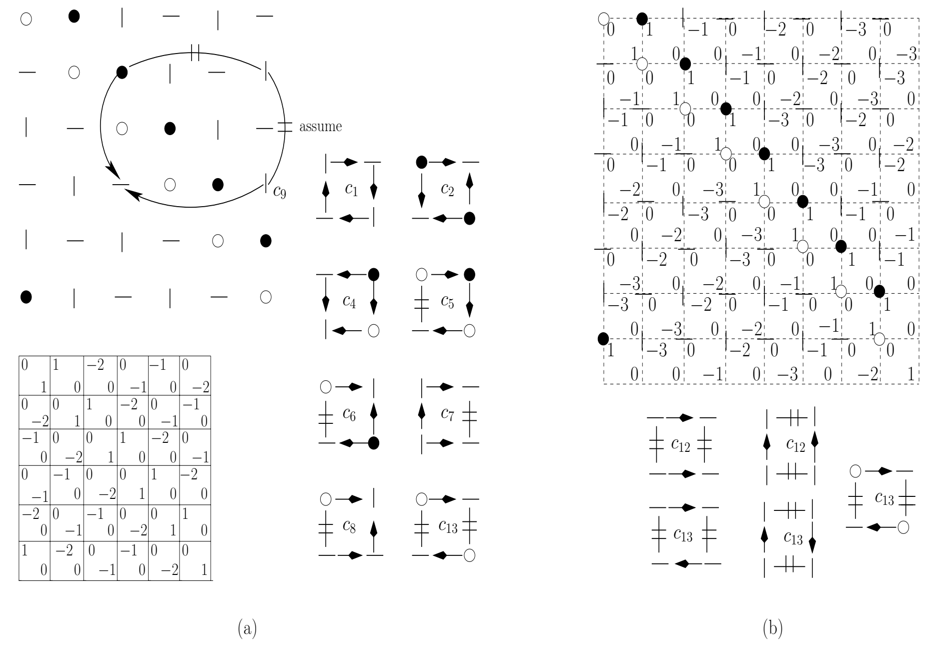


Fig. 6. Two examples from F_6 and F_8: horizontal (respectively, vertical) bars indicate second largest elements with respect to U_1 (respectively U_2)

In both cases each sum is taken mod (n) (in particular, $n = 0$) and $1 \leq j < n/2$ (in particular, j takes values $1, 2$, and 3 for $n = 7$ and $n = 8$).

If $n > 1$ is odd we immediately get a contradiction, since in this case, by (8), $x_{1,1} = x_{n-1,1}$, while, by (7), $x_{1,1} > x_{n-1,1}$ for all $n > 1$. Yet, for each even n, the family F_n of all locally minimal NE-free and (c_3, c_9)-free configurations is not empty.

Up to an isomorphism, F_2 (respectively, F_4) consists of a unique configuration: c_1 in Figure 1 (respectively, (c_2, c_4, c_5, c_6) in Figure 2). Two larger examples, from F_6 and F_8, are given in Figures 6 (a) and (b), respectively.

We already know that each configuration $P \in F_{2k}$ must satisfy (5) - (9). Yet, P has one more important property:

$$x_{i,i+2j+1} \neq x_{i,i+2j'+1}, \; x_{i+2j,i} \neq x_{i+2j',i} \tag{10}$$

for all $i \in [n]$ and for all positive distinct $j, j' < n/2$. Indeed, it is easy to see that otherwise c_9 appears; see Figure 6(a).

Let us denote by G_n the family of all configurations satisfying (5) - (10). We already know that $F_n \subseteq G_n$ and $F_n = G_n = \emptyset$ if $n > 1$ is odd. Let us show that $F_n = G_n$ for even n. Obviously, G_4 consists of a unique configuration (c_2, c_4, c_5, c_6) in Figure 2 and $G_2 = \{c_1\}$. Examples of configurations from G_6 and G_8 are given in Figures 6 (a) and (b). It is easy to verify that each configuration of G_n contains eight 2-squares $C' = \{c_2, c_4, c_5, c_6, c_1, c_7, c_8, c_{13}\}$ whenever $n \geq 6$; see Figure 6 (a). Moreover, c_{12} appears, too, when $n \geq 10$.

On the other hand, no configuration $P \in G_n$ contains c_9, c_{10}, c_{11}, c_{14}, or c_{15}, since no 2-square in P can have two adjacent equalities. It is also easy to verify that P cannot contain c_3. Thus, P can contain only nine 2-squares of $C'' = C' \cup \{c_{12}\}$. In particular, each $P \in G_n$ is (c_3, c_9)-free; in other words, $G_n \subseteq F_n$ and, hence, $G_n = F_n$ for each n. This implies Theorem 3 and provides an explicit characterization for family F_n of locally minimal NE-free and (c_3, c_9)-free configurations. $\square$

Remark 4. Interestingly, for even n each configuration $P \in F_n = G_n$ contains the same set of nine 2-squares C'' if $n \geq 10$; for $P \in G_8$ there are two options: C'' or C' (see example in Figure 6 (b), where c_{12} does not appear); for $P \in G_6$ only C'; furthermore, G_4 consists of a unique configuration (c_2, c_4, c_5, c_6) in Figure 2 and G_2 only of c_1; finally, $F_n = G_n$ is empty if $n > 1$ is odd.

Acknowledgments. We are thankful to Kukushkin who promoted the idea of generalizing Shapley's (1964) theorem to bimatrix games and various concepts of solution.

References

1. Boros, E., Gurvich, V., Makino, K.: Minimal and locally minimal games and game forms, Rutcor Research Report 28-2007, Rutgers University
2. Chvatal, V., Lenhart, W.J., Sbihi, N.: Two-colourings that decompose perfect graphs. J. Combin. Theory Ser. B 49, 1–9 (1990)
3. Fredman, M., Khachiyan, L.: On the Complexity of Dualization of Monotone Disjunctive Normal Forms. J. Algorithms 21(3), 618–628 (1996)
4. Gurvich, V.A., Gvishiani, A.D.: Dual set systems and their applications. Izvestiya Akad. Nauk SSSR, ser. Tekhnicheskaya Kibernetika (in Russian) 4, 31–39 (1983); English translation in: Soviet J. of Computer and System Science (formerly Engineering Cybernetics)
5. Gurvich, V., Khachiyan, L.: On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. Discrete Applied Mathematics 96-97, 363–373 (1999)
6. Gurvich, V., Libkin, L.: Absolutely determined matrices. Mathematical Social Sciences 20, 1–18 (1990)
7. Kukushkin, N.S.: Shapley's 2 × 2 theorem for game forms. Economics Bullletin, `http://economicsbulletin.vanderbilt.edu/2007/volume3/EB-07C70017A.pdf`; see also Technical report, Department of Mathematical Methods for Economic Decision Analysis, Computing Center of Russian Academy of Sciences, `http://www.ccas.ru/mmes/mmeda/ququ/Shapley.pdf`
8. Shapley, L.S.: Some topics in two-person games. In: Drescher, M., Shapley, L.S., Tucker, A.W. (eds.) Advances in Game Theory. Annals of Mathematical Studies, AM52, pp. 1–28. Princeton University Press, Princeton (1964)

Synchronization of Grammars

Didier Caucal[1] and Stéphane Hassen[2]

[1] IGM–CNRS
didier.caucal@univ-mlv.fr
[2] IREMIA
shassen@univ-reunion.fr

Abstract. Deterministic graph grammars are finite devices which generate the transition graphs of pushdown automata. We define the notion of synchronization by grammars, generalizing previous sub-classes such as visibly and height-deterministic pushdown automata. The languages recognized by grammars synchronized by a given grammar form an effective boolean algebra lying between regular languages and deterministic context-free languages. We also provide a sufficient condition to obtain the closure under concatenation and its iteration.

1 Introduction

In recent literature, several restrictions of pushdown automata have been studied in order to define classes of languages which generalize regular languages while retaining good closure properties (namely closure under boolean operations, concatenation and its iteration). All these approaches consist in defining a notion of synchronization between pushdown automata.

The first such approach is to partition the input alphabet between pushing, popping and internal symbols, and to impose that stack movement only depend on the type of symbol read, yielding the so-called visibly pushdown automata [AM04]. This enforces that the stack height variation be entirely characterized by the input word.

A first generalization is to replace the partition of the input alphabet by a finite transducer assigning a weight to every input word. A pushdown automaton is synchronized by a transducer if two initial computations ending in the same configuration are labelled by words of the same weight, and we can only reach a finite number of configurations by initial computations of a given weight [Ca06].

A last generalization, defining the height-deterministic pushdown automata, is to synchronize a pushdown automaton by another pushdown automaton. The notion of synchronization is simply that two initial computations with the same input word end in configurations with the same height [NS07].

The classes of languages accepted by pushdown automata synchronized by a given partition of the input alphabet [AM04], a finite transducer [Ca06] or a pushdown automaton [NS07] are boolean algebras but, for the last two approaches, are not in general closed under concatenation and its iteration.

Instead of using the stack height of pushdown automata and doing a special treatment of the ε-moves, a general approach is to define the synchronization at a graph level. The transition graphs of the pushdown automata are generated by infinite parallel rewritings by (deterministic graph) grammars [MS85, Ca07]. The weight of a vertex in the generated graph is defined as the minimal number of steps of parallel rewritings necessary to produce it. The notion of synchronization is defined for grammars generating deterministic graphs which can have vertices of infinite in-degree. This allows a uniform treatment of real-time and non-real-time deterministic pushdown automata.

A grammar G is synchronized by a grammar H if for any initial path of (the graph generated by) G, there exists an initial path of H with the same label and these paths end in vertices of same weight. By extending usual constructions from finite automata to grammars, we show that the languages recognized by all grammars synchronized with a given grammar form a boolean algebra containing the regular languages, and we provide a simple sufficient condition for the closure under concatenation and its iteration.

It appears that the boolean algebras yielded by the previous notions of synchronization can all be captured using synchronization by grammars. We also show that the family of balanced languages [BB02], which is not synchronized according to the previous notions, fit our formalism.

2 Deterministic Graph Grammars

In this section, we recall the notion of deterministic graph grammar together with the family of graphs they generate: the regular graphs. For these graphs, we introduce the level of a vertex and for deterministic regular graphs, we define the weight of the label of an initial path. We end with a classical result on their recognized languages: the labels of their accepting paths are the context-free languages, and are the deterministic (resp. and real-time) context-free languages by restricting to deterministic regular graphs (resp. and of finite degree).

Let $\mathbb{N}$ be the set of natural numbers. For a set E, we write $|E|$ its cardinality, 2^E its powerset and for any $n \geq 0$, $E^n = \{\, (e_1, \ldots, e_n) \mid e_1, \ldots, e_n \in E \,\}$ is the set of n-tuples of elements of E. Thus $E^* = \bigcup_{n\geq 0} E^n$ is the free monoid generated by E for the *concatenation*: $(e_1, \ldots, e_m)\cdot(e'_1, \ldots, e'_n) = (e_1, \ldots, e_m, e'_1, \ldots, e'_n)$, and whose neutral element is the 0-tuple $()$. A finite set E of symbols is an *alphabet* of *letters*, and E^* is the set of *words* over E. Any word $u \in E^n$ is of *length* $|u| = n$ and is also represented by a mapping from $[n] := \{1, \ldots, n\}$ into E, or by the juxtaposition of its letters: $u = u(1)\ldots u(|u|)$. The neutral element is the word of length 0 called the *empty word* and denoted by ε.

Let F be a set of symbols called *labels*, ranked by a mapping $\varrho : F \longrightarrow \mathbb{N}$ associating to each label f its *arity* $\varrho(f) \geq 0$, and such that

$$F_n \ := \ \{\ f \in F \mid \varrho(f) = n\ \} \text{ is countable for every } n \geq 0.$$

We consider simple, oriented and labelled hypergraphs: a *hypergraph* G is a subset of $\bigcup_{n\geq 0} F_n V^n$, where V is an arbitrary set, such that

its *vertex* set $V_G \ := \ \{\ v \in V \mid FV^*vV^* \cap G \neq \emptyset\ \}$ is finite or countable,
its *label* set $F_G \ := \ \{\ f \in F \mid fV^* \cap G \neq \emptyset\ \}$ is finite.

Any $f v_1 \ldots v_{\varrho(f)} \in G$ is a *hyperarc* labelled by f and of successive vertices $v_1, \ldots, v_{\varrho(f)}$; it is depicted for

$\varrho(f) \geq 2$ as an arrow labelled f and successively linking $v_1, \ldots, v_{\varrho(f)}$;
$\varrho(f) = 1$ as a label f on vertex v_1 and f is called a *colour* of v_1 ;
$\varrho(f) = 0$ as an isolated label f called a *constant*.

This is illustrated in the figures throughout the paper. Note that a vertex v is depicted by a dot named (v) where parentheses are used to differentiate a vertex name from a vertex label (a colour).

For a subset $E \subseteq F$ of labels, we write

$$V_{G,E} \ := \ \{ \ v \in V \mid EV^* v V^* \cap G \neq \emptyset \ \} \quad = \quad V_{G \ \cap \ EV_G^*}$$

the set of vertices of G linked by a hyperarc labelled in E.

A *graph* G is a hypergraph whose labels are only of arity 1 or 2 : $F_G \subset F_1 \cup F_2$.

Hence a graph G is a set of *arcs* $a v_1 v_2$ identified with the labelled transition $v_1 \underset{G}{\xrightarrow{a}} v_2$ or directly $v_1 \xrightarrow{a} v_2$ if G is understood, plus a set of coloured vertices $f\,v$. A tuple $(v_0, a_1, v_1, \ldots, a_n, v_n)$ for $n \geq 0$ and $v_0 \underset{G}{\xrightarrow{a_1}} v_1 \ \ldots \ v_{n-1} \underset{G}{\xrightarrow{a_n}} v_n$ is a *path* from v_0 to v_n labelled by $u = a_1 \ldots a_n$; we write $v_0 \underset{G}{\overset{u}{\Longrightarrow}} v_n$ or directly $v_0 \overset{u}{\Longrightarrow} v_n$ if G is understood. For $P, Q \subseteq V_G$ and $u \in F_2^*$, we write

$$P \underset{G}{\overset{u}{\Longrightarrow}} Q \ \text{ if } \ p \underset{G}{\overset{u}{\Longrightarrow}} q \quad \text{for some } p \in P \text{ and } q \in Q$$
$$\text{and } \ L(G, P, Q) := \{ \ u \mid P \underset{G}{\overset{u}{\Longrightarrow}} Q \ \}$$

is the language recognized by G from P to Q. In these notations, we can replace P (and/or Q) by a colour i to designate the subset $V_{G,i}$. In particular $i \underset{G}{\overset{u}{\Longrightarrow}} Q$ means that there is a path labelled by u from a vertex coloured by i to a vertex in Q, and $L(G, i, f)$ is the label set of the paths from a vertex coloured by i to a vertex coloured by f.

In this paper, we only use two colours $i, f \in F_1$ to mark respectively the initial vertices and the final vertices. For any graph G, we denote

$L(G) \ := \ L(G, i, f)$ the *language recognized* by G
$L(G, i) \ := \ L(G, i, V_G)$ the *complete language recognized* by G.

Recall that the *regular languages* over an alphabet $T \subset F_2$ form the set:

$$Rat(T^*) \ := \ \{ \ L(G) \mid G \text{ finite } \wedge \ F_G \ \subseteq T \cup \{i, f\} \ \}.$$

A graph *grammar* R is a finite set of rules of the form $f x_1 \ldots x_{\varrho(f)} \longrightarrow H$ where $f x_1 \ldots x_{\varrho(f)}$ is a hyperarc joining pairwise distinct vertices $x_1 \neq \ldots \neq x_{\varrho(f)}$ and H is a finite hypergraph with $\{x_1, \ldots, x_{\varrho(f)}\} \ \subseteq \ V_H$; we denote by

$N_R := \{ \ f \in F \mid \exists \ x_1, \ldots, x_{\varrho(f)} \ f x_1 \ldots x_{\varrho(f)} \in Dom(R) \ \}$ the *non-terminals* of R, the labels of the left hand sides,

$T_R := \{ \ f \in F - N_R \mid \exists \ P \in Im(R), \ \ V_{P,f} \neq \emptyset \ \}$ the *terminals* of R, the labels of R which are not non-terminals,

$F_R := N_R \cup T_R$ the labels of R.

We use grammars to generate graphs. Hence in the following, we may assume that any terminal is of arity 1 or 2 : $T_R \subset F_1 \cup F_2$.

As for context-free grammars (on words), a graph grammar has an axiom: an initial finite hypergraph. To indicate this axiom, we assume that any grammar R has a unique constant non-terminal $Z \in N_R \cap F_0$; the *axiom* of R is the right hand side H of the rule of Z : $Z \longrightarrow H$.

To simplify, we add the condition that i only colours vertices of the axiom.

Starting from the axiom, we want that R generates a unique graph up to isomorphism. So we finally assume that any grammar R is *deterministic* meaning that there is only one rule per non-terminal:

$$(X,H)\,,\,(Y,K) \in R \;\wedge\; X(1) = Y(1) \quad\Longrightarrow\quad (X,H) = (Y,K).$$

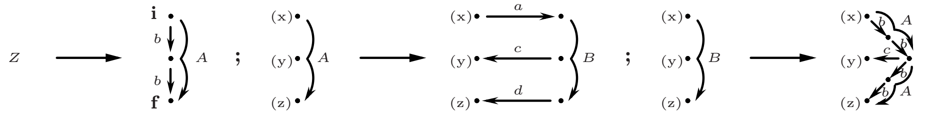
For any rule $X \longrightarrow H$, we say that

$$\begin{array}{lll} & V_X \cap V_H & \text{are the } \textit{inputs} \text{ of } H \\ \text{and} & \bigcup\{\, V_Y \mid Y \in H \;\wedge\; Y(1) \in N_R \,\} & \text{are the } \textit{outputs} \text{ of } H. \end{array}$$

To work with these grammars, it is simpler to assume that any grammar R is *terminal-outside* [Ca07] : any terminal arc or colour in a right hand side links a non input vertex:

$$H \;\cap\; (T_R V_X V_X \,\cup\, T_R V_X) \;=\; \emptyset \quad \text{for any rule } \;(X,H) \in R.$$

We will use upper-case letters $A, B, C, \ldots$ for non-terminals and lower-case letters $a, b, c \ldots$ for terminals. Here is an example of a (deterministic graph) grammar R :



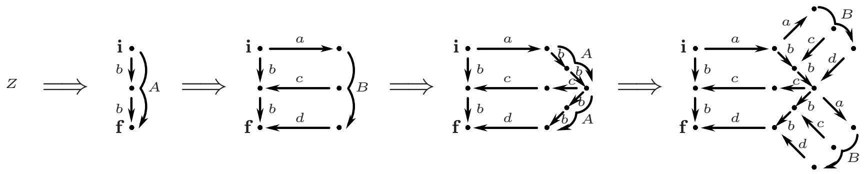
For this grammar R, we have $N_R \;=\; \{Z, A, B\}$, $T_R \;=\; \{a, b, c, d\}$ and x, y, z are the inputs of the last two rules (the axiom rule having no input).

Given a grammar R, the *rewriting* $\underset{R}{\longrightarrow}$ is the binary relation between hypergraphs defined as follows: M rewrites into N, written $M \underset{R}{\longrightarrow} N$, if we can choose a non-terminal hyperarc $X = As_1 \ldots s_p$ in M and a rule $Ax_1 \ldots x_p \longrightarrow H$ in R such that N can be obtained by replacing X by H in M :

$$N \;=\; (M - X) \;\cup\; h(H)$$

for some function h mapping each x_i to s_i, and the other vertices of H injectively to vertices outside of M; this rewriting is denoted by $M \underset{R,\,X}{\longrightarrow} N$. The rewriting $\underset{R,\,X}{\longrightarrow}$ of a hyperarc X is extended in an obvious way to the rewriting $\underset{R,\,E}{\longrightarrow}$ of any set E of non-terminal hyperarcs. The *complete parallel rewriting* $\underset{R}{\Longrightarrow}$ is the rewriting according to the set of all non-terminal hyperarcs: $M \underset{R}{\Longrightarrow} N$ if $M \underset{R,\,E}{\longrightarrow} N$ where E is the set of all non-terminal hyperarcs of M.

Taking the previous grammar, we depict in the next figure the first four steps of the parallel derivation from its constant non-terminal Z :



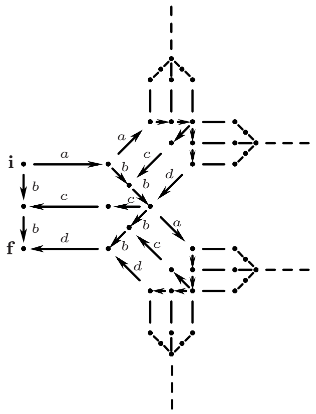
Given a deterministic grammar R and a hypergraph H, we denote

$$[H] \;:=\; H \cap T_R V_H^* \;=\; H \cap (T_R V_H V_H \cup T_R V_H)$$

the set of terminal arcs and of terminal coloured vertices of H.
A graph G is *generated* by R (from its axiom) if G belongs to the following set R^ω of isomorphic graphs:

$$R^\omega \;:=\; \{\, \bigcup_{n\geq 0}[H_n] \mid Z \underset{R}{\longrightarrow} H_0 \underset{R}{\Longrightarrow} \ldots H_n \underset{R}{\Longrightarrow} H_{n+1} \ldots \,\}\,.$$

For instance by iterating indefinitely the previous derivation, we get the infinite graph depicted below.



Grammars R and S are *equivalent* if they generate the same graph(s): $R^\omega = S^\omega$. A *regular graph* is a graph generated by a (deterministic graph) grammar.
Given a (regular) graph G generated by a grammar R:

$$G \;=\; \bigcup_{n\geq 0}[H_n] \quad \text{with} \quad Z \underset{R}{\longrightarrow} H_0 \underset{R}{\Longrightarrow} \ldots H_n \underset{R}{\Longrightarrow} H_{n+1} \ldots$$

we define the *level* $\ell(s)$ of a vertex $s \in V_G$, denoted also $\ell_G^R(s)$ to precise G and R, as being

$$\ell(s) \;:=\; \min\{\, n \mid s \in V_{H_n} \,\}$$

the minimal number of rewritings applied from the axiom to get s.
For any grammar R and for $G \in R^\omega$, we denote

$L(R) \;:=\; L(G)$ the *language recognized* by R

$L(R,i) \;:=\; L(G,i)$ the *complete language recognized* by R.

These languages are well-defined since generated graphs by a grammar are isomorphic. For instance, the previous grammar R recognizes the language $L(R)$ generated from A by the following context-free grammar:

$$A \;=\; bb \;+\; aAAd \;+\; aAccb$$

As the generated graph(s) by R is co-accessible from f (from any vertex, there is a path to a final vertex), $L(R,i)$ is the set of prefixes of the words in $L(R)$.

A graph G is (label) *complete* if for any arc label $a \in F_G \cap F_2$, any vertex $s \in V_G$ is source of an a-arc: $\exists\ t,\ s \underset{G}{\xrightarrow{a}} t$. For a grammar R generating a complete graph R^ω, we have $L(R,i) = (T_R - \{i,f\})^*$.

A graph G is *deterministic* if i colours a unique vertex, and two arcs with the same source have distinct labels: $r \underset{G}{\xrightarrow{a}} s \wedge r \underset{G}{\xrightarrow{a}} t \implies s = t$.

For a deterministic graph G generated by R, we can define the *weight* $\|\, u \,\|_R$ of any word $u \in L(R,i)$ by the level of the ending vertex of the initial path labelled by u:

$$\|\, u \,\|_R := \ell(s) \quad \text{for} \quad i \underset{G}{\overset{u}{\Longrightarrow}} s.$$

For the previous grammar R, we have $\|\, \varepsilon \,\|_R = \|\, b \,\|_R = \|\, bb \,\|_R = 0$ and

$$\|\, a \,\|_R = \|\, abbc \,\|_R = \|\, abbbb \,\|_R = 1 \ ; \ \|\, aa \,\|_R = 3.$$

The regular graphs of finite degree (any vertex is linked by a finite number of edges) are the transition graphs of pushdown automata restricted to a regular set of configurations. So the regular graphs of finite degree recognize the context-free languages. By adding ε-transitions, any regular graph (allowing vertices of infinite degree) recognizes a context-free language. Furthermore by identifying vertices linked by ε-edges on the transition graphs of deterministic pushdown automata, we get the deterministic regular graphs [Ca07].

Proposition 2.1. *The grammars (resp. generating deterministic graphs, deterministic graphs of finite degree) recognize the (resp. deterministic, deterministic and real-time) context-free languages.*

3 Synchronization of Grammars

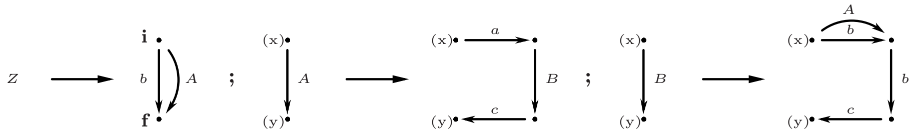
We introduce the synchronization as a binary relation between grammars generating deterministic graphs. To each grammar R, we associate the family $Sync(R)$ of the languages recognized by its synchronized grammars. By applying standard constructions on finite automata to synchronized grammars, we show that $Sync(R)$ is an extension of the regular languages which remains a boolean algebra (cf. Theorem 3.5).

In this section, we restrict to any grammar R generating a deterministic graph. Note that the determinism of R^ω is a first order sentence which is decidable.

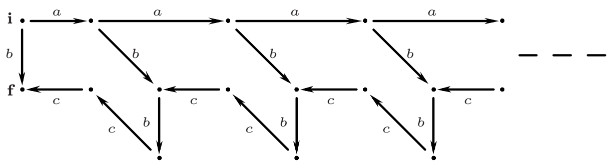
A grammar R *synchronizes* a grammar S and we write $R \rhd S$ or $S \lhd R$ if

$$\forall\, u \in L(S,i) \ \ (u \in L(R,i) \ \wedge \ \|\, u \,\|_R = \|\, u \,\|_S)$$

meaning that the label u of any initial path of the graph(s) generated by S is the label of an initial path of the graph generated by R, and these paths end in vertices of the same level. For instance the grammar of the previous section synchronizes the following one:



whose generated graph(s) is represented below by vertices of increasing level (the vertices of a same level are the vertices in a same vertical line).



Note that $\triangleright$ is a reflexive and transitive relation which is not antisymmetric; we denote $\bowtie$ the *bi-synchronization* relation:

$$R \bowtie S \quad \text{if} \quad R \triangleright S \text{ and } S \triangleright R.$$

So $R \bowtie S \iff R \triangleright S \wedge L(R,i) = L(S,i)$. Let us give a basic property of $\triangleright$.

Lemma 3.1. *$R \triangleright S$ and R^ω of finite degree $\Longrightarrow$ S^ω of finite degree.*

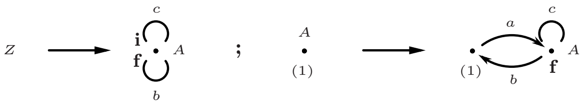
The grammar synchronization does not depend on colour f *i.e.* on final vertices. For instance the language recognized by the previous graph is $\{ a^n b (bcc)^n \mid n \geq 0 \}$ which has no common word with the language recognized by the graph of the previous section. For each grammar R, we associate the following family:

$$Sync(R) := \{ L(R) \cap L(S) \mid R \triangleright S \} \text{ of } \textit{synchronized languages} \text{ by } R.$$

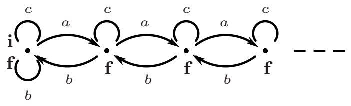
In particular $L(R) \in Sync(R)$

$$Sync(R) = \{ L(S) \mid R \triangleright S \} \text{ if any vertex of } R^\omega \text{ is final.}$$

Taking the following grammar R:

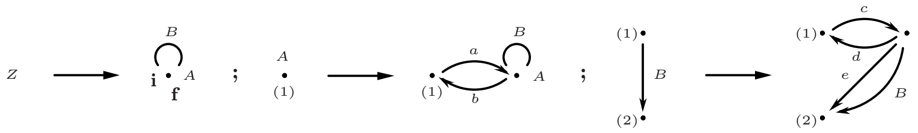


generating the following (label) complete graph:



its set $Sync(R)$ is the family of *visibly pushdown languages* for a a pushing letter, b a popping letter and c an internal letter [AM04]. This family is a boolean algebra containing all the regular languages and closed under concatenation $\cdot$ and under its Kleene iteration $*$. Except for the last two operations, we extend these closure properties to $Sync(R)$ for any grammar R, and we will give a general condition on R such that $Sync(R)$ is also closed under $\cdot$ and $*$.

Due to Proposition 2.1 and Lemma 3.1, we have more families of synchronized languages by allowing grammars generating graphs of infinite in-degree. For instance the following grammar R:



has a family $Sync(R) \neq Sync(S)$ for any grammar S such that S^ω is of finite degree.

The closure properties of $Sync(R)$ are just obtained by translating usual constructions on finite automata to synchronized grammars.

Recall that the *synchronization product* $G{\times}H$ of two graphs G and H is the graph

$$\begin{aligned} G{\times}H := \{\ (s,p) \xrightarrow{a} (t,q) \mid s \underset{G}{\xrightarrow{a}} t \ \wedge\ p \underset{H}{\xrightarrow{a}} q\ \} \\ \cup\ \{\ i(s,p) \mid is \in G \ \wedge\ ip \in H\ \} \ \cup\ \{\ f(s,p) \mid fs \in G \ \wedge\ fp \in H\ \}\,. \end{aligned}$$

So $L(G{\times}H,i) \ = \ L(G,i) \cap L(H,i)$ and $L(G{\times}H) \ = \ L(G) \cap L(H)$.

By synchronization product of a grammar R by a finite deterministic automaton, we get that any regular language included in $L(R)$ is a synchronized language of R.

Lemma 3.2. *For any grammar R and any regular language L, $L(R) \cap L \in Sync(R)$.*

Proof
We want to define a grammar synchronized by R and recognizing $L(R) \cap L$.

Let K be a finite deterministic graph (automaton) recognizing L: $L(K) = L$.

We order the vertices (states) of K: $V_K \ = \ \{q_1,\ldots,q_n\}$ with $n = |V_K|$.

To each non-terminal $A \in N_R$ of R, we associate a new symbol A' of arity $\varrho(A){\times}n$ except that $Z' = Z$.

To each non-terminal hyperarc $As_1\ldots s_m$ (with $A \in N_R$ and $m = \varrho(A)$), we associate the following hyperarc:

$$(As_1\ldots s_m)' \ := \ A'(s_1,q_1)\ldots(s_1,q_n)\ldots(s_m,q_1)\ldots(s_m,q_n)\,.$$

The synchronization product of R by K is the following grammar:

$$R{\times}K \ := \ \{\ \big(X',\, [H]{\times}K \cup \{\ Y' \mid Y \in H \ \wedge\ Y(1) \in N_R\ \}\big) \mid (X,H) \in R\ \}.$$

This grammar is synchronized by R.

As $G{\times}K \ \in (R{\times}K)^\omega$ for any $G \in R^\omega$, the grammar $R{\times}K$ recognizes

$$L(R{\times}K) \ = \ L(R) \cap L(K) \ = \ L(R) \cap L\,. \qquad \square$$

The synchronization product of a grammar by a deterministic finite automaton can be extended for two grammars synchronized by a given one.

Let S and S' be grammars synchronized by R: $R \ \triangleright\ S$ and $R \ \triangleright\ S'$.

To each non-terminal couple $(A,B) \in N_S{\times}N_{S'}$, we associate a new symbol $(A,B)'$ of arity $\varrho(A){\times}\varrho(B)$ except that $(Z,Z)' = Z$.

To each non-terminal hyperarc $As_1\ldots s_m$ of S ($A \in N_S$ and $m = \varrho(A)$) and each non-terminal hyperarc $Bt_1\ldots t_n$ of S' ($B \in N_{S'}$ and $n = \varrho(B)$), we associate the hyperarc:

$$(As_1\ldots s_m\,,\, Bt_1\ldots t_n)' \ := \ (A,B)'(s_1,t_1)\ldots(s_1,t_n)\ldots(s_m,t_1)\ldots(s_m,t_n)\,.$$

The *synchronization product* $S{\times}S'$ of S by S' is the grammar of the rules
$$(X,Y)' \longrightarrow [H]{\times}[K] \cup \{\, (fU,gV)' \mid fU \in H \wedge f \in N_S \wedge gV \in K \wedge g \in N_{S'} \,\}$$
for each $(X,H) \in S$ and $(Y,K) \in S'$.

The synchronization product of grammars synchronized by a grammar R is used to deduce the closure by intersection of $Sync(R)$.

Lemma 3.3. *For any grammars R, S, S', we have*
$$S \lhd R \wedge S' \lhd R \implies S{\times}S' \bowtie S'{\times}S \lhd S \quad \text{and} \quad (S{\times}S')^\omega = S^\omega{\times}S'^\omega$$
$$S \lhd R \implies R{\times}S \bowtie S$$
$Sync(R)$ is closed under intersection.

Proof
The grammar $S{\times}S'$ is synchronized by S and S' hence by R and
$$(S{\times}S')^\omega = S^\omega{\times}S'^\omega := \{\, G{\times}G' \mid G \in S^\omega \wedge G' \in S'^\omega \,\}.$$
Thus $L(S{\times}S') = L(S) \cap L(S')$ and $L(S{\times}S',i) = L(S,i) \cap L(S',i)$.
The closure under intersection of $Sync(R)$ results from the equality:
$$\big(L(R) \cap L(S)\big) \cap \big(L(R) \cap L(S')\big) = L(R) \cap L(S{\times}S').$$
Finally $R{\times}S \lhd S$ and $L(R{\times}S,i) = L(S,i)$ thus $R{\times}S \bowtie S$. □

The closure under intersection of $Sync(R)$ implies that
$$Sync(R) = \{\, L(S) \mid R \rhd S \wedge L(S) \subseteq L(R) \,\}.$$
Using Lemma 3.3, we show that we have the same families of synchronized languages by restricting to grammars R generating (deterministic) graphs R^ω which are accessible from i and co-accessible from f; in that case, $L(R,i)$ is the set of prefixes of $L(R)$.

For the closure under union, we define a generalized synchronization product $G{\otimes}H$ of graphs G and H such that $L(G{\otimes}H,i) = L(G,i) \cup L(H,i)$ and $L(G{\otimes}H) = L(G) \cup L(H)$.

As for the synchronization product $S{\times}S'$ of grammars S and S' synchronized by a grammar R, we define the product $S{\otimes}S'$ generating $(S{\otimes}S')^\omega = S^\omega \otimes S'^\omega$.

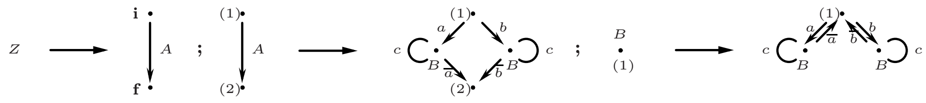
It follows the closure of $Sync(R)$ under union and complement with respect to $L(R)$. This also permits to decide whether R synchronizes S.

Lemma 3.4. *The synchronization relation $\rhd$ is recursive.*

We summarize previous results.

Theorem 3.5. *For any grammar R, the family $Sync(R)$ of synchronized languages is an effective boolean algebra with respect to $L(R)$, of deterministic context-free languages, containing all the regular languages included in $L(R)$.*

Let us give another family of synchronized languages than the visibly pushdown languages. Taking an internal letter c, two pushing letters a, b and their corresponding popping letters $\overline{a}, \overline{b}$, the following grammar R:



defines by synchronization the family $Sync(R)$ of balanced languages [BB02].

This family $Sync(R)$ is not closed under $\cdot$ and $*$. By extending standard constructions on finite automata for the closure under $\cdot$ and $*$, we will get families of synchronized languages closed under $\cdot$ and $*$. However the constructions need to extend the synchronization to grammars generating non deterministic graphs.

4 Synchronization of Weighted Grammars

We generalize the notion of synchronization to grammars, called weighted grammars, generating non-deterministic graphs. A grammar is weighted if in the generated graph two initial paths with the same label end in vertices of same weight. We show that weighted grammars can be in a certain sense determinized (cf. Proposition 4.2) and hence do not allow to capture new boolean algebras. However they allow to use on grammars the standard constructions for concatenation and its iteration (which introduce non-determinism). As a consequence, we provide a simple sufficient condition for the boolean algebras, defined by synchronization of grammars, to be closed under concatenation and iteration (cf. Theorem 4.3).

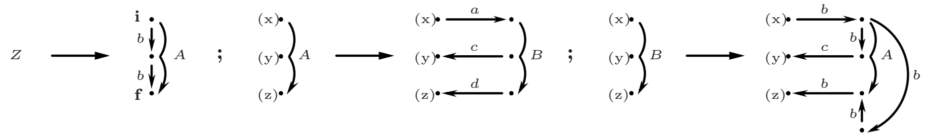
In this section, a deterministic graph grammar R can generate a non deterministic graph $G \in R^\omega$. We say that R is a *weighted grammar* if two initial paths with the same label end in vertices of the same level:

$$i \underset{G}{\overset{u}{\Longrightarrow}} s \ \wedge \ i \underset{G}{\overset{u}{\Longrightarrow}} t \quad \Longrightarrow \quad \ell(s) = \ell(t)$$

which allows to define the weight of any $u \in L(R,i)$ as above:

$$\|\, u \,\|_R := \ell(s) \quad \text{for} \quad i \underset{G}{\overset{u}{\Longrightarrow}} s.$$

Here is an example of a weighted grammar generating a non deterministic graph:



Any grammar generating a deterministic graph is weighted. Any weighted grammar generates a finite out-degree graph which can be of infinite in-degree. The decidability that a grammar generates a deterministic graph can be extended to the weighted property.

Lemma 4.1. *We can decide whether a grammar is weighted.*

The synchronization is generalized to weighted grammars R and S:

$$R \ \rhd \ S \quad \text{if} \quad \forall\, u \in L(S,i),\ u \in L(R,i) \ \wedge \ \|\, u \,\|_R \ = \ \|\, u \,\|_S$$

$$\text{and} \quad Sync_w(R) \ := \ \{\ L(R) \cap L(S) \mid R \rhd S \ \wedge \ S \text{ weighted} \,\}.$$

For instance the above grammar is synchronized by the grammar of the previous section. A key property is that any weighted grammar can be determinized.

Proposition 4.2. *Any weighted grammar can be transformed into an equivalent bi-synchronized grammar generating a deterministic graph.*

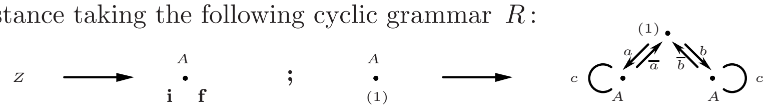
This implies that for any weighted grammar R, we can construct a grammar S generating a deterministic graph such that $Sync_w(R) = Sync(S)$.

Weighted grammars have been introduced for the closure under $\cdot$ and $*$ of $Sync(R)$ for grammars R generating deterministic graphs.

We say that R is a *cyclic grammar* if R^ω is deterministic and its initial vertex is the unique final vertex. In that case $L(R)$ is closed under concatenation.

Theorem 4.3. *For any cyclic grammar R, the family $Sync(R)$ is closed under $\cdot$ and under $*$.*

For instance taking the following cyclic grammar R:



the family $Sync(R)$ is the closure under concatenation and under iteration of the concatenation of the balanced languages [BB02].

5 Synchronization of Pushdown Automata

The synchronization of height-deterministic pushdown automata [NS07] and the synchronization with a transducer [Ca06] define language families synchronized by grammars.

We begin with the last approach of [NS07]. A (real-time) *pushdown automaton* in a weak form S over an alphabet T of *terminals* is a finite set of rules of the form:

$$Ap \xrightarrow{a} q \quad ; \quad Ap \xrightarrow{a} Aq \quad ; \quad Ap \xrightarrow{a} ABq$$

with $A, B \in P$, $p, q \in Q$, $a \in T$, where P, Q are disjoint alphabets of respectively *stack letters* and *states*. We associate to S a subset $F \subseteq Q$ of *final states*, and an *initial configuration* $c = \bot r$ for $r \in Q$ and $\bot \in P$ which cannot be popped ($\bot p \xrightarrow{a} q$ is not a rule of S). The *transition graph* $\mathrm{Tr}(S)$ of S is the set of its transitions

$$\{\ wu \xrightarrow{a} wv \mid u \underset{S}{\xrightarrow{a}} v \ \wedge\ w \in P^*\ \} \ \cup\ \{i\,c\} \ \cup\ \{\ fu \mid u \in P^*F\ \}$$

restricted to the vertices accessible from c. We denote $L(S) := L(Tr(S))$ the *language recognized* by S, and we say that S is *complete* if $L(Tr(S), i) = T^*$.

A complete pushdown automaton S is *height-deterministic* [NS07] if

$$c \underset{Tr(S)}{\overset{u}{\Longrightarrow}} xp \ \wedge\ c \underset{Tr(S)}{\overset{u}{\Longrightarrow}} yq \quad \Longrightarrow \quad |x| = |y|$$

meaning that two initial paths with the same label end in vertices of same length. Finally two height-deterministic pushdown automata S and S' are synchronized if

$$c \underset{Tr(S)}{\overset{u}{\Longrightarrow}} xp \ \wedge\ c' \underset{Tr(S')}{\overset{u}{\Longrightarrow}} yq \quad \Longrightarrow \quad |x| = |y|$$

and the family of languages synchronized by S is

$$Sync(S) \;:=\; \{\; L(S') \mid S' \text{ synchronized by } S \;\}.$$

As S is complete and $Sync(S)$ does not depend on the final states of S, $T^* \in Sync(S)$.

Any family $Sync(S)$ can be obtained by synchronization with a grammar.

Proposition 5.1. *We can transform any height-deterministic pushdown automaton S into a grammar R with R^ω deterministic of finite degree and $Sync(R) = Sync(S)$.*

Any family $Sync(S)$ contains T^*, hence cannot be the set of balanced languages. However and redefining $Sync(S) := \{\; L(S) \cap L(S') \mid S' \lhd S \;\}$, Proposition 5.1 remains true and its converse must be studied.

The synchronization of pushdown automata, and more generally grammars, by a transducer [Ca06] can also be captured using synchronization by a *linear grammar*: each right hand side has at most one non-terminal hyperarc. Taking the visibly pushdown languages, the converse is false.

In conclusion, the synchronization by grammars strictly generalizes the known synchronization notions of pushdown automata.

Many thanks to Arnaud Carayol and Antoine Meyer for their remarks and comments on this paper.

References

[AM04] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) 36^{th} STOC, ACM Proceedings, pp. 202–211 (2004)

[BB02] Berstel, J., Boasson, L.: Balanced grammars and their languages. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, vol. 2300, pp. 3–25. Springer, Heidelberg (2002)

[Ca07] Caucal, D.: Deterministic graph grammars. In: Flum, J., Grädel, E., Wilke, T. (eds.) Texts in Logic and Games 2, pp. 169–250. Amsterdam University Press (2007)

[Ca06] Caucal, D.: Synchronization of pushdown automata. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 120–132. Springer, Heidelberg (2006)

[MS85] Muller, D., Schupp, P.: The theory of ends, pushdown automata, and second-order logic. Theoretical Computer Science 37, 51–75 (1985)

[NS07] 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)

Lower Bounds for Depth-2 and Depth-3 Boolean Circuits with Arbitrary Gates

Dmitriy Yu. Cherukhin

Mech. and Math. Faculty, Moscow State University,
Leninskie Gory, Moscow, 119992, Russia

Abstract. We consider depth-2 and 3 circuits over the basis consisting of *all* Boolean functions. For depth-3 circuits, we prove a lower bound $\Omega(n \log n)$ for the size of any circuit computing the cyclic convolution. For depth-2 circuits, a lower bound $\Omega(n^{3/2})$ for the same function was obtained in our previous paper [10]. Here we present an improved proof of this bound. Both lower bounds are the best known for depth-3 and depth-2 circuits, respectively.

Keywords: Boolean function, circuit, complexity, depth, lower bound, cyclic convolution.

1 Introduction

Proving circuit lower bounds is one of the central mathematical problem in Computer Science. A considerable progress in this area has been made only for *weak* types of circuits, i.e. circuits satisfying certain strong restrictions, like monotone circuits or circuits of bounded depth over weak bases. For such circuits, exponential lower bounds are known.

For more traditional models, which have structural (and not computational) constraints, like formulas over the full basis, or switching and switching-and-rectifier networks, only polynomial lower bounds are known. We classify such circuits as *medium strength* circuits.

For the most practical model, namely, for unrestricted circuits over the full basis, only linear lower bounds are known. We classify this model as the *strong* one.

According to this classification, the circuits we consider in this paper are of medium strength. Specifically, we consider bounded depth circuits having arbitrary gates. In this model, the size of a circuit is defined as the number of wires in it. For every fixed depth d, there are explicit Boolean multi-output functions[1] that require circuits of superlinear size (in the maximum of the number of inputs and the number of outputs).

For $d = 2$, the best known lower bound is of the order $n^{3/2}$. It was obtained in our previous paper [10]. For $d > 2$, all known lower bounds are "almost" linear,

[1] A Boolean multi-output function is a mapping from $\{0,1\}^n$ to $\{0,1\}^k$ (for certain n, k).

that is, they are of the order $nf(n)$ where $f(n)$ is a function that grows slower than any function of the form n^{ε}. For $d = 3$, the function $f(n)$ is of order $\log n$, and for other $d > 3$ it is even smaller.

Note that every Boolean function of n variables is computed in our model by a circuit of size n and depth 1 (recall that we allow arbitrary gates). Likewise, any k-output Boolean function of n input variables can be computed by a circuit of size nk and depth 1. Thus superlinear lower bounds could be obtained only for k being an unbounded function of n. And there are no exponential lower bounds (in $\max\{n, k\}$) in our model.

In this paper, we present a slightly modified proof from the paper [10] of $\Omega(n^{3/2})$ lower bound for the size of depth-2 circuits. The new result in this paper is $\Omega(n \log n)$ lower bound for the size of depth-3 circuits. The best lower bound for depth-3 circuits known before was of the order $n \log\log n$ [6]. To prove the new lower bound we reduce depth-3 circuits to depth-2 circuits and then we use a method similar to that of [10].

We obtain our lower bounds for the *cyclic convolution* function (see the definition below). The same function was used in [10]. Our method applies also to other "multiplicative" functions, namely, to matrix multiplication (for depth-2 circuits) and to multiplication of polynomials over the field $\mathbb{Z}_2$. For multiplication-of-matrices-$n \times n$ function, for depth-2 circuits, we are able to prove the lower bound $\Omega(n^3)$, which matches the (trivial) upper bound $\mathcal{O}(n^3)$; see also new paper [11].

2 Previous Results and Proof Methods

In all the previous papers known to the author, the proof of a circuit lower bound (in the considered model) is based on a property of the graph underlying the circuit. Specifically, one defines a graph property such that any circuit computing the given function has that property. Then one proves that the number of edges in any graph having that property must exceed the lower bound one wants to show.

The graph property that is mostly used in this context is the following. A circuit (with n input nodes and n output nodes) has the property if it is a *superconcentrator*, that is, for every $k \leqslant n$ every set of k inputs is connected to every set of k outputs by a family of k vertex disjoint paths. For instance, every circuit computing the convolution function must be a superconcentrator [1]. It is known that the number of edges in every superconcentrator of constant depth is superlinear in n, which implies superlinear lower bounds for the size of any circuit of constant depth computing the convolution.

The first superlinear lower bound for superconcentrators of constant depth (depth 2) is due to Pippenger [2]. His result was improved and generalized to larger depths in a series of papers [3,5,6,8]. Now we know minimal size of a superconcentrator for every specific depth (up to a multiplicative constant). For the survey of these results, we refer to the paper [8].

For depth 2, the minimal number of edges in a superconcentrator is $\Theta(n \frac{\log^2 n}{\log\log n})$, and for depth 3 it is $\Theta(n \log\log n)$. These bounds were also the best known lower bounds for depth-2 and 3 circuits in our model. The papers [4,7,9] use even weaker graph properties than being a superconcentrator. Thus the lower bounds in those papers are weaker than the above ones.

Our method also uses a property of the graph underlying the circuit. We do not state that property explicitly, as we think that the property is not interesting in its own right, at least not as much, as being a superconcentrator. The idea of the proof (for depth-2 circuits) is the following. The function we consider (the cyclic convolution) depends on two groups of variables, $\tilde{x}$ and $\tilde{y}$. We pick a subset I of the first group $\tilde{x}$ and a subset O of output variables. For every evaluation of variables $\tilde{y}$ and remaining variables in $\tilde{x}$, we obtain a function from $\{0,1\}^I$ to $\{0,1\}^O$.

For cyclic convolution, there are many functions (for all choices of values of $\tilde{y}$ and remaining variables from $\tilde{x}$) obtained in this way. Therefore, there must be many edges in the circuit between inputs in I and outputs in O (to transmit the controlling information from the inputs $\tilde{y}$ and remaining inputs from $\tilde{x}$). As the depth equals 2, those edges are incident either to inputs, or to outputs. We obtain our lower bound by summing the number of such edges over all choices of I and O and taking into account the cyclic shifts.

Then we reduce depth-3 circuits to depth-2 circuits by modifying the underlying graph.

3 Basic Definitions and Main Results

A *Boolean* function of n variables is a function $f\colon \{0,1\}^n \to \{0,1\}$. A *multi-output Boolean* function is a function $f\colon \{0,1\}^n \to \{0,1\}^k$.

We define, for each integer n, an n-output Boolean function $H_n = (h_1, \ldots, h_n)$ of $2n$ input variables. Each h_j is a Boolean function of $2n$ variables that are the same variables for all h_j and are called $x_1, \ldots, x_n$, $y_1, \ldots, y_n$. The function h_j computes the value of the j-th output of H_n:

$$h_j(x_1, \ldots, x_n,\ y_1, \ldots, y_n) = x_1 y_j \oplus x_2 y_{j+1} \oplus \ldots \oplus x_n y_{j-1}\ . \quad (1)$$

We call H_n the *cyclic convolution*.

Now we are going to define the notion of a Boolean *circuit* of depth d with arbitrary gates that has $2n$ inputs and n outputs (and that computes H_n). Such circuit is identified by a triple $(G, g, \prec)$ satisfying the following conditions.

1) G is a finite directed graph.

2) The graph G has $2n$ *inputs* and n *outputs*. A node is called an input if it has no in-going edges. A node is called an output if it has no outgoing edges.

3) g is a mapping that assigns to each node v (which is not an input) a Boolean function which is *locally computed* in v; let $g[v]$ denote that function. The fan-in of $g[v]$ must be equal to the in-degree of v.

4) $\prec$ is a linear ordering on the nodes of G that has the following property: if there is an edge from a node v to a node w then $v \prec w$. This property implies

that G has no directed cycles. We also assume that the maximal length of a directed path in G is at most d.

Using the ordering $\prec$, we identify the edges going to a node v and the arguments of $g[v]$: the in-going edges are ordered according to the order on their origins.

Besides, using the ordering $\prec$ on inputs and outputs of the circuit, we identify the inputs with variables $x_1, \ldots, x_n,\ y_1, \ldots, y_n$ and the outputs with variables $z_1, \ldots, z_n$.

5) For every $j = 1, \ldots, n$ the function "globally" computed by output z_j must coincide with h_j. In the following two paragraphs we define formally the notion of the function $f[v]$ that is *globally computed* in a node v.

If v is an input then we let $f[v]$ be equal to the value of the variable identified with that input.

Assume that v is not an input and let $v_1, \ldots, v_k$ be all nodes such that there is an edge from v_i to v. Number them so that $v_1 \prec \ldots \prec v_k$ (where $\prec$ is the ordering in the definition of the circuit). Reasoning by induction (on the ordinal number of v in the order $\prec$), we may assume that $f[v_1], \ldots, f[v_k]$ are defined. Let

$$f[v](\tilde{x}) \equiv g[v](f[v_1](\tilde{x}), \ldots, f[v_k](\tilde{x})) \ , \tag{2}$$

where $\tilde{x} = (x_1, \ldots, x_n,\ y_1, \ldots, y_n)$.

The number of edges in G is called the *size* of S; we use the notation $L(S)$ for the size of a circuit S.

In this paper, we prove the following theorems.

Theorem 1. *If $d = 2$ then $L(S) = \Omega(n^{3/2})$.*

Theorem 2. *If $d = 3$ then $L(S) = \Omega(n \log n)$.*

In the rest of the paper, we prove these theorems. First we prove Lemma 1 (in Section 4), which is the main complexity-theoretic ingredient in our method. Then we derive certain its corollaries, and Theorem 1 is one of them. The other one is used in the proof of Theorem 2. In Section 5, we present a general Lemma 2, which generalizes a lemma from [5]. Finally, we prove Theorem 2 in Section 6.

We conclude this section by a remark. The set of nodes in a circuit can be partitioned into *levels*. A node v is on the level k if k is the maximal length of a directed path from an input to v. For instance, all inputs belong to level 0. By our assumptions, the number of levels in the circuit S is at most d. Without loss of generality we may assume that, in the circuit S, all outputs belong to level d and every edge goes from a level i to the level $i + 1$ (for some $i < d$).

Indeed, we can insert fictitious nodes into every edge going from a level i to level $j > i + 1$. This transformation increases the number of edges at most d times. As d is constant and the bounds of Theorems 1 and 2 are asymptotic, we can afford such increase.

4 Depth-2 Circuits. Lemma 1 and Its Corollaries

Assume that f and $f_1, \ldots, f_k$ are Boolean functions of variables $\tilde{y} = (y_1, \ldots, y_n)$. We say that f is *expressible* through $f_1, \ldots, f_k$ if for some Boolean function Φ of k variables we have

$$f(\tilde{y}) \equiv \Phi(f_1(\tilde{y}), \ldots, f_k(\tilde{y})) \ .$$

As there are 2^{2^k} different functions Φ of k variables, there are at most that much functions expressible through $f_1, \ldots, f_k$. On the other hand, if $f_1, \ldots, f_k$ are different variables, that bound is attained — we can express every of 2^{2^k} different functions of k variables through $f_1 = y_1, \ldots, f_k = y_k$.

In this section, we assume that S is a circuit of depth $d = 2$. Recall that we assume that every edge in S goes from a level $i-1$ to the level i, for some i. In this case we say that the edge belongs to level i. We number levels in S by 0,1,2, where 0 is the bottom level (containing inputs) and 2 is the top level (containing outputs). Let L_i denote the number of edges in the i-th level. Some of the nodes of the *middle* (i.e., first) level connected to all inputs will be called *special.* Let L_2^* stand for the number of edges in the second level that are not incident to special nodes (that is, edges connecting outputs with non-special nodes). The number L_2^* depends on the choice of special nodes. The following Lemma holds for every choice of special nodes, satisfying the above constraint.

Lemma 1. *Let k and l be natural numbers and $kl \leqslant n$. Then we have*

$$kL_1 + lL_2^* \geqslant nkl \ .$$

Proof. Let v be a vertex in G. Consider the function $f[v]$ of the input variables that is globally computed in v. Define functions $f_0[v], f_1[v], \ldots, f_n[v]$ of variables $y_1, \ldots, y_n$ as follows. The function $f_0[v]$ is obtained by substituting zeroes for all variables $x_1, \ldots, x_n$ in $f[v]$. The function $f_i[v]$ is obtained by substituting 1 for x_i and zeroes for the remaining variables $x_1, \ldots, x_n$ in $f[v]$. Thus $f_0[v], f_1[v], \ldots, f_n[v]$ are sub-functions of $f[v]$.

Let J be the set of the first l natural numbers that are congruent to 1 modulo k, that is, $J = \{1, k+1, \ldots, lk-k+1\}$. Let z_j be j-th output node and $\mathcal{F}$ the set of all functions $f_i[z_j]$, for $1 \leqslant i \leqslant k$ and $j \in J$. As j-th output of S computes h_j, the equality (1) implies that $f_i[z_j]$ is equal to y_{i+j-1}. Note $i+j-1$ takes all values in the range $1, \ldots, kl$, as i ranges over $1, \ldots, k$, and j over J. Hence the set $\mathcal{F}$ consists of independent variables $y_1, \ldots, y_{kl}$.

Let X_i stand for the set of all nodes in the middle level that are connected to the input x_i and let Z_j denote the set of nodes in the middle level connected to the output z_j. Note that by (2) the function $f[z_j]$ is expressible through the functions $f[v]$ for $v \in Z_j$. Then the function $f_i[z_j]$ is expressible through the functions $f_i[v]$ for $v \in Z_j$, since substitutions of constants for variables preserve equalities and thus the expressibility property.

Let $\mathcal{G}$ denote the set of all the functions $f_i[v]$, where $1 \leqslant i \leqslant k$, $v \in Z_j$ and $j \in J$. It is easy to see that all functions from the set $\mathcal{F}$ are expressible through the functions from $\mathcal{G}$. However, $\mathcal{F}$ consists of kl independent variables. Thus, there are $2^{2^{kl}}$ functions expressible through $\mathcal{F}$. However, every function expressible through $\mathcal{F}$ is also expressible through $\mathcal{G}$ (since the expressibility property is transitive). Thus there are at least $2^{2^{kl}}$ functions expressible through $\mathcal{G}$, and hence

$$|\mathcal{G}| \geqslant kl \ . \tag{3}$$

We now come to the central point of the proof. If node v of the middle level is not connected to the input x_i, then the function $f[v]$ does not depend on x_i, hence, $f_i[v] = f_0[v]$. Let us thus replace $f_i[v]$ by $f_0[v]$ everywhere in $\mathcal{G}$ where it is possible. Now the set $\mathcal{G}$ contains only those functions $f_i[v]$ for which the node v is connected to the input x_i, i.e., $v \in X_i$. In addition, the set $\mathcal{G}$ contains the functions $f_0[v]$ such that $v \in Z_j$, $j \in J$, and the node v is not connected to at least one of the inputs $x_1, \ldots, x_k$.

Recall that every special node is connected to all inputs. Therefore, the set $\mathcal{G}$ contains only the functions $f_i[v]$ for $v \in X_i$, $1 \leqslant i \leqslant k$, and the functions $f_0[v]$ for non-special $v \in Z_j$ and $j \in J$. Let Z_j^* be the set of non-special nodes from Z_j. Then

$$|\mathcal{G}| \leqslant \sum_{i=1}^{k} |X_i| + \sum_{j \in J} |Z_j^*| \ .$$

Together with (3) it yields

$$kl \leqslant \sum_{i=1}^{k} |X_i| + \sum_{j \in J} |Z_j^*| \ . \tag{4}$$

Note that the proof above does not change when i ranges not over $1, 2, \ldots, k$, but over any other set obtained from it by a cyclic shift modulo n. Similarly, we can change the range of j (i.e., the set J). For simplicity, we will shift i and j synchronously. For each of the resulting n shifts an inequality similar to (4) holds. Summing all these inequalities, we get

$$nkl \leqslant k \sum_{i=1}^{n} |X_i| + l \sum_{j=1}^{n} |Z_j^*| \ . \tag{5}$$

To conclude the proof, we note that the first sum in (5) equals L_1, and the second one is L_2^*. □

Corollary 1 (Theorem 1). $L(S) = \Omega(n^{3/2})$.

Proof. Choose special nodes in an arbitrary way so that the above constraint is satisfied (say, declare all nodes non-special). Applying the lemma to $k = l = [\sqrt{n}]$ we get

$$[\sqrt{n}]L(S) = [\sqrt{n}](L_1 + L_2) \geqslant [\sqrt{n}](L_1 + L_2^*) \geqslant n \cdot [\sqrt{n}]^2 \ . \qquad \square$$

A by-product of the lemma is the following corollary that will be used in the proof of our second theorem.

Corollary 2. *If there are at most $\frac{n}{2}$ special nodes, then $L_1 \cdot L_2^* \geqslant \frac{n^3}{16}$.*

Proof. Note that there are at least n nodes in the middle level. It follows from the information transmission between the inputs and the outputs. Namely, the functions $f_1[z_1], \ldots, f_1[z_n]$ are in fact different variables; however, they are expressible through the functions $f_1[v]$ for nodes v from the middle level. Thus, the middle level contains at least n nodes.

Therefore, by the assumption of our corollary there are at least $\frac{n}{2}$ non-special nodes in the middle level. Every node of the middle level has an outgoing edge (since they are not outputs), thus $L_2^* \geqslant \frac{n}{2}$. Furthermore, every input also has an outgoing edge, thus $L_1 \geqslant n$.

Let us distinguish three cases:

1. $L_1 \geqslant \frac{n^2}{2}$;
2. $L_2^* \geqslant n^2$;
3. $L_1 < \frac{n^2}{2}$ and $L_2^* < n^2$.

Case 1: We thus have $L_1 \cdot L_2^* \geqslant \frac{n^2}{2} \cdot \frac{n}{2} = \frac{n^3}{4}$.
Case 2: Similarly, $L_1 \cdot L_2^* \geqslant n \cdot n^2 = n^3$.
Case 3: Choose k and l as follows:

$$k = \left[\left(n \cdot \frac{L_2^*}{L_1}\right)^{1/2}\right], \quad l = \left[\left(n \cdot \frac{L_1}{L_2^*}\right)^{1/2}\right].$$

The condition of this case implies that $nL_1 \geqslant n^2 > L_2^*$ and $nL_2^* \geqslant \frac{n^2}{2} > L_1$. Thus, both k and l are positive integer numbers.

Furthermore, $kl \leqslant n$. Applying lemma 1 and using the inequality $[x] \geqslant \frac{x}{2}$ for $x \geqslant 1$, we get

$$n \leqslant \frac{L_1}{l} + \frac{L_2^*}{k} = \frac{L_1}{\left[\left(n \cdot \frac{L_1}{L_2^*}\right)^{1/2}\right]} + \frac{L_2^*}{\left[\left(n \cdot \frac{L_2^*}{L_1}\right)^{1/2}\right]} \leqslant$$

$$\leqslant \frac{2L_1}{\left(n \cdot \frac{L_1}{L_2^*}\right)^{1/2}} + \frac{2L_2^*}{\left(n \cdot \frac{L_2^*}{L_1}\right)^{1/2}} = 4\left(\frac{L_1 L_2^*}{n}\right)^{1/2}.$$

Thus, $L_1 L_2^* \geqslant \frac{n^3}{16}$. □

5 Lemma 2

The following lemma is a generalization of a lemma by Pudlák [5, Lemma 4]. Its original proof was simplified by an anonymous referee.

Lemma 2. *Assume* $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ *are nonnegative real numbers such that* $a_1 \geqslant \ldots \geqslant a_n$ *and*

$$\begin{aligned} a_1 + a_2 + \ldots + a_n &\geqslant b_1 + b_2 + \ldots + b_n \ , \\ a_2 + \ldots + a_n &\geqslant b_2 + \ldots + b_n \ , \\ &\ldots \\ a_n &\geqslant b_n \ . \end{aligned} \tag{6}$$

Let $\varphi \colon \mathbb{R}_+ \to \mathbb{R}_+$ *be an increasing concave function. Then*

$$\varphi(a_1) + \ldots + \varphi(a_n) \geqslant \varphi(b_1) + \ldots + \varphi(b_n) \ .$$

Note that the requirement that a_i's and b_i's are nonnegative is redundant if these numbers are in the domain of φ. We state this requirement since we will apply this lemma just to the function $\sqrt{x}$, which is defined on nonnegative numbers.

Proof. First we prove that if $a \geqslant b \geqslant \varepsilon > 0$, then

$$\varphi(a) + \varphi(b) \geqslant \varphi(a + \varepsilon) + \varphi(b - \varepsilon) \ . \tag{7}$$

In words: if the largest of the numbers of a, b is increased by ε, and the smallest one is decreased by ε, then the sum $\varphi(a) + \varphi(b)$ does not increase.

Indeed, denote $\delta = \frac{\varepsilon}{a-b+2\varepsilon}$. Then by Jensen's inequality

$$\begin{aligned} \delta\varphi(b - \varepsilon) + (1 - \delta)\varphi(a + \varepsilon) \leqslant \varphi(\delta(b - \varepsilon) + (1 - \delta)(a + \varepsilon)) = \\ \varphi(a + \varepsilon - \delta(a - b + 2\varepsilon)) = \varphi(a) \ . \end{aligned}$$

Similarly,

$$(1 - \delta)\varphi(b - \varepsilon) + \delta\varphi(a + \varepsilon) \leqslant \varphi(b) \ .$$

Summing the last two inequalities we get (7).

We prove this lemma by induction on n. The base is $n = 1$. In this case the claim follows from (6) and the assumption that φ is increasing.

We now prove the induction step ($n \geqslant 2$). Increase a_1 and decrease a_n by ε, where ε is the maximum possible number such that all the inequalities (6) are satisfied after this change. The sum $\varphi(a_1) + \ldots + \varphi(a_n)$ does not increase due to this change. This follows from the inequality $a_1 \geqslant a_n$ and the inequality (7). Hence, if we are able to prove the claim for the new numbers $a_1, \ldots, a_n$, the claim for the former numbers will follow.

We now prove the claim for the new numbers $a_1, \ldots, a_n$. By the maximality of ε, at least one of the inequalities (6) (except for the first one) becomes an equality. Indeed, increasing a_1 by ε and decreasing a_n by ε does not change the sum $a_1 + \ldots + a_n$. Thus the first inequality of the system (6) remains intact. However, the left-hand side of all subsequent inequalities decreases. Thus the maximality of ε implies that one of the subsequent inequalities has become an equality.

Thus, for some $k \geqslant 2$ we have

$$a_k + \ldots + a_n = b_k + \ldots + b_n \ . \tag{8}$$

Denote the system (6) by $\Phi(a_1, \ldots, a_n;\ b_1, \ldots, b_n)$. Subtract the equality (8) from the first $k-1$ inequalities of this system. Then the system (6) splits into two independent systems of the same type, namely,

$$\Phi(a_1, \ldots, a_{k-1};\ b_1, \ldots, b_{k-1}) \ ,$$
$$\Phi(a_k, \ldots, a_n;\ b_k, \ldots, b_n) \ .$$

Applying the induction hypothesis to these two systems, we get

$$\varphi(a_1) + \ldots + \varphi(a_{k-1}) \geqslant \varphi(b_1) + \ldots + \varphi(b_{k-1}) \ ,$$
$$\varphi(a_k) + \ldots + \varphi(a_n) \geqslant \varphi(b_k) + \ldots + \varphi(b_n) \ .$$

Finally, summing the two last inequalities we get the desired claim. □

6 Circuits of Depth 3. Proof of Theorem 2

In this section we assume that $d = 3$. Denote the nodes of the second level (i.e., the level preceding the outputs) by $v_1, \ldots, v_t$. Let $d^+(v_i)$ be the number of edges going into v_i, and $d^-(v_i)$ the number of edges going out v_i. Define the number a_i as

$$a_i = d^+(v_i) \cdot d^-(v_i) \ , \quad i = 1, \ldots, t \ .$$

Re-numbering v_i if needed, we can assume that $a_1 \geqslant a_2 \geqslant \ldots \geqslant a_t$.

Let $m = [n/2]$. For each $p = 1, 2, \ldots, m$, we transform the circuit S into a new depth-2 circuit S_p that implements the same function H_n. Namely, we move the nodes $v_1, \ldots, v_{p-1}$ to the first level of the circuit and connect each input to each such node (and we change the gates in $v_1, \ldots, v_{p-1}$ so that the function globally computed in v_i is preserved).

Then we remove the nodes $v_p, \ldots, v_t$ from the circuit. To preserve the functionality, we add new edges when eliminating v_i. Namely, if there is an edge from node w of the first level to node v_i and an edge from v_i to an output z_j, then we add a new edge directly from w to z_j. We proceed this way for each pair (w, z_j). Thus eliminating v_i results in adding a_i new edges to the circuit. Then we change the gates in each output w so that the function globally computed in w is preserved. The resulting circuit is denoted by S_p.

The nodes $v_1, \ldots, v_{p-1}$ will be *special* for S_p (recall the notion of a special node from Sect. 4). The number of special nodes is at most $n/2$, thus we can apply Corollary 2. Recall that L_i denotes the number of edges of the i-th level, and L_2^* denotes the number of edges of the second level leaving non-special nodes. We specify the circuit for which we count the number of these edges in parentheses (for example, $L_2^*(S_p)$). Corollary 2 yields

$$L_1(S_p) \cdot L_2^*(S_p) \geqslant \frac{n^3}{16} \ . \tag{9}$$

We now compute $L_1(S_p)$ and $L_2^*(S_p)$. Every edge of the first level of S_p either was there in S or was added. Since we have added edges connecting the $2n$ inputs and the nodes $v_1, \ldots, v_{p-1}$ the number of additional edges is $2n(p-1)$. Therefore,

$$L_1(S_p) = L_1(S) + 2n(p-1) \ .$$

The edges of the second level that leave non-special nodes are exactly the edges that we added to S_p when eliminating the nodes $v_p, \ldots, v_t$, i.e.,

$$L_2^*(S_p) = \sum_{i=p}^{t} a_i \ .$$

Denote $\theta = \frac{L_1(S)}{2n}$. By substituting the values for $L_1(S_p)$ and $L_2^*(S_p)$ into the inequality (9) we get

$$\sum_{i=p}^{t} a_i \geqslant \frac{n^3}{16(L_1(S) + 2(p-1)n)} = \frac{n^2}{32(\theta + p - 1)} \ . \tag{10}$$

To apply Lemma 2, we introduce the numbers $b_1, \ldots, b_t$ as follows:

$$b_p = \tfrac{n^2}{32}\left(\tfrac{1}{\theta+p-1} - \tfrac{1}{\theta+p}\right) \ , \quad p = 1, \ldots, m-1 \ ,$$
$$b_m = \tfrac{n^2}{32}\left(\tfrac{1}{\theta+m-1}\right) \ , \quad b_{m+1} = \ldots = b_t = 0 \ .$$

Note that the system (6) now follows from the inequalities (10) for different values of p. Indeed, b_i is defined as the difference between the two numbers from the right-hand side of (10). Thus, after intermediate terms cancel in $b_p + \ldots + b_t$, we are left with the first number, which is on the right in (10). Note also that the last inequalities of the system (6) that do not have matching inequalities in (10) hold since $b_{m+1} = \ldots = b_t = 0$ and $a_i \geqslant 0$.

Applying Lemma 2 to the function $\varphi(x) = \sqrt{x}$ we get

$$\sum_{i=1}^{t} \sqrt{a_i} \geqslant \sum_{i=1}^{m} \left(\frac{n^2}{32}\left(\frac{1}{\theta+i-1} - \frac{1}{\theta+i}\right)\right)^{1/2} = \sum_{i=1}^{m} \frac{n}{(32(\theta+i-1)(\theta+i))^{1/2}} \geqslant$$
$$\frac{n}{4\sqrt{2}} \sum_{i=1}^{m} \frac{1}{\theta+i} = \frac{n}{4\sqrt{2}} \left(\ln(\theta+m) - \ln\theta + O(1)\right) \ . \tag{11}$$

On the other hand, by the definition of a_i's, we have

$$\sqrt{a_i} = \left(d^+(v_i) \cdot d^-(v_i)\right)^{1/2} \leqslant \frac{d^+(v_i) + d^-(v_i)}{2} \ . \tag{12}$$

Finally, note that the edges of the second level of S are exactly the edges entering the nodes $v_1, \ldots, v_t$, and the edges of the third level are the edges leaving these nodes. Hence,

$$L_2(S) = \sum_{i=1}^{t} d^+(v_i) \ , \quad L_3(S) = \sum_{i=1}^{t} d^-(v_i) \ .$$

In total, by summing the inequalities (12) we get

$$\sum_{i=1}^{t} \sqrt{a_i} \leqslant \frac{1}{2} \sum_{i=1}^{t} \left(d^+(v_i) + d^-(v_i)\right) = \frac{1}{2} \left(L_2(S) + L_3(S)\right) \ . \qquad (13)$$

To conclude, we consider the two possible cases: if $\theta \geqslant \ln n$, then $L(S) \geqslant L_1(S) = 2\theta n \geqslant 2n \ln n$; otherwise (11) and (13) imply

$$L(S) \geqslant L_2(S) + L_3(S) = \Omega\left(\frac{n}{2\sqrt{2}} \ln\left(1 + \frac{m}{\theta}\right)\right) = \Omega(n \ln n) \ .$$

Therefore, in both cases $L(S) = \Omega(n \log n)$, which proves Theorem 2.

7 Conclusion

We have a feeling that, using a more elaborate graph transformations, it is possible to improve the known lower bounds for every fixed depth. The first step in this direction would be a transformation of a depth-4 circuit into a depth-2 circuit. On the other hand, it is also interesting to generalize our method for depth-2 circuits to circuits of larger depth.

Acknowledgements. The author is grateful to the anonymous referee for simplifying the original proof of Lemma 2 and to the Program Committee of CSR-2008 (specially to Nikolay Vereshchagin and Edward A. Hirsch) for the interest to the paper and for the help in translating the camera ready copy into English. The work is partially supported by President RF grant for leading scientific school support NSh–4470.2008.1.

References

1. Valiant, L.G.: Graph-theoretic arguments in low-level complexity. In: Gruska, J. (ed.) MFCS 1977. LNCS, vol. 53, pp. 162–176. Springer, Heidelberg (1977)
2. Pippenger, N.: Superconcentrators of depth 2. J. of Computer and System Sciences 24, 82–90 (1982)
3. Dolev, D., Dwork, C., Pippenger, N., Wigderson, A.: Superconcentrators, generalizers and generalized connectors with limited depth. In: Proc. 15th ACM STOC, pp. 42–51 (1983)
4. Pudlák, P., Savický, P.: On shifting networks. Theoretical Comput. Sci. 116, 415–419 (1993)
5. Pudlák, P.: Communication in Bounded Depth Circuits. Combinatorica 14(2), 203–216 (1994)
6. Alon, N., Pudlák, P.: Superconcentrators of depth 2 and 3; odd levels help (rarely). J. of Computer and System Sciences 48, 194–202 (1994)
7. Pudlák, P., Rödl, V., Sgall, J.: Boolean circuits, tensor ranks and communication complexity. SIAM J. on Computing 26(3), 605–633 (1997)
8. Radhakrishnan, J., Ta-Shma, A.: Bounds for dispersers, extractors and depth-two superconcentrators. SIAM J. of Discrete Mathematics 13(1), 2–24 (2000)

9. Raz, R., Shpilka, A.: Lower Bounds for Matrix Product, in Bounded Depth Circuits with Arbitrary Gates. SIAM J. Comput. 32(2), 488–513 (2003)
10. Cherukhin, D.Y.: The lower estimate of complexity in the class of schemes of depth 2 without restrictions on a basis. Moscow Univ. Math. Bull. 60(4), 42–44 (2005)
11. Jukna, S.: Entropy of operators or why matrix multiplication is hard for depth-two circuits (manuscript, 2008), `www.thi.informatik.uni-frankfurt.de/~yukna`

A Semantic Proof of Polytime Soundness of Light Affine Logic

Ugo Dal Lago[1] and Martin Hofmann[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna
[2] Institut für Informatik, LMU München

Abstract. We define a denotational semantics for Light Affine Logic (LAL) which has the property that denotations of functions are polynomial time computable by construction of the model. This gives a new proof of polytime-soundness of LAL which is considerably simpler than the standard proof based on proof nets and also is entirely semantical in nature. The model construction uses a new instance of a resource monoid; a general method for interpreting variations of linear logic with complexity restrictions introduced earlier by the authors.

1 Introduction

In recent years, a large number of characterizations of complexity classes based on logics and lambda calculi have appeared. At least three different principles have been exploited, namely linear types [3,10], restricted modalities in the context of linear logic [8,2,13] and non-size-increasing computation [9,1]. These systems have been studied with different, often unrelated methodologies. In particular, proofs of soundness (any function which is representable in the system lies in a complexity class) are usually quite complex and cannot be easily generalized. As a consequence, unifying, reasonably simple frameworks for the analysis of quantitative properties of computation are desirable. This would help to improve the understanding on existing systems, since proofs of soundness, especially conceptually simple ones, often shed light on the *reasons why* the system under consideration enjoys certain quantitative properties.

While we take the significance of LAL itself more or less for granted in this paper we may point out that it is the first system that characterises polynomial time without recourse to explicit resource bounds as found e.g. in Bounded Arithmetic and in addition allows one to define inductive datatypes as certain fomulas by impredicative quantification. Functions acting on those datatypes can thus be naturally represented as proofs via the well-known Curry-Howard correspondence (e.g. logical implication corresponds to functional types). And, noticeably, the class of representable first-order functions *equals* the polynomial time functions. One can thus view LAL as the first resource-free and purely proof-theoretic characterisation of polynomial time.

In a previous paper [7], we have introduced a new semantical framework which consists of an innovative modification of realizability whereby realizers and their runtime are bounded by elements of a certain algebraic structure, a *resource monoid*. The axioms for resource monoids are such that for any resource monoid the category of corresponding realizability sets is symmetric monoidal closed and supports second-order quantification, i.e., impredicative polymorphism. With particular resource monoids one can then realize further constructs and type formers such as modalities or recursors. In [7] we have introduced resource monoids and provided concrete instances for LFPL [9] and Elementary Affine Logic (EAL, see [5]). A fairly complicated resource monoid for LAL with a consequently rather technical and unenlightening proof of correctness has been presented in [7].

In this paper we provide a very simple resource monoid for LAL. Not only do we obtain in this way a new, simpler, and conceptually appealing proof of polytime soundness (all definable functions on binary strings are polynomial time computable) for LAL; we also find that the resource monoid we obtain is quite natural; its members are triples (n, m, f) with $n, m \in \mathbb{N}$ and f a monotonically increasing polynomial-time function; the monoid operation which interprets tensor product is given by

$$(n, m, f) + (l, k, g) = (n + l, \max(m, k), \max(f, g)).$$

The order relation between these monoid elements is given by

$$(n, m, f) \leq (l, k, g) \iff (n \leq l) \wedge (n + m \leq l + k) \wedge (f \leq g).$$

The interpretation of the modalities ! and § of LAL uses the functional $f \mapsto \lambda x.x^2 f(x^2)$ which explains that bounding functions extracted from the interpretation are polynomials whose degree grows exponentially with the nesting depth of the modalities as is expected from the known proof based on proof nets and also the known hardness examples. Some formulae which are not provable syntactically can be justified in the semantics. An example is the distributive law $\S(A \otimes B) \multimap \S A \otimes \S B$.

The formal similarity of our resource monoid with the one for LFPL from [7] raises hopes for a system that somehow combines LFPL and LAL; we have to admit that these hopes have not, as yet, materialised if one discounts trivial solutions like the disjoint union of the two systems.

Related work. Semantic models for LAL exist [15,14]; however none of these models yields a proof of polytime soundness. More generally, the method of realizability has been used in connection with resource-bounded computation in several places. The most prominent is Cook and Urquhart's work [4], where terms of a language called PV^ω are used to realize formulas of bounded arithmetic. The contribution of that paper is related to ours in that realizability is used to show "polytime soundness" of a logic. There are important differences though. First, realizers in [4] are typed and very closely related to the logic that is being realized. Second, the language of realizers PV^ω only contains first-order recursion and is therefore too weak for systems with like LFPL or LAL that contain or define recursion with higher-order result types. In contrast, we use untyped realizers and interpret types as certain partial equivalence relations on those. This links our work to the untyped realizability model HEO (due to Kreisel [12]). This, in turn, has also been done by Crossley et al. [6]. There, however, one proves externally that untyped realizers (in this case of bounded arithmetic formulas) are polytime, whereas our realizers are polytime bounded by construction.

2 Preliminaries

As in any realizability model, we need to introduce a language L in which to write realizers. We stay abstract here: all the results presented in this paper hold for every L satisfying some basic conditions, which we will now explain.

Let $L \subseteq \Sigma^*$ be the set of finite sequences over some finite alphabet Σ. We assume a pairing function $\langle \cdot, \cdot \rangle : L \times L \to L$ and a length function $| \cdot | : L \to \mathbb{N}$ such that $|\langle x, y \rangle| = |x| + |y| + cp$ and $|x| \leq \mathit{length}(x)$ yet $|x| = \Omega(\mathit{length}(x)^\varepsilon)$ for some $\varepsilon > 0$, where $\mathit{length}(x)$ is the number of symbols in x and $cp > 0$ is a fixed constant. We assume a reasonable encoding of algorithms as elements of L. We write $\{e\}(x)$ for the (possibly undefined) application of algorithm $e \in L$ to input $x \in L$. We furthermore assume an abstract time measure $\mathit{Time}(\{e\}(x)) \in \mathbb{N}$ such that $\mathit{Time}(\{e\}(x))$ is defined whenever $\{e\}(x)$ is and $\{e\}(x)$ can be evaluated on a Turing machine in time bounded by $p(\mathit{Time}(\{e\}(x)), |e|, |x|)$, where $p : \mathbb{N}^3 \to \mathbb{N}$ is a fixed polynomial. We require that algorithms manipulating higher-order functions and 0-1 strings can be represented in L and their abstract time measures satisfy intuitive bounds. For example, we assume the existence of e_{comp} (composition) and e_{contr} (duplication, copying) such that for every x, y it holds that $\{e_{comp}\}(\langle x, y \rangle) = z$ where $|z| = |x| + |y| + O(1)$ and $\{z\}(w) = \{y\}(\{x\}(w))$ and $\{e_{contr}\}(x) = \langle x, x \rangle$. Moreover, $\mathit{Time}(\{e_{contr}\}(x)) = O(|x|)$ and $\mathit{Time}(\{e_{comp}\}(\langle x, y \rangle)) = O(1)$ and $\mathit{Time}(\{z\}(w)) = \mathit{Time}(\{x\}(w)) + \mathit{Time}(\{y\}(\{x\}(w))) + O(1)$.

This abstract framework can be instantiated with call-by-value lambda terms [7] or Turing machines [10]. Since the instantiation is irrelevant for our purposes we do not give any details.

3 Resource Monoids and Length Spaces

In this section, we recall the notion of a resource monoid [7] and the corresponding category of realizability sets, called *length spaces*, as well as its general properties.

A *resource monoid* is a quadruple $M = (|M|, +, \leq_M, \mathcal{D}_M)$ where
(i) $(|M|, +)$ is a commutative monoid;
(ii) $\leq_M$ is a pre-order on $|M|$ which is compatible with $+$;
(iii) $\mathcal{D}_M : \{(\alpha, \beta) \mid \alpha \leq_M \beta\} \to \mathbb{N}$ is a function such that for every α, β, γ

$$\begin{aligned} \mathcal{D}_M(\alpha, \beta) + \mathcal{D}_M(\beta, \gamma) &\leq \mathcal{D}_M(\alpha, \gamma) \\ \mathcal{D}_M(\alpha, \beta) &\leq \mathcal{D}_M(\alpha + \gamma, \beta + \gamma) \end{aligned}$$

and, moreover, for every $n \in \mathbb{N}$ there is α such that $\mathcal{D}_M(0, \alpha) \geq n$.

Given a resource monoid $M = (|M|, +, \leq_M, \mathcal{D}_M)$, the function $\mathcal{F}_M : |M| \to \mathbb{N}$ is defined by putting $\mathcal{F}_M(\alpha) = \mathcal{D}_M(0, \alpha)$.

We shall use elements of a resource monoid to bound data, algorithms, and runtimes in the following way: an element φ bounds an algorithm e if $\mathcal{F}_M(\varphi) \geq |e|$ and, more importantly, whenever α bounds an input x to e then there must be a bound $\beta \leq_M \varphi + \alpha$ for the result $y = \{e\}(x)$ and, most importantly, the runtime of that computation must be bounded by $\mathcal{D}_M(\beta, \varphi + \alpha)$. So, in a sense, we have the option of either producing a large output fast or to take a long time for a small output. The "inverse triangular" law above ensures that the composition of two algorithms bounded by φ_1 and φ_2, respectively, can be bounded by $\varphi_1 + \varphi_2$ or a simple modification thereof. In particular, the contribution of the unknown intermediate result in a composition cancels out using that law. Another useful intuition is that $\mathcal{D}_M(\alpha, \beta)$ behaves like the difference $\beta - \alpha$, indeed, $(\beta - \alpha) + (\gamma - \beta) \leq \gamma - \alpha$.

A *length space* on a resource monoid $M = (|M|, +, \leq_M, \mathcal{D}_M)$ is a pair $A = (|A|, \Vdash_A)$, where $|A|$ is a set and $\Vdash_A \subseteq |M| \times L \times |A|$ is a(n infix) relation satisfying the following conditions:

(i) if $\alpha, e \Vdash_A a$, then $\mathcal{F}_M(\alpha) \geq |e|$;
(ii) for every $a \in |A|$, there are α, e such that $\alpha, e \Vdash_A a$;
(iii) if $\alpha, e \Vdash_A a$ and $\alpha \leq_M \beta$, then $\beta, e \Vdash_A a$;
(iv) if $\alpha, e \Vdash_A a$ and $\alpha, e \Vdash_A b$, then $a = b$.

The last requirement implies that each element of $|A|$ is uniquely determined by the (nonempty) set of its realizers and in particular limits the cardinality of any length space to the number of partial equivalence relations on L.

A *morphism* from length space $A = (|A|, \Vdash_A)$ to length space $B = (|B|, \Vdash_B)$ (on the same resource monoid $M = (|M|, +, \leq_M, \mathcal{D}_M)$) is a function $f : |A| \to |B|$ such that there exist $e \in L = \Sigma^*$, $\varphi \in |M|$ with $\mathcal{F}_M(\varphi) \geq |e|$ and whenever $\alpha, d \Vdash_A a$, there must be β, c such that

(i) $\beta, c \Vdash_B f(a)$;
(ii) $\beta \leq_M \varphi + \alpha$;
(iii) $\{e\}(d) = c$;
(iv) $\mathit{Time}(\{e\}(d)) \leq \mathcal{D}_M(\beta, \varphi + \alpha)$.

We call e a *realizer* of f and φ a *majorizer* of f. The set of all morphisms from A to B is denoted as $\mathit{Hom}(A, B)$. If f is a morphism from A to B realized by e and majorized by φ, then we will write $f : A \xrightarrow{e,\varphi} B$ or $\varphi, e \Vdash_{A \multimap B} f$.

Given two length spaces $A = (|A|, \Vdash_A)$ and $B = (|B|, \Vdash_B)$ on the same resource monoid M, we define $A \otimes B = (|A| \times |B|, \Vdash_{A \otimes B})$ (on M) where $e, \alpha \Vdash_{A \otimes B} (a, b)$ iff $\mathcal{F}_M(\alpha) \geq |e|$ and there are f, g, β, γ with

$$\begin{gathered} f, \beta \Vdash_A a \\ g, \gamma \Vdash_B b \\ e = \langle f, g \rangle \\ \alpha \geq_M \beta + \gamma \end{gathered}$$

The following result is from [7]:

Theorem 1. *The category of length spaces for any resource monoid is symmetric monoidal closed with respect to the tensor product given above. In particular, there is a neutral object I and for any two length spaces an exponential $A \multimap B$.*

3.1 Interpreting Multiplicative Affine Logic

We can now formally show that second order multiplicative affine logic (i.e., multiplicative linear logic plus full weakening) can be interpreted inside the category of length spaces on any monoid M. Doing this will simplify the analysis of LAL, since the latter can be obtained by enriching multiplicative affine logic with two modalities. Formulae of (intuitionistic, second order) multiplicative affine logic are generated by the following productions:

$$A ::= \alpha \mid A \multimap A \mid A \otimes A \mid \forall \alpha.A$$

where α ranges over a countable set of atoms. Rules are reported in Figure 1. A *realizability environment* is a partial function assigning length spaces (on the

Identity, Cut and Weakening.

$$\frac{}{A \vdash A}\ I \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ U \qquad \frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ W$$

Multiplicative Logical Rules.

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ L_{\otimes} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ R_{\otimes}$$

$$\frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\ L_{\multimap} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ R_{\multimap}$$

Second Order Logical Rules.

$$\frac{\vdash \Gamma, A[C/\alpha] \vdash B}{\Gamma, \forall\alpha.A \vdash B}\ L^{\forall} \qquad \frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \forall\alpha.A}\ R^{\forall}$$

Fig. 1. Intuitionistic Multiplicative Affine Logic

same resource monoid) to atoms. Realizability semantics $[\![A]\!]^{\mathscr{R}}_{\eta}$ of a formula A on the realizability environment η is defined by induction on A:

$$\begin{aligned}
[\![\alpha]\!]^{\mathscr{R}}_{\eta} &= \eta(\alpha) \\
[\![A \otimes B]\!]^{\mathscr{R}}_{\eta} &= [\![A]\!]^{\mathscr{R}}_{\eta} \otimes [\![B]\!]^{\mathscr{R}}_{\eta} \\
[\![A \multimap B]\!]^{\mathscr{R}}_{\eta} &= [\![A]\!]^{\mathscr{R}}_{\eta} \multimap [\![B]\!]^{\mathscr{R}}_{\eta} \\
[\![\forall\alpha.A]\!]^{\mathscr{R}}_{\eta} &= (|[\![\forall\alpha.A]\!]^{\mathscr{R}}_{\eta}|, \Vdash_{[\![\forall\alpha.A]\!]^{\mathscr{R}}_{\eta}})
\end{aligned}$$

where

$$|[\![\forall\alpha.A]\!]^{\mathscr{R}}_{\eta}| = \prod_{C\in\mathscr{U}} |[\![A]\!]^{\mathscr{R}}_{\eta[\alpha\to C]}|$$
$$\alpha, e \Vdash_{[\![\forall\alpha.A]\!]^{\mathscr{R}}_{\eta}} a \Longleftrightarrow \forall C.\alpha, e \Vdash_{[\![A]\!]^{\mathscr{R}}_{\eta[\alpha\to C]}} a$$

Here $\mathscr{U}$ stands for the class of all length spaces. Some care is needed when defining the product since strictly speaking it does not exist for size reasons. The standard way out is to let the product range over those length spaces whose underlying set equals the set of equivalence classes of a partial equivalence relation on L. As already mentioned, every length space is isomorphic to one of that form. When working with the product one has to insert these isomorphisms in appropriate places which, however, we elide to increase readability.

If $n \geq 0$ and $A_1, \ldots, A_n$ are formulas, the expression $[\![A_1 \otimes \ldots \otimes A_n]\!]^{\mathscr{R}}_{\eta}$ stands for I if $n = 0$ and $[\![A_1 \otimes \ldots \otimes A_{n-1}]\!]^{\mathscr{R}}_{\eta} \otimes [\![A_n]\!]^{\mathscr{R}}_{\eta}$ if $n \geq 1$.

4 Light Length Spaces

Light Affine Logic extends Multiplicative Affine Logic by two modalities ! and § which are governed by the rules in Figure 2. In LAL, we can use variations on

Exponential Rules and Contraction.

$$\frac{\Gamma, \Delta \vdash A}{\S\Gamma, !\Delta \vdash \S A}\ P_{\S} \qquad \frac{A \vdash B}{!A \vdash !B}\ P^1_! \qquad \frac{\vdash A}{\vdash !A}\ P^2_! \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ C$$

Fig. 2. Intuitionistic Light Affine Logic

the usual impredicative encodings of natural numbers, lists, etc. For example, binary lists can be encoded as cut-free proofs for

$$\mathit{List}_{\mathsf{LAL}} \equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$$

while natural numbers correspond to proofs for

$$\mathit{Int}_{\mathsf{LAL}} \equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha).$$

Now, let π be an LAL proof with conclusion $\mathit{List}_{\mathsf{LAL}} \vdash \mathit{List}_{\mathsf{LAL}}$. If we cut π against proofs corresponding to binary lists and we normalize the obtained proof, we get a proof corresponding to a binary list. So any proof like π *represents* a function from binary lists to binary lists. The above definition can be easily generalized to the cases when π has conclusion in the form $\{!, \S\}^j \mathit{List}_{\mathsf{LAL}} \vdash \{!, \S\}^k \mathit{List}_{\mathsf{LAL}}$.

But what is the expressive power of Light Affine Logic? The class of representable functions include all the polytime functions (see [16]):

Theorem 2 (Polytime Completeness). *Every polytime function on binary lists is represented by a* LAL *proof* $\pi : \mathit{List}_{\mathsf{LAL}} \multimap \S^n \mathit{List}_{\mathsf{LAL}}$.

We will now describe a resource monoid with the property that the ensuing category of length spaces provides structure for the interpretation of these modalities while allowing us to extract polytime bounds for functions of basic type. For ease of notation we denote $\max(m,n)$ by $m \mid n$:

Definition 1. *The algebraic structure* $\mathcal{L}$ *is the quadruple* $(|\mathcal{L}|, +, \leq_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}})$ *such that:*

- *Elements of* $|\mathcal{L}| \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N}^{\mathbb{N}}$ *are triples* (n,m,f) *such that* $f : \mathbb{N} \to \mathbb{N}$ *is a monotonically increasing polytime function.*
- *For every* $(n,m,f),(l,k,g) \in |\mathcal{L}|$, $(n,m,f) + (l,k,g) = (n+l, m \mid k, f \mid g)$.
- *For every* $(n,m,f),(l,k,g) \in |\mathcal{L}|$, $(n,m,f) \leq_{\mathcal{L}} (l,k,g)$ *iff* $n \leq l$, $n+m \leq l+k$ *and* $f \leq g$.
- *For every* $(n,m,f),(l,k,g) \in |\mathcal{L}|$ *such that* $(n,m,f) \leq_{\mathcal{L}} (l,k,g)$,

$$\mathcal{D}_{\mathcal{L}}((n,m,f),(l,k,g)) = (l-n)g(l+k).$$

The triple $(0,0,0) \in |\mathcal{L}|$*, denoted* $0_{\mathcal{L}}$*, is an identity for* $+$.

The binary relation $\leq_{\mathcal{L}}$ is trivially reflexive. Moreover, it is transitive:

Lemma 1 (Transitivity). *If* α,β,γ *are in* $|\mathcal{L}|$, $\alpha \leq_{\mathcal{L}} \beta$ *and* $\beta \leq_{\mathcal{L}} \gamma$*, then* $\alpha \leq_{\mathcal{L}} \gamma$.

Proof. Let $(n,m,f),(l,k,g),(p,q,h) \in |\mathcal{L}|$. Moreover, let $(n,m,f) \leq_{\mathcal{L}} (l,k,g)$ and $(l,k,g) \leq_{\mathcal{L}} (p,q,h)$. Trivially:

$$\begin{aligned} n \leq l &\leq p; \\ n+m \leq l+k &\leq p+q; \\ f \leq g &\leq h. \end{aligned}$$

In other words $(n,m,f) \leq_{\mathcal{L}} (p,q,h)$. □

But $\leq_{\mathcal{L}}$ is even compatible with $+$:

Lemma 2 (Compatibility). $0_{\mathcal{L}} \leq_{\mathcal{L}} \alpha$ *for every* $\alpha \in |\mathcal{L}|$. *Moreover, if* α,β,γ *are in* $|\mathcal{L}|$ *and* $\alpha \leq_{\mathcal{L}} \beta$*, then* $\alpha + \gamma \leq_{\mathcal{L}} \beta + \gamma$.

The following is now immediate.

Lemma 3. $\mathcal{L}$ *is a resource monoid.*

Definition 2. *A* light length space *is a length space over the resource monoid* $\mathcal{L}$. *Given a light length space* $A = (|A|, \Vdash_A)$*, the light spaces* $!A = (|A|, \Vdash_{!A})$ *and* $\S A = (|A|, \Vdash_{\S A})$ *both with underlying set* $|A|$*, are defined by:*

$$\begin{aligned} (n,m,f), e \Vdash_{!A} a &\iff \exists (l,k,g) \in |\mathcal{L}|.(l,k,g), e \Vdash_A a \wedge (1, l+k, g^+) \leq_{\mathcal{L}} (n,m,f) \\ (n,m,f), e \Vdash_{\S A} a &\iff \exists (l,k,g) \in |\mathcal{L}|.(lk,k,g), e \Vdash_A a \wedge (l,k,g^+) \leq_{\mathcal{L}} (n,m,f) \end{aligned}$$

where $g^+(x) = x^2 g(x^2)$.

The constructions ! and § on light length spaces serve to capture the exponential modalities of light affine logic. Their definition is the crucial contribution of this paper. The relations $\Vdash_{!A}, \Vdash_{\S A}$ can equivalently be defined inductively by the following rules:

$$\frac{\alpha, e \Vdash_{!A} a \quad \alpha \leq \beta}{\beta, e \Vdash_{!A} a} \quad \frac{\alpha, e \Vdash_{\S A} a \quad \alpha \leq \beta}{\beta, e \Vdash_{\S A} a} \quad \frac{(l,k,g), e \Vdash_A a}{(1,(l+k),g^+), e \Vdash_{!A} a} \quad \frac{(lk,k,g), e \Vdash_A a}{(l,k,g^+), e \Vdash_{\S A} a}$$

Before we embark on the verification that these settings admit an interpretation of all the constructions of LAL let us illustrate the definitions using the particular length space $\mathbf{N} = (\mathbb{N}, \Vdash_{\mathbf{N}})$ where $(l,k,g), e \Vdash_{\mathbf{N}} n$ if e encodes n and $l \geq n$ and $k \geq 0$ and $g(x) \geq c$ for c a constant large enough so that $lc \geq |e|$. Note that the constant c may be chosen independent of n. This length space is isomorphic to the denotation in the model of the LAL-type $\forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$ which is the LAL-version of Church numerals.

Then $(l,k,g), e \Vdash_{\mathbf{N}\otimes\mathbf{N}} (n_1, n_2)$ if $l \geq n_1 + n_2$ and $e = \langle e_1, e_2 \rangle$ where e_i encodes n_i and $g(x) \geq c$ and $lg(l+k) \geq |e|$. Note that the latter can be achieved by choosing $g(x) = c + d$ for some fixed constant d.

We see that the diagonal map $n \mapsto (n,n)$ cannot be realized for then we would need a fixed l_0 such that $l_0 + l \geq l + l$ for all l. Similarly, we see that all realisable maps f from $\mathbf{N}$ to $\mathbf{N}$ must satisfy $f(x) \leq x + O(1)$. The runtime of such a function is governed by the third ("g") component of its realiser and is hence an arbitrary polynomial.

On the other hand, the length space $!\mathbf{N}$ has $(l,k,g), e \Vdash_{!\mathbf{N}} n$ if $l \geq 1$ and $k \geq n$ and $g(x) \geq cx^2$. Now note that $lg(k) \geq c(1+k)^2 \geq cn \geq |e|$. On the other hand, since the l-slot (first component) may be chosen 1 we find that the diagonal map $!\mathbf{N} \to !\mathbf{N}\otimes!\mathbf{N}$ is realisable. The identity function from $!\mathbf{N}$ to $\mathbf{N}$ is not realisable because the first component l_0 of its realiser would have to satisfy $l_0 + 1 \geq n$ for all n.

Now consider the length space $\S\mathbf{N}$. We have $(l,k,g), e \Vdash_{\S\mathbf{N}} n$ if $lk \geq n$ and $g(x) \geq cx^2$. We are now able to realise the identity from $!\mathbf{N}$ to $\S\mathbf{N}$ noticing that, in particular $(1, n, \lambda x.cx^2), e \Vdash_{\S\mathbf{N}} n$ when e encodes n. We also note that, for instance the doubling function can be realised as a map from $\mathbf{N}$ to $\S\mathbf{N}$. To do this, we need a realiser with first component $l_0 = 1$. Given input n realised by $(n, 1, g)$ we then realise the result $2n$ by $(n, 2, \lambda x.cx^2)$. We can even realise the function $1/4n^2 + O(n)$ but not n^2. For this, two $\S$-s are needed.

Proving light length spaces to be a model for LAL amounts to prove that certain constructions involving the modalities ! and $\S$ can be justified in the model. First of all, the diagonal map is a morphism, as can be easily proved:

Lemma 4. *Given light length spaces A, B, there is the morphism: contr $:!A \to !A\otimes!A$ where contr$(a) = (a,a)$.*

Proof. The majoriser for the obvious realiser e_{contr} is given by a suitably padded version of $(2, 0, x \mapsto 0)$. The central part of the verification is the observation that

$$\begin{aligned} 2.(1, l+k, g^+) &= (2, 2(l+k), g^+) \leq_{\mathcal{L}} (2, 0, x \mapsto 0) + (1, l+k, g^+) \\ &= (3, l+k, g^+) \end{aligned} \qquad \square$$

On the other hand, ! is a functor.

Lemma 5 (Functoriality of !). *If $f : A \xrightarrow{e,\alpha} B$, then there is β such that $f : !A \xrightarrow{e,\beta} !B$.*

Proof (of Lemma 5). Let α be (n,m,f) and suppose $d,(l,k,g) \Vdash_{!A} a$. Then $(l,k,g) \geq_{\mathcal{L}} (1, p+q, h^+)$, where $d,(p,q,h) \Vdash_A a$. Observe that there must be $(i,j,r), c$ such that $c,(i,j,r) \Vdash_B f(a)$, $(i,j,r) \leq_{\mathcal{L}} (n,m,f) + (p,q,h)$ and $\mathit{Time}(\{e\}(d)) \leq \mathcal{D}_{\mathcal{L}}((i,j,r),(n,m,f)+(p,q,h))$. As a consequence, $c,(1,i+j,r^+) \Vdash_{!B} f(a)$. But

$$(1, i+j, r^+) \leq_{\mathcal{L}} (n+m+1, m, f^+) + (1, p+q, h^+)$$

because:

- The inequality $1+i+j \leq n+m+2+m \mid (p+q)$ holds, because if $m \leq q$, then $1+i+j \leq 1+n+p+m \mid q = 1+n+p+q \leq 2+n+m+m \mid (p+q)$. and if $m > q$, then $1+i+j \leq 1+n+p+m \mid q = 1+n+p+m \leq 2+n+m+p+q \leq 2+n+m+m \mid (p+q)$.
- For every $x \in \mathbb{N}$,

$$\begin{aligned} r^+(x) &= x^2 r(x^2) \leq x^2 (f \mid h)(x^2) \\ &\leq x^2 (f(x^2) \mid h(x^2)) = (x^2 f(x^2) \mid x^2 h(x^2)) \\ &= (f^+(x) \mid h^+(x)) = (f \mid h)^+(x). \end{aligned}$$

Moreover:

$$\begin{aligned} \mathit{Time}(\{e\}(d)) &\leq \mathcal{D}_{\mathcal{L}}((i,j,r),(n,m,f)+(p,q,h)) \\ &\leq (n+p)(f \mid h)(n+p+m \mid q) \\ &\leq (n+m+1)\cdot(n+m+2+m \mid (p+q))^2 \cdot (f \mid h)((n+m+2+m \mid (p+q))^2) \\ &= (n+m+1)\cdot(f \mid h)^+(n+m+2+m \mid (p+q)) \\ &= (n+m+1)\cdot(f^+ \mid h^+)(n+m+2+m \mid (p+q)) \\ &= \mathcal{D}_{\mathcal{L}}((1,i+j,r^+),(n+m+2, m \mid (p+q), f^+ \mid h^+)) \\ &\leq \mathcal{D}_{\mathcal{L}}((1,i+j,r^+),(n+m+1,m,f^+)+(1,p+q,h^+)) \end{aligned}$$

This means that $f : !A \xrightarrow{e,(n+m+1,m,f^+)} !B$. □

Notice that the distributive law $!A \otimes !B \multimap !(A \otimes B)$ *cannot* be proved in the syntax and is not validated in the model either. Indeed, its introduction would collapse LAL to elementary affine logic, which is elementary time complete. The modality § is a functor itself:

Lemma 6 (Functoriality of §). *If $f : A \xrightarrow{e,\alpha} B$, then there is β such that $f : \S A \xrightarrow{e,\beta} \S B$.*

Proof (of Lemma 6). Let α be (n,m,f) and suppose $d,(l,k,g) \Vdash_{\S A} a$. Then $(l,k,g) \geq_{\mathcal{L}} (p,q,h^+)$, where $d,(pq,q,h) \Vdash_A a$. Observe that there must be $(i,j,r), c$ such that $c,(i,j,r) \Vdash_B f(a)$, $(i,j,r) \leq_{\mathcal{L}} (n,m,f) + (pq,q,h)$ and $\mathit{Time}(\{e\}(d)) \leq \mathcal{D}_{\mathcal{L}}((i,j,r),(n,m,f)+(pq,q,h))$. As a consequence, we obtain $c,(n,m,f)+(pq,q,h) \Vdash_B f(a)$. But notice that

$$\begin{aligned} (n,m,f)+(pq,q,h) &= (n+pq, m \mid q, f \mid h) \\ &\leq_{\mathcal{L}} (((m+1) \mid q)(n+1+p), (m+1) \mid q, f \mid h). \end{aligned}$$

which implies $c, (n+1+p, (m+1) \mid q, (f \mid h)^+) \Vdash_{\S B} f(a)$. Now:

$$\begin{aligned}(n+1+p, (m+1) \mid q, (f \mid h)^+) &= (n+1, m+1, f^+) + (p, q, h^+)\\ &\leq_{\mathcal{L}} (n+2, m+1, f^+) + (l, k, g).\end{aligned}$$

Moreover:

$$\begin{aligned}Time(\{e\}(d)) &\leq \mathcal{D}_{\mathcal{L}}((i,j,r), (n,m,f) + (pq, q, h))\\ &\leq (n+pq)(f \mid h)(n+pq+m \mid q)\\ &\leq (n+p+2+q(m+1))^2 (f \mid h)((n+p+2+q \mid (m+1))^2)\\ &= (f \mid h)^+(n+p+2+q \mid (m+1))\\ &= (f^+ \mid h^+)(n+p+2+q \mid (m+1))\\ &= (p+n+2-(p+n+1))(f^+ \mid h^+)(n+p+2+q \mid (m+1))\\ &= \mathcal{D}_{\mathcal{L}}((n+1+p, (m+1), (f \mid h)^+), (n+2, m+1, f^+) + (p, q, h^+))\\ &= \mathcal{D}_{\mathcal{L}}((n+1+p, (m+1), (f \mid h)^+), (n+2, m+1, f^+) + (l, k, g)).\end{aligned}$$

This means that $f : \S A \overset{e,(n+2,m+1,f^+)}{\longrightarrow} \S B$. □

The following lemma whose proof we elide establishes the remaining properties required to model LAL: distributivity of § over $\otimes$ and the dereliction axiom relating the two modalities.

Lemma 7. *Given light length spaces A, B, there are morphisms: derelict $: !A \to \S A$ and distr $: \S A \otimes \S B \to \S(A \otimes B)$ where, for every $a \in |A|$ and $b \in |B|$, derelict$(a) = a$ and distr$(a,b) = (a,b)$.*

As anticipated in the introduction, a principle which cannot be proved syntactically, *can* be justified in the semantics:

Lemma 8. *Given light length spaces A_1, A_2, there is a morphism codistr : $\S(A_1 \otimes A_2) \to \S A_1 \otimes \S A_2$ where codistr$(a_1, a_2) = (a_1, a_2)$.*

Proof. Let $e_{codistr} = e_{id}$. We know $\{e_{id}\}(d)$ takes constant time.

Suppose that $\langle d_1, d_2\rangle, \gamma \Vdash_{\S(A_1 \otimes A_2)} (a_1, a_2)$. We have $\gamma \geq_{\mathcal{L}} (l, k, f^+)$ and $(lk, k, f) \geq_{\mathcal{L}} (l_1, k_1, f_1) + (l_2, k_2, f_2)$ where $d_i, (l_i, k_i, f_i) \Vdash_{A_i} a_i$ for $i = 1, 2$. By upward closure we also have $d_i, (l_i, k, f_i) \Vdash_{A_i} a_i$ and then $d_i, (\lceil l_i/k \rceil, k, f_i^+) \Vdash_{\S A_i} a_i$ and finally $\langle d_1, d_2\rangle, (\lceil l_1/k \rceil + \lceil l_2/k \rceil, k, f^+) \Vdash_{\S A_i \otimes \S A_2} (a_1, a_2)$. But now

$$(\lceil l_1/k \rceil + \lceil l_2/k \rceil, k, f^+) \leq_{\mathcal{L}} (2, 0, 0) + (l, k, f^+)$$

so that a realiser for $e_{codistr}$ may be given by padding $(2, 0, 0)$ so as to cover the (constant) runtime of this algorithm.

This shows that light length spaces are not fully complete as a model of LAL. On the other hand, results like Lemma 8 are potentially very interesting, since soundness holds for any extension of LAL which can be interpreted in the model.

4.1 Interpreting Light Affine Logic

Interpretations of the modalities § and ! are the obvious ones: $[\![!A]\!]^{\mathscr{R}}_\eta = ![\![A]\!]^{\mathscr{R}}_\eta$ and $[\![\S A]\!]^{\mathscr{R}}_\eta = \S[\![A]\!]^{\mathscr{R}}_\eta$. Since all the axioms needed are justifiable in our semantics, we get:

Theorem 3. *Light length spaces form a model of* LAL.

As a consequence, we can prove that the set of functions which can be represented in LAL is a subset of the class of polytime functions:

Corollary 1 (Soundness). *Let π be an* LAL *proof with conclusion of the form $\vdash \{!, \S\}^j List_{\mathsf{LAL}} \multimap \{!, \S\}^k List_{\mathsf{LAL}}$ and let $f : B \to B$ be the function induced by $[\![\pi]\!]^{\mathscr{R}}$. Then f is computable in polynomial time.*

Proof (Sketch). Intuitively, this is clear since runtimes of realizers are always bounded by the third components of majorizers. A slight complication arises from the fact that the third component of the argument to a map also influences the runtime; indeed, without this feature maps like application from $(A \multimap B) \otimes A$ to B could not be interpreted in the model. However, we can define a light length space of binary lists $\mathbf{B}$ whose realisers have constant third component analogous to the light length space $\mathbf{N}$ in the motivation after Definition 2. We then show using definable iterators that this length space is isomorphic to the denotation of the type B above. It is, however, obvious that all functions on $\mathbf{B}$ are polytime. For details see [11] where such an argument has been carried out in detail for Bounded Linear Logic. □

5 Conclusions

We have introduced a new model for LAL based on realizability. This allows us to give a simplified proof of soundness for the same logic. As any kind of semantics, our model can be used to identify certain axioms as not derivable in LAL (if it's not in the model it can't be in the syntax). Examples of such principles are the identification of the two modalities or commutation of the-modality with tensor. More interestingly, there are formulas which are syntactically not provable but are justified in the semantics. The fact that our semantics has polynomial time computability built-in means that such formulas can be added to LAL without compromising soundness for polynomial time. One example of such a formula has been given in Lemma 8 (distributivity of § over tensor.). We are confident that more examples can be found; of particular interest would be principles that enable more algorithms to be expressed in their natural form.

References

1. Amadio, R.M.: Max-plus quasi-interpretations. In: Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications, pp. 31–45 (2003)
2. Asperti, A., Roversi, L.: Intuitionistic light affine logic. ACM Transactions on Computational Logic 3(1), 137–175 (2002)

3. Bellantoni, S., Niggl, K.H., Schwichtenberg, H.: Higher type recursion, ramification and polynomial time. Annals of Pure and Applied Logic 104, 17–30 (2000)
4. Cook, S., Urquhart, A.: Functional interpretations of feasible constructive arithmetic. Annals of Pure and Applied Logic 63(2), 103–200 (1993)
5. Coppola, P., Martini, S.: Typing lambda terms in elementary logic with linear constraints. In: Proceedings of the 6th International Conference on Typed Lambda-Calculus and Applications, pp. 76–90 (2001)
6. Crossley, J., Mathai, G., Seely, R.: A logical calculus for polynomial-time realizability. Journal of Methods of Logic in Computer Science 3, 279–298 (1994)
7. Lago, U.D., Hofmann, M.: Quantitative models and implicit complexity. In: Proc. Foundations of Software Technology and Theoretical Computer Science, pp. 189–200 (2005)
8. Girard, J.-Y.: Light linear logic. Information and Computation 143(2), 175–204 (1998)
9. Hofmann, M.: Linear types and non-size-increasing polynomial time computation. In: Proceedings of the 14th IEEE Syposium on Logic in Computer Science, pp. 464–473 (1999)
10. Hofmann, M.: Safe recursion with higher types and BCK-algebra. Annals of Pure and Applied Logic 104, 113–166 (2000)
11. Hofmann, M., Scott, P.: Realizability models for BLL-like languages. Theoretical Computer Science 318(1-2), 121–137 (2004)
12. Kreisel, G.: Interpretation of analysis by means of constructive functions of finite types. In: Heyting, A. (ed.) Constructiviey in Mathematics, pp. 101–128. North-Holland, Amsterdam (1959)
13. Lafont, Y.: Soft linear logic and polynomial time. Theoretical Computer Science 318, 163–180 (2004)
14. Lago, U.D., Martini, S.: Phase semantics and decidability of elementary affine logic. Theor. Comput. Sci. 318(3), 409–433 (2004)
15. Murawski, A.S., Luke Ong, C.-H.: Discreet games, light affine logic and ptime computation. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 427–441. Springer, Heidelberg (2000)
16. Roversi, L.: A p-time completeness proof for light logics. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 469–483. Springer, Heidelberg (1999)

On Subword Complexity of Morphic Sequences

Rostislav Deviatov

Moscow State University and Independent University of Moscow
deviatov1@rambler.ru

Abstract. We sketch the proof of the following result: the subword complexity of arbitrary morphic sequence is either $\Theta(n^2)$, or $O(n^{3/2})$.

1 Introduction

Morphisms and morphic sequences are well known and well studied in combinatorics on words (e. g., see [1]). We study their subword complexity.

Let Σ be a finite alphabet. A mapping $\varphi\colon \Sigma^* \to \Sigma^*$ is called a *morphism* if $\varphi(uv) = \varphi(u)\varphi(v)$ for all $u, v \in \Sigma^*$. A morphism is determined by its values on single-letter words. A morphism is *non-erasing* if $|\varphi(a)| \geq 1$ for each $a \in \Sigma$, and is called *coding* if $|\varphi(a)| = 1$ for each $a \in \Sigma$. Let $|\varphi|$ denote $\max_{a\in\Sigma} |\varphi(a)|$.

Let $\varphi(s) = su$ for some $s \in \Sigma$, $u \in \Sigma^*$, and suppose $\forall n\ \varphi^n(u)$ is not empty. Then an infinite sequence $\varphi^\infty(s) = \lim_{n\to\infty} \varphi^n(s)$ is well-defined and is called *pure morphic*. Sequences of the form $\psi(\varphi^\infty(s))$ with coding ψ are called *morphic*.

In this paper we study a natural combinatorial characteristics of sequences, namely subword complexity. *Subword complexity* of a sequence β is a function $p_\beta\colon \mathbb{N} \to \mathbb{N}$ where $p_\beta(n)$ is the number of all different n-length subwords occurring in β. For a survey on subword complexity, see, e. g., [2]. Pansiot showed [4] that the subword complexity of an arbitrary pure morphic sequence adopts one of the five following asymptotic behaviors: $O(1)$, $\Theta(n)$, $\Theta(n \log\log n)$, $\Theta(n \log n)$, or $\Theta(n^2)$. Since codings can only decrease subword complexity, the subword complexity of every morphic sequence is $O(n^2)$. We formulate the following main result.

Theorem 1. *The subword complexity p_β of a morphic sequence β is either $p_\beta(n) = \Theta(n^{1+\frac{1}{k}})$ for some $k \in \mathbb{N}$, or $p_\beta(n) = O(n \log n)$.*

Note that for each k the complexity class $\Theta(n^{1+\frac{1}{k}})$ is non-empty [3].

However in this extended abstract we show the technics used in the proof of this main result considering the following weaker case of Theorem 1.

Theorem 2. *The subword complexity p_β of a morphic sequence β is either $p_\beta(n) = \Theta(n^2)$, or $p_\beta(n) = O(n^{3/2})$.*

We give an example of a morphic sequence β with $p_\beta = \Theta(n^{3/2})$ in Section 6.

Let Σ be a finite alphabet, $\varphi\colon \Sigma^* \to \Sigma^*$ be a morphism, $\psi\colon \Sigma^* \to \Sigma^*$ be a coding, α be a pure morphic sequence generated by φ, and $\beta = \psi(\alpha)$ be a morphic sequence. By Theorem 7.7.1 from [1] every morphic sequence can be generated by a non-erasing morphism, so further we assume that φ is non-erasing.

To prove Theorem 2, we prove the following two propositions:

Proposition 1. *If there are evolutions of 2-blocks arising in α that are not continuously periodic, the subword complexity of β is $\Omega(n^2)$.*

Proposition 2. *If all the evolutions of 2-blocks arising in α are continuously periodic, the subword complexity of β is $O(n^{3/2})$.*

Most part of the paper is devoted to formulation of what k-blocks, evolutions and continuously periodic evolutions are. Similarly, one can generalize the notion of a continuously periodic evolution of k-blocks to each $k \in \mathbb{N}$, generalize Propositions 1 and 2 to an arbitrary k, and thus prove Theorem 1. (Actually, the notion of continuously periodic evolution of k-blocks needs more technical details, but Propositions 1 and 2 can be reformulated easily: 2-blocks, n^2 and $n^{3/2}$ should be replaced by k-blocks, $n^{1+1/(k-1)}$ and $n^{1+1/k}$, respectively). However, the full detailed proof of Theorem 1 (and Theorem 2 as well) needs much more space and will be published elsewhere.

We will speak about occurrences in α. Strictly speaking, we call a pair of a word γ and a location i in α *an occurrence* if the subword of α that starts from position i in α and is of length $|\gamma|$ is γ. This occurrence is denoted by $\alpha_{i\ldots j}$ if j is the index of the last letter that belongs to the occurrence. In particular, $\alpha_{i\ldots i}$ denotes a single-letter occurrence, and $\alpha_{i\ldots i-1}$ denotes an occurrence of the empty word between the $(i-1)$-th and the i-th letters. Since $\alpha = \alpha_1\alpha_2\alpha_3\ldots = \varphi(\alpha) = \varphi(\alpha_1)\varphi(\alpha_2)\varphi(\alpha_3)\ldots$, φ might be considered either as a morphism on words (which we call abstract words sometimes), or as a mapping on the set of occurrences in α. Usually we speak of the latter, unless stated otherwise.

We call a finite word γ *p-periodic* with *left* (resp. *right*, *complete*) period δ if $|\delta| = p$ and $\gamma = \delta\delta\ldots\delta\delta_{1\ldots k}$ (resp. $\gamma = \delta_{p-k+1\ldots p}\delta\ldots\delta$, $\gamma = \delta\ldots\delta$). If δ is known, we will shortly call γ a *left* (resp. *right*, *completely*) *δ-periodic* word. δ will be always considered as an abstract word. The subword $\gamma_{|\gamma|-k+1\ldots|\gamma|}$ is called the *incomplete occurrence*, where $0 \le k < |\delta|$. All the same is with sequences of symbols or numbers.

The function $r_a : \mathbb{N} \to \mathbb{N}$, $r_a(n) = |\varphi^n(a)|$ is called *the growth rate* of a. Let us define *orders of letters* with respect to φ. We say that $a \in \Sigma$ has *order k* if $r_a(n) = \Theta(n^{k-1})$, and has *order* ∞ if $r_a(n) = \Omega(q^n)$ for some $q > 1$ ($q \in \mathbb{R}$).

Consider a directed graph G defined as follows. Vertices of G are letters of Σ. For every $a, b \in \Sigma$, for each occurrence of b in $\varphi(a)$, construct an edge $a \to b$. For instance, if $\varphi(a) = abbab$, we construct two edges $a \to a$ and three edges $a \to b$. Fig. 1 shows an example of graph G.

Using the graph G, one can prove the following

Lemma 1. *For every $a \in \Sigma$, either a has some order $k < \infty$, or has order ∞. For every a of order $k < \infty$, either a never appears in $\varphi^n(a)$ (and then a is called* pre-periodic*), or for each n a unique letter b_n of order k occurs in $\varphi^n(a)$, and the sequence $(b_n)_{n\in\mathbb{Z}_{\ge 0}}$ is periodic (then a is called* periodic*). If a is a periodic letter of order $k > 1$, then at least one letter of order $k-1$ occurs in $\varphi(a)$.*

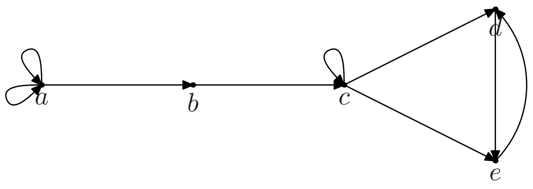


Fig. 1. An example of graph G for the following morphism φ: $\varphi(a) = aab$, $\varphi(b) = c$, $\varphi(c) = cde$, $\varphi(d) = e$, $\varphi(e) = d$. Here a is a letter of order ∞, b is a pre-periodic letter of order 2, c is a periodic letter of order 2, d and e are periodic letters of order 1.

2 Blocks and Semiblocks

A (possibly empty) occurrence $\alpha_{i...j}$ is *a k-block* if it consists of letters of order $\leq k$, and the letters α_{i-1} and α_{j+1} both have order $> k$. The letter α_{i-1} is called the *left border* of this block and is denoted by $\mathrm{LB}(\alpha_{i...j})$. The letter α_{j+1} is called the *right border* of this block and is denoted by $\mathrm{RB}(\alpha_{i...j})$.

The image under φ of a letter of order $\leq k$ cannot contain letters of order $> k$. Let $\alpha_{i...j}$ be a k-block. Then $\varphi(\alpha_{i...j})$ is a suboccurrence of some k-block which is called the *descendant* of $\alpha_{i...j}$ and is denoted by $\mathrm{Dc}(\alpha_{i...j})$. The l-th superdescendant (denoted by $\mathrm{Dc}^l(\alpha_{i...j})$) is the descendant of ... of the descendant of $\alpha_{i...j}$ (l times).

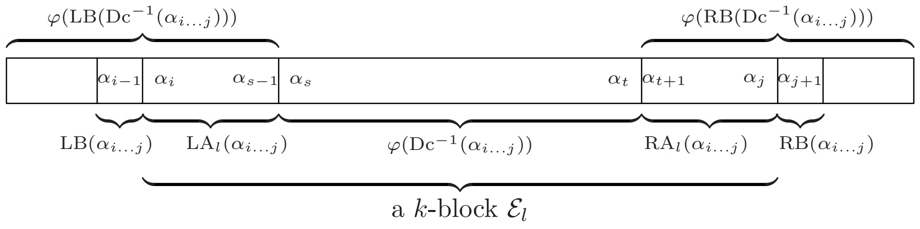
Let $\alpha_{s...t}$ be a k-block in α. Then a unique k-block $\alpha_{i...j}$ such that $\mathrm{Dc}(\alpha_{i...j}) = \alpha_{s...t}$, is called the *ancestor* of $\alpha_{s...t}$ and is denoted $\mathrm{Dc}^{-1}(\alpha_{s...t})$. The l-th superancestor (denoted by $\mathrm{Dc}^{-l}(\alpha_{s...t})$) is the ancestor of ... of the ancestor of $\alpha_{s...t}$ (l times). If $\mathrm{Dc}^{-1}(\alpha_{s...t})$ does not exist (it can happen only if α_{s-1} and α_{t+1} belong to an image of the same letter), then $\alpha_{s...t}$ is called an *origin*. A sequence $\mathcal{E}$ of k-blocks, $\mathcal{E}_0 = \alpha_{i...j}, \mathcal{E}_1 = \mathrm{Dc}(\alpha_{i...j})$, $\mathcal{E}_2 = \mathrm{Dc}^2(\alpha_{i...j}), \ldots, \mathcal{E}_l = \mathrm{Dc}^l(\alpha_{i...j}), \ldots$, where $\alpha_{i...j}$ is an origin, is called an *evolution*.

Lemma 2. *The set of all abstract words that can be origins in α, is finite.*

Corollary 1. *The set of all possible evolutions in α (considered as sequences of abstract words rather than sequences of occurrences in α), is finite.*

Now we define *atoms* inside k-blocks. The l-th left and right atoms exist in a k-block $\mathcal{E}_m$, where $\mathcal{E}$ is an evolution, iff $m \geq l > 0$. First, define the l-th atoms inside the k-block $\mathcal{E}_l$. Let $\mathcal{E}_l = \alpha_{i...j}$. There is a suboccurrence $\alpha_{s...t} = \varphi(\mathrm{Dc}^{-1}(\alpha_{i...j}))$ inside of it. The occurrence $\alpha_{i...s-1}$ that comes from the image of the left border of the ancestor, is called *the l-th left atom* of the block and is denoted by $\mathrm{LA}_l(\alpha_{i...j})$. Similarly, the occurrence $\alpha_{t+1...j} = \mathrm{RA}_l(\alpha_{i...j})$ is called *the l-th right atom* of the block. Then, $\mathrm{LA}_l(\mathcal{E}_m) = \varphi^{m-l}(\mathrm{LA}_l(\mathcal{E}_l))$, and the same for right atoms. See Fig. 2.

Let $\mathcal{E}$ be an evolution of k-blocks. Consider a sequence $\mathrm{LB}(\mathcal{E}_0), \ldots, \mathrm{LB}(\mathcal{E}_l), \ldots$ All these letters are of order $> k$, and the letter $\mathrm{LB}(\mathcal{E}_{l+1})$ is the rightmost letter of order $> k$ in $\varphi(\mathrm{LB}(\mathcal{E}_l))$. Hence, not later than starting from $\mathrm{LB}(\mathcal{E}_{|\Sigma|})$, this



$\varphi(\mathrm{LB}(\mathrm{Dc}^{-1}(\alpha_{i\ldots j})))$
$\varphi(\mathrm{RB}(\mathrm{Dc}^{-1}(\alpha_{i\ldots j})))$
α_{i-1}
α_i
α_{s-1}
α_s
α_t
α_{t+1}
α_j
α_{j+1}
$\mathrm{LB}(\alpha_{i\ldots j})$
$\mathrm{LA}_l(\alpha_{i\ldots j})$
$\varphi(\mathrm{Dc}^{-1}(\alpha_{i\ldots j}))$
$\mathrm{RA}_l(\alpha_{i\ldots j})$
$\mathrm{RB}(\alpha_{i\ldots j})$
a k-block $\mathcal{E}_l$

Fig. 2. Structure of a k-block

sequence (of abstract letters) is periodic. Its period length is denoted by $\mathrm{LB}\mathbf{P}(\mathcal{E})$. The same can be said about the sequence of right borders (the period length is $\mathrm{RB}\mathbf{P}(\mathcal{E})$). Their l. c. m. is denoted by $\mathrm{B}\mathbf{P}(\mathcal{E})$. The exact place of the evolution where both sequences become periodic (i. e. both of them have reached at least the first positions of their first periods) is denoted by $\mathrm{F}(\mathcal{E})$. The block $\mathcal{E}_{\mathrm{F}(\mathcal{E})}$ is called *first pre-stable*. The block $\mathcal{E}_{\mathrm{F}(\mathcal{E})}$ and all its superdescendants are called *pre-stable*.

Notice that $\mathrm{LA}_{l+1}(\mathcal{E}_{l+1})$ depends only on $\mathrm{LB}(\mathcal{E}_l)$, not on the whole $\mathcal{E}_l$. Hence, the sequence $\mathrm{LA}_{\mathrm{F}(\mathcal{E})+1}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+1}), \ldots, \mathrm{LA}_l(\mathcal{E}_l), \ldots$ is periodic with a period of length $\mathrm{LB}\mathbf{P}(\mathcal{E})$ if considered as a sequence of abstract words. Consider one of its periods, e. g., $\mathrm{LA}_{\mathrm{F}(\mathcal{E})+1}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+1}), \ldots,$ $\mathrm{LA}_{\mathrm{F}(\mathcal{E})+\mathrm{LB}\mathbf{P}(\mathcal{E})}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+\mathrm{LB}\mathbf{P}(\mathcal{E})})$. There are three possible cases.

Case I. At least one of these words contains a letter of order k.

Case II. None of these words contain a letter of order k, but at least one of them is not empty.

Case III. All these words are empty.

Similarly, cases I, II and III are defined for right borders and atoms. These cases happen independently at right and at left, in any combination.

Now we define the *core* of a pre-stable k-block (notation: C). Consider the first pre-stable k-block of the evolution. Its core is the whole block. Then, the core of $\mathcal{E}_{l+1}$ is $\varphi(\mathrm{C}(\mathcal{E}_l))$. Thus, $\mathrm{C}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+l}) = \varphi^l(\mathcal{E}_{\mathrm{F}(\mathcal{E})})$. The suboccurrence in the block between the core and its left (right) border is called its *left (right) component*.

We know that $\mathrm{F}(\mathcal{E}) \leq |\Sigma|$. Thus $|\,\mathrm{C}(\mathcal{E}_{\mathrm{F}(\mathcal{E})})| \leq D := 2|\varphi|^{|\Sigma|+1}$.

Lemma 3. *Consider a k-block and its evolution. If case I holds at right (at left), the right (left) component has growth rate $\Theta(n^k)$. If case II or III holds, it has growth rate $O(n^{k-1})$.*

Now we introduce semiblocks to consider evolutions of words that grow 'at one side' (at left or at right only), while evolutions of blocks represent sequences of words that grow (that may have atoms) 'at both sides'.

Let $\alpha_{i\ldots j}$ be a (possibly empty) occurrence in α consisting of letters of order $\leq k$, and suppose α_{j+1} has order $> k$. Suppose also that $j - i + 1 \leq D$. Then $\alpha_{i\ldots j}$ is called a *right k-semiblock*, and α_{j+1} is called its *right border* (a left border of a right k-semiblock does not exist). The image of the k-semiblock under φ prolonged right upto the leftmost letter of order $> k$ is also called a k-semiblock — the *descendant* of $\alpha_{i\dots j}$. Evolution of right k-semiblocks, with arbitrary k-semiblock of length $\leq D$ as an origin, is defined analogously to how it was defined for k-blocks.

Left k-semiblocks and their evolutions are defined in a similar way.

The length of origins of k-semiblock evolutions is bounded by definition, so the set of all k-semiblock evolutions (considered as sequences of abstract words) is finite. Like an evolutional sequence of borders of k-block, an evolutional sequence of left (right) borders of left (right) k-semiblock is eventually periodic with pre-period not greater than $|\Sigma|$.

Pre-stable k-semiblocks and the first pre-stable k-semiblocks are defined analogously to those of k-blocks. All the notation we introduced for k-blocks, is used for k-semiblocks as well. Fig. 3 shows the structure of a k-semiblock.

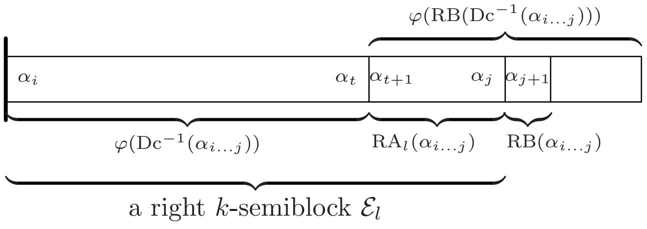

$\varphi(\mathrm{RB}(\mathrm{Dc}^{-1}(\alpha_{i\dots j})))$
α_i
α_t
α_{t+1}
α_j
α_{j+1}
$\varphi(\mathrm{Dc}^{-1}(\alpha_{i\dots j}))$
$\mathrm{RA}_l(\alpha_{i\dots j})$
$\mathrm{RB}(\alpha_{i\dots j})$
a right k-semiblock $\mathcal{E}_l$

Fig. 3. Structure of a right k-semiblock

The *core* (C) of a k-semiblock is defined similarly too. Namely, let $\mathcal{E}$ be an evolution of k-semiblocks. Consider the first pre-stable k-semiblock of $\mathcal{E}$. Its core is the whole semiblock. Then, the core of $\mathcal{E}_{l+1}$ is $\varphi(\mathrm{C}(\mathcal{E}_l))$. Thus, $\mathrm{C}(E_{\mathrm{F}(\mathcal{E})+l}) = \varphi^l(E_{\mathrm{F}(\mathcal{E})})$. The suboccurrence in the semiblock between the core and its left border is called its left component.

3 1-Blocks and 1-Semiblocks

Now we will consider 1-blocks and 1-semiblocks more accurately.

From the fact that every 1-block or 1-semiblock consists of letters of order 1 only, and from Corollary 1, it follows that *all core lengths are bounded by a single constant that depends on φ and Σ only.* For an evolution $\mathcal{E}$, cores of the block $\mathcal{E}_{\mathrm{F}(\mathcal{E})+|\Sigma|}$ and its descendants consist of periodic letters only.

A core of 1-block or 1-semiblock is called its (unique) *central kernel.*

Consider a 1-block or a right 1-semiblock $\mathcal{E}_l$ where $\mathcal{E}$ is an evolution and $l \geq |\Sigma|$. The concatenation $\mathrm{RpreP}(\mathcal{E}_l) := \mathrm{RA}_{l-|\Sigma|+1}(\mathcal{E}_l) \dots \mathrm{RA}_{l-1}(\mathcal{E}_l)\,\mathrm{RA}_l(\mathcal{E}_l)$ is called the *right pre-period* of $\mathcal{E}_l$. Left pre-periods are defined similarly. All letters in the right component of a 1-block or a 1-semiblock outside its right pre-period are periodic.

The left (right) component of a 1-block or a 1-semiblock except its left (right) pre-period is called the *left (right) repetition* (notation: LR, RR).

Lemma 4. *Let $\mathcal{E}$ be an evolution of 1-blocks or left 1-semiblock, $l > \mathrm{F}(\mathcal{E}) + |\Sigma|$. Then there are such abstract words $\mathrm{LP}(\mathcal{E}_l)$ (for each l), that $\mathrm{LR}(\mathcal{E}_l)$ is a left $\mathrm{LP}(\mathcal{E}_l)$-periodic word and:*

The sequence $\mathrm{LP}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+|\Sigma|}), \dots, \mathrm{LP}(\mathcal{E}_l), \dots$ is periodic. In particular, lengths of these words are bounded by a constant that depends on Σ and φ only.

The lengths of the incomplete occurrences become periodic starting from $\mathrm{LR}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+|\Sigma|})$. The sequence (of abstract words) $\mathrm{LpreP}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+|\Sigma|}), \dots,$ $\mathrm{LpreP}(\mathcal{E}_l), \dots$ is periodic. Lengths of these words are bounded by a constant that depends on Σ and φ only. The sequence (of abstract words) $\mathrm{C}(\mathcal{E}_{\mathrm{F}(\mathcal{E})+|\Sigma|}), \dots, \mathrm{C}(\mathcal{E}_l), \dots$ is also periodic.

Let $\mathcal{E}$ be an evolution of 1-blocks or of 1-semiblocks. If case I holds at left or at right, we say that $\mathcal{E}_l$ is stable if $l \geq \mathrm{F}(\mathcal{E}) + |\Sigma| + 2\,\mathrm{B}\mathbf{P}(\mathcal{E})|\Sigma|!$. If case III holds both at left and at right (case II is impossible for 1-blocks), $\mathcal{E}_l$ is stable if $l \geq \mathrm{F}(\mathcal{E}) + |\Sigma|$. If $\mathcal{E}_l$ is stable then:

1. It is pre-stable.

2. The pre-periods $\mathrm{LpreP}(\mathcal{E}_l)$ and $\mathrm{RpreP}(\mathcal{E}_l)$ and the core $\mathrm{C}(\mathcal{E}_l)$ belong to the periodic parts of the corresponding sequences from Lemma 4.

3. If case I holds at left or at right, $\mathrm{LR}(\mathcal{E}_l)$ and $\mathrm{RR}(\mathcal{E}_l)$ consist of at least two their periods (when considered as left $\mathrm{LP}(\mathcal{E}_l)$-periodic and right $\mathrm{RP}(\mathcal{E}_l)$-periodic words, respectively).

The least number l such that $\mathcal{E}_l$ is stable is denoted by $\mathrm{S}(\mathcal{E})$.

The following lemma is an easy corollary of Lemma 4.

Lemma 5. *Let $\mathcal{E}$ be an evolution of 1-blocks or 1-semiblocks. The sequence $|\,\mathrm{LR}(\mathcal{E}_{\mathrm{S}(\mathcal{E})})|, \dots, |\,\mathrm{LR}(\mathcal{E}_l)|, \dots$ considered modulo s, is periodic for all $s \in \mathbb{N}$.*

Fig. 4 shows the detailed structure of a 1-block.

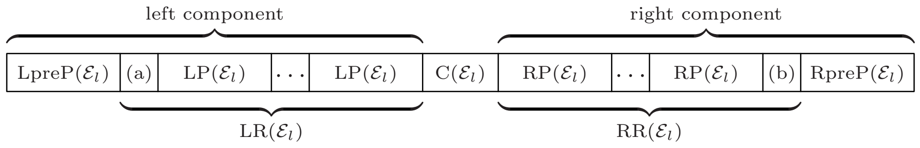


Fig. 4. Detailed structure of the 1-block $\mathcal{E}_l$: (a), (b) — the incomplete occurrences of periods

4 2-Blocks

Now consider 2-blocks more accurately. First, let us give definitions concerning 2-blocks that are necessary to define continuously periodic evolutions. Through this section, we will give examples based on the following morphism φ: $\varphi(s) = saca$, $\varphi(a) = bad$, $\varphi(b) = dbd$, $\varphi(c) = ece$, $\varphi(d) = d$, $\varphi(e) = e$. Then $\varphi^\infty(s) = \alpha = s\,aca\,badecebad\,dbdbaddeeceedbdbadd\dots$. Here s is a letter of order 4, a is a letter of order 3, b and c are letters of order 2 and d and e are letters of order 1. Consider an evolution $\mathcal{E}$ of 2-blocks, whose origin is $\alpha_{3\ldots3} = c$. A 2-block $\mathcal{E}_l$ where l is large enough looks as follows:

$$\underbrace{dd..d}_{\text{left component}}\underbrace{ee..ecee..e}_{\text{core}}\underbrace{dd..db\ldots bdd..dbdd..db\ldots ddbdddbdb}_{\text{right component}}.$$

Here case I holds at left and case II holds at right. Intervals denoted by ... may contain many intervals denoted by ..

First, let us define *stable 2-blocks*. We will impose requirements on the number of iterations to be made from the beginning of the evolution, to guarantee for the block to be stable.

If a 2-block is stable, it is required to be pre-stable. Other requirements depend on the case that holds at left and at right of the given block.

Case I (e. g., at right). Consider an alphabet Σ' which is Σ without all the letters of order 1. Let us define a morphism φ' as follows: for $a \in \Sigma'$, to obtain $\varphi'(a)$, we take the word $\varphi(a)$ and remove all the letters of order 1 from it. Let α' be a new pure morphic sequence generated by φ'. (If α is non-periodic, then α' is infinite.) In other words, α' is obtained from α by removing all letters of order 1. All the 2-blocks in α become 1-blocks in α'. In the example, the corresponding 1-block in α' is $cbb\ldots b$. For a given 2-block in α to be stable, we require the corresponding 1-block in α' to be stable too.

Let $\mathcal{E}$ be an evolution of 2-blocks in α, $\alpha'_{s\ldots t}$ be the 1-block corresponding to $\alpha_{i\ldots j} \in \mathcal{E}$. Consider an occurrence of $\mathrm{RP}(\alpha'_{s\ldots t})$ in $\alpha'_{s\ldots t}$. We may assume that the number of atoms it consists of is divisible by $\mathrm{B}\mathbf{P}(\mathcal{E})$. A suboccurrence in $\alpha_{i\ldots j}$ containing all the letters of $\mathrm{RP}(\alpha'_{s\ldots t})$ and all the letters of order 1 to the right of $\mathrm{RP}(\alpha'_{s\ldots t})$ upto the closest letter of order 2, is called a *pseudo-period*. In the example, both left and right borders are always a, and $\mathrm{B}\mathbf{P}(\mathcal{E}) = 1$. Thus, a right pseudo-period is an occurrence $bdd..d$ (the amount of d's may differ in different pseudo-periods).

For a given 2-block to be stable, its right component is required to contain at least two pseudo-periods. Moreover, all the 1-blocks inside two leftmost of them should be stable.

Now consider an occurrence of the empty word immediately at left of $\mathrm{RB}(\mathcal{E}_{\mathrm{F}(\mathcal{E})})$. This occurrence is a right 1-semiblock (an origin). It is called the *right outer central 1-semiblock* of $\mathcal{E}_{\mathrm{F}(\mathcal{E})}$ and is denoted by $\mathrm{RO}(\mathcal{E}_{\mathrm{F}(\mathcal{E})})$. Then, $\mathrm{RO}(\mathcal{E}_{l+1}) := \mathrm{Dc}(\mathrm{RO}(\mathcal{E}_l))$. For a 2-block $\mathcal{E}_l$ to be stable, we require $\mathrm{RO}(\mathcal{E}_l)$ to be stable too. In the example, the right outer central semiblock is the occurrence $dd..d$ between the letters e and b (the leftmost in the right component).

Case II (at right). If case II holds, the right component consists of the right outer central 1-semiblock (and noting else). For a 2-block to be stable, we require this 1-semiblock to be stable. In the example, case II holds at left, thus the left component of the block is its left outer semiblock.

Case III. No more requirements.
There are still other requirements concerning the core. If it contains letters of order 1 only, all these letters are required to become periodic. Let it contain

some letters of order 2. Consider $\mathrm{C}(\mathcal{E}_{\mathrm{F}(\mathcal{E})})$. Its length is not greater than D, so its suboccurrence between its right end and the rightmost letter of order 2 may be considered as a left 1-semiblock. It is called the *right inner central 1-semiblock* (notation: RI). Then, $\mathrm{RI}(\mathcal{E}_{l+1}) := \mathrm{Dc}(\mathrm{RI}(\mathcal{E}_l))$. In the example, the right inner 1-semiblock is the word *ee..e* between the letter c and the right component. For the 2-block $\mathcal{E}_l$ to be stable, we require $\mathrm{LI}(\mathcal{E}_l)$ and $\mathrm{RI}(\mathcal{E}_l)$ to be stable. Moreover, the letters of order 2 inside $\mathrm{C}(\mathcal{E}_l)$ should be periodic, all the 1-blocks between them have to be stable too.

These are all the requirements we impose on 2-block to be stable. The least number l such that $\mathcal{E}_l$ is stable is denoted by $\mathrm{S}(\mathcal{E})$.

Now we define left and right pre-periods of 2-blocks. Let $\alpha_{i...j}$ be a stable 2-block and $\alpha'_{s...t}$ be the corresponding 1-block. If case I holds (e. g. at right), the occurrence $\alpha_{u...j}$ is called the right pre-period (RpreP) if u is the maximal index such that

1. $\mathrm{RpreP}(\alpha'_{s...t})$ is contained in $\alpha_{u...j}$.
2. 1-blocks between letters of order 2 in the right component of $\alpha_{i...j}$ outside $\alpha_{u...j}$ are all stable.
3. α_u is a letter of order 2.

Notice that $\alpha_{u...j}$ is then completely inside the last L atoms, where $L = |\Sigma| + \max\{S(\mathcal{F}) : \mathcal{F}$ is an evolution of 1-blocks$\}$. Hence, the sequence (of abstract words) $\mathrm{RpreP}(\mathcal{E}_{\mathrm{S}(\mathcal{E})}), \ldots, \mathrm{RpreP}(\mathcal{E}_l), \ldots$ is periodic. In the example, the right pre-period is really long since all the 1-blocks outside it are stable, and the length of a stable 1-block is greater than $|\Sigma|! = 120$. However, its structure can be written as follows: *bd..dbd..d . . . ddbdb*.

If case II holds, $\mathrm{RpreP}(\alpha_{i...j}) := \mathrm{RpreP}(\mathrm{RO}(\alpha_{i...j}))$. If case III holds, the right pre-period is the occurrence of the empty word immediately at left of α_{j+1}.

Now we define central kernels of stable 2-blocks. Let $\alpha_{i...j}$ be a 2-block.

There may be two kinds of central kernels: ones that are suboccurrences of $\mathrm{C}(\alpha_{i...j})$ and ones that are outside it. To define central kernels inside $\mathrm{C}(\alpha_{i...j})$, consider two cases. If it consists of letters of order 1 only, it is called a central kernel itself. Let $\mathrm{C}(\alpha_{i...j})$ contain letters of order 2. Then, if $\alpha_{i...j}$ is stable, $\mathrm{C}(\alpha_{i...j})$ can be decomposed into $\mathrm{LI}(\alpha_{i...j}), \mathrm{RI}(\alpha_{i...j})$ and some 1-blocks between them (they all evolute together with the 2-block). Central kernels of the 2-block are all the cores, left and right pre-periods of these 1-blocks and semiblocks and letters of order 2 themselves. In the example, central kernels inside the core are the occurrences of the word *eeeee* immediately at left and at right of c, the letter c itself and two occurrences of the empty word between the core and the left and the right components.

To define central kernels of $\alpha_{i...j}$ at right (or at left) of $\mathrm{C}(\alpha_{i...j})$, consider cases I, II and III again. If case I holds at right, central kernels at right of $\mathrm{C}(\alpha_{i...j})$ are $\mathrm{RpreP}(\mathrm{RO}(\alpha_{i...j}))$ and $\mathrm{C}(\mathrm{RO}(\alpha_{i...j}))$. If case II holds, the only central kernel at right is $\mathrm{C}(\mathrm{RO}(\alpha_{i...j}))$. In case III there are no more cental kernels. In the example, central kernels outside the core are the occurrence of the word *ddddd* immediately at left of the leftmost letter b and two occurrences of the empty word between the core and the left and the right components once more.

Central kernels of a block or a semiblock, as well as its right and left pre-periods, are called *simple kernels.* A concatenation of consecutive simple kernels (with no simple kernel *immediately* at left and at right of it) is called *kernel.*

The following remarks are easy corollaries of Lemma 4 and the definition of central kernels. Every kernel of a stable 2-block $\mathcal{E}_l$ corresponds to some kernel of $\mathcal{E}_{l+1}$, and vice versa. Thus, we have evolutional sequences of kernels concerned with $\mathcal{E}$. (The amount of these sequences equals the amount of kernels in *each* $\mathcal{E}_l$). These sequences are periodic. A part of $\mathcal{E}_l$ between its (consecutive) central kernels is actually either an empty word or a (left or right) repetition of some 1-block or semiblock $\alpha_{s\ldots t}$. The part of $\mathcal{E}_{l+1}$ between the corresponding central kernels is the (left or right, respectively) repetition of $\mathrm{Dc}(\alpha_{s\ldots t})$.

The part of a 2-block between its rightmost (leftmost) central kernel and its right (left) pre-period is called the *right (left) pseudorepetition* of the 2-block (notation: $\mathrm{RpR}, \mathrm{LpR}$).

Now we can give an example of a 2-block with all its parts marked:

$$\underbrace{\underbrace{\overline{(ddddd)}}_{\text{LpreP}}\underbrace{d..d}_{\text{LpR}}}_{\text{left component} = \text{LO}}\overline{()()}\underbrace{\underbrace{e..\overline{(eeeee)}}_{\text{LI}}\overline{(c)}\underbrace{\overline{(eeeee)}..e}_{\text{RI}}}_{\text{core}}\overline{()()}\underbrace{\underbrace{d..\overline{(ddddd)}}_{\text{RO}}\underbrace{b\ldots\overbrace{bd..d}^{\text{a pseudo-period}}}_{\text{RpR}}\underbrace{\overline{(bd..db\ldots ddbdb)}}_{\text{RpreP}}}_{\text{right component}}.$$

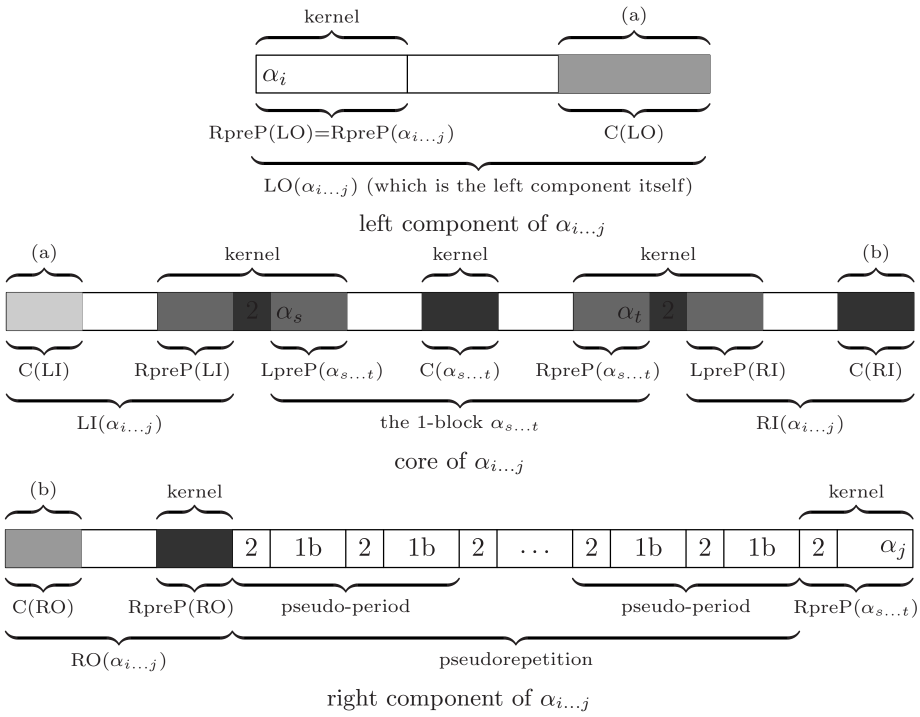
Here parentheses denote simple kernels and lines above denote kernels. Fig. 5 shows a detailed structure of a 2-block in a more general case.

These are all the notions concerning 2-blocks that are necessary to define continuously periodic evolutions. We have to say another several words concerning left and right pseudorepetitions. Let $\alpha_{i\ldots j}$ be a 2-block that belongs to an evolution $\mathcal{E}$ and such that case I holds at right. Consider then all the 1-blocks between the letters of order 2 in $\mathrm{RpR}(\alpha_{i\ldots j})$. The sequence of evolutions they belong to is denoted by $\mathfrak{E}_R(\alpha_{i\ldots j})$. $\mathfrak{E}_R(\alpha_{i\ldots j})$ ($\mathfrak{E}_L(\alpha_{i\ldots j})$) is considered beginning at left (resp. at right).

Let l be divisible by $\mathrm{B}\mathbf{P}(\mathcal{E})$ and $l-\mathrm{B}\mathbf{P}(\mathcal{E}) > \mathrm{F}(\mathcal{E})$. Consider the concatenation of (right) atoms $\mathrm{RA}_{l-\mathrm{B}\mathbf{P}(\mathcal{E})}(\mathcal{E}_l)\ldots\mathrm{RA}_{l-1}(\mathcal{E}_l)\,\mathrm{RA}_l(\mathcal{E}_l)$. This concatenation does not depend on l. It may contain some pre-periodic letters of order 2, but the same atoms of the $|\Sigma|$-th and further superdescendants of $\mathcal{E}_l$ do not contain them. The same atoms of $\mathcal{E}_{l+|\Sigma|}$ form some sequence of periodic letters of order 2, separated by 1-blocks, and all of them do not depend on l. 1-blocks inside the same atoms of $\mathcal{E}_{l+|\Sigma|+m}$ are exactly the m-th superdescendants of ones inside $\mathcal{E}_{l+|\Sigma|}$, new 1-blocks will no longer arise in that atoms.

All we have said above and Lemma 4 imply the following lemma:

Lemma 6. *Let $\mathcal{E}_l$ be a 2-block such that case I holds at right, and let $\alpha'_{s\ldots t}$ be the corresponding 1-block. Consider all the letters of order 2 inside its right component. These letters, except for not more than Q rightmost ones that form* $\mathrm{RpreP}(\alpha'_{s\ldots t})$*, form a p_l-periodic sequence. Here Q is a constant that depends on Σ and φ only. p_l possibly depends on l, but the sequence $(p_l)_{l=\mathrm{S}(\mathcal{E})}^{\infty}$ is periodic.*



kernel
(a)
α_i
RpreP(LO)=RpreP($\alpha_{i\ldots j}$)
C(LO)
LO($\alpha_{i\ldots j}$) (which is the left component itself)
left component of $\alpha_{i\ldots j}$
(a)
kernel
kernel
kernel
(b)
2
α_s
α_t
2
C(LI)
RpreP(LI)
LpreP($\alpha_{s\ldots t}$)
C($\alpha_{s\ldots t}$)
RpreP($\alpha_{s\ldots t}$)
LpreP(RI)
C(RI)
LI($\alpha_{i\ldots j}$)
the 1-block $\alpha_{s\ldots t}$
RI($\alpha_{i\ldots j}$)
core of $\alpha_{i\ldots j}$
(b)
kernel
kernel
2 1b 2 1b 2 … 2 1b 2 1b 2 α_j
C(RO)
RpreP(RO)
pseudo-period
pseudo-period
RpreP($\alpha_{s\ldots t}$)
RO($\alpha_{i\ldots j}$)
pseudorepetition
right component of $\alpha_{i\ldots j}$

Fig. 5. Detailed structure of the 2-block $\alpha_{i\ldots j}$, where case II holds at left and case I holds at right: 2 denotes a letter of order 2, 1b denotes a 1-block, two letters (a) denote two parts of a single kernel, two letters (b) denote another kernel. Central kernels are filled.

Everything we have said about components of 1-blocks, can be said about this sequence too.

During the evolution (as l grows), $\mathfrak{E}_R(\mathcal{E}_l)$ is prolonged to the right (and is not changed elsewhere). The amount of blocks added at right per iteration is periodic (the period length is $\mathrm{B}\mathbf{P}(\mathcal{E})$*). Thus we can get an infinite sequence, which is denoted by $\mathfrak{E}_R(\mathcal{E})$.*

$\mathfrak{E}_R(\mathcal{E})$ is a periodic sequence if it is considered as sequence of abstract words with known left and right borders. If two 1-blocks inside $\mathfrak{E}_R(\mathcal{E}_l)$ are at the same places in two consecutive periods of $\mathfrak{E}_R(\mathcal{E})$, then one of them is the $\mathrm{B}\mathbf{P}(\mathcal{E})$*-th superdescendant of another.*

The following lemma is a corollary of the periodicities we have noticed above.

Lemma 7. *Let $\mathcal{E}$ be an evolution of 2-blocks, $s \in \mathbb{N}$. The sequences $|\mathcal{E}_{\mathrm{S}(\mathcal{E})}|, \ldots, |\mathcal{E}_l|, \ldots$ and $|\operatorname{RpR}(\mathcal{E}_{\mathrm{S}(\mathcal{E})})|, \ldots, |\operatorname{RpR}(\mathcal{E}_l)|, \ldots$ modulo s are periodic.*

5 Continuously Periodic Evolutions

Let $\mathcal{E}$ be an evolution of k-blocks ($k = 1, 2$) or k-semiblocks ($k = 1$). Let case II or III hold at right. Let k_0 be the order of $a := \mathrm{RB}(\mathcal{E}_{\mathrm{S}(\mathcal{E})})$. Then $\varphi(a)$ contains a letter of order $k_0 - 1$. Hence, the part of $\varphi^{\mathrm{B}\mathbf{P}(\mathcal{E})}(\mathrm{RB}(\mathcal{E}_{\mathrm{S}(\mathcal{E})}))$ outside $\mathrm{Dc}^{\mathrm{B}\mathbf{P}(\mathcal{E})}(\mathcal{E}_{\mathrm{S}(\mathcal{E})})$ will be longer than one letter and will start with a again. Thus we can construct an abstract sequence (it is built like a pure morphic sequence and can be prolonged infinitely) starting with a. It is called *the right bounding sequence* of $\mathcal{E}_{\mathrm{S}(\mathcal{E})}$ (notation: RBS). If we consider $\mathcal{E}_{\mathrm{S}(\mathcal{E})+m}$ $(1 \le m \le \mathrm{B}\mathbf{P} - 1)$ instead of $\mathcal{E}_{\mathrm{S}(\mathcal{E})}$, we can build another sequence in the similar way. These sequences are also called *right bounding sequences.* These sequences are abstract, they are not occurrences in α. However, there is a beginning of one of these sequences in α at right of each block $\mathcal{E}_l$, and the lengths of these beginnings grow as evolution passes. Moreover, the length of this beginning is $\Theta(l^{k_0})$.

Left bounding sequences are defined in a similar way.

Consider all the evolutions $\mathcal{E}$ of 1-blocks and 1-semiblocks. Consider all the words $\psi(\mathrm{LP}(\mathcal{E}_l))$ and $\psi(\mathrm{RP}(\mathcal{E}_l))$ (where $\mathcal{E}_l$ is stable and case I holds at left and at right, respectively). If one of these encoded periods is *completely* p-periodic itself (where p is less than its length), consider the smallest its complete period instead of it. This (finite due to Lemma 4) set of words is called the set of *admissible periods.* A cyclic shift of an admissible period is also called an admissible period. Since the set of admissible periods is finite, we assume they are enumerated in some way.

The following definition is very significant for the proof.

Definition 1. *Let $\mathcal{E}$ be an evolution of k-blocks ($k = 1, 2$) or k-semiblocks ($k = 1$) such that $|\mathcal{E}_l| \to \infty$ as $l \to \infty$ (it means that either the core contains letters of order > 1 or case I or II holds at left or at right). $\mathcal{E}$ is called* continuously periodic *if for every stable block $\mathcal{E}_l = \alpha_{i\ldots j}$ it is possible to choose some (actually, not more than three) of its kernels $\alpha_{i_1\ldots j_1}, \ldots, \alpha_{i_s\ldots j_s}$ so that:*

1. The total amount of letters of order k in any of the words $\alpha_{j_t\ldots i_{t+1}}$, grows unboundedly as $l \to \infty$ (thus, in particular, no two central kernels can be chosen);

2. All the words $\psi(\alpha_{j_t\ldots i_{t+1}})$, are (left or right) γ-periodic for some admissible periods γ;

3. If $j_s < j$, then case II or III should hold at right (and if $\mathcal{E}$ is an evolution of semiblocks, they should be right *ones), the infinite word $\psi(\alpha_{j_s\ldots j}\,\mathrm{RBS}(\mathcal{E}_l))$ is periodic and its period is admissible,*
And similar condition at left.

In fact, Lemma 4 implies that all the evolutions of 1-blocks or 1-semiblocks are continuously periodic.

After we gave the definition of a continuously periodic evolution, the statements of Propositions 1 and 2 are completely formulated.

6 An Example

Here we give an example of a sequence with complexity $\Theta(n^{3/2})$.

Let $\Sigma = \{1,2,3,4\}$. Consider the following morphism φ: $\varphi(4) = 43, \varphi(3) = 32, \varphi(2) = 21, \varphi(1) = 1$ and the following coding ψ: $\psi(4) = 4, \psi(3) = 3, \psi(2) = \psi(1) = 1$. Then $\varphi^\infty(4) = \alpha = 4332322132212113221211211113\ldots$, and $\psi(\varphi^\infty(4)) = \beta = 4331311131111113111111111113\ldots$ In α there is a unique evolution $\mathcal{E}$ of 2-blocks, whose origin is the occurrence of the empty word between 4 and 3. Case I holds at left, so $|\mathcal{E}_l| = \Theta(l^2)$. It is clear that $\mathcal{E}$ is continuously periodic evolution, since $\psi(\mathcal{E}_l)$ consists of repeating letters 1. Proposition 2 asserts $p_\beta = O(n^{3/2})$. Let us show $p_\beta = \Omega(n^{3/2})$.

Indeed, β consists of alternating letters 3 and 2-blocks $\mathcal{E}_l$. Given a subword $\beta_{i\ldots j}$ of length n, consider the longest 2-block $\beta_{s\ldots t}$ that is completely inside it. If the longest 2-block in another subword $\beta_{i'\ldots j'}$ is of another length or if it is of the same length but starts from the different location in $\beta_{i'\ldots j'}$ than $\beta_{s\ldots t}$ in $\beta_{i\ldots j}$, the words $\beta_{i\ldots j}$ and $\beta_{i'\ldots j'}$ are obviously distinct.

Assume $3n/4 > |\beta_{s\ldots t}| > n/2$. Then there are $\Theta(\sqrt{n})$ possibilities for its length. There are $\Theta(n)$ possibilities for its location in $\beta_{i\ldots j}$, thus we obtain $\Theta(n\sqrt{n})$ different subwords of length n in β. Therefore, $p_\beta(n) = \Omega(n^{3/2})$, and hence $p_\beta(n) = \Theta(n^{3/2})$.

Acknowledgements

The author is kindly grateful to Y. Pritykin for a useful discussion.

References

1. Allouche, J.-P., Shallit, J.: Automatic Sequences. Cambridge University Press, Cambridge (2003)
2. Ferenczi, S.: Complexity of sequences and dynamical systems. Discrete Math. 206(1-3), 145–154 (1999)
3. Nicolas, F.: Master's thesis
4. Pansiot, J.-J.: Complexité des facteurs des mots infinis engendrés par morphimes itérés. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 380–389. Springer, Heidelberg (1984)

Comparing Universal Covers in Polynomial Time

Jiří Fiala[1] and Daniël Paulusma[2]

[1] Charles University, Faculty of Mathematics and Physics,
DIMATIA and Institute for Theoretical Computer Science (ITI)[⋆],
Malostranské nám. 2/25, 118 00, Prague, Czech Republic
fiala@kam.mff.cuni.cz

[2] Department of Computer Science, Durham University[⋆⋆],
Science Laboratories, South Road,
Durham DH1 3EY, England
daniel.paulusma@durham.ac.uk

Abstract. The universal cover T_G of a connected graph G is the unique (possible infinite) tree covering G, i.e., that allows a locally bijective homomorphism from T_G to G. Universal covers have major applications in the area of distributed computing. It is well-known that if a graph G covers a graph H then their universal covers are isomorphic, and that the latter can be tested in polynomial time by checking if G and H share the same degree refinement matrix. We extend this result to locally injective and locally surjective homomorphisms by following a very different approach. Using linear programming techniques we design two polynomial time algorithms that check if there exists a locally injective or a locally surjective homomorphism, respectively, from a universal cover T_G to a universal cover T_H. This way we obtain two heuristics for testing the corresponding locally constrained graph homomorphisms. As a consequence, we have obtained a new polynomial time algorithm for testing (subgraph) isomorphism between universal covers, and for checking if there exists a role assignment (locally surjective homomorphism) from a given tree to an arbitrary fixed graph H.

1 Introduction

In this paper, we consider simple, undirected, possibly infinite but connected graphs. See [5] for undefined graph terminology. A *(graph) homomorphism* $f : G \to H$ from a graph $G = (V_G, E_G)$ to a graph $H = (V_H, E_H)$ is a mapping $V_G \to V_H$ such that $(f(u), f(v)) \in E_H$ whenever $(u, v) \in E_G$. Graph homomorphisms have a great deal of applications in graph theory, computer science and other fields, see the monograph [16].

A graph homomorphism f from a graph G to a graph H can be required to satisfy some local constraint [9]. If, for every $u \in V_G$ the restriction of f,

[⋆] Supported by the Ministry of Education of the Czech Republic as project 1M0021620808.

[⋆⋆] Supported by EPSRC as project EP/D053633/1.

i.e. the mapping $f_u : N(u) \rightarrow N(f(u))$, is bijective, we say that f is *locally bijective* [1,19], and we write $G \xrightarrow{B} H$. If, for every $u \in V_G$, f_u is injective, we say that f is *locally injective* [10,11], and we write $G \xrightarrow{I} H$. If, for every $u \in V_G$, f_u is surjective, we say that f is *locally surjective* [13,20], and we write $G \xrightarrow{S} H$.

Locally bijective homomorphisms, also called graph coverings, originally arose in topological graph theory [22], and have applications in distributed computing [4], in recognizing graphs by networks of processors [2], and in constructing highly transitive regular graphs [3]. Locally injective homomorphisms, also called partial graph coverings, have been studied due to their applications in models of telecommunication [11], in distance constrained labelings of graphs with applications to frequency assignment [12], and as indicators of the existence of homomorphisms of derivate graphs (line graphs) [24]. Locally surjective homomorphisms, also called role assignments, have applications in distributed computing [6] and social science [8,26].

The main computational question is whether for every graph H the problem of deciding if an input graph G has a homomorphism of given type $* = B, I$ or S to the fixed graph H can be classified as either NP-complete or polynomially solvable. For the locally surjective homomorphisms this classification is known [13], with the problem for every connected H on at least three vertices being NP-complete. For the locally bijective and injective cases there are many partial results, see e.g. [11,19], but even conjecturing a classification for these two cases is problematic. In this paper, we continue the study started in [14] in order to get more insight in the structure of these computational issues.

1.1 Problem Formulation

The existence of a locally constrained homomorphism imposes a partial order on the class of connected graphs $\mathcal{C}$ for each of the three local constraints B, I, and S [14]. We can relax these three orders in two different ways. This leads to two different heuristics for testing if $G \xrightarrow{*} H$ for two given graphs G and H under each type $* = B, I, S$.

Firstly, we can transform the partial orders from the domain of finite graphs to the domain of matrices. An *equitable partition* of a connected graph G is a partition of its vertex set in blocks $B_1, \ldots, B_k$ such that each vertex in each B_i has the same number $m_{i,j}$ of neighbors in B_j, and we call the $k \times k$ matrix $M = (m_{i,j})_{1 \le i,j \le k}$ a *degree matrix* of G. We say that a vertex u is of the *i-th sort* if $u \in B_i$. Equitable partitions are well-known in algebraic graph theory, see e.g. [15]. Note that the degree refinement matrix of G is the degree matrix corresponding to the equitable partition of G with the smallest number of blocks (which are ordered in a unique way), and an adjacency matrix of G can be seen as a degree matrix with the maximum number of rows.

Let $\mathcal{M}$ be the set of all degree matrices. We define three relations $(\mathcal{M}, \xrightarrow{\exists B})$, $(\mathcal{M}, \xrightarrow{\exists I})$ and $(\mathcal{M}, \xrightarrow{\exists S})$ imposed on the set of degree matrices by the existence of graph homomorphisms of the corresponding local constraint, i.e., $M \xrightarrow{\exists *} N$ if and only if there exist two graphs $G, H \in \mathcal{C}$ with degree matrix M, N, respectively, such that $G \xrightarrow{*} H$. All three relations are partial orders [14], and a successful matrix comparison of each type is a necessary condition for the corresponding graph comparison.

Secondly, we can transform the partial orders from the domain of finite graphs to the domain of possibly infinite trees. The *universal cover* T_G of a connected graph G is the only tree that allows a locally bijective homomorphism $T_G \xrightarrow{B} G$. A generic construction of the universal cover takes as vertices of T_G all finite walks in G that start from an arbitrary fixed vertex in G and that does not traverse the same edge in two consecutive steps. Two such vertices are adjacent in T_G if the associated walks differ only in the presence of the last edge. The required homomorphism $T_G \xrightarrow{B} G$ can be taken as the mapping that assigns every walk its last vertex. One can easily see that the universal cover is unique up to an isomorphism (in particular, if we take walks that start in another fixed vertex). As a matter of fact, if two subtrees of a universal cover rooted at two different vertices are isomorphic to depth $n-1$, then they are isomorphic to all depths [25]. Universal covers are also called infinite unfoldings or views of graphs and have applications in finite automata theory[23], distributed computing [18,27] and existential pebble games [7].

Also universal covers can be equipped with a structure that impose a necessary condition for the existence of a locally constrained homomorphism. There are two options: either the existence of a locally constrained homomorphism or a simple inclusion (as a subtree). In the latter case, $T_G = T_H$, $T_G \subseteq T_H$, and $T_G \supseteq T_H$ are necessary conditions for $G \xrightarrow{B} H$, $G \xrightarrow{I} H$ and $G \xrightarrow{S} H$, respectively, see [14] for more details.

Moreover, a result in [14] states that the universal cover T_G is equal to the *universal cover* T_M of any degree matrix M of G which is constructed in the following way. We take as root a vertex corresponding to row 1 of M, thus of the 1st sort, and inductively adding a new level of vertices while maintaining the property that each vertex of the i-th sort has exactly $m_{i,j}$ neighbors of the j-th sort. Hence, a successful universal cover comparison is a necessary condition for the corresponding graph comparison as well. More precisely, we have shown the forward implications in the following theorem.

Theorem 1. *Let G and H be connected graphs with degree matrices M and N, resp. Then the following holds:*

$$\begin{array}{c}
G \xrightarrow{B} H \Longrightarrow M \xrightarrow{\exists B} N \Longleftrightarrow T_G \xrightarrow{B} T_H \Longleftrightarrow T_G = T_H \\
G \xrightarrow{I} H \Longrightarrow M \xrightarrow{\exists I} N \Longrightarrow T_G \xrightarrow{I} T_H \Longleftrightarrow T_G \subseteq T_H \\
G \xrightarrow{S} H \Longrightarrow M \xrightarrow{\exists S} N \Longrightarrow T_G \xrightarrow{S} T_H \Longrightarrow T_G \supseteq T_H
\end{array}$$

The backward implications in Theorem 1 for locally bijective homomorphism are consequences of the theorem of Leighton [21]. The equivalence $T_G \xrightarrow{I} T_H \Longleftrightarrow T_G \subseteq T_H$ follows form the fact that a locally injective homomorphisms between two trees is indeed globally injective [14].

Observe that $C_4 \overset{*}{\not\to} C_3$ while both graphs allow the 1×1 degree matrix $M = (2)$. This example excludes the implication $G \xrightarrow{*} H \Longleftarrow M \xrightarrow{\exists *} N$ for $* = B, I, S$. If G itself is a tree then $T_G = G$. We then find that $T_G \xrightarrow{S} T_H \not\Longleftarrow T_G \supseteq T_H$ for the choice $G = P_4$, $H = P_3$, since $P_4 \supseteq P_3$ but $P_4 \overset{S}{\not\to} P_3$. This example shows that the relations $T_G \xrightarrow{S} T_H$ and $T_G \supseteq T_H$ are different. By using linear programming techniques, the backward implication $M \xrightarrow{\exists I} N \Longleftarrow T_M \xrightarrow{I} T_N$ can be excluded [14]. So, the inclusion of universal covers does not imply the relation on matrices for the locally injective constraint. What about the remaining backward implication?

Question 1. Does there exist a counter example for the backward implication $M \xrightarrow{\exists S} N \Longleftarrow T_G \xrightarrow{S} T_H$ in Theorem 1?

The problem of deciding $G \xrightarrow{*} H$ is NP-complete for all three local constraints, and remains NP-hard for many particular fixed targets H, as we mentioned earlier on. We have shown that $M \xrightarrow{\exists B} N$ can be verified in polynomial time, but so far only membership to the class NP could be shown for the matrix comparison problem $M \xrightarrow{\exists *} N$ for $* = I, S$ [14]. It is not expected that a polynomial algorithm would solve these two problems. Testing if $T_G = T_H$ can be done in polynomial time by checking if G and H share the same the degree refinement matrix [2]. Especially given the above, it would be useful to have a polynomial heuristic for checking the other universal problem comparisons as well.

Question 2. How hard is it to decide if $T_G \xrightarrow{I} T_H$ (or equivalently $T_G \subseteq T_H$) holds and to decide if $T_G \xrightarrow{S} T_H$ holds for two given connected graphs G and H?

In this paper we answer Question 1 in Section 2 as well as Question 2 in Section 3.

2 Excluding the Remaining Implication

We show that the relation $T_M \xrightarrow{S} T_N$ lies strictly between $M \xrightarrow{\exists S} N$ and $T_M \supseteq T_N$.

Proposition 1. *For degree matrices*

$$M = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \text{and} \qquad N = \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix}$$

it holds that $T_M \xrightarrow{S} T_N$ but $M \not\xrightarrow{\exists S} N$.

Proof. Observe first that N is a matrix of a finite tree T_N and no other connected simple graph allows this degree matrix. The infinite tree T_M consist of pairwise disjoint paths that are of infinite length and induced by vertices of the first sort (white vertices). These paths are linked by vertices of the second sort (each is adjacent to three paths) and every vertex of the second sort is joined to the middle vertex of a unique P_3. The trees T_M and T_N together with a homomorphism witnessing $T_M \xrightarrow{S} T_N$ are depicted in Fig. 1.

This homomorphism is obtained inductively. We first map one infinite white path into T_N such that the sorts of the images alternate. Every vertex u of the

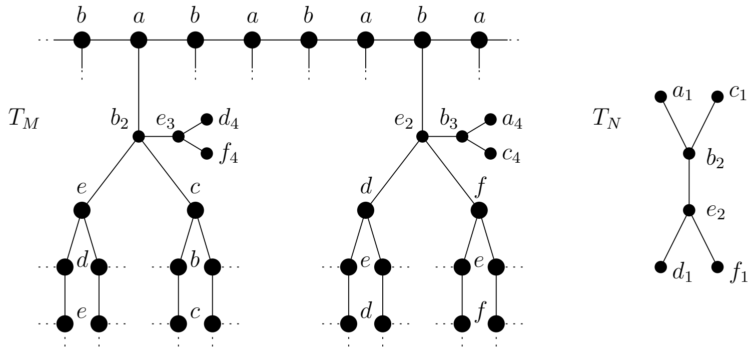


Fig. 1. Showing $T_M \xrightarrow{S} T_N$. White vertices in T_M are of the 1st sort. Sorts of the remaining vertices are indicated by subscripts.

second sort in T_M must be mapped on a vertex of the second sort in T_N so that the homomorphism can be extended to the pending claw.

Then, depending of whether the image of the already processed neighbor of u was of the first or of the second sort, we extend the mapping to the two infinite white paths that contains the remaining two neighbors of u. Both cases are depicted in Fig. 1.

Now, in order to obtain a contradiction, assume that a finite graph G with degree matrix M and a mapping $f : G \xrightarrow{S} T_N$ exists (recall that the target graph T_N is unique for this choice of N). Consider the vertices of the first sort of G, call them red. These red vertices induce a disjoint union of cycles in G.

Denote by a the number of red vertices u such that $f(u)$ is of the first sort in T_N and call them light-red. Analogously, let b be the number of red vertices v such that $f(v)$ is of the second sort, and call them dark-red. Since N prescribes that both red neighbors of every light-red u must be dark we have $a \leq b$.

On the other hand, due to the pending claws (which also exist in G), every vertex of the third sort in G is mapped to a vertex of the second sort in T_N, and every vertex of the fourth sort in G is mapped to a vertex of the first sort in T_N. Then every vertex u' of the second sort in G is mapped to a vertex of the second sort in T_N. Since already its neighbor of the third sort is mapped to a vertex of the second sort in T_N, u' must have at least two light-red neighbors and, consequently, at most one dark-red neighbor. Hence, $a \geq 2b$ which is in contradiction with $a \leq b$. We conclude that $M \not\xrightarrow{\exists S} N$. □

3 Testing Locally Injective and Surjective Homomorphisms between Universal Covers

In this section we focus on the decision problems whether $T_M \xrightarrow{*} T_N$ holds for local constraints $* = I, S$. As the algorithms are almost the same for both constraints, we treat both cases simultaneously, pointing only at the differences where the particular local constraint plays different role. We first need some new terminology. For an integer $k \geq 1$ we define $[k] := \{1, 2, \ldots, k\}$ and abbreviate $[k] \times [l]$ by $[k \times l]$.

Definition 1. *Let M and N be two degree matrices of order k and l, resp. We say that a vector $\mathbf{p}^{r,s}$ consisting of kl nonnegative integers is a* distribution row *for indices $(r, s) \in [k \times l]$ if the condition 1 holds. A distribution row $\mathbf{p}^{r,s}$ is called* injective *if in addition condition 2 holds. It is called* surjective *if in addition conditions 3 and 4 hold.*

$$\sum_{j=1}^{l} p_{i,j}^{r,s} = m_{r,i} \qquad \text{for all } i \in [k], \tag{1}$$

$$\sum_{i=1}^{k} p_{i,j}^{r,s} \leq n_{s,j} \qquad \text{for all } j \in [l], \tag{2}$$

$$n_{s,j} \geq 1 \qquad \Longrightarrow \qquad \sum_{i=1}^{k} p_{i,j}^{r,s} \geq n_{s,j} \qquad \text{for all } j \in [l], \tag{3}$$

$$n_{s,j} = 0 \qquad \Longrightarrow \qquad \sum_{i=1}^{k} p_{i,j}^{r,s} = 0 \qquad \text{for all } j \in [l]. \tag{4}$$

As an example, consider the matrices M and N from Proposition 1. The locally surjective homomorphism from T_M and T_N in Figure 1 defines exactly the following surjective distribution rows $\mathbf{p}^{r,s} = (p_{1,1}^{r,s}, p_{1,2}^{r,s}, p_{2,1}^{r,s}, p_{2,2}^{r,s}, p_{3,1}^{r,s}, p_{3,2}^{r,s}, p_{4,1}^{r,s}, p_{4,2}^{r,s})$:

$$\begin{aligned}
\mathbf{p}^{1,1} &= (0, 2, 0, 1, 0, 0, 0, 0)\\
\mathbf{p}^{1,2} &= (2, 0, 0, 1, 0, 0, 0, 0)\\
\mathbf{p}^{2,2} &= (2, 1, 0, 0, 0, 1, 0, 0)\\
\mathbf{p}^{3,2} &= (0, 0, 0, 1, 0, 0, 2, 0)\\
\mathbf{p}^{4,1} &= (0, 0, 0, 0, 0, 1, 0, 0)
\end{aligned}$$

Distribution rows play a central role in the NP algorithms for the degree matrix comparison problems $M \xrightarrow{\exists I} N$ and $M \xrightarrow{S} N$ [14]. Suppose $G \xrightarrow{*} H$ via f is indeed a witness for $M \xrightarrow{\exists *} N$ for $* \in \{I, S\}$. Let f map $u \in V_G$ of the r-th sort to $v \in V_H$ of the s-th sort, and denote the number of neighbors of the i-th sort in $N_G(u)$ that are mapped to neighbors of the j-th sort in $N_H(v)$ by $p_{i,j}^{r,s}$. Then the vector $\mathbf{p}^{r,s}$ defined by entries $p_{i,j}^{r,s}$ is a (surjective or injective) distribution row that we call suitable. Our NP algorithms try to identify suitable distribution rows. The difficulty is that there may be exponentially many distribution rows. Therefore, these algorithms could only use the nondeterministic choice of suitable distribution rows to verify whether $M \xrightarrow{\exists *} N$ holds for $* = I, S$, respectively, see [14] for more details. However, for the decision problem on the existence of a locally constrained homomorphism between universal covers we prove that we may reduce the number of suitable distribution rows to only a polynomial number. For showing this we need some more terminology. For a degree matrix M we say that matrix rows r and i are *adjacent* if $m_{r,i} > 0$.

Definition 2. *We say that a distribution row* $\mathbf{p}^{r,s}$ *is* a witness of type (s,j) *for (adjacent) matrix rows* r *and* i *if* $\mathbf{p}^{r,s}_{i,j} \geq 1$.

Definition 3. *We say that a distribution row* $\mathbf{p}^{r,s}$ respects the allowed set $X \subseteq [k \times l]$ *if* $\mathbf{p}^{r,s}_{i',j'} \geq 1$ *implies* $(i',j') \in X$ *for all* $(i',j') \in [k \times l]$.

Note that if $\mathbf{p}^{r,s}$ is a witness of type (s,j) for matrix rows r and i that respects an allowed set X then $(i,j) \in X$. We need the following lemma for our algorithms.

Lemma 1. *For given* r *and* i *the existence of an injective or surjective witness* $\mathbf{p}^{r,s}$ *of type* (s,j) *respecting an allowed set* X *can be tested in a polynomial time.*

Proof. We can do this by translating the problem to the integer flow problem. It is well-known [17] that this problem can be solved in polynomial time on flow networks with integer edge capacities (if such a network has a flow, then this flow may be assumed to be integer). We first define our auxiliary flow network F and then explain it afterwards. We let $V_F = \{p, u_{i'}, v_{j'}, q \,|\, (i',j') \in [k \times l]\}$ and $E_F = \{(p,u_{i'}), (u_{i'},v_{j'}), (v_{j'},q) \,|\, (i',j') \in [k \times l]\}$. The sought flow g goes from p to q and must satisfy the following edge constraints:

$$g(p,u_{i'}) = m_{r,i'} \qquad \text{for } * = I: \; g(v_{j'},q) \leq n_{s,j'}$$

$$g(u_{i'},v_{j'}) \begin{cases} \geq 1 & \text{if } (i',j') = (i,j) \\ = 0 & \text{if } (i',j') \notin X \\ \geq 0 & \text{otherwise} \end{cases} \qquad \text{for } * = S: \; g(v_{j'},q) \begin{cases} \geq n_{s,j'} & \text{if } n_{s,j'} \geq 1 \\ = 0 & \text{if } n_{s,j'} = 0 \end{cases}$$

We claim that F has an integer flow g if and only if there exists an injective, or respectively, surjective witness $\mathbf{p}^{r,s}$ of type (s,j) for r and i respecting X. First suppose F allows an integer flow g. Choose $p^{r,s}_{i',j'} = g(u_{i'},v_{j'})$ for all $(i',j') \in [k \times l]$. Because $\sum_{j'=1}^{l} p^{r,s}_{i',j'} = \sum_{j'=1}^{l} g(u_{i'},v_{j'}) = g(p,u_{i'}) = m_{r,i'}$ for all $i' \in [k]$, $\mathbf{p}^{r,s}$ is a distribution row. For all $j' \in [l]$, $\sum_{i'=1}^{k} p^{r,s}_{i',j'} = \sum_{i'=1}^{k} g(u_{i'},v_{j'}) = g(v'_j,q)$, which is at most $n_{s,j'}$ if $* = I$, at least $n_{s,j'}$ if $* = S$ and $n_{s,j'} \geq 1$, and 0 otherwise, $\mathbf{p}^{r,s}$ is injective or surjective, respectively. Since $p^{r,s}_{i,j} = g(u_i,v_j) \geq 1$, $\mathbf{p}^{r,s}$ is a witness. Finally, since $p^{r,s}_{i',j'} = g(u_{i'},v_{j'}) = 0$ for all $(i',j') \notin X$, $\mathbf{p}^{r,s}$ respects X.

Now suppose there exists an injective, or respectively, surjective witness $\mathbf{p}^{r,s}$ of type (s,j) for r and i respecting X. By Definition 2, $p^{r,s}_{i,j} \geq 1$. It is easy to verify that $\mathbf{p}^{r,s}$ satisfies the other edge constraints in F as well. Hence F allows $\mathbf{p}^{r,s}$ as integer flow. □

Our two algorithms can now be presented as one generic iterative algorithm.

Algorithm 1. The test whether $T_M \xrightarrow{*} T_N$ holds for $* = I$ or S

```
Input: Degree matrices $M$ and $N$
Parameter: Local constraint $* \in \{I, S\}$
initialize $X^{r,s} = \{(i,j) \mid m_{r,i} > 0$ and $n_{s,j} > 0\}$ for all $(r,s) \in [k \times l]$;
repeat
    foreach $(r,s) \in [k \times l]$ and $(i,j) \in X^{r,s}$ do
        if $(r,i)$ has no witness of type $(s,j)$ respecting $X^{r,s}$ then
            remove $(i,j)$ from $X^{r,s}$ and remove $(r,s)$ from $X^{i,j}$;
        end
    end
until no removal happens during the whole foreach loop ;
if there exists an $r \in [k]$ with $X^{r,s}$ empty for all $s \in [l]$ then
    return $T_M \not\xrightarrow{*} T_N$
else
    return $T_M \xrightarrow{*} T_N$
end
```

Theorem 2. *Algorithm 1 is correct and runs in polynomial time.*

Proof. For each $X^{r,s}$, one iteration of Algorithm 1 takes polynomial time due to Lemma 1. Since the number of different allowed sets $X^{r,s}$ is kl, a complete iteration, i.e., an iteration over all $X^{r,s}$, then takes polynomial time as well. At the start of the algorithm each $X^{r,s}$ contains at most kl elements, and after each complete iteration the size of each $X^{r,s}$ has never increased. Since the algorithm finishes as soon as all $X^{r,s}$ have stable size, the number of iterations is at most kl. We conclude that Algorithm 1 runs in polynomial time.

We now show that Algorithm 1 is correct. Suppose $T_M \xrightarrow{*} T_N$ via f. Then f induces witnesses of type (s,j) for all adjacent matrix rows r, i such that (r,i) has a witness of type (s,j) if and only if (i,r) has a witness of type (j,s). Hence f defines nonempty sets $X^{r,s}$ for all matrix rows r.

It remains to show that if Algorithm 1 terminates in the affirmative state, then a locally constrained homomorphism $f : T_M \xrightarrow{*} T_N$ can be constructed. Pick an arbitrary vertex $u \in T_M$. Let u be of the r-th sort. By definition of the algorithm, there exists a (final) allowed set $X^{r,s} \neq \emptyset$. Define $f(u) = v$ for any $v \in T_N$ that is of the s-th sort. Choose an arbitrary $(i,j) \in X^{r,s}$. By definition of $X^{r,s}$, we can find a witness $\mathbf{p}^{r,s}$ for (r,i) of type (s,j) respecting $X^{r,s}$.

We use $\mathbf{p}^{r,s}$ to extend f. By definition of $\mathbf{p}^{r,s}$, for every $p^{r,s}_{i',j'} \geq 1$, we can let f map $p^{r,s}_{i',j'}$ different neighbors of u that all are of the i'-th sort onto neighbors of v that all are of the j'-the sort in such a way that, from $N(u)$ to $N(v)$, f is injective when $* = I$, and surjective when $* = S$. Whenever the mapping f is defined along an edge (u,u'), we iteratively extend f to the whole neighborhood $N(u')$ of u' by the same procedure as above in case $N(u') \supsetneq \{u\}$. We only have to make sure to choose a witness $\mathbf{p}^{i',j'}$ for (i',r) of type (j',s), where $u, u', f(u)$ and $f(u')$ are of the r, i', s- and j'-th sort respectively. Then $p^{i',j'}_{r,s} \geq 1$ by definition of a witness, and indeed we can use $\mathbf{p}^{i',j'}$ to extend our mapping f that already maps

$u \in N(u')$ of the r-th sort to $v \in N(f(u'))$ of the s-th sort. The reason why such a witness $\mathbf{p}^{i',j'}$ exists follows from the reciprocal removal of pairs (i', j') from $X^{r,s}$ and (r, s) from $X^{i',j'}$. When the condition of the **repeat** loop is satisfied, it holds that

$$\begin{gathered} (r, i') \textit{ has a witness of type } (s, j') \textit{ respecting } X^{r,s} \\ \text{if and only if} \\ (i', r) \textit{ has a witness of type } (j', s) \textit{ respecting } X^{i',j'}. \end{gathered} \qquad \square$$

We are even able to construct in polynomial time a locally constrained homomorphism $f : T_M \xrightarrow{*} T_N$ if Algorithm 1 approves that $T_M \xrightarrow{*} T_N$. This can be seen as follows. We use the method described in the proof of Theorem 2 to construct f. Finding witnesses respecting certain allowed sets can be done in polynomial time using the flow network of the proof of Lemma 1. If f is defined along edge (u, u') then we always choose for the same extension of f on $N(u')$, i.e., how we extend f only depends on the sort of u and the sort of u'. As it is sufficient to keep only at most kl possibilities, the claim follows.

4 Conclusions

We have answered questions 1 and 2 of Section 1.1 in Proposition 1 and Theorem 2, respectively. We conclude with some other applications.

The H-Role Assignment problem asks whether $G \xrightarrow{S} H$ for a graph G and a fixed target graph H. This problem is NP-complete for all connected graphs H on at least three vertices [13]. It becomes polynomially solvable for every fixed target H when restricted to the class of trees. This follows from Theorem 2, and the fact that $T \not\xrightarrow{S} G$ if T is a tree and G contains a cycle, together with the fact that $T_G = T$ for every tree G. Since $T_G \xrightarrow{I} T_H$ if and only if $T_G \subseteq T_H$, Algorithm 1 tests for infinite subtree isomorphism as well. Since $T_G = T$ for every tree G, it can also be used for (sub-)tree isomorphism for finite trees, especially if these trees can be encoded in terms of degree (refinement) matrices independent of their original size (as otherwise much faster algorithms exist). Finally, we note that there exist matrices that are not the degree matrix of a finite graph. If such a matrix M has the property that $m_{i,j} > 0$ whenever $m_{j,i} > 0$ then it still possible to construct a universal cover T_M of M (or disjoint submatrices of M) in the same way as before. Algorithm 1 can then be used for universal cover comparison of those matrices as well.

Acknowledgments. We thank Jan Arne Telle for fruitful discussions on this topic.

References

1. Abello, J., Fellows, M.R., Stillwell, J.C.: On the complexity and combinatorics of covering finite complexes. Australian Journal of Combinatorics 4, 103–112 (1991)
2. Angluin, D.: Local and global properties in networks of processors. In: Proceedings of the 12th ACM Symposium on Theory of Computing, pp. 82–93 (1980)

3. Biggs, N.: Constructing 5-arc transitive cubic graphs. Journal of London Mathematical Society II 26, 193–200 (1982)
4. Bodlaender, H.L.: The classification of coverings of processor networks. Journal of Parallel Distributed Computing 6, 166–182 (1989)
5. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Macmillan, London, Elsevier, New York (1976)
6. Chalopin, J., Métivier, Y., Zielonka, W.: Local computations in graphs: the case of cellular edge local computations. Fund. Inform. 74, 85–114 (2006)
7. Dantchev, S., Martin, B.D., Stewart, I.A.: On non-definability of unsatisfiability (manuscript)
8. Everett, M.G., Borgatti, S.: Role coloring a graph. Mathematical Social Sciences 21, 183–188 (1991)
9. Fiala, J., Heggernes, P., Kristiansen, P., Telle, J.A.: Generalized H-coloring and H-covering of trees. Nordic Journal of Computing 10, 206–224 (2003)
10. Fiala, J., Kratochvíl, J.: Complexity of partial covers of graphs. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 537–549. Springer, Heidelberg (2001)
11. Fiala, J., Kratochvíl, J.: Partial covers of graphs. Discussiones Mathematicae Graph Theory 22, 89–99 (2002)
12. Fiala, J., Kratochvíl, J., Kloks, T.: Fixed-parameter complexity of λ-labelings. Discrete Applied Mathematics 113, 59–72 (2001)
13. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignment problem. Theoretical Computer Science 349, 67–81 (2005)
14. Fiala, J., Paulusma, D., Telle, J.A.: Locally constrained graph homomorphisms and equitable partitions. European Journal of Combinatorics (to appear)
15. Godsil, C.: Algebraic Combinatorics. Chapman and Hall, Boca Raton (1993)
16. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press, Oxford (2004)
17. Hoffman, A.J., Kruskal, J.B.: Integral boundary points of convex polyhedra. Annals of Mathematics Studies 38, 223–246 (1956)
18. Kranakis, E., Krizanc, D., Van den Berg, J.: Computing boolean functions on anonymous networks. Information and Computation 114, 214–236 (1994)
19. Kratochvíl, J., Proskurowski, A., Telle, J.A.: Covering regular graphs. Journal of Combinatorial Theory B 71, 1–16 (1997)
20. Kristiansen, P., Telle, J.A.: Generalized H-coloring of graphs. In: Lee, D.T., Teng, S.-H. (eds.) ISAAC 2000. LNCS, vol. 1969, pp. 456–466. Springer, Heidelberg (2000)
21. Leighton, F.T.: Finite common coverings of graphs. Journal of Combinatorial Theory B 33, 231–238 (1982)
22. Massey, W.S.: Algebraic Topology: An Introduction. Harcourt (1967)
23. Moore, E.F.: Gedanken-experiments on sequential machines. Annals of Mathematics Studies 34, 129–153 (1956)
24. Nešetřil, J.: Homomorphisms of derivative graphs. Discrete Math. 1, 257–268 (1971)
25. Norris, N.: Universal covers of graphs: isomorphism to depth $n-1$ implies isomorphism to all depths. Discrete Applied Mathematics 56, 61–74 (1995)
26. Roberts, F.S., Sheng, L.: How hard is it to determine if a graph has a 2-role assignment? Networks 37, 67–73 (2001)
27. Yamashita, M., Kameda, T.: Computing on anonymous networks: Part I - Characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems 7, 69–89 (1996)

S4LP and Local Realizability

Melvin Fitting

Lehman College – CUNY
250 Bedford Park Boulevard West
Bronx, NY 10548, USA
melvin.fitting@lehman.cuny.edu

Abstract. The logic S4LP combines the modal logic S4 with the justification logic LP, both axiomatically and semantically. We introduce a simple restriction on the behavior of constants in S4LP, having no effect on the LP sublogic. Under this restriction some powerful derived rules are established. Then these are used to show completeness relative to a semantics having what we call the *local realizability* property: at each world and for each formula true at that world there is a realization also true at that world, where a realization is the result of replacing all modal operators with explicit justification terms. This is a part of a project to understand the deeper aspects of Artemov's Realization Theorem.

1 Introduction

Logics of knowledge, Hintikka style, are familiar tools, [1]. Recently a family of *justification logics* has been created. In these, instead of a modal operator, *known*, there is an infinite family of explicit reasons by which something is known. There are justification logic analogs of several standard single-knower Hintikka style logics, and work proceeds on multiple-knower versions. Connections between Hintikka logics and explicit logics are quite close, via *Realization Theorems*. They say any theorem of a standard Hintikka logic of knowledge can be *realized*, its knowledge operators can be replaced with explicit justifications, to produce a theorem of the corresponding explicit logic of knowledge. Thus the usual knowledge operators carry hidden explicit content.

Justification logics began with an analog of S4, due to Sergei Artemov, [2]. The motivation was to create an arithmetic semantics for propositional intuitionistic logic, completing a project begun by Gödel. Artemov succeeded in this. The justification logic created was called LP, for "logic of proofs;" explicit justifications represent formal arithmetic proofs. It was soon realized that proofs were only one kind of justification, and LP was one of a family of similar logics. In order to keep the discussion manageable here, I will frame it in terms of LP, thinking of it as a representative member of the family but having historical precedence. The work, in fact, applies to a range of logics.

The Realization Theorem connects LP with S4. Each theorem of S4 has a realization—a replacement of modal operators with explicit justification terms that produces a theorem of LP. (The converse is also true, and trivial.) Indeed, a realization can be chosen that is *normal*, negative occurrences of necessity can be replaced with distinct variables. Moreover a realization can be extracted constructively from a proof in S4. Given this fundamental relationship between S4 and LP, it is natural to consider a logic that combines of LP and S4, so that both explicit and implicit notions of knowledge are present. This has been done, it is known as S4LP, [3,4]. Axiomatically, one simply provides the machinery of S4, the machinery of LP, and a connecting axiom saying that explicit knowledge implies implicit knowledge (see Section 3.1). S4LP is a conservative extension of both S4 and LP. The Realization Theorem becomes a result about this single logic, rather than one connecting two different ones. Unfortunately, the only proofs known for the single-logic version of the Realization Theorem detour through the older proofs, via conservativity. (Unfortunately too, this paper does not shed any fresh light on this important issue.)

A Hintikka/Kripke semantics for S4 is standard. In [5,6] a semantics for LP was presented, combining justification logic machinery originating in [7] with the usual S4 semantics. This semantics has been adapted to S4LP in two distinct ways. First, one can use the LP semantics without change, since there is an underlying Kripke structure for the interpretation of the modal operator. Axiomatic soudness is in [3,4] and a completeness theorem is in [8]. The second semantics is the single agent version of an n-agent logic of knowledge with explicit common knowledge, [9,10]. In this, separate accessibility relations are used for the modal operator and for explicit justification terms. Again, soundness and completeness results have been shown. We are not concerned here with the two-accessibility-relation version of S4LP semantics, but only the single accessibility version, as investigated in [3,4,8]. In this, justifications can be thought of as supplying an analysis of an individual's knowledge, and the connection between justifications and the modal/knowledge operator can be expected to be quite close.

We say an S4LP model meets the *local realizability condition* provided, at each possible world of the model, each formula that is true at that world has a realization that is also true at that world (normaliity is not required). The main result of this paper is that axiomatic S4LP is complete with respect to models meeting the local realizability condition, provided a certain condition is placed on the constant specifications allowed. What was called *strong* completeness for LP in [6] is an easy corollary.

I want to thank Sergei Artemov for comments on an earlier draft of this paper.

2 The Logic LP

We begin with a sketch of the oldest justification logic, LP, from [2]. First, the language and an axiom system, and then a standard semantics.

2.1 LP Axiomatically

Justification terms or *proof terms* are built up from *variables*, x_1, x_2, ..., and *constant symbols*, c_1, c_2, ... They are built up using the following *operation* *symbols*: $+$ and $\cdot$, both binary infix, and $!$, unary prefix. The reader is referred to [2] and to [11] for a discussion of the intended meaning of these.

Formulas are built up from *propositional letters*, P_1, P_2, $\ldots$, and a *falsehood constant*, $\bot$, using $\supset$, together with an additional rule of formation, $t{:}X$ is a formula if t is a justification term and X is a formula. Read it as "t is a justification for X." Other propositional connectives are introduced as abbreviations.

Axiom (schemes) for LP, and rules, are as follows.

Classical Axioms:	all tautologies
Truth Axioms:	$t{:}X \supset X$
$+$ Axioms:	$t{:}X \supset (t+u){:}X$
	$u{:}X \supset (t+u){:}X$
$\cdot$ Axioms:	$t{:}(X \supset Y) \supset (u{:}X \supset (t \cdot u){:}Y)$
! Axioms:	$t{:}X \supset !t{:}t{:}X$

$$\text{Modus Ponens: } \frac{X \qquad X \supset Y}{Y}$$

$$\text{Axiom Necessitation: If } X \text{ is an axiom and } c \text{ is a constant: } \frac{}{c{:}X}$$

A *constant specification* $\mathcal{C}$ is an assignment of axioms to constants; take it to be a set of formulas of the form $c{:}X$, where c is a constant and X is an axiom. A proof *meets constant specification* $\mathcal{C}$ provided that whenever $c{:}X$ is introduced using the Axiom Necessitation rule, then X is an axiom that $\mathcal{C}$ assigns to constant c. A constant specification can be given ahead of time, or can be created during the course of a proof. In this paper we will assume a constant specification has been fixed ahead of time. Various conditions can be imposed on constant specifications. A constant specification is *axiomatically appropriate* if all instances of axiom schemes have proof constants—here *this will always be assumed.* Another common condition is being *injective*: at most one formula is associated with each constant. We will need a condition, given in Section 4, that conflicts with injectivity, but which is nonetheless natural to consider.

2.2 LP Semantics

The usual semantics for LP comes from [6], and amounts to a blending of an earlier semantics from [7] with the usual Hintikka semantics for logics of knowledge. A model is $\mathcal{M} = \langle \mathcal{G}, \mathcal{R}, \mathcal{A}, \mathcal{V} \rangle$, where $\langle \mathcal{G}, \mathcal{R} \rangle$ is a frame, with $\mathcal{R}$ a reflexive and transitive relation on $\mathcal{G}$. $\mathcal{V}$ maps propositional variables to subsets of $\mathcal{G}$. The item not standard in Kripke models is $\mathcal{A}$, an *admissible evidence function.* For each justification term t and formula X, $\mathcal{A}(t, X)$ is some subset of $\mathcal{G}$. Intuitively, $\mathcal{A}(t, X)$ is the set of worlds at which t is admissible evidence for X. This does not mean conclusive evidence—just evidence that is relevant. Admissible evidence functions must meet certain conditions which we give next. (In earlier work a related mapping $\mathcal{E}$, called an evidence function, was used in place of $\mathcal{A}$. The change in notation is essentially cosmetic.)

Constant Specification Condition. For the given constant specification $\mathcal{C}$, if $c{:}X \in \mathcal{C}$ then $\mathcal{A}(c, X) = \mathcal{G}$. If this condition is met, we say $\mathcal{A}$ *meets constant specification* $\mathcal{C}$.
$\cdot$ **Condition.** $\mathcal{A}(s, X \supset Y) \cap \mathcal{A}(t, X) \subseteq \mathcal{A}(s \cdot t, Y)$.
$+$ **Condition.** $\mathcal{A}(s, X) \cup \mathcal{A}(t, X) \subseteq \mathcal{A}(s + t, X)$.
$\mathcal{R}$ **Closure Condition.** $\Gamma\mathcal{R}\Delta$ and $\Gamma \in \mathcal{A}(t, X)$ imply $\Delta \in \mathcal{A}(t, X)$.
! **Condition.** $\mathcal{A}(t, X) \subseteq \mathcal{A}(!t, t{:}X)$.

Let $\mathcal{M} = \langle \mathcal{G}, \mathcal{R}, \mathcal{A}, \mathcal{V} \rangle$ be an LP model. $\mathcal{M}, \Gamma \Vdash X$ is read: formula X is true at world $\Gamma \in \mathcal{G}$, of LP model $\mathcal{M}$. The conditions for it are as follows.

Atomic Condition. For a propositional letter P, $\mathcal{M}, \Gamma \Vdash P$ if $\Gamma \in \mathcal{V}(P)$.
Classical Conditions. $\mathcal{M}, \Gamma \Vdash X \supset Y$ iff $\mathcal{M}, \Gamma \not\Vdash X$ or $\mathcal{M}, \Gamma \Vdash Y$. Also $\mathcal{M}, \Gamma \not\Vdash \bot$.
Justification Condition. $\mathcal{M}, \Gamma \Vdash t{:}X$ iff $\Gamma \in \mathcal{A}(t, X)$ and $\mathcal{M}, \Delta \Vdash X$ for all $\Delta \in \mathcal{G}$ with $\Gamma\mathcal{R}\Delta$.

We say X is *true at world* Γ if $\mathcal{M}, \Gamma \Vdash X$, and otherwise X is *false at* Γ. X is *valid* in a model $\mathcal{M}$ if X is true at every world of it.

The Justification Condition says we have $t{:}X$ at Γ if X is knowable at Γ in the Hintikka sense, and t is admissible evidence for X at Γ. If Hintikka semantics captures *true belief*, then the present machinery captures is *justified* true belief.

The semantics just given is the *weak model semantics*; there is a stronger version. A model $\mathcal{M}$ is *fully explanatory* provided, if $\mathcal{M}, \Delta \Vdash X$ for all $\Delta \in \mathcal{G}$ with $\Gamma\mathcal{R}\Delta$ then there is some justification t such that $\mathcal{M}, \Gamma \Vdash t{:}X$. That is, $\mathcal{M}$ is fully explanatory provided Hintikka-knowability of X at Γ implies there is a justification for X at Γ. Fully explanatory is examined in Section 6.

In [6] soundness and completeness was shown. If $\mathcal{C}$ is an axiomatically appropriate constant specification, then X has an axiomatic proof using $\mathcal{C}$ if and only if X is valid in every weak LP model meeting $\mathcal{C}$ if and only if X is valid in every fully explanatory LP model meeting $\mathcal{C}$. Being fully explanatory is an interesting condition that has not yet found applications. This is a puzzling circumstance, which the results of this paper will only make more puzzling.

3 The Logic S4LP

LP and S4 are connected intimately via the Realization Theorem, as noted in the Introduction. So it is natural to consider a logic combining the two, S4LP, originating in [3,4].

3.1 S4LP Axioms

First, the language of LP is extended with the formation rule: if X is a formula, so is $\Box X$. Next, the axiomatization of LP as given in Section 2.1 is extended with S4 machinery, and a connecting axiom.

$\Box$ Axioms: $\Box X \supset X$

$\Box(X \supset Y) \supset (\Box X \supset \Box Y)$

$\Box X \supset \Box\Box X$

Connecting Axiom: $t{:}X \supset \Box X$

The usual necessitation rule is added.

$$\Box \text{ Necessitation: } \frac{X}{\Box X}$$

Finally, it is assumed that the LP Axiom Necessitation rule also applies to the new axioms just added, and that constant specifications also take these new axioms into account.

$\Box$ Necessitation can be shown to be a redundant rule, but doing so involves proving an Internalization Theorem, whose statement and proof we skip here.

3.2 An S4LP Semantics

As was noted in the Introduction, there are two semantics for S4LP, with different motivations. One, from [9,10], allows not just one but multiple agents, each with its own knowledge operator, K_i, but with justifications meaningful to all and playing the role of justified common knowledge. In this semantics one has multiple accessibility relations, one for each agent, and one for justification terms. If there is a single agent the logic reduces to S4LP; there are two relations, one Hintikka style to supply an interpretation for $\Box$, the other is combined with an admissible evidence function as in Section 2.2. Both weak and strong completeness theorems are provable. However, this is not the semantics that will concern us here, and it will not be mentioned further.

The semantics examined in this paper understands knowledge as having an explicit (justification term) aspect and an implicit (modal) aspect. Justification terms provide an analysis of our knowledge, rather than being the items of knowledge we share with other agents. This approach is from [3,4,8]. There is a *single* accessibility relation for both implicit and explicit knowledge. Now details.

First, LP models are fundamental. These are as in Section 2.2. This provides the semantics for justification terms. But since we are using an extended language now, we can also adopt the following, familiar from modal logic.

Necessitation Condition. $\mathcal{M}, \Gamma \Vdash \Box X$ iff $\mathcal{M}, \Delta \Vdash X$ for all $\Delta \in \mathcal{G}$ with $\Gamma \mathcal{R} \Delta$.

In other words, justification terms are interpreted using the accessibility relation and the admissible evidence function, while the modal operator uses the accessibility relation but does not take the admissible evidence function into account. The Justification Condition now can be given a somewhat simpler expression: $\mathcal{M}, \Gamma \Vdash t{:}X$ iff $\Gamma \in \mathcal{A}(t, X)$ and $\mathcal{M}, \Gamma \Vdash \Box X$.

As with LP, we can introduce notions of weak and strong models, but now the Fully Explanatory condition is simpler to state: for each Γ and for each

X, if $\mathcal{M}, \Gamma \Vdash \Box X$ then for some justification term t we have $\mathcal{M}, \Gamma \Vdash t{:}X$. Equivalently, if $\mathcal{M}, \Gamma \Vdash \Box X$ then there is some t such that $\Gamma \in \mathcal{A}(t, X)$.

Axiomatic soundness comes from [3,4]. Completeness of S4LP with respect to the weak model semantics for S4LP is in [8]. Completeness with respect to the strong model semantics, with models satisfying the fully explanatory condition, is an open problem. Here it will be a special case of a more general result, but the general result requires a special condition on constant specifications.

4 Some Derived S4LP Rules

The rules presented here will be used in our S4LP completeness proof in the next section. But we must impose a restriction on constant specifications that is at odds with injectivity. We begin with the replacement of terms containing a variable with a $\Box$ operator.

Definition 1. *Let Z be a formula, and let x be a variable. $Z(x/\Box)$ is the result of replacing every justification term in Z that contains x with $\Box$:*

$$\begin{aligned} A(x/\Box) &= A \text{ for } A \text{ atomic} \\ [X \supset Y](x/\Box) &= [X(x/\Box) \supset Y(x/\Box)] \\ [\Box X](x/\Box) &= \Box[X(x/\Box)] \\ [t{:}X](x/\Box) &= \begin{cases} t{:}[X(x/\Box)] & \text{if } x \text{ does not occur in } t \\ \Box[X(x/\Box)] & \text{if } x \text{ occurs in } t \end{cases} \end{aligned}$$

If we replace justification terms containing variable x with $\Box$ this turns axioms into axioms except for the $\cdot$ Axiom, where the results are theorems but not axioms. *We now modify the formulation of* S4LP *by adding these theorems to our axiom list.* This affects the role of constants in applications of the Axiom Necessitation Rule, and keeps things simpler than they otherwise would be.

New Axioms. From now on our axiomatization of S4LP also contains the following two schemes: $t{:}(X \supset Y) \supset (\Box X \supset \Box Y)$ and $\Box(X \supset Y) \supset (u{:}X \supset \Box Y)$

Lemma 1. *If Z is an axiom of* S4LP*, so is $Z(x/\Box)$ for every variable x.*

Proof. We give one case as an example. Consider the axiom $t{:}X \supset (t + u){:}X$. There are three subcases.

1. x does not occur in either t or u. Then $[t{:}X \supset (t+u){:}X](x/\Box) = [t{:}X(x/\Box) \supset (t + u){:}X(x/\Box)]$, which is also a + axiom.
2. x occurs in u but not in t. Then $[t{:}X \supset (t + u){:}X](x/\Box) = [t{:}X(x/\Box) \supset \Box X(x/\Box)]$, which is a connecting axiom.
3. x occurs in t. Then $[t{:}X \supset (t + u){:}X](x/\Box) = [\Box X(x/\Box) \supset \Box X(x/\Box)]$, a classical axiom.

Definition 2 ($\Box$ Closed). *Let $\mathcal{C}$ be a constant specification for* S4LP*. $\mathcal{C}$ is $\Box$* closed *provided, whenever $c{:}Z \in \mathcal{C}$ then also $c{:}Z(x/\Box) \in \mathcal{C}$, for each variable x.*

Note that a constant specification that is $\Box$ closed cannot be injective (though it may be when restricted to LP formulas, not containing $\Box$). Here is the main result concerning $\Box$ closed constant specifications.

Theorem 1. *Suppose the constant specification $\mathcal{C}$ is $\Box$ closed. If Z is provable in* S4LP *using constant specification $\mathcal{C}$, so is $Z(x/\Box)$, for each variable x.*

Proof. By induction on axiomatic proof length. Lemma 1 takes care of axioms. Modus ponens and $\Box$ Necessitation are straightforward. Axiom Necessitation is covered by the assumption that the constant specification is $\Box$ closed.

If we do not assume the constant specification is $\Box$ closed, it is still true that if Z is provable in S4LP so is $Z(x/\Box)$, but using a modified constant specification.

Corollary 1. *Assuming a constant specification that is $\Box$ closed, the following is a derived rule of* S4LP*. If x does not occur in A or B:*

$$\frac{x{:}A \supset B}{\Box A \supset B}$$

This corollary points out a similarity in behavior between the $\Box$ operator in S4LP and the existential quantifier in first-order logic. This similarity has been an important motivating factor in the development of justification logics.

5 Completeness

We begin with a definition of local realizability and a statement of the completeness theorem. The completeness proof has similarities with one often used for first-order modal logic with the Barcan formula.

Definition 3. *For a formula X of* S4LP*, a* realization *is a formula X' whose structure is like that of X, but in which every occurrence of $\Box$ has been replaced with a justification term:*

$$\begin{gathered}
A \text{ realizes } A \text{ if } A \text{ is atomic}\\
X' \supset Y' \text{ realizes } X \supset Y \text{ if } X' \text{ realizes } X \text{ and } Y' \text{ realizes } Y\\
t{:}X' \text{ realizes } t{:}X \text{ if } X' \text{ realizes } X\\
t{:}X' \text{ realizes } \Box X \text{ if } t \text{ is a term and } X' \text{ realizes } X
\end{gathered}$$

This extends the usual notion of LP realization. Ordinarily it is S4 formulas that are realized, while here we include a case covering justification terms which are not part of the language of S4. We are not considering *normal* realizations.

Definition 4. *Let $\mathcal{M} = \langle \mathcal{G}, \mathcal{R}, \mathcal{A}, \mathcal{V} \rangle$ be an* S4LP *model. $\mathcal{M}$* meets the local realizability condition *provided: for every world $\Gamma \in \mathcal{G}$ and for every formula X, if $\mathcal{M}, \Gamma \Vdash X$ then there is some realization X' of X such that $\mathcal{M}, \Gamma \Vdash X'$.*

Theorem 2 (Completeness). *Let $\mathcal{C}$ be a constant specification that is $\Box$ closed (Definition 2). If Z_0 is not provable using $\mathcal{C}$ then Z_0 is false at some world of an* S4LP *model meeting $\mathcal{C}$ and* meeting the local realizability condition.

The rest of this section contains the proof of Completeness. *$\mathcal{C}$ is a $\Box$ closed constant specification fixed for the sectio*n (the first few results don't use $\Box$ closure). For a set S of formulas and a formula X, we write $S \vdash X$ if there is some finite subset $\{Y_1, \ldots, Y_n\}$ of S such that $(Y_1 \wedge \ldots \wedge Y_n) \supset X$ is a theorem of S4LP using $\mathcal{C}$. With this definition the deduction theorem and compactness are immediate. We say S is *inconsistent* if $S \vdash \bot$, and *consistent* if it is not inconsistent. *Maximal* consistency has its usual meaning.

Definition 5. *Let X be a formula, and assume an occurrence of $\Box$ in X has been designated. Let t be a justification term. By $X(t)$ we mean the result of replacing the designated occurrence of $\Box$ in X with t.*

Say X is $\Box(P \supset x{:}\dot{\Box}Q) \supset \Box R$ and the designated occurrence of $\Box$ is marked with a dot. Then $X(t) = \Box(P \supset x{:}t{:}Q) \supset \Box R$. The notation $X(t)$ is incomplete since which occurrence of $\Box$ in X is designated is understood, and is not represented in the notation itself.

Definition 6. *A set S of formulas has the $\Box$* instantiation property *provided, for every formula X with a designated occurrence of $\Box$, if $S \cup \{X\}$ is consistent then there is some term t such that $S \cup \{X(t)\}$ is also consistent.*

In canonical models, maximally consistent sets are possible worlds. We will require the $\Box$ instantiation property too. For maximally consistent sets the $\Box$ instantiation property becomes: if $X \in S$ then there is some t such that $X(t) \in S$. This is what we need, but the more general version comes in along the way.

Definition 7. *Let S be a set of formulas. $S^\sharp$ is $\{X \mid \Box X \in S\}$.*

Proposition 1. *Suppose S is a maximally consistent set of formulas that has the $\Box$ instantiation property. Then $S^\sharp$ also has the $\Box$ instantiation property.*

Proof. Assume the hypothesis. Let X be a formula with a designated occurrence of $\Box$. Suppose $S^\sharp \cup \{X\}$ is consistent. We show that for some t, $S^\sharp \cup \{X(t)\}$ is consistent. For convenience we use the defined modal operator $\Diamond$.

Since $S^\sharp \cup \{X\}$ is consistent, so is $S \cup \{\Diamond X\}$ by the following argument. If it were not consistent, $S, \Diamond X \vdash \bot$, and so $S \vdash (\Diamond X \supset \bot)$, that is, $S \vdash \Box\neg X$. Since S is maximal, $\Box\neg X \in S$, hence $\neg X \in S^\sharp$, so $S^\sharp \cup \{X\}$ is not consistent.

Since S has the $\Box$ instantiation property, for some justification term t, $S \cup \{\Diamond X(t)\}$ is consistent.

Since $S \cup \{\Diamond X(t)\}$ is consistent, so is $S^\sharp \cup \{X(t)\}$, which finishes the argument. Again we have a proof by contradiction. Suppose $S^\sharp \cup \{X(t)\}$ is not consistent. Then $S^\sharp, X(t) \vdash \bot$, so $S^\sharp \vdash \neg X(t)$. Then for some $Y_1, \ldots, Y_n \in S^\sharp$, the formula $(Y_1 \wedge \ldots \wedge Y_n) \supset \neg X(t)$ is provable. Using the Rule of Necessitation and standard modal theorems, $(\Box Y_1 \wedge \ldots \wedge \Box Y_n) \supset \Box\neg X(t)$ is provable. Since $Y_1, \ldots, Y_n \in S^\sharp$, we must have $\Box Y_1, \ldots, \Box Y_n \in S$, and since S is maximally consistent, we must also have $\Box\neg X(t) \in S$, contradicting the fact that $S \cup \{\neg\Box\neg X(t)\}$ is consistent.

Next we address the problem of extending a set that has the $\Box$ instantiation property to a maximally consistent set that still has this property.

Lemma 2. *Let S be a set of formulas with the $\Box$ instantiation property, and F be a finite set of formulas. If $S \cup F$ is consistent then $S \cup F$ also has the $\Box$ instantiation property.*

Proof. Let X be a formula with a designated occurrence of $\Box$. Assume $S \cup F \cup \{X\}$ is consistent. We show that for some t, $S \cup F \cup \{X(t)\}$ is also consistent. The argument is very simple.

Say $F = \{Y_1, \ldots, Y_n\}$. Then $S \cup \{Y_1 \wedge \ldots \wedge Y_n \wedge X\}$ is consistent. Since S has the $\Box$ instantiation property for some t, $S \cup \{Y_1 \wedge \ldots \wedge Y_n \wedge X(t)\}$ is consistent, using the original designated occurrence of $\Box$ in X. But this implies that $S \cup F \cup \{X(t)\}$ is consistent.

Proposition 2. *If S is a consistent set of formulas that has the $\Box$ instantiation property, then S can be extended to a set that is maximally consistent and has the $\Box$ instantiation property.*

Proof. Assume S is consistent and has the $\Box$ instantiation property. We extend S using a modified Lindenbaum construction. Enumerate all formulas , say X_0, X_1, ... Then define a sequence of sets of formulas, S_0, S_1, ..., in which each set extends its predecessor, is consistent, and has the $\Box$ instantiation property.

To start, $S_0 = S$.

Suppose S_n has been defined, is consistent, and has the $\Box$ instantiation property. If $S_n \cup \{X_n\}$ is not consistent, set $S_{n+1} = S_n$. Otherwise, proceed as follows. X_n has a finite number of $\Box$ occurrences, say k of them. Choose one of them as designated. $S_n \cup \{X_n\}$ is consistent and, by Lemma 2, it has the $\Box$ instantiation property. Then, using the designated occurrence of $\Box$, there must be some justification term t such that $S_n \cup \{X_n, X_n^1\}$ is consistent, where $X_n^1 = X_n(t)$. By Lemma 2 this set too has the $\Box$ instantiation property. Now repeat this with a different designated occurrence of $\Box$ in X_n, getting a set $S_n \cup \{X_n, X_n^1, X_n^2\}$, consistent and with the $\Box$ instantiation property. And so on for each of the k occurrences of $\Box$ in X_n. Let $S_{n+1} = S_n \cup \{X_n, X_n^1, X_n^2, \ldots, X_n^k\}$.

Let $S = S_0 \cup S_1 \cup S_2 \cup \ldots$. As usual, S is maximally consistent. But also it has the $\Box$ instantiation property by the following argument. Suppose X is a formula with a designated occurrence of $\Box$, and suppose $S \cup \{X\}$ is consistent. Say $X = X_n$. Then $S \cup \{X_n\}$ is consistent and so at stage n of the construction above, not only is $X_n = X$ in S_{n+1}, but also $X(t)$ is in S_{n+1} for some justification term t, and hence $X(t)$ is in S, so trivially $S \cup \{X(t)\}$ is consistent.

All results so far say: some set has the $\Box$ instantiation property provided some other set does. We do not yet know there are any such sets at all (except for inconsistent ones). This is taken care of by the following Lemma and Proposition.

Lemma 3. *Let F be a consistent finite set of formulas, and let X be a single formula with a designated occurrence of $\Box$. Then $F \cup \{X \supset X(x)\}$ is consistent, where x is a variable that does not occur in F or in X.*

Proof. Assume the hypothesis, but also assume $F \cup \{X \supset X(x)\}$ is not consistent, where x does not occur in F or in X. Then $[\bigwedge F \wedge (X \supset X(x))] \supset \bot$ is provable. By Theorem 1 we also have provability of $\{[\bigwedge F \wedge (X \supset X(x))] \supset \bot\}(x/\Box)$, but this is just $[\bigwedge F \wedge (X \supset X)] \supset \bot$, and it follows that F is not consistent.

Proposition 3. *If F is a finite, consistent set then F extends to a consistent set with the $\Box$ instantiation property.*

Proof. Enumerate the formulas, X_0, X_1, X_2, ... We define a chain F_0, F_1, F_2, ... of consistent finite sets, as follows.

$F_0 = F$.

Assume F_n has been defined. Consider formula X_n. Choose an occurrence of $\Box$ in X_n and take it to be designated (if there are none, this step is vacuous). Let x be a variable that does not occur in the finite set F_n or in X_n. Then $F_n \cup \{X_n \supset X_n(x)\}$ is consistent, by Lemma 3. Extend this set by consistently adding an instantiation implication for a different designated occurrence of $\Box$ in X_n, and so on until each occurrence has had a corresponding implication added. Call the resulting set F_{n+1}. Clearly it is finite and consistent.

Set F^* to be $\cup_n F_n$. Then F^* is consistent, has the $\Box$ instantiation property.

Completeness for LP was proved in [6], and that proof was extended to S4LP, as usually formulated, in [8]. Now we modify that proof to establish our main result, which we restate here for convenience.

Theorem 2. Let $\mathcal{C}$ be a constant specification that is $\Box$ closed (Definition 2). If Z_0 is not provable using $\mathcal{C}$ then Z_0 is false at some world of an S4LP model meeting $\mathcal{C}$ and *meeting the local realizability condition.*

Proof. Since Z_0 is not provable, $\{\neg Z_0\}$ is consistent. Using Proposition 3, this set extends to a set that has the $\Box$ instantiation property, and by Proposition 2, this further extends to a set that is maximally consistent and has the $\Box$ instantiation property. Call this set Γ_0.

Construct a model as follows. Let $\mathcal{G}$ be the set of all maximally consistent sets of formulas that have the $\Box$ instantiation property. (Note that $\Gamma_0 \in \mathcal{G}$.) If $\Gamma \in \mathcal{G}$, let $\Gamma^\sharp = \{X \mid \Box X \in \Gamma\}$, and set $\Gamma \mathcal{R} \Delta$ if $\Gamma^\sharp \subseteq \Delta$. This gives us a frame, $\langle \mathcal{G}, \mathcal{R}\rangle$. It is easily shown to be reflexive and transitive. Define a mapping $\mathcal{A}$ by setting $\mathcal{A}(t, X) = \{\Gamma \in \mathcal{G} \mid t{:}X \in \Gamma\}$. Finally, define a mapping $\mathcal{V}$ by specifying that for an atomic formula P, $\Gamma \in \mathcal{V}(P)$ if and only if $P \in \Gamma$. This gives us a structure $\mathcal{M} = \langle \mathcal{G}, \mathcal{R}, \mathcal{A}, \mathcal{V}\rangle$. We begin by showing that $\mathcal{M}$ is an S4LP model.

First we verify that $\mathcal{A}$, meets the $\cdot$ Condition. Suppose we have $\Gamma \in [\mathcal{A}(s, X \supset Y) \cap \mathcal{A}(t, X)]$. By the definition of $\mathcal{A}$, we must have $t{:}X \in \Gamma$ and $s{:}(X \supset Y) \in \Gamma$. Since $s{:}(X \supset Y) \supset (t{:}X \supset (s \cdot t){:}Y)$ is an S4LP axiom, and Γ is maximally consistent, it follows that $(s \cdot t){:}Y \in \Gamma$, and hence $\Gamma \in \mathcal{A}(s \cdot t, Y)$.

Next we verify the $\mathcal{R}$ Closure Condition. Suppose $\Gamma, \Delta \in \mathcal{G}$ and $\Gamma \mathcal{R} \Delta$. Also assume $\Gamma \in \mathcal{A}(t, X)$. By definition of $\mathcal{A}$, we have $t{:}X \in \Gamma$. But $t{:}X \supset !t{:}t{:}X$ is an S4LP axiom, and so is $!t{:}t{:}X \supset \Box t{:}X$, so we have $\Box t{:}X \in \Gamma$, and hence $t{:}X \in \Gamma^\sharp \subseteq \Delta$. Then $\Delta \in \mathcal{A}(t, X)$.

Verifying that $\mathcal{A}$ meets the + and ! Conditions is similar, and is omitted. Likewise it is straightforward to check that $\mathcal{M}$ meets constant specification $\mathcal{C}$.

We have now verified that $\mathcal{M}$ is an S4LP model.

Next, a Truth Lemma can be shown: for each formula X and each $\Gamma \in \mathcal{G}$

$$X \in \Gamma \Longleftrightarrow \mathcal{M}, \Gamma \Vdash X \tag{1}$$

Many of the cases are familiar from standard S4 completeness proofs. I'll verify only one. Suppose (1) is known for X, and we are considering the formula $t{:}X$.

Suppose first that $t{:}X \in \Gamma$. Then, using the Connecting Axiom, $t{:}X \supset \Box X$, $\Box X \in \Gamma$, and so $X \in \Gamma^\sharp$. Then if Δ is an arbitrary member of $\mathcal{G}$ with $\Gamma \mathcal{R} \Delta$ we have $\Gamma^\sharp \subseteq \Delta$ and hence $X \in \Delta$. By the induction hypothesis, $\mathcal{M}, \Delta \Vdash X$. Also since $t{:}X \in \Gamma$, we have $\Gamma \in \mathcal{A}(t, X)$. It follows that $\mathcal{M}, \Gamma \Vdash t{:}X$.

Next, suppose $\mathcal{M}, \Gamma \Vdash t{:}X$. This case is trivial. Part of the definition of $\Vdash$ tells us $\Gamma \in \mathcal{A}(t, X)$, and by definition of $\mathcal{A}$ for $\mathcal{M}$, we must have $t{:}X \in \Gamma$.

Thus we have the Truth Lemma. It follows immediately that $\mathcal{M}$ meets the local realizability condition. Here is the argument. Suppose $\mathcal{M}, \Gamma \Vdash X$. Then by the Truth Lemma, $X \in \Gamma$. Designate an occurrence of $\Box$ in X. Since members of $\mathcal{G}$ have the $\Box$ instantiation property, for some t, $\Gamma \cup \{X(t)\}$ is consistent, hence $X(t) \in \Gamma$ since Γ is maximal. If there are occurrences of $\Box$ in $X(t)$ repeat this step, eliminating a second occurrence. And so on. When all occurrences of $\Box$ are gone, we have a realization X' of X, with $X' \in \Gamma$. But then $\mathcal{M}, \Gamma \Vdash X'$, by the Truth Lemma again.

Since $\neg Z_0 \in \Gamma_0$, and $\Gamma_0 \in \mathcal{G}$, we have $\mathcal{M}, \Gamma_0 \not\Vdash Z_0$, completing the proof.

6 Conclusion

S4LP is an explicit/implicit analog of S4. In a similar way explicit/implicit analogs of weaker logics can be introduced, essentially by dropping axioms from S4LP. The results of this paper carry over to the analogs of K4, T, and K. In the other direction one can strengthen LP from an S4 counterpart to an S5 one, but it requires adding machinery. This affects what is needed for a completeness proof [12,13], and the status of an explicit/implicit version of S5 has not been checked yet.

The LP semantics in Section 2.2 comes from [5,6] in two versions, weak and strong. The weak version is basic in this paper. For the strong version an additional condition is placed on models. An LP model $\mathcal{M} = \langle \mathcal{G}, \mathcal{R}, \mathcal{A}, \mathcal{V} \rangle$ is *fully explanatory* provided, for each world $\Gamma \in \mathcal{G}$, if $\mathcal{M}, \Delta \Vdash X$ for every $\Delta \in \mathcal{G}$ such that $\Gamma \mathcal{R} \Delta$, then $\mathcal{M}, \Gamma \Vdash t{:}X$ for some justification term t. Strong models are weak models that are fully explanatory. LP is complete with respect to strong models, using a proof that very closely mirrors the usual proof of completeness for modal logics. Conventionally one shows that if $\{\Box X_1, \ldots, \Box X_n, \neg\Box Y\}$ is consistent, then so is $\{X_1, \ldots, X_n, \neg Y\}$. In the LP case one shows that, for fixed choice of t_1, …, t_n, if $\{t_1{:}X_1, \ldots, t_n{:}X_n, \neg u{:}Y\}$ is consistent for every choice of u, then $\{X_1, \ldots, X_n, \neg Y\}$ is also consistent.

The work of this paper provides a second proof of strong completeness for LP, along completely different lines, as follows. Suppose X is a formula of LP, and X is not LP provable using an axiomatically appropriate constant specification $\mathcal{C}$. It is simple to extend $\mathcal{C}$ to a constant specification that is axiomatically appropriate for all of S4LP and is $\Box$ closed. Call the extension $\mathcal{C}^*$. Then X is not provable in S4LP using $\mathcal{C}^*$ either, since an LP counter model for X easily extends to an S4LP countermodel for X, using the weak notion of S4LP model from [8]. Since X is not provable in S4LP using $\mathcal{C}^*$, by the completeness proof of the present paper, X is falsified at some world of an S4LP model meeting the local realizability condition. If we ignore $\Box$ in such a model we have an LP model, and local realizability immediately gives us the fully explanatory condition.

It is curious that the fully explanatory condition can be approached from such seemingly different directions—via a generalization of a standard modal argument, and via a generalization of a Henkin completeness argument. It is also curious that no use has been found for the condition. This report ends on a note of genuine puzzlement.

References

1. Hintikka, J.: Knowledge and Belief. Cornell University Press (1962)
2. Artemov, S.: Explicit provability and constructive semantics. The Bulletin for Symbolic Logic 7(1), 1–36 (2001)
3. Artemov, S., Nogina, E.: Logic of knowledge with justifications from the provability perspective. Technical report, Technical Report TR-2004011, CUNY Ph. D. Program in Computer Science, 2004 (2004)
4. Artemov, S., Nogina, E.: Introducing Justification into Epistemic Logic. Journal of Logic and Computation 15(6), 1059–1073 (2005)
5. Fitting, M.C.: A semantics for the logic of proofs. Technical Report TR-2003012, CUNY Ph.D. Program in Computer Science (2003), `http://www.cs.gc.cuny.edu/tr/`
6. Fitting, M.C.: The logic of proofs, semantically. Annals of Pure and Applied Logic 132, 1–25 (2005)
7. Mkrtychev, A.: Models for the logic of proofs. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 266–275. Springer, Heidelberg (1997)
8. Fitting, M.C.: Semantics and tableaus for LPS4. Technical Report TR-2004016, CUNY Ph.D. Program in Computer Science (2004), `http://www.cs.gc.cuny.edu/tr/`
9. Artemov, S.: Evidence-based common knowledge. Technical report, Technical Report TR-2004018, CUNY Ph. D. Program in Computer Science, 2004 (2004)
10. Artemov, S.: Justified common knowledge. Theoretical Computer Science 357(1-3), 4–22 (2006)
11. Fitting, M.C.: Reasoning with justifications. Studia Logica (Forthcoming, 2008)
12. Rubtsova, N.: Evidence-based knowledge for S5. The Bulletin of Symbolic Logic 12(2), 344 (2006)
13. Rubtsova, N.: Semantics for Logic of explicit knowledge corresponding to S5. In: Proceedings of the Workshop on Rationality and Knowledge, ESSLLI, Malaga, pp. 124–131 (2006)

On the Expressive Power of Permanents and Perfect Matchings of Matrices of Bounded Pathwidth/Cliquewidth (Extended Abstract)

Uffe Flarup[1] and Laurent Lyaudet[2]

[1] Department of Mathematics and Computer Science
Syddansk Universitet, Campusvej 55, 5230 Odense M, Denmark
Fax: +45 65 93 26 91
flarup@imada.sdu.dk

[2] Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, 46, allée d'Italie, 69364 Lyon Cedex 07, France
Fax: +33 4 72 72 80 80
laurent.lyaudet@ens-lyon.fr

Abstract. Some 25 years ago Valiant introduced an algebraic model of computation in order to study the complexity of evaluating families of polynomials. The theory was introduced along with the complexity classes VP and VNP which are analogues of the classical classes P and NP. Families of polynomials that are difficult to evaluate (that is, VNP-complete) includes the permanent and hamiltonian polynomials.

In a previous paper the authors together with P. Koiran studied the expressive power of permanent and hamiltonian polynomials of matrices of bounded treewidth, as well as the expressive power of perfect matchings of planar graphs. It was established that the permanent and hamiltonian polynomials of matrices of bounded treewidth are equivalent to arithmetic formulas. Also, the sum of weights of perfect matchings of planar graphs was shown to be equivalent to (weakly) skew circuits.

In this paper we continue the research in the direction described above, and study the expressive power of permanents, hamiltonians and perfect matchings of matrices that have bounded pathwidth or bounded cliquewidth. In particular, we prove that permanents, hamiltonians and perfect matchings of matrices that have bounded pathwidth express exactly arithmetic formulas. This is an improvement of our previous result for matrices of bounded treewidth. Also, for matrices of bounded weighted cliquewidth we show membership in VP for these polynomials.

1 Introduction

In this paper we continue the work that was started in [8]. Our focus is on easy special cases of otherwise difficult to evaluate polynomials, and their relation to various classes of arithmetic circuits. It is conjectured that the permanent and hamiltonian polynomials are hard to evaluate. Indeed, in Valiant's model

[17,18] these families of polynomials are both VNP-complete (VP and VNP are analogues of the classical classes P and NP). In the boolean framework they are complete for the complexity class $\sharp$P [19]. However, for matrices of bounded treewidth the permanent and hamiltonian polynomials can efficiently be evaluated - the number of arithmetic operations being polynomial in the size of the matrix [4].

The sum of weights of perfect matchings in a weighted (undirected) graph is another hard to evaluate polynomial, but for planar graphs it can be evaluated efficiently due to Kasteleyn's theorem [11].

By means of reductions these evaluation methods can all be seen as general-purpose evaluation algorithms for certain classes of polynomials. As an example, if an arithmetic formula represents a polynomial P then one can construct a matrix A of bounded treewidth such that:

(i) The entries of A are variables of P, or constants from the underlying field.
(ii) The permanent of A is equal to P.

It turns out that the converse holds as well. Here we would like to study to what extent this can be done, when the matrix A has bounded pathwidth or bounded cliquewidth instead of bounded treewidth. In [8] the following results (with abuse of notation) were established:

(i) permanent/hamiltonian(bounded treewidth matrix) $\equiv$ arithmetic formulas.
(ii) perfect matchings(planar matrix) $\equiv$ arithmetic skew circuits.

One can also by similar proofs show that:

(iii) perfect matchings(bounded treewidth matrix) $\equiv$ arithmetic formulas.

In this paper we establish the following results:

(i) per/ham/perfect matchings(bounded pathwidth matrix) $\equiv$ arithmetic skew circuits of bounded width $\equiv$ arithmetic weakly skew circuits of bounded width $\equiv$ arithmetic formulas.
(ii) arithmetic formulas $\subseteq$ per/ham/perf. match.(bounded weighted cliquewidth matrix) $\subseteq$ VP.

Weighted cliquewidth is a natural extension of cliquewidth for weighted graphs defined in this paper. It differs from the cliquewidth of the underlying unweighted graph since this notion is not adapted to our problematic.

Overview of paper: The second section of the paper introduces definitions used throughout the paper and gives some small technical results related to graph-widths. Sections 3 and 4 are devoted to our main results, namely the expressiveness of the permanent, hamiltonian, and perfect matchings of graphs of bounded pathwidth and of bounded weighted cliquewidth respectively. We prove in Section 3 that the permanent, hamiltonian, and perfect matchings limited to bounded pathwidth graphs express arithmetic formulas. In Section 4, we show that for all three polynomials the complexity is between arithmetic formulas and VP for graphs of bounded weighted cliquewidth.

Due to space restriction, we had to remove several proofs that can be found in the full version [9]. We will not emphasize this point but all reductions can be done in polynomial time (most of them can be done in linear time).

2 Definitions and Preliminary Results

2.1 Arithmetic Circuits

Definition 1. *An arithmetic circuit is a finite, acyclic, directed graph. Vertices have indegree 0 or 2, where those with indegree 0 are referred to as* inputs*. A single vertex must have outdegree 0, and is referred to as* output*. Each vertex of indegree 2 must be labeled by either + or $\times$, thus representing computation. Vertices are commonly referred to as* gates *and edges as* arrows*.*

In this paper various subclasses of arithmetic circuits will be considered: For *weakly skew* circuits we have the restriction that for every multiplication gate, at least one of the incoming arrows is from a subcircuit whose only connection to the rest of the circuit is through this incoming arrow. For *skew* circuits we have the restriction that for every multiplication gate, at least one of the incoming arrows is from an input gate. For *formulas* all gates (except output) have outdegree 1. Thus, reuse of partial results is not allowed.

For a detailed description of various subclasses of arithmetic circuits, along with examples, we refer to [15].

Definition 2. *The* size *of a circuit is the total number of* gates *in the circuit. The* depth *of a circuit is the length of the longest path from an input gate to the output gate.*

2.2 Pathwidth and Treewidth

Since the definition of pathwidth is closely related to the definition of treewidth (bounded pathwidth is a special case of bounded treewidth) we also include the definition of treewidth in this paper. Treewidth for undirected graphs is most commonly defined as follows:

Definition 3. *Let $G = \langle V, E \rangle$ be a graph. A k-tree-decomposition of G is:*

(i) *A tree $T = \langle V_T, E_T \rangle$.*
(ii) *For each $t \in V_T$ a subset $X_t \subseteq V$ of size at most $k+1$.*
(iii) *For each edge $(u, v) \in E$ there is a $t \in V_T$ such that $\{u, v\} \subseteq X_t$.*
(iv) *For each vertex $v \in V$ the set $\{t \in V_T | v \in X_t\}$ forms a (connected) subtree of T.*

The treewidth of G is the smallest k such that there exists a k-tree-decomposition for G.

A k-path-decomposition of G is then a k-tree-decomposition where the "tree" T is a path (each vertex $t \in V_T$ has at most one child in T).

The pathwidth (treewidth) of a directed, weighted graph is naturally defined as the pathwidth (treewidth) of the underlying, undirected, unweighted graph. The pathwidth (treewidth) of an $(n \times n)$ matrix $M = (m_{i,j})$ is defined as the pathwidth (treewidth) of the directed graph

2.3 Cliquewidth, NLCwidth and m-Cliquewidth

We have seen that the definition of pathwidth and treewidth for weighted graphs was straightforwardly defined as the width of the underlying, unweighted graph. This is a major difference compared to cliquewidth. We can see that if we consider non-edges as edges of weight 0 then every weighted graph has a clique (which has bounded cliquewidth 2) as its underlying, unweighted graph.

One motivation for studying bounded cliquewidth matrices was to obtain efficient algorithms for evaluating polynomials like the permanent and hamiltonian for such matrices. For this reason, it is not reasonable to define the cliquewidth of a weighted graph as the cliquewidth of the underlying, unweighted graph, because then computing the permanent of a matrix of cliquewidth 2 is as difficult as the general case. Hence, we put restrictions on how weights are assigned to edges: Edges added in the same operation between vertices having the same pair of labels will all have the same weight.

We now recall the definitions of cliquewidth, NLCwidth and m-cliquewidth for undirected, unweighted graph, and then introduce the new notions of W-cliquewidth, W-NLCwidth and W-m-cliquewidth which are variants of the preceding ones for *weighted*, directed graphs.

For the definitions of W-cliquewidth, W-NLCwidth and W-m-cliquewidth, we will consider simple, weighted, directed graphs where the weights are in some set W. Below, operations solely for the weighted case are indicated by **bold** font. In the three following constructions, a (directed) arc from a vertex x to a vertex y is only added by relevant operations if there is not already an arc from x to y.

Definition 4 ([3,5]). *A graph G has cliquewidth (W-**cliquewidth**) at most k iff there exists a set of source labels $\mathcal{S}$ of cardinality k such that G can be constructed using a finite number of the following operations (named clique operations **or W-clique operations**):*

- *(i) ver_a, $a \in \mathcal{S}$ (basic construct: create a single vertex with label a).*
- *(ii) $\rho_{a \to b}(H)$, $a, b \in \mathcal{S}$ (rename all vertices with label a to have label b instead).*
- *(iii) $\eta_{a,b}(H)$, $a, b \in \mathcal{S}$, $a \neq b$ (add edges between all pairs of vertices where one of them has label a and the other has label b).*
- ***(iii)'** $\alpha^w_{a,b}(H)$, $a, b \in \mathcal{S}$, $a \neq b$, $w \in W$ (add arcs of weight w from all vertices with label a to all vertices with label b).*
- *(iv) $H \oplus H'$ (disjoint union of graphs).*

Definition 5 ([20]). *A graph G has NLCwidth (W-**NLCwidth**) at most k iff there exists a set of source labels $\mathcal{S}$ of cardinality k such that G can be constructed using a finite number of the following operations (named NLC operations or **W-NLC operations**):*

(i) ver_a, $a \in \mathcal{S}$ *(basic construct: create a single vertex with label a).*
(ii) $\circ_R(H)$ *for any mapping* R *from* $\mathcal{S}$ *to* $\mathcal{S}$ *(for every source label* $a \in \mathcal{S}$ *rename all vertices with label* a *to have label* $R(a)$ *instead).*
(iii) $H \times_E H'$ *for any* $E \subseteq \mathcal{S}^2$ *(disjoint union of graphs to which are added edges between all couples of vertices* $x \in H$ *(with label* l_x*),* $y \in H'$ *(with label* l_y*) having* $(l_x, l_y) \in E$*).*
(iii)' $H \times_E H'$ *for any partial function* $E : \mathcal{S}^2 \times \{-1, 1\} \to W$ *(disjoint union of graphs to which arcs of weight* w *are added for each couple of vertices* $x \in H$, $y \in H'$ *whose labels* l_x, l_y *are such that* $E(l_x, l_y, s) = w$*; the arc is from* x *to* y *if* $s = 1$ *and from* y *to* x *if* $s = -1$*).*

One important distinction between cliquewidth, NLCwidth on one side and m-cliquewidth (to be defined below) on the other side is that in the first two each vertex is assigned exactly *one* label, and in the last one each vertex is assigned a *set* of labels (possibly empty).

Definition 6 ([6]). *A graph* G *has m-cliquewidth (*W***-m-cliquewidth****) at most* k *iff there exists a set of source labels* $\mathcal{S}$ *of cardinality* k *such that* G *can be constructed using a finite number of the following operations (named m-clique operations or* W***-m-clique operations****):*

(i) ver_A *(basic construct: create a single vertex with a set of labels* A, $A \subseteq \mathcal{S}$*).*
(ii) $H \otimes_{E,h,h'} H'$ *for any* $E \subseteq \mathcal{S}^2$ *and any* $h, h' : \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$ *(disjoint union of graphs to which are added edges between all couples of vertices* $x \in H$, $y \in H'$ *whose sets of labels* L_x, L_y *contain a couple of labels* l_x, l_y *such that* $(l_x, l_y) \in E$*. Then the labels of vertices from* H *are changed via* h *and the labels of vertices from* H' *are changed via* h'*).*
(ii)' $H \otimes_{E,h,h'} H'$ *for any partial function* $E : \mathcal{S}^2 \times \{-1, 1\} \to W$ *and any* $h, h' : \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$ *(disjoint union of graphs to which arcs of weight* w *are added for each couple of vertices* $x \in H$, $y \in H'$ *whose sets of labels* L_x, L_y *contain* l_x, l_y *such that* $E(l_x, l_y, s) = w$*; the arc is from* x *to* y *if* $s = 1$ *and from* y *to* x *if* $s = -1$*. Then the labels of vertices from* H *are changed via* h *and the labels of vertices from* H' *are changed via* h'*).*

In the last operation for W-m-cliquewidth, there is a possibility that two (or more) arcs are added from a vertex x to a vertex y during the same operation and then the obtained graph is not simple. For this reason, we will consider as well-formed terms only the terms where this does not occur.

It is stated in [6] (a proof sketch of some of this result is given in [6], and one of the inequalities is proven in [10]) that

$$mcwd(G) \leq wd_{NLC}(G) \leq cwd(G) \leq 2^{mcwd(G)+1} - 1.$$

Hence, cliquewidth, NLC-width and m-cliquewidth are equivalent with respect to boundedness.

The following theorem shows that the inequalities between the three widths are still valid in the weighted case. It justifies our definitions of weighted cliquewidth. A proof of this result can be obtained by collecting the ideas in [6,10] and combining them with our weighted definitions.

Theorem 1. *For any weighted graph G,*

$$Wmcwd(G) \leq Wwd_{NLC}(G) \leq Wcwd(G) \leq 2^{Wmcwd(G)+1} - 1.$$

The three preceding constructions of graphs can be extended to weighted graphs with loops by adding the basic constructs $verloop_a^w$ or $verloop_A^w$ which creates a single vertex with a loop of weight w and label a or set of labels A. One can then easily show the following result.

Let G be a weighted graph (directed or not) with loops. Let $Unloop(G)$ denote the weighted graph (directed or not) obtained from G by removing all loops. Then:

- $Wcwd(G) = Wcwd(Unloop(G))$.
- $Wwd_{NLC}(G) = Wwd_{NLC}(Unloop(G))$.
- $Wmcwd(G) = Wmcwd(Unloop(G))$.

2.4 Permanent and Hamiltonian Polynomials

In this paper we take a graph theoretic approach to deal with permanent and hamiltonian polynomials. The reason for this is that a natural way to define pathwidth, treewidth or cliquewidth of a matrix M, is by the width of the graph with adjacency matrix M, also see e.g. [13].

Definition 7. *A* cycle cover *of a directed graph is a subset of the edges, such that these edges form disjoint, directed cycles (loops are allowed). Furthermore, each vertex in the graph must be in one (and only one) of these cycles. The* weight *of a cycle cover is the* product *of weights of all participating edges.*

Definition 8. *The* permanent *of an $(n \times n)$ matrix $M = (m_{i,j})$ is the sum of weights of all cycle covers of G_M.*

The permanent of M is usually defined by the formula $\text{per}(M) = \sum_{\sigma \in S_n} \prod_{i=1}^n m_{i,\sigma(i)}$ but here we emphasize on the graph theoretic approach. The equivalence with Definition 8 is clear since any permutation can be written down as a product of disjoint cycles, and this decomposition is unique. The *hamiltonian* polynomial $\text{ham}(M)$ is defined similarly, except that we only sum over cycle covers consisting of a *single* cycle (hence the name).

There is a natural way of representing polynomials by permanents. Indeed, if the entries of M are variables or constants from some field K, $f = \text{per}(M)$ is a polynomial with coefficients in K (in Valiant's terminology, f is a projection of the permanent polynomial). In the next section we study the power of this representation in the case where M has bounded pathwidth or bounded cliquewidth.

2.5 Connections between Permanents and Sum of Weights of Perfect Matchings

Another combinatorial characterization of the permanent is by sum of weights of perfect matchings in a bipartite graph. We will use this connection to deduce results for the permanent from results for the sum of weights of perfect matchings, and reciprocally.

Definition 9. *Let G be a directed graph (weighted or not). We define the* inside-outside graph *of G, denoted $IO(G)$, as the bipartite, undirected graph (weighted or not) obtained as follows:*

- *we split each vertex $u \in V(G)$ in two vertices u^+ and u^-;*
- *each arc uv (of weight w) is replaced by an edge between u^+ and v^- (of weight w). A loop on u (of weight w) is replaced by an edge between u^+ and u^- (of weight w).*

It is well-known that the permanent of a matrix M can be defined as the sum of weights of all perfect matchings of $IO(G_M)$. We can see that the adjacency matrix of $IO(G_M)$ is $\begin{pmatrix} 0 & M \\ M^t & 0 \end{pmatrix}$. The two following lemmas can be easily proved.

Lemma 1. *If G has treewidth (pathwidth) k, then $IO(G)$ has treewidth (pathwidth) at most $2k+1$.*

Lemma 2. *If G has W-cliquewidth k, then $IO(G)$ has W-cliquewidth at most $2k$.*

3 Expressiveness of Matrices of Bounded Pathwidth

In this section we study the expressive power of permanents, hamiltonians and perfect matchings of matrices of bounded pathwidth. We will prove that in each case we capture exactly the families of polynomials computed by polynomial size skew circuits of bounded width. A by-product of these proofs will be a proof of the equivalence between polynomial size skew circuits of bounded width and polynomial size *weakly* skew circuits of bounded width. This equivalence can not be immediately deduced from the already known equivalence between polynomial size skew circuits and polynomial size weakly skew circuits in the unbounded width case [16] (the proofs in [16] use a combinatorial characterization of the complexity of the determinant as the sum of weights of s,t-paths in a graph of polynomial size with distinguished vertices s and t. The additional difficulties to extend these proofs to circuits and graphs of bounded width would be equivalent to the ones we deal with). We will then prove that skew circuits of bounded width are equivalent to arithmetic formulas.

Definition 10. *An arithmetic circuit φ has* width *$k \geq 1$ if there exists a finite set of totally ordered layers such that:*

- *Each gate of φ is contained in exactly 1 layer.*
- *Each layer contains at most k gates.*
- *For every non-input gate of φ if that gate is in some layer n, then both inputs to it are in layer $n+1$.*

Theorem 2. *The polynomial computed by a weakly skew circuit of bounded width can be expressed as the permanent of a matrix of bounded pathwidth. The size of the matrix is polynomial in the size of the circuit. All entries in the matrix are either 0, 1, constants of the polynomial, or variables of the polynomial.*

Proof. Let φ be a weakly skew circuit of bounded width $k \geq 1$ and $l > 1$ the number of layers in φ. The graph G we construct will have pathwidth at most $4 \cdot k - 1$ (each bag in the path-decomposition will contain at most $4 \cdot k$ vertices) and the number of bags in the path-decomposition will be $l - 1$. G will have two distinguished vertices s and t, and the sum of weights of all directed paths from s to t equals the value computed by φ.

Since φ is a weakly skew circuit we consider a decomposition of it into disjoint subcircuits defined recursively as follows: The output gate of φ belongs to the *main subcircuit*. If a gate in the main subcircuit is an addition gate, then both of its input gates are in the main subcircuit as well. If a gate g in the main subcircuit is a multiplication gate, then we know that at least one input to g is the output gate of a subcircuit which is disjoint from φ except for its connection to g. This subcircuit forms a *disjoint multiplication-input subcircuit*. The other input to g belongs to the main subcircuit. If some disjoint multiplication-input subcircuit φ' contains at least one multiplication gate, then we make a decomposition of φ' recursively. Note that such a decomposition of a weakly skew circuit is not necessarily unique, because *both* inputs to a multiplication gate can be disjoint from the rest of the circuit, and any one of these two can be chosen as the one that belongs to the main subcircuit.

Let $\varphi_0, \varphi_1, \ldots, \varphi_d$ be the disjoint subcircuits obtained in the decomposition (φ_0 is the main subcircuit). The graph G will have a vertex v_g for every gate g of φ and $d + 1$ additional vertices $s = s_0, s_1, \ldots, s_d$ (t will correspond to v_g where g is the output gate of φ). For every gate g in the subcircuit φ_i, the following construction will ensure that the sum of weights of directed paths from s_i to v_g is equal to the value computed at g in φ.

For the construction of G we process the *decomposition* of φ in a bottom-up manner. Let subcircuit φ_i be a leaf in the decomposition of φ (so φ_i consists solely of addition gates and input gates). Assume that φ_i is located in layers top_i through bot_i ($1 \geq top_i \geq bot_i \geq l$) of φ. First we add a vertex s_i to G in bag $bot_i - 1$, and for each input gate with value w in the bottom layer bot_i of φ_i we add a vertex to G also in bag $bot_i - 1$ along with an edge of weight w from s_i to that vertex. Let n range from $bot_i - 1$ to top_i: Add the already created vertex s_i to bag $n - 1$ and handle input gates of φ_i in layer n as previously described. For each addition gate of φ_i in layer n we add a new vertex to G (which is added to bags n and $n - 1$ of the path-decomposition of G). In bag n we already have two vertices that represent inputs to this addition gate, so we add edges of weight 1 from both of these to the newly added vertex. The vertex representing the output gate of the circuit φ_i is denoted by t_i. The sum of weighted directed paths from s_i to t_i equals the value computed by the subcircuit φ_i.

Let φ_i be a subcircuit in the decomposition of φ that contains multiplication gates. Addition gates and input gates in φ_i are handled as before. Let g be a multiplication gate in φ_i in layer n and φ_j the disjoint multiplication-input subcircuit that is one of the inputs to g. We know that vertices s_j and t_j already are in bag n, so we add an edge of weight 1 from the vertex representing the other input to g to the vertex s_j, and an edge of weight 1 from t_j to a newly created vertex v_g that represents gate g, and then v_g is added to bags n and $n-1$.

For every b $(1 \geq b \geq l-1)$ we need to show that only a constant number of vertices are added to bag b during the entire process. Every gate in layer b of φ is represented by a vertex, and these vertices may all be added to bag b. Every gates in layer $b+1$ are also represented by a vertex, and all of these may be added to bag b (because they are used as input here). In addition to this, a number of s_i vertices are also added to bag b. For each gate of subcircuit φ_j in layer b or $b+1$, we have the corresponding s_j vertex in bag b. In total up to $4 \cdot k$ vertices are added to bag b.

Note that in layer 1 of φ we just have the output gate. This gate is represented by the vertex t of G which is in bag 1 of the path-decomposition.

The sum of weights of all directed paths from s to t in G can by induction be shown to be equal to the value computed by φ. The final step in the reduction to the permanent polynomial is to add an edge of weight 1 from t back to s and loops of weight 1 at all nodes different from s and t. □

With a longer proof one can show that the graph constructed in the preceding proof has pathwidth at most $\left\lfloor \frac{7 \cdot k}{2} \right\rfloor - 1$. Also, it can be modified to work for the hamiltonian polynomial as well, by adapting the idea in [14]: For the permanent polynomial each bag in the path-decomposition contains at most $4 \cdot k$ vertices; for each of these vertices we now need to introduce one extra vertex in the same bag. In addition each bag must contain 2 more vertices in order to establish a connection to adjacent bags in the path-decomposition. In total each bag now contains at most $8 \cdot k + 2$ vertices.

Theorem 3. *The polynomial computed by a weakly skew circuit of bounded width can be expressed as the sum of weights of perfect matchings of a symmetric matrix of bounded pathwidth. The size of the matrix is polynomial in the size of the circuit. All entries in the matrix are either 0, 1, constants of the polynomial, or variables of the polynomial.*

Proof. It is a direct consequence of Theorem 2 and Lemma 1. □

Theorem 4. *The hamiltonian of a matrix of bounded pathwidth can be expressed as a skew circuit of bounded width. The size of the circuit is polynomial in the size of the matrix.*

Proof. Let M be a matrix of bounded pathwidth k and let G_M be the underlying, directed graph. Each bag in the path-decomposition of G_M contains at most $k+1$ vertices. We refer to one end of the path-decomposition as the *leaf* of the path-decomposition and the other as the *root* (recall that path-decompositions are special cases of tree-decompositions).

We process the path-decomposition of G_M from the leaf towards the root. The overall idea is the same as the proof of Theorem 5 in [8] - namely to consider weighted partial path covers (i.e. partial covers consisting solely of paths) of subgraphs of G_M that are induced by the path-decomposition of G_M. During the processing of the path-decomposition of G_M at every level distinct from the root, new partial path covers are constructed by taking one previously generated partial path cover and then add at most $(k+1)^2$ new edges, so all the multiplication gates we have in our circuit are skew. For any bag in the path-decomposition of G_M we only need to consider a number of partial path covers that depends solely on k, so the circuit we produce has bounded width. At the root we add sets of edges to partial path covers to form hamiltonian cycles. □

By a similar proof, one can show the following theorem.

Theorem 5. *The sum of weights of perfect matchings of a symmetric matrix of bounded pathwidth can be expressed as a skew circuit of bounded width. The size of the circuit is polynomial in the size of the matrix.*

Theorem 6. *The permanent of a matrix of bounded pathwidth can be expressed as a skew circuit of bounded width. The size of the circuit is polynomial in the size of the matrix.*

Proof. It is a direct consequence of Theorem 5 and Lemma 1. □

The following corollary can be deduced from Theorem 2 and Theorem 6.

Corollary 1. *A family of polynomials is computable by polynomial size skew circuits of bounded width if and only if it is computable by polynomial size weakly skew circuits of bounded width.*

We need the following theorem from [1] to prove the equivalence between polynomial size skew circuits of bounded width and polynomial size arithmetic formulas.

Theorem 7. *Any arithmetic formula can be computed by a linear bijection straight-line program of polynomial size that uses three registers.*

This result of Ben-Or and Cleve is a generalisation of the celebrated result of Barrington in boolean complexity proving the equivalence between NC^1 and bounded width branching programs.

Let $R_1, \dots, R_m$ be a set of m registers, a linear bijection straight-line (LBS) program is a vector of m initial values given to the registers plus a sequence of instructions of the form

(i) $R_j \leftarrow R_j + (R_i \times c)$, or
(ii) $R_j \leftarrow R_j - (R_i \times c)$, or
(iii) $R_j \leftarrow R_j + (R_i \times x_u)$, or
(iv) $R_j \leftarrow R_j - (R_i \times x_u)$,

where $1 \leq i, j \leq m$, $i \neq j$, $1 \leq u \leq n$, c is a constant, and $x_1, \dots, x_n$ are variables (n is the number of variables). We suppose without loss of generality that the value computed by the LBS program is the value in the first register after all instructions have been executed.

Theorem 8. *A family of polynomials is computed by polynomial size skew circuits of bounded width if and only if it is a family of polynomial size arithmetic formulas.*

4 Expressiveness of Matrices of Bounded Weighted Cliquewidth

In this section we study the expressive power of permanents, hamiltonians and perfect matchings of matrices that have bounded weighted cliquewidth. We first prove that every arithmetic formula can be expressed as the permanent, hamiltonian, or sum of perfect matchings of a matrix of bounded W-cliquewidth using the results for the bounded pathwidth matrices and the following lemma.

Lemma 3. *Let G be a weighted graph (directed or not) with weights in W. If G has pathwidth k, then G has W-cliquewidth at most $k+2$.*

Theorem 9. *Every arithmetic formula can be expressed as the permanent of a matrix of W-cliquewidth at most 25 and size polynomial in n, where n is the size of the formula. All entries in the matrix are either 0, 1, constants of the formula, or variables of the formula.*

Proof. Let φ be a formula of size n. Due to the proof of Theorem 8, we know that it can be computed by a skew circuit of width 6 and size $O(n^{O(1)})$. Hence it is equal to the permanent of a graph of size $O(n^{O(1)})$, pathwidth at most $4 \cdot 6 - 1 = 23$ by Theorem 2, and W-cliquewidth at most $23 + 2 = 25$ by Lemma 3. □

Similarly, one obtains the two following results.

Theorem 10. *Every arithmetic formula can be expressed as the hamiltonian of a matrix of W-cliquewidth at most $(8 \cdot 6 + 2 - 1) + 2 = 51$ and size polynomial in n, where n is the size of the formula. All entries in the matrix are either 0, 1, or constants of the formula, or variables of the formula.*

Theorem 11. *Every arithmetic formula can be expressed as the sum of weights of perfect matchings of a symmetric matrix of W-cliquewidth at most $25 \cdot 2 = 50$ and size polynomial in n, where n is the size of the formula. All entries in the matrix are either 0, 1, constants of the formula, or variables of the formula.*

We can modify the constructions for bounded treewidth graphs expressing formulas in [8] to obtain results similar to the ones above. These modifications require more work than the preceding proofs but we obtain smaller constants since we obtain graphs of W-cliquewidth at most 13/34/26 (instead of 25/51/50) whose permanent/hamiltonian/sum of perfect matchings are equal to formulas.

Due to our restrictions on how weights are assigned in our definition of bounded W-cliquewidth it is not true that weighted graphs of bounded treewidth also have bounded W-cliquewidth. In fact, if one tries to follow the proofs in [5,2] that show that graphs of bounded treewidth have bounded cliquewidth, then one obtains that a weighted graph G of treewidth k has W-cliquewidth at most $3 \cdot (|W_G| + 1)^{k-1}$ or $3 \cdot (\Delta + 1)^{k-1}$. W_G denotes the set of weights on the edges of G and Δ is the maximum degree of G. Weighted trees still have bounded weighted cliquewidth (the bound is 3). But we can show that there exist families of weighted graphs with treewidth two and unbounded W-cliquewidth [12].

We now turn to the upper bound on the complexity of permanent, hamiltonian, and perfect matchings of graphs of bounded weighted cliquewidth. We show that in all three cases the complexity is at most the complexity of VP.

The decision version of the hamiltonian cycle problem has been shown to be polynomial time solvable in [7] for matrices of bounded cliquewidth. Here we extend these ideas in order to compute the hamiltonian polynomial efficiently (in VP) for bounded W-m-cliquewidth matrices.

Definition 11. *A* path cover *of a directed graph G is a subset of the edges of G, such that these edges form disjoint, directed, non-cyclic paths in G. We require that every vertex of G is in (exactly) one path. For technical reasons we allow "paths" of length 0, by having paths that start and end in the same vertex. Such constructions do* not *have the same interpretation as a loop. The* weight *of a path cover is the product of weights of all participating edges (in the special case where there are no participating edges the weight is defined to be 1).*

Theorem 12. *The hamiltonian of an $n \times n$ matrix of bounded W-m-cliquewidth can be expressed as a circuit of size $O(n^{O(1)})$ and thus is in* VP.

Proof. Let M be an $n \times n$ matrix of bounded W-m-cliquewidth. By G we denote the underlying, directed, weighted graph for M. The circuit is constructed based on the parse tree T for G (T is the parse tree of a term over the W-m-clique algebra which constructs the graph G with a bounded number of labels). By T_t we denote the subtree of T rooted at t for some node $t \in T$. By G_t we denote the subgraph of G constructed from the parse tree T_t.

The overall idea is to produce a circuit that computes the sum of weights of all hamiltonian cycles of G. To obtain this there will be non-output gates that compute weights of all path covers of all G_t graphs, and then we combine these subresults. Of course, the total number of path covers can grow exponentially with the size of G_t, so we will not "describe" path covers directly by the edges participating in the covers. Instead we describe a path cover of some G_t graph by the labels associated with the start- and end-vertices of the paths in the cover. Such a description do not uniquely describe a path cover, because two different path covers of the same graph can contain the same number of paths and all these paths can have the same labels associated. However, we do not need the weight of each individual path cover. If multiple path covers of some graph G_t share the same description, then we just compute the sum of weights of these path covers.

For a leaf in the parse tree T of G we construct a single gate of constant weight 1, representing a path cover consisting of a single "path" of length 0, starting and ending in a vertex with the given labels. Per definition this path cover has weight 1.

For an internal node $t \in T$ the grammar rule describes which edges to add and how to relabel vertices. We obtain new path covers by considering a path cover from the left child of t and a path cover from the right child of t: For each such pair of path covers consider all subsets of edges added at node t, and for every subset of edges check if the addition of these edges to the pair of path covers will result in a valid path cover. If it does, then add a gate that computes the weight of this path cover, by multiplying the weight of the left path cover, the weight of the right path cover and the total weight of the newly added edges. After all pairs of path covers have been processed, check if any of the resulting path covers have the same description - namely that the number of paths in some path covers are the same, and that these paths have the same labels for start- and end-vertices. If multiple path covers have the same description then add addition gates to the circuit and produce a single gate which computes the sum of weights of all these path covers.

For the root node r of T we combine path covers from the children of r to produce hamiltonian cycles, instead of path covers. Finally, the output of the circuit is a summation of all gates computing weights of hamiltonian cycles.

Proof of correctness: The first step of the proof is by induction over the height of the parse tree T. We will show that for each non-root node t of T there is for every path cover description of G_t a corresponding gate in the circuit that computes the sum of weights of all path covers of G_t with that description. For the base cases - leaves of T - it is trivially true.

For the inductive step we consider two disjoint graphs that are being connected with edges at a node t of the parse tree T. Edges added at node t are *only* added in here, and not at any other nodes in T, so every path cover of G_t can be split into 3 parts: A path cover of G_{t_l}, a path cover of G_{t_r} and a polynomial number of edges added at node t. Consider a path cover description along with all path covers of G_t that have this description. All of these path covers can be split into 3 such parts, and by our induction hypothesis the weights of the path covers of G_{t_l} and G_{t_r} are computed in already constructed gates.

In order to complete the proof of correctness we have to handle the root t of T in a special way. At the root we do not compute weights of path covers, but instead compute weights of hamiltonian cycles. Every hamiltonian cycle of G can (similarly to path covers) be split into 3 parts: A path cover of G_{t_l}, a path cover of G_{t_r} and a polynomial number of edges added at the root of T. By our induction hypothesis all the needed weights are already computed.

The size of the circuit is polynomial since at each step the number of path cover descriptions is polynomially bounded once the W-m-cliquewidth is bounded. □

By a similar proof, one can prove the following theorem.

Theorem 13. *The sum of weights of perfect matchings of an $n \times n$ symmetric matrix of bounded W-NLCwidth can be expressed as a circuit of size $O(n^{O(1)})$ and thus is in* VP.

Theorem 14. *The permanent of an $n \times n$ matrix of bounded W-m-cliquewidth can be expressed as a circuit of size $O(n^{O(1)})$ and thus is in* VP.

Proof. It is a direct consequence of Theorem 13 and Lemma 2. □

References

1. Ben-Or, M., Cleve, R.: Computing Algebraic Formulas Using a Constant Number of Registers. In: STOC 1988, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 254–257. ACM, New York (1988)
2. Corneil, D., Rotics, U.: On the Relationship Between Clique-Width and Treewidth. SIAM Journal on Computing 34, 825–847 (2005)
3. Courcelle, B., Engelfriet, J., Rozenberg, G.: Context-free Handle-rewriting Hypergraph Grammars. In: Graph-Grammars and Their Application to Computer Science, pp. 253–268 (1990)
4. 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, 23–52 (2001)
5. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101, 77–114 (2000)
6. Courcelle, B., Twigg, A.: Compact Forbidden-Set Routing. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 37–48. Springer, Heidelberg (2007)
7. Espelage, W., Gurski, F., Wanke, E.: How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204. Springer, Heidelberg (2001)
8. Flarup, U., Koiran, P., Lyaudet, L.: On the expressive power of planar perfect matching and permanents of bounded treewidth matrices. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 124–136. Springer, Heidelberg (2007)
9. Flarup, U., Lyaudet, L.: On the expressive power of permanents and perfect matchings of matrices of bounded pathwidth/cliquewidth. Research Report RR2008-05, ENS Lyon (2008), http://aps.arxiv.org/pdf/0801.3408
10. Johansson, O.: Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. Congressus Numerantium 132, 39–60 (1998)
11. Kasteleyn, P.W.: Graph theory and crystal physics. In: Harary, F. (ed.) Graph Theory and Theoretical Physics, pp. 43–110. Academic Press, London (1967)
12. Lyaudet, L., Todinca, I.: Private communication (2007)
13. Makowsky, J.A., Meer, K.: Polynomials of bounded treewidth. In: Cucker, F., Maurice Rojas, J. (eds.) Foundations of Computational Mathematics, Proceedings of the Smalefest 2000, vol. 2002, pp. 211–250. World Scientific, Singapore (2002)
14. Malod, G.: Polynômes et coefficients. Ph.D. thesis (2003)
15. Malod, G., Portier, N.: Characterizing Valiant's Algebraic Complexity Classes. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 704–716. Springer, Heidelberg (2006)
16. Toda, S.: Classes of arithmetic circuits capturing the complexity of computing the determinant. IEICE Transactions on Information and Systems E75-D, 116–124 (1992)
17. Valiant, L.G.: Completeness classes in algebra. In: Proc. 11th ACM Symposium on Theory of Computing, pp. 249–261 (1979)
18. Valiant, L.G.: Reducibility by algebraic projections. In: Logic and Algorithmic (an International Symposium held in honour of Ernst Specker). Monographie n^o 30 de L'Enseignement Mathématique, pp. 365–380 (1982)
19. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 181–201 (1979)
20. Wanke, E.: k-NLC Graphs and Polynomial Algorithms. Discrete Applied Mathematics 54, 251–266 (1994)

The Most General Conservation Law for a Cellular Automaton

Enrico Formenti[1,⋆], Jarkko Kari[2,⋆⋆], and Siamak Taati[2,3]

[1] Université de Nice-Sophia Antipolis, Départment d'Informatique, Parc Valrose, 06108 Nice Cedex 2, France
[2] Department of Mathematics, University of Turku, FI-20014 Turku, Finland
[3] Turku Centre for Computer Science, FI-20520 Turku, Finland

Abstract. We study the group-valued and semigroup-valued conservation laws in cellular automata (CA). We provide examples to distinguish between semigroup-valued, group-valued and real-valued conservation laws. We prove that, even in one-dimensional case, it is undecidable if a CA has any non-trivial conservation law of each type. For a fixed range, each CA has a *most general* (group-valued or semigroup-valued) conservation law, encapsulating all conservation laws of that range. For one-dimensional CA the semigroup corresponding to such a most general conservation law has an effectively constructible finite presentation, while for higher-dimensional ones no such effective construction exists.

1 Introduction

Conservation laws in physics are numerical invariants of the dynamics of a system. In cellular automata, a similar concept has already been defined and studied (see e.g. [8, 3, 14, 5, 6]). We first choose a finite window, through which we can recognize the pattern made by the states of a finite number of cells on the lattice. We associate a real value to each possible local pattern that may be seen through this window, resembling the "energy" (or "mass", or ...) of that pattern. The total "energy" of a configuration is obtained by sliding this window all over the lattice and summing up the energy values of the local patterns we see. We have a conservation law for that energy provided the evolution of the CA preserves the total energy of each configuration.

In physics, conservation laws are used to write equations about the dynamics of the system. Each conserved quantity extracts certain information about the dynamics. In many cases, different conservation laws extract enough information to allow the reconstruction of the whole dynamics. In other cases, conservation laws concretize the physicist's insight into the behavior of the system by refuting those sequences of events that do not respect their preservation.

We study what happens if instead of mere real numbers, we allow energy valuations from a commutative group or semigroup. A remarkable (but trivial)

⋆ This work has been supported by the French ANR Blanc Projet "Sycomore".
⋆⋆ Research supported by the Academy of Finland grant 211967.

fact is that for each CA and a fixed window, there is a *most general* conservation law that extracts whatever information can be expressed in terms of conservation laws using that window. Any other more specific conservation law using that window can be derived from that by applying an algebraic homomorphism.

We provide examples that the group-valued conservation laws give strictly more information than the real-valued ones, and examples in which the semigroup-valued conservation laws are strictly more general than the group-valued ones. Needless to say, the semigroup-valued conservation laws can be quite expressive. Nevertheless, we prove that for one-dimensional CA, the most general conservation law of each range, and a finite presentation of the corresponding semigroup can be effectively constructed. This is a good news, because the Word Problem for commutative semigroups is also decidable (see e.g. [1]). Therefore the whole theory, in one-dimensional case, turns out to be algorithmically effective! For example we can effectively determine whether two (finite) configurations have the same total energy, or if a given CA conserves a given energy valuation. In higher dimensions, however, no such construction for the most general semigroup-valued conservation laws is possible.

Increasing the size of our window, we obtain more and more general (more and more discriminating) conservation laws. We may ask if there is a large enough window that provides the *absolutely most general* conservation law amongst all. This we still cannot answer adequately. However, we show that there are CA whose non-trivial conservation laws require very large windows. In fact, we prove that it is undecidable (even in one-dimensional case) whether a CA has any non-trivial conservation law at all, answering a formerly open question (see [10]). This is valid, no matter we are looking for real-valued, group-valued, or semigroup-valued conservation laws.

We note that the concept of the most general conservation law is closely related to the Conway's tiling group [4, 16], when we look at the space-time diagram of CA as tilings of the plane. Unlike Conway's group, in this paper we restrict our study to *commutative* groups or semigroups. That is because non-commutative conservation laws in higher-than-one dimensional CA do not make much sense. Furthermore, in the non-commutative case, the Word Problem is undecidable even for groups, and we much prefer to stay in the algorithmic realm.

2 Preliminaries

A cellular automaton (CA) is a collection of identical *cells* arranged regularly on a *lattice* where a natural notion of *neighborhood* is present. Each cell is assigned a *state* from a finite number of possible states. The state of the cells are updated synchronously, in discrete time steps, according to a *local update rule*, which takes into account the current state of each cell and its neighbors.

The cells are often indexed by $\mathbb{Z}^d$ ($d \geq 1$), where we obtain a *d-dimensional* CA. The state set is a finite set S. An assignment $c : \mathbb{Z}^d \to S$ of states to the cells of the lattice is referred to as a *configuration* (of the lattice). For each state $q \in S$, a *q-uniform* configuration is a configuration with all cells in state q, and a *q-finite* configuration is one in which all cells but finitely many of them are in the state q. The set of all q-finite configurations is denoted by $C_q[S]$. A *pattern* over a finite set $A \subseteq \mathbb{Z}^d$ is an assignment $p : A \to S$. We use the notation $g|_X$ for the restriction of a mapping g to a subset X of its domain. Therefore, for example, $c|_A$ denotes the pattern seen over $A \subseteq \mathbb{Z}^d$ in the configuration $c \in S^{\mathbb{Z}^d}$.

The neighborhood is specified by a finite set $N \subseteq \mathbb{Z}^d$. The neighborhood of a cell $i \in \mathbb{Z}^d$ is the set $i + N = \{i + a : a \in N\}$. The local update rule is a function $f : S^N \to S$. The local rule f naturally induces a mapping $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$, called the *global mapping*, that maps each configuration c, to its follower configuration $F(c)$, which when starting from c, appears on the lattice after one time-step. Namely, $F(c)[i] \triangleq f(c|_{i+N})$; i.e., the state of the cell i in $F(c)$ is the result of the application of the local rule on the pattern of the neighborhood of i in c. We often identify a CA with its global mapping. A *quiescent* state is a state $q \in S$ such that F maps the q-uniform configuration to itself; i.e., $f(q^N) = q$. If q is a quiescent state, the image of every q-finite configuration is also q-finite.

For any $a \in \mathbb{Z}^d$, the *translation by* a is the operator $\sigma^a : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ defined by $(\sigma^a c)[i] \triangleq c[a + i]$. Notice that σ^a is a CA with neighborhood $\{a\}$. When $d = 1$, we may write σ for σ^1.

Let $A \subseteq \mathbb{Z}^d$ be a finite set. The *A-block-presentation* of a configuration $c \in S^{\mathbb{Z}^d}$ is a configuration $e \in \left(S^A\right)^{\mathbb{Z}^d}$ where $e[i] = c|_{i+A}$. That is, the state of the cell i in e is the overall state of the cells $i + A$ in c.

One-dimensional CA have a natural representation (upto translations) using edge-labeled De Bruijn graphs. The *De Bruijn graph* of order k ($k > 0$) over an alphabet S, is a graph $B_k[S]$ with vertex set $V = S^k$ and edge set $E = S^{k+1}$, where for any $a, b \in S$ and $u \in S^{k-1}$, there is an edge aub from au to ub.

Let $F : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ be a one-dimensional CA with neighborhood $[-l, r] = \{-l, -l+1, \ldots, r\}$ and local rule $f : S^{[-l,r]} \to S$. For any $k \geq l + r$, the CA can be represented on the De Bruijn graph $B_k[S]$ with labeling $\lambda : E \to S^{k-(l+r)}$ which is defined as follows. For every edge $u_0 u_1 \cdots u_k \in S^{k+1}$, let $\lambda(u_0 u_1 \cdots u_k) = v_l v_{l+1} \cdots v_{k-r}$ where $v_i = f(u[i - l, i + r])$. The edge sequence $p = \{p[i]\}_{i \in \mathbb{Z}}$ of each bi-infinite path on $B_k[S]$ is the $[0, k]$-block-presentation of a unique configuration $c \in S^{\mathbb{Z}}$, while its label sequence $\lambda(p) = \{\lambda(p[i])\}_{i \in \mathbb{Z}}$ is the $[l, k - r]$-block-presentation of $F(c)$. Conversely, for every configuration $c \in S^{\mathbb{Z}}$, there is a unique infinite path on $B_k[S]$ whose edge sequence is the $[0, k]$-block-presentation of c.

A *two-counter machine* is a finite automaton equipped with two unbounded *counters*. The machine can increase or decrease the value of each counter, and can test if either has value zero. Two-counter machines are known to be equivalent in power with Turing machines — any algorithm can be implemented on a two-counter machine (see e.g. [11]).

3 Universal Conservation Law

Let $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ be a CA, and to avoid cumbersome technicalities, let us assume that the CA has a quiescent state q. Let Φ be a commutative (additive) semigroup, and $W \subseteq \mathbb{Z}^d$ a finite set, and assume that to each pattern $p \in S^W$ we have associated a value $\mu(p) \in \Phi$ as its *energy*. We would like to define the total energy $M(c)$ of a configuration $c \in S^{\mathbb{Z}^d}$ as the sum

$$\sum_{i \in \mathbb{Z}^d} \mu(c|_{i+W}) \tag{1}$$

by sliding the *window* W over c and adding up all the local energy values we see. However, there is no uniform way to interpret an infinite sum over a semigroup! To overcome this, we choose the following approach: we assume that Φ contains an identity element 0, and that $\mu(q^W) = 0$. Then the above sum will have a natural meaning for all q-finite configurations — only finite number of the terms have values other than 0. For all other configurations we leave the total energy undefined.[1]

We say that the CA *conserves* the energy valuation μ, if

$$M(F(c)) = M(c) \tag{2}$$

for every q-finite configuration c. Then M is a Φ-valued conserved energy defined using window W, and (2) is a *conservation law* for the CA. More concisely, the conservation law can be identified by the pair (Φ, μ).

Notice that the values of M do not depend on the displacement of the window. That is, for every translation σ^a, $M \circ \sigma^a = M$. The conservation of M by a CA F means further that its value is constant over the orbits of F; i.e., $M \circ F = M$. Therefore, the kernel of M is a partition of the orbits of F. The finer this partition, the more information the conservation law extracts about the dynamics of the CA.

An energy function which assigns the same value to every finite configuration is trivially conserved by every CA. We call such a conservation law *trivial*. Note that the total energy mapping $M : C_q[S] \to \Phi$ of a conservation law is not necessarily onto (even if the local energy function μ is onto). The uncovered part of Φ bears no information about the dynamics of F. Hence it is convenient to name the *realizable* sub-monoid $\breve{\Phi} \triangleq M(C_q[S])$ of Φ. So (Φ, μ) is trivial, if and only if, the realizable sub-monoid $\breve{\Phi}$ is trivial.

Let us now fix a CA $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ with a quiescent state q, and a finite window $W \subseteq \mathbb{Z}^d$. Every conserved energy valuation μ for F satisfies (2) for every q-finite configuration c. The key observation in this paper is that the largest semigroup generated by the (formal) values of μ for which (2) holds for every q-finite configuration c provides the *most general* conservation law for F defined using window W.

Put it precisely, let $\Sigma \triangleq S^W - \{q^W\}$ be the set of non-quiescent patterns on W, and let us denote by $\mathbb{N}^\Sigma$ the free commutative (additive) monoid generated by Σ. Define the energy assignment $\mu_\circ : S^W \to \mathbb{N}^\Sigma$ with $\mu_\circ(q^W) = 0$, and $\mu_\circ(p) = p$ for any other pattern p over W, and let $M_\circ : C_q[S] \to \mathbb{N}^\Sigma$ be the corresponding total energy. Let $\cong \subseteq \mathbb{N}^\Sigma \times \mathbb{N}^\Sigma$ be the coarsest monoid congruence where $M_\circ(F(c))$

[1] This is not the only possible approach, but it sounds most natural to us.

$\cong M_\circ(c)$ for every q-finite c. Define $\Phi_F \triangleq \mathbb{N}^\Sigma/\cong$ and let $h_\cong : \mathbb{N}^\Sigma \to \Phi_F$ be the natural homomorphism. Define $\mu_F : W \to \Phi_F$ with $\mu_F = h_\cong \circ \mu_\circ$. Clearly the pair (Φ_F, μ_F) identifies a conservation law, because

$$M_F(F(c)) = h_\cong(M_\circ(F(c))) = h_\cong(M_\circ(c)) = M_F(c) \tag{3}$$

Furthermore, for every conservation law (Φ, μ) with window W, there is a monoid homomorphism $h : \Phi_F \to \Phi$ so that $\mu = h \circ \mu_F$; schematically,

$$\begin{tikzcd} (S^{\mathbb{Z}}, F) \arrow[r, "M_F"] \arrow[dr, "M"'] & \Phi_F \arrow[d, dashed, "h"] \\ & \Phi \end{tikzcd} \tag{4}$$

One can verify that using $\breve{\Phi}_F$ instead of Φ_F, the choice of h would be unique.

It is easy to see that when $W' \subseteq W \subseteq \mathbb{Z}^d$, the most general conservation law based on window W' is a factor of the most general conservation law based on window W; i.e., $(\Phi_F^{(W)}, \mu_F^{(W)})$ is *more general* than $(\Phi_F^{(W')}, \mu_F^{(W')})$. That is because every energy valuation on W' can be seen also as a energy valuation on W, by adding some *dummy* elements to W'.

Following a similar trail of reasoning, if instead of semigroup-valued energies we consider group-valued energies, we can define the *most general* group-valued conservation law (G_F, ρ_F) based on W for F. Likewise, we define the realizable subgroup $\breve{G}_F \triangleq P_F(C_q[S])$ of G_F, where P_F is the total energy mapping corresponding to ρ. The conservation law $(\breve{G}_F, \rho_F)$ satisfies a similar universal property (4) among group-valued conservation laws.

Example 1 (Spreading 1*'s).* Consider the one-dimensional CA F with state set $\{0, 1\}$, neighborhood $\{-1, 0, 1\}$, and local rule $f(a, b, c) = a \vee b \vee c$; see Fig. 1 for a typical snapshot. The time axis in the figure goes downward. It is easy to see that every group-valued conservation law for F is trivial. Notice that every non-quiescent finite configuration eventually turns into a single ever-growing block of ones. In contrast, F has a non-trivial semigroup-valued conservation law. Let $\Phi = \{0, 1\}$ be the commutative semigroup with binary operation $a + b \triangleq a \vee b$. Let $W = \{0\}$ be the singleton window, and $\mu : \{0, 1\} \to \Phi$ be the identity. Under this energy valuation, the 0-uniform has total energy 0, while every other 0-finite configuration has total energy 1.

Example 2 (XOR). The second example is again a one-dimensional CA F' with binary state set, and neighborhood $\{-1, 0, 1\}$. The local rule is the XOR rule $f'(a, b, c) = a + b + c \pmod 2$. Figure 2 shows a typical snapshot. A well-known property of the XOR CA (and in general every linear CA) is its replicating behavior. Specifically, every finite pattern, after a finite number of steps, is replicated into three copies with large 0 blocks in between (Fig. 2 depicts an example. This is easy to verify using generating functions; see e.g. [15]). This implies that F'

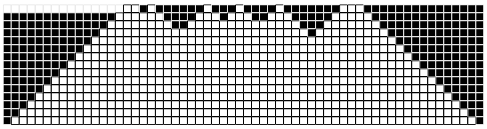


Fig. 1. A space-time snapshot from the CA in Ex. 1

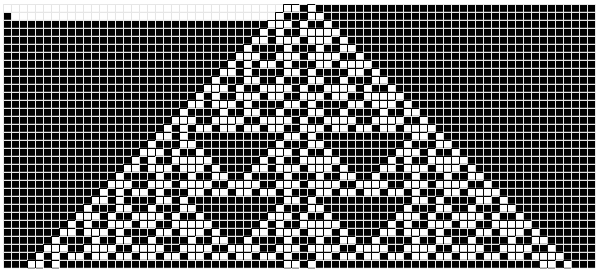


Fig. 2. A space-time snapshot from the CA in Ex. 2

cannot have any non-trivial real-valued conservation law. On the other hand, G preserves the parity of the configurations. Let $G = \mathbb{Z}_2$ be the binary cyclic group, and consider the identity energy function $\rho : \{0,1\} \to \mathbb{Z}_2$ on window $\{0\}$. The total energy $P(c)$ is simply the parity of the number of 1's in c, and is preserved by F'.

4 Semigroup-Valued Conservation Laws

The definition of the most general conservation law of certain range, given above, is based on an infinite presentation of the corresponding semigroup. A standard theorem from the theory of semigroups states that any finitely generated commutative semigroup has a finite presentation (see e.g. [7]). A question arises that how one can find such a finite presentation. A finite presentation is needed if, for example, we want to algorithmically verify whether two configurations have the same total energy. It turns out there is no algorithm to construct such a finite presentation for the semigroup of the most general conservation law in 2- or higher-dimensional CA. In one-dimensional case, we can construct these semigroups effectively.

Let $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ be a CA with a quiescent state $q \in S$. Clearly, (q,q) is a quiescent state for the product $F \times F$. Let $\Phi = \{0,1\}$ be the Boolean semigroup with $a + b \triangleq a \vee b$, for any $a, b \in \Phi$. Define the range-$\{0\}$ energy valuation $\mu : S \times S \to \Phi$ with

$$\mu(a,b) = \begin{cases} 0 \text{ if } a = b, \\ 1 \text{ otherwise.} \end{cases} \tag{5}$$

The energy μ is conserved by $F \times F$, if and only if, F is injective on finite configurations. According to the Garden-of-Eden theorem [12, 13], F is injective on finite configurations if and only if it is surjective. However, when $d \geq 2$, it is undecidable whether a given d-dimensional CA is surjective [9]. Therefore no algorithm could verify, for a given F, whether $F \times F$ conserves μ.

Theorem 1. *There is no algorithm, that given a 2- or higher-dimensional CA F with state set S, and an energy valuation $\mu : S \to \Phi$ from a finitely presented commutative semigroup Φ, determines if F conserves μ.*

Corollary 1. *There is no algorithm, that given a 2- or higher-dimensional CA F, computes a finite presentation of the semigroup Φ_F and the energy valuation μ_F of the most general conservation law for F with window $\{0\}$.*

Let us now focus on one-dimensional CA. Let $F : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ be a 1d CA. Without loss of generality we assume that F has a neighborhood $[-l, r] = \{-l, -l+1, \ldots, r\}$ with $l + r \geq 0$. Let q be the designated quiescent state of F. Let $W \subseteq \mathbb{Z}$ be a finite set, and $\mu : S^W \to \Phi$ be an energy valuation on window W with values from a commutative monoid Φ. Again without loss of generality we assume that $W = [0, m) = \{0, 1, \ldots, m-1\}$.

For $k = l + r + m$, consider the k'th order De Bruijn representation $(B_k[S], \lambda)$ of F. This has a vertex q^k, with a loop edge q^{k+1} which is labeled by q^m. Any path corresponding to a q-finite configuration starts by circulating in this loop, and after possibly passing through a finite number of other edges, eventually returns back to this loop.

To each edge $u_0 u_1 \cdots u_k \in S^{k+1}$ let us assign two elements

$$\alpha(u_0 u_1 \cdots u_k) \triangleq \mu(u_0 u_1 \cdots u_{m-1}) \tag{6}$$

and

$$\beta(u_0 u_1 \cdots u_k) \triangleq \mu(v_l v_{l+1} \cdots v_{l+m-1}) \tag{7}$$

from Φ_F, where $v_l v_{l+1} \cdots v_{k-r} = \lambda(u_0 u_1 \cdots u_k)$ is the label of $u_0 u_1 \cdots u_k$. The total energy of a q-finite configuration x can be calculated by adding up the values of α over the edges of the corresponding bi-infinite path on $B_k[S]$. Likewise, the sum of β values on this path gives the total energy of $F(x)$. Note that the initial and final parts of such a path, where it is circulating in the loop q^{k+1} do not contribute to the total energy, because $\mu(q^m) = 0$. For any path $p = p_1 p_2 \cdots p_n$ (p_i is the i'th edge of the path), let us use the notation $\alpha(p)$ for the sum of the values of α over the edges of p; i.e.,

$$\alpha(p) \triangleq \sum_{i=1}^{n} \alpha(p_i) \tag{8}$$

and similarly for β.

The requirements imposed by the conservation of μ can now be translated in terms of the values of α and β over finite paths on the graph $B_k[S]$: The pair (Φ, μ) specifies a conservation law, if and only if, for any finite path p starting and ending at vertex q^k, $\alpha(p) = \beta(p)$.

Proposition 1. *Let G be a (finite, directed) graph with vertex set V and edge set E, and $\varDelta$ a finite symbol set. Let $\alpha, \beta : E \to \varDelta$ and $A, B \subseteq V$. Let $\varPhi$ be the largest commutative monoid generated by $\varDelta$, satisfying the equations*

$$\alpha(p) = \beta(p) \tag{9}$$

for any finite path p starting from A and ending at B. Then, there is an algorithmically constructible finite subset of the above equations, such that any commutative monoid generated by $\varDelta$ satisfying those equations is a factor of $\varPhi$.

From Prop. 1, we immediately obtain what we were after in this section:

Theorem 2. *For any one-dimensional CA F and any finite window $W \subseteq \mathbb{Z}$, the semigroup $\varPhi_F$ of the most general conservation law for F based on W is effectively finitely presentable.*

This does not say much about the realizable sub-monoid $\breve{\varPhi}_F \subseteq \varPhi_F$. For example, it is not even clear if $\breve{\varPhi}_F$ is finitely generated or not. Following a similar construction, however, one can decide whether $\breve{\varPhi}_F$ is trivial or not.

Proposition 2. *Let F be a one-dimensional CA, and $(\varPhi, \mu)$ a semigroup-valued conservation law for F. It is decidable whether $\breve{\varPhi}$ is trivial.*

5 Group-Valued Conservation Laws

Group-valued conservation laws are more easily tractable. There is a simple algorithm which tests whether a given CA F (of any dimension) conserves a given group-valued energy function ρ. (This is similar to the real-valued case; see e.g. [10]). In particular, for any fixed window W one can effectively construct the most general group-valued conservation law $(G_F^{(W)}, \rho_F^{(W)})$ based on W. The next challenge would be to classify all conservation laws, based on all windows. For example, given a CA, is it possible to decide if it has any conservation law at all? In this section we prove that the answer to the latter question is negative.

Lemma 1. *Let $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ be a 1d CA with a designated quiescent state q. Suppose that F is nilpotent over q-finite configurations. Then F does not have any non-trivial (real-valued/group-valued/semigroup-valued) conservation law.*

Theorem 3 (Blondel, Cassaigne and Nichitiu [2]). *Given a two-counter machine A with 2 counters and no halting state, it is undecidable whether A has a periodic orbit.*

Theorem 4. *There is no algorithm that given a one-dimensional cellular automaton F determines if F has a non-trivial (real-valued/group-valued/semigroup-valued) conservation law.*

Sketch of the proof. We show how to reduce the problem of whether a given counter machine has a periodic orbit to finding out if a 1d CA has any non-trivial conservation law. Since the former is undecidable, we conclude that so is the latter.

Let A be a two-counter machine with state set Q, two registers x_1 and x_2, and transition function $\delta : Q \times \{0,1\}^2 \to Q \times \{1,2\} \times \{-,0,+\}$. We construct a CA F with a designated quiescent state q such that

a) if A has a periodic configuration, F has a non-trivial (real-valued) conservation law, while
b) if A has no periodic orbit, F is nilpotent over q-finite configurations (hence, has no non-trivial conservation law).

The CA F has two states L and R which are *end-markers*. In the interval between a left end-marker L and a right end-marker R, the CA simulates the machine A. The CA also constantly verifies the syntax of the block between two end-markers, to make sure it corresponds with an actual simulation. If a syntax error is found, or if the simulation overflows the end-markers, the CA erases the whole block, by replacing the cell contents with q. Blocks not having end-markers are also erased.

If the machine A has no periodic orbit, every syntactically correct simulation of A on a finite block eventually overflows the boundaries. Therefore, every q-finite configuration eventually goes quiescent; i.e., F is nilpotent over q-finite configurations.

On the other hand, if A has a periodic configuration, one can choose a sufficiently large simulation block in F which evolves periodically, and never overflows. Let us fix a snapshot of such a periodic simulation block (including its end-markers), and denote it by B. Since no new end-marker is ever created by F, and since the end-markers block the flow of information inwards and outwards the simulation blocks, we can argue that if a simulation block in a configuration eventually turns into B, it does so independent of the rest of the configuration, and after a bounded number of steps. Let us denote by $\mathcal{B}$ the set of all simulation blocks that eventually turn into B. This is a stable set. Once we know that a block from $\mathcal{B}$ occurs in a certain position on a configuration c, we also know that a block from $\mathcal{B}$ occurs in the same position on any pre-image of c. This immediately gives a non-trivial (real-valued) conservation law that states that the number of occurrences of blocks from $\mathcal{B}$ is conserved by F. □

6 Conclusion and Open Problems

In this paper we examined a number of algorithmic problems that arise from studying the algebraic conservation laws for cellular automata. The semigroup-valued conservation laws are highly expressive, still not so tractable. The group-valued conservation laws are more expressive than the real-valued ones, yet as accessible as they are.

Since reversible CA are particularly attractive for modeling physical processes, it would be useful to examine the same problems in the restricted case of reversible CA. In particular, is it decidable whether a given *reversible* CA has any conservation law? Notice that our proof of Theorem 4 takes advantage of the existence of very long transients to construct CA whose conservation laws need very large windows. We conjecture that every reversible CA has a non-trivial conservation law with a relatively small window.

References

[1] Biryukov, A.P.: Some algorithmic problems for finitely defined commutative semigroups. Siberian Mathematical Journal 8, 384–391 (1967)

[2] Blondel, V.D., Cassaigne, J., Nichitiu, C.: On the presence of periodic configurations in Turing machines and in counter machines. Theoretical Computer Science 289, 573–590 (2002)

[3] Boccara, N., Fukś, H.: Number-conserving cellular automaton rules. Fundamenta Informaticae 52, 1–13 (2002)

[4] Conway, J.H., Lagarias, J.C.: Tiling with polyominoes and combinatorial group theory. Journal of Combinatorial Theory A 53, 183–208 (1990)

[5] Durand, B., Formenti, E., Róka, Z.: Number conserving cellular automata I: decidability. Theoretical Computer Science 299, 523–535 (2003)

[6] Formenti, E., Grange, A.: Number conserving cellular automata II: dynamics. Theoretical Computer Science 304, 269–290 (2003)

[7] Grillet, P.A.: Semigroups: An Introduction to the Structure Theory. Dekker, New York (1995)

[8] Hattori, T., Takesue, S.: Additive conserved quantities in discrete-time lattice dynamical systems. Physica D 49, 295–322 (1991)

[9] Kari, J.: Reversibility and surjectivity problems of cellular automata. Journal of Computer and System Sciences 48, 149–182 (1994)

[10] Kari, J.: Theory of cellular automata: A survey. Theoretical Computer Science 334, 3–33 (2005)

[11] Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)

[12] Moore, E.F.: Machine models of self-reproduction. In: Proceedings of Symposia in Applied Mathematics, pp. 17–33. AMS (1962)

[13] Myhill, J.: The converse of Moore's Garden-of-Eden theorem. Proceedings of the American Mathematical Society 14, 685–686 (1963)

[14] Pivato, M.: Conservation laws in cellular automata. Nonlinearity 15, 1781–1793 (2002)

[15] Robison, A.D.: Fast computation of additive cellular automata. Complex Systems 1, 211–216 (1987)

[16] Thurston, W.P.: Conway's tiling groups. American Mathematical Monthly 97, 757–773 (1990)

Lower Bounds on Frequency Estimation of Data Streams[⋆] (Extended Abstract)

Sumit Ganguly

Indian Institute of Technology, Kanpur

Abstract. We consider a basic problem in the general data streaming model, namely, to estimate a vector $f \in \mathbb{Z}^n$ that is arbitrarily updated (i.e., incremented or decremented) coordinate-wise. The estimate $\hat{f} \in \mathbb{Z}^n$ must satisfy $\|\hat{f} - f\|_\infty \le \epsilon \|f\|_1$, that is, $\forall i \; (|\hat{f}_i - f_i| \le \epsilon \|f\|_1)$. It is known to have $\tilde{O}(\epsilon^{-1})$ randomized space upper bound [6], $\Omega(\epsilon^{-1} \log(\epsilon n))$ space lower bound [4] and deterministic space upper bound of $\tilde{\Omega}(\epsilon^{-2})$ bits.[1] We show that any deterministic algorithm for this problem requires space $\Omega(\epsilon^{-2}(\log \|f\|_1)(\log n)(\log^{-1}(\epsilon^{-1})))$ bits.

1 Introduction

A data stream σ over the domain $[1, n] = \{1, 2, \ldots, n\}$ is modeled as a sequence of records of the form $(pos, i, \delta v)$, where, pos is the current sequence index, $i \in [1, n]$ and $\delta v \in \{+1, -1\}$. Here, $\delta v = 1$ signifies an insertion of an instance of i and $\delta v = -1$ signifies a deletion of an instance of i. For each data item $i \in [1, n]$, its frequency $(\mathsf{freq}\ \sigma)_i$ is defined as $\sum_{(pos,i,\delta v)\ \in\ \text{stream}} \delta v$. The *size* of σ is defined as $|\sigma| = \max\{\|\mathsf{freq}\ \sigma'\|_\infty \mid \sigma' \text{ prefix of } \sigma\}$. In this paper, we consider the *general stream model*, where, the n-dimensional frequency vector $\mathsf{freq}\ \sigma \in \mathbb{Z}^n$. The data stream model of processing permits online computations over the input sequence using sub-linear space. The data stream computation model has proved to be a viable model for a number of application areas, such as network monitoring, databases, financial data processing, etc..

We consider the problem ApproxFreq(ϵ): given a data stream σ, return $\hat{f}$, such that $err(\hat{f}, \mathsf{freq}\ \sigma) \le \epsilon$, where, the function err is given by (1). Equivalently, the problem may be formulated as: given $i \in [1, n]$, return $\hat{f}_i$ such that $|\hat{f}_i - (\mathsf{freq}\ \sigma)_i| \le \epsilon \cdot \|\mathsf{freq}\ \sigma\|_1$, where, $\|f\|_1 = \sum_{i \in [1,n]} |f_i|$.

$$err(\hat{f}, f) \overset{\text{def}}{=} \frac{\|\hat{f} - f\|_\infty}{\|f\|_1} \le \epsilon. \tag{1}$$

The problem ApproxFreq(ϵ) is of fundamental interest in data streaming applications. For general streams, this problem is known to have a space lower bound

[⋆] The full version of the paper together with all proofs that could not be included here due to lack of space may be found at arXiv:cs/0701004v3.

[1] The $\tilde{O}$ and $\tilde{\Omega}$ notations suppress poly-logarithmic factors in $n, \log \epsilon^{-1}, \|f\|_\infty$ and $\log \delta^{-1}$, where, δ is the error probability (for randomized algorithm).

of $\Omega(\epsilon^{-1}\log(n\epsilon))$ [4], a randomized space upper bound of $\tilde{O}(\epsilon^{-1})$ [6], and a deterministic space upper bound of $\tilde{O}(\epsilon^{-2})$ bits [9]. For insert-only streams (i.e., $\mathsf{freq}\ \sigma \geq 0$), there exist deterministic algorithms that use $O((\epsilon^{-1})(\log(mn)))$ space [7,14,15]; however extensions of these algorithms to handle deletions in the stream are not known.

Mergeability. Data summary structures for summarizing data streams for frequency dependent computations (e.g., approximate frequent items, frequency moments, etc.; formally defined in Section 2) typically exhibit the property of *arbitrary mergeability.* If D is a data structure for processing a stream and D_j, $j = 1, \ldots, k$ for k arbitrary, be the respective current state of the structure after processing streams S_j, then, there exists a simple operation *Merge* such that *Merge*$(D_1, \ldots, D_k)$ reconstructs the state of D that would be obtained by processing the union of streams S_j, $j = 1, 2, \ldots, k$. For randomized summaries, this might require initial random seeds to be shared. Thus, a summary of a distributed stream can be constructed from the summaries of the individual streams, followed by the *Merge* operation. Almost all known data streaming structures are arbitrarily mergeable, including, sketches [3], Countsketch [5], Count-Min sketches [6], Flajolet-Martin sketches [8] and its variants, k-set[10], CR-precis structure [9] and random subset sums [13]. In this paper, we ask the question, namely, when are stream summaries mergeable?

Contributions. We present a space lower bound of $\Omega(\epsilon^{-2}(\log\epsilon^{-1})^{-1}(\log m)(\log n))$ bits for any *deterministic uniform* algorithm A_n for the problem ApproxFreq(ϵ) over input streams of size m over the domain $[1, n]$. The uniformity is in the sense that A_n must be able to solve ApproxFreq(ϵ) for all *general* input streams over the domain $[1, n]$. The lower bound implies that the CR-precis structure [9] is nearly space-optimal for ApproxFreq(ϵ), up to poly-logarithmic factors. The uniformity requirement is essential since there exists an algorithm that solves ApproxFreq(ϵ) for all input streams σ with $|\sigma| \leq 1$ using space $O(\epsilon^{-1}\text{polylog}(n))$ [11].

We also show that for any deterministic and uniform algorithm A_n over general streams, there exists another algorithm B_n such that (a) the state of B_n is arbitrarily mergeable, (b) B_n uses at most $O(\log n)$ bits of extra space than A_n, and, (c) for every input stream σ, the output of B_n on σ is the same as the output of A_n on some stream σ' such that $\mathsf{freq}\ \sigma = \mathsf{freq}\ \sigma'$. In other words, if A_n correctly solves a given frequency dependent problem, so does B_n; further, the state of B_n is arbitrarily mergeable and B_n uses $O(\log n)$ bits of extra space. This shows that deterministic data stream summaries for frequency dependent computation are essentially arbitrarily mergeable.

2 Stream Automaton

In this section, we define a stream automaton and study some basic properties.

Definition 1 (Stream Automaton). *A stream automaton A_n over the domain $[1, n]$ is a deterministic Turing machine that uses two tapes, namely, a* *two-way read-write work-tape and a one-way read-only input tape. The input tape contains the input stream σ. After processing its input, the automaton writes an output, denoted by output$_{A_n}(\sigma)$, on the work-tape.* $\square$

Effective space usage. We say that a stream automaton uses space $s(n,m)$ bits if *for all* input streams σ having $|\sigma| \leq m$, the number of cells (bits) on the work-tape in use, after having processed σ, is bounded by $s(n,m)$. In particular, this implies that for $m \geq m'$, $s(n,m) \geq s(n,m')$. The space function $s(n,m)$ does not count the space required to actually write the answer on the work-tape, or to process the $s(n,m)$ bits of the work-tape once the end of the input tape is observed. The proposed model of stream automata is non-uniform over the domain size n, (and uniform over the stream size parameter $m = |\sigma|$), since, for each $n \geq 1$, there is a stream automata A_n for solving instances of a problem over domain size n. This creates a problem in quantifying *effective space usage*, particularly, for low-space computations, that is, $s(n,m) = o(n \log m)$. Let $Q(A_n)$ denote the set of states in the finite control of the automaton A_n. If $|Q(A_n)| \geq m2^n$, then, for all $m' \leq m$, the automaton can map the frequency vector isomorphically into its finite control, and $s(n,m) = 0$. This problem is caused by non-uniformity of the model as a function of the domain size n, and can be avoided as follows. We define the effective space usage of A_n as

$$\text{Space}(A_n, m) \overset{\text{def}}{=} s(n,m) + \log s(n,m) + |Q(A_n)| \enspace .$$

Although, the model of stream automata does not explicitly allow queries, this can be modeled by a stream automaton's capability of writing vectors as answers, whose space is not counted towards the effective space usage. So if $\{q_i\}_{i \in I}$ denotes the family of all queries that are applicable for the given problem, where, I is a finite index set of size $p(n)$ then, the output of the automaton can be thought of as the $p(n)$-dimensional vector output$_{A_n}(\sigma)$.

A *frequency dependent problem* over a data stream is characterized by a family of binary predicates $P_n(\hat{f}, \mathsf{freq}\ \sigma)$, $\hat{f} \in \mathbb{Z}^{p(n)}$, $n \geq 1$, called the characteristic predicate for the domain $[1,n]$. P_n defines the acceptability (or good approximations) of the output. A stream automaton A_n *solves* a problem provided, for every stream σ, $P_n(\text{output}_{A_n}(\sigma), \mathsf{freq}\ \sigma)$ holds. For example, the characteristic predicate corresponding to the problem ApproxFreq(ϵ) is $err(\hat{f}, f) \leq \epsilon$, where, $\hat{f} \in \mathbb{Z}^n$ and $err(\cdot,\cdot)$ is defined by (1). Examples of frequency dependent problems are approximating frequencies and finding frequent items, approximate quantiles, histograms, estimating frequency moments, etc..

Given stream automata A_n and B_n, B_n is said to be an *output restriction* of A, provided, for every stream σ, there exists a stream σ' such that, $\mathsf{freq}\ \sigma = \mathsf{freq}\ \sigma'$ and $\text{output}_{B_n}(\sigma) = \text{output}_{A_n}(\sigma')$. The motivation of this definition is the following straightforward lemma.

Lemma 1. *Let P_n be the characteristic predicate of a frequency-dependent problem over data streams and suppose that a stream automaton A_n solves P_n. If B_n is an output restriction of A_n, then, B_n also solves P_n.* $\square$

Notation. Fix a value of the domain size $n \geq 2$. Each stream record of the form $(i, 1)$ and $(i, -1)$ is equivalently viewed as e_i and and $-e_i$ respectively, where, $e_i = [0, \ldots, 0, 1 \text{ (position } i), 0 \ldots, 0]$ is the i^{th} standard basis vector of $\mathbb{R}^n$. A stream is thus viewed as a sequence of elementary vectors (or its inverse). The notation $\sigma \circ \tau$ refers to the stream obtained by concatenating the stream τ to the end of the stream σ. In this notation, $\mathsf{freq}\ e_i = e_i$, $\mathsf{freq}\ -e_i = -e_i$ and $\mathsf{freq}\ \sigma \circ \tau = \mathsf{freq}\ \sigma + \mathsf{freq}\ \tau$. The *inverse stream* corresponding to σ is denoted as σ^r and is defined inductively as follows: $e_i^r = -e_i$, $-e_i^r = e_i$ and and $(\sigma \circ \tau)^r = \tau^r \circ \sigma^r$. The configuration of A_n is modeled as the triple (q, h, w), where, q is the current state of the finite control of A_n, h is the index of the current cell of the work tape, and w is the current contents of the work-tape. The processing of each record by A_n can be viewed as a transition function $\oplus_{A_n}(a, v)$, where, a is the current configuration of A_n, and v is the next stream record, that is, one of the e_i's. The transition function is written in infix form as $a \oplus_{A_n} v$. We assume that $\oplus_{A_n}$ associates from the left, that is, $a \oplus_{A_n} u_1 \circ u_2$ means $(a \oplus_{A_n} u_1) \oplus_{A_n} u_2$. Given a stream automaton A_n, the space of possible configurations of A_n is denoted by $C(A_n)$. Let $C_m(A_n)$ denote the subset of configurations that are reachable from the initial state o and after processing an input stream σ with $|\sigma| = \|\mathsf{freq}\ \sigma\|_\infty \leq m$. We now define two sub-classes of stream automata.

Definition 2. *A stream automaton A_n is said to be **path independent**, if for each configuration s of A_n and input stream σ, $s \oplus_{A_n} \sigma$ is dependent only on* $\mathsf{freq}\ \sigma$ *and s. A stream automaton A_n is said to be **path reversible** if for every stream σ and configuration s, $s \oplus_{A_n} \sigma \circ \sigma^r = s$, where, σ^r is the* inverse *stream of σ.* □

Overview of Proof. The proof of the lower bound on the space complexity of ApproxFreq(ϵ) proceeds in three steps. A subclass of path independent stream automata, called *free automata* is defined and is proved to be the class of path independent automata whose transition function $\oplus_{A_n}$ can be modeled as a linear mapping of $\mathbb{R}^n$, with input restricted to $\mathbb{Z}^n$. We then derive a space lower bound for ApproxFreq(ϵ) for free automata (Section 4.1). In the second step, we show that a path independent automaton that solves ApproxFreq(ϵ) can be used to design a free automaton that solves ApproxFreq(4ϵ)(Section 4.2). In the third step, we prove that for any frequency-dependent problem with characteristic predicate P_n and a stream automaton A_n that solves it, there exists an output-restricted stream automaton B_n that also solves P_n, is path-independent, and, $\text{Space}(B_n, m) \leq \text{Space}(A_n, m) + O(\log n)$. This step has two parts— the property is first proved for the class of path-reversible automata A_n (Section 5) and then generalized to all stream automata (Section 6). Combining the results of the three steps, we obtain the lower bound.

3 Path-Independent Stream Automata

In this section, we study the properties of path independent automata. Let A_n be a path-independent stream automaton over the domain $[1, n]$ and let $\oplus$ abbreviate $\oplus_{A_n}$. Define the function $+ : \mathbb{Z}^n \times C(A_n) \rightarrow C(A_n)$ as follows.

$$x + a = a \oplus \sigma \text{ where, } \mathsf{freq}\ \sigma = x.$$

Since A_n is a path independent automaton, the function $x + a$ is well-defined. The kernel M_{A_n} of a path independent automaton is defined as follows. Let the initial configuration be denoted by o.

$$M_{A_n} = \{x \in \mathbb{Z}^n \mid x + o = 0 + o\}$$

The subscript A_n in M_{A_n} is dropped when A_n is clear from the context.

Lemma 2. *The kernel of a path independent automaton is a sub-module of $\mathbb{Z}^n$.*

Proof. Let $x \in M$. Then, $0 + o = -x + x + o = -x + o$, or $-x \in M$. If $x, y \in M$, then, $0 + o = x + o = x + y + o$, or, $x + y \in M$. So M is a sub-module of $\mathbb{Z}^n$. □

The quotient set $\mathbb{Z}^n/M = \{x + M \mid x \in \mathbb{Z}^n\}$ together with the well-defined addition operation $(x + M) + (y + M) = (x + y) + M$, forms a module over $\mathbb{Z}$.

Lemma 3. *Let M be the kernel of a path independent automaton A_n. The mapping $x + M \mapsto x + o$ is a set isomorphism between $\mathbb{Z}^n/M$ and the set of reachable configurations $\{x + o \mid x \in \mathbb{Z}^n\}$. The automaton A_n gives the same output for each $y \in x + M$, $x \in \mathbb{Z}^n$.*

Proof. $y \in x + M$ iff $x - y \in M$ or $-y + x + o = o$, or, $x + o = y + o$. Thus, A_n attains the same configuration after processing both x and y and therefore A_n gives the same output for both x and y. Since, $x + o = y + o$ iff $x - y \in M$, which implies that the mapping $x + M \mapsto x + o$ is an isomorphism. □

Let $\mathbb{Z}^n_m$ denote the subset $\{-m, \ldots, m\}^n$ of $\mathbb{Z}^n$.

Lemma 4. *Let A_n be a path independent automaton with kernel M. Then,*

$$\mathit{Space}(A_n, m) \geq \lceil\ \log|\{x + M \mid x \in \mathbb{Z}^n_m\}|\ \rceil \geq (n - \dim M)\log(2m + 1).$$

Proof. The set of distinct configurations of A_n after it has processed a stream with frequency $x \in \mathbb{Z}^n_m$ is isomorphic to $\{x + M \mid x \in \mathbb{Z}^n_m\}$. The number of configurations using workspace of $s = s(n, m)$ is at most $|Q_{A_n}| \cdot s \cdot 2^s$. Therefore,

$$2^{\text{Space}(A_n, m)} = |Q_{A_n}| \cdot s \cdot 2^s \geq \left|\{x + M \mid x \in \mathbb{Z}^n_m\}\right| \ . \tag{2}$$

We now obtain an upper bound on the size $|M \cap \mathbb{Z}^n_m|$. Let $b_1, b_2, \ldots, b_r$ be a basis for M. The set

$$P_m = \{\alpha_1 b_1 + \ldots + \alpha_r b_r \mid |\alpha_i| \leq m \text{ and integral}, i = 1, 2, \ldots, n\}$$

defines the set of all integral points generated by $b_1, b_2, \ldots, b_r$ with multipliers in $\{-m, \ldots, m\}$. Thus,

$$|M \cap \mathbb{Z}^n_m| \leq |P_m| = (2m + 1)^r \ . \tag{3}$$

It follows that

$$\left|\{x + M \mid x \in \mathbb{Z}_m^n\}\right| \geq \frac{|\mathbb{Z}_m^n|}{|M \cap \mathbb{Z}_m^n|} \geq (2m+1)^{n-r} \ .$$

Since, $r = \dim M$, substituting in (2) and taking logarithms, we have

$$\text{Space}(A_n, m) \geq \log\left|\{x + M \mid x \in \mathbb{Z}_m^n\}\right| \geq (n-r)\log(2m+1) \ . \qquad \square$$

Lemma 5 shows that given a sub-module M, a path-independent automaton with a given M as a kernel can be constructed using nearly optimal space. The transition function $(x + M) + (y + M) = (x + y) + M$ implies that the state of a path independent automaton is arbitrarily mergeable.

Lemma 5. *For any sub-module M of $\mathbb{Z}^n$, one can construct a path-independent automaton with kernel M that uses nearly optimal space $s(n,m) = \log|\{x + M \mid x \in [-m \dots m]^n\}| + O(\log n)$ and uses $n^{O(1)}$ states in its finite control.* □

Proof. Let M be a given sub-module of $\mathbb{Z}^n$ with basis $b_1, \dots, b_r$ (say). It is sufficient to construct a path independent automaton whose configurations are isomorphic to $E = \mathbb{Z}^n/M$. Since, $\mathbb{Z}^n$ is free, $\mathbb{Z}^n/M$ is finitely generated using any basis of $\mathbb{Z}^n$. Therefore, the basic module decomposition theorem states that

$$\mathbb{Z}^n/M = \mathbb{Z}/(q_1) \oplus \cdots \oplus \mathbb{Z}/(q_r) \ . \tag{4}$$

where, $q_1|q_2|\cdots|q_r$. (Here, $\oplus$ refers to the direct sum of modules.) The finite control of the automaton stores $q_1, \dots, q_r$ and the machinery required to calculate 1 mod q_j and -1 mod q_j for each j. For the frequency vector f, the residue vector $f + M$ is maintained as a vector of residues with respect to the q_j's as given by (4). Since, (4) is a direct sum, hence, the space used by this representation is optimal and equal to $|\{x + M \mid x \in [-m \dots m]^n\}|$. □

Definition 3 (Free Automaton). *A path independent automaton A_n with kernel M is said to be free if $\mathbb{Z}^n/M$ is a free module.* □

Lemma 6 shows that the transition function $\oplus$ of a free automata can be represented as a linear mapping and is proved in the full version [12].

Lemma 6. *Let A_n be free automaton with kernel M. There exists a unique vector subspace M^e of $\mathbb{R}^n$ of the smallest dimension containing M. The mapping $x + M \mapsto x + M^e$ is an injective mapping from $\mathbb{Z}^n/M$ to $\mathbb{R}^n/M^e$. If $\dim \mathbb{Z}^n/M = r$, then, there exists an orthonormal basis $V = [V_1, V_2]$ of $\mathbb{R}^n$ such that* $\text{rank}(V_1) = r$, $\text{rank}(V_2) = n - r$, *M^e is the linear span of V_2 and $\mathbb{R}^n/M^e$ is the linear span of V_1.* □

4 Frequency Estimation

In this section, we present a space lower bound for ApproxFreq(ϵ) using path-independent automaton. Recall that a stream automaton A_n solves

ApproxFreq(ϵ), provided, after processing any input stream σ with $\mathsf{freq}\ \sigma = x$, A_n returns a vector $\hat{x} \in \mathbb{R}^n$ satisfying $err(\hat{x}, x) = \frac{\|\hat{x}-x\|_\infty}{\|x\|_1} \leq \epsilon$. In general, if an estimation algorithm returns the same estimate u for all elements of a set S, then, $err(u, S)$ is defined as $\max_{y \in S} err(u, y)$. Given a set S, let $\min_{\ell_1}(S)$ denote the element in S with the smallest ℓ_1 norm: $\min_{\ell_1}(S) = \mathrm{argmin}_{y \in S}\ \|y\|_1$.

Lemma 7. *If $S \subset \mathbb{Z}^n$ and there exists $h \in \mathbb{R}^n$ such that $err(h, S) \leq \epsilon$, then $err(\min_{\ell_1}(S), S) \leq 2\epsilon$.* □

4.1 Frequency Estimation Using Free Automata

In this section, let A_n be a free automaton with kernel M that solves the problem ApproxFreq(ϵ).

Lemma 8. *Let M be a sub-module of $\mathbb{Z}^n$. (1) if there exists h such that $err(h, M) \leq \epsilon$, then, $err(0, M) \leq \epsilon$, and, (2) if $err(0, M) \leq \epsilon$ then $err(0, M^e) \leq \epsilon$.*

For a free automaton A_n with kernel M and corresponding $n \times r$ orthonormal matrix V_1 whose columns are a basis for $\mathbb{R}^n / M^e$ (as given by Lemma 6), the minimum ℓ_2 estimator is defined as

$$\bar{x}_2 = est_2(x) = V_1 V_1^T x \ . \tag{5}$$

It is easy to show that the ℓ_2 estimator is well-defined. It is called the minimum ℓ_2 estimator since it returns a point in the coset $x + M^e$ that is closest to the origin in terms of the ℓ_2 distance. We now show that there is a subset J of the set of the standard unit vectors $\{e_1, e_2, \ldots, e_n\}$ such that $|J| = \Theta(n)$ and the minimum ℓ_2 estimator is nearly optimal for the unit vectors in J. We first prove a technical lemma.

Lemma 9. *For any real $C \geq 1$, let $J_C = \{i : 1 \leq i \leq n \text{ and } \|V_1 V_1^T e_i\|_1 \geq C\}$. Then, $|J_C| \leq \frac{n}{C}$.*

Proof. Since, V_1 has orthonormal columns, $\|V_1\|_2 = \|V_1 V_1^T\|_2 = 1$. By a standard identity between norms, we have $\|V_1 V_1^T\|_F \leq \sqrt{n} \|V_1 V_1^T\|_2 = \sqrt{n}$. Therefore,

$$|J_C| \cdot C \leq \sum_{i \in J_C} \|V_1 V_1^T e_i\|_1 \leq \|V_1 V_1^T\|_F^2 \leq n, \text{ or, } |J_C| \leq \frac{n}{C} \ .$$ □

Lemma 10. *Let A_n be a free automaton that solves* ApproxFreq(ϵ) *. Then $\exists\ J' \subset \{1, 2, \ldots, n\}, |J'| \geq \lceil n/2 \rceil$ such that for $i \in J'$, $err(est_2(e_i), e_i) \leq 3\epsilon$.*

Proof. For $C > 1$, let J'_C be the index set $J'_C = \{i : \|V_1 V_1^T e_i\|_1 < C\}$.

$$\begin{aligned} err(est_2(e_i), e_i) &= \frac{\|est_2(e_i) - e_i\|_\infty}{\|e_i\|_1} = \frac{\|est_2(e_i) - e_i\|_\infty}{\|est_2(e_i) - e_i\|_1} \cdot \|est_2(e_i) - e_i\|_1 \\ &= err(0, est_2(e_i) - e_i) \cdot \|est_2(e_i) - e_i\|_1 \ . \end{aligned}$$

The vector $w = est_2(e_i) - e_i \in M^e$. By Lemma 8, part (1), $err(0, M) \leq \epsilon$. By Lemma 8, part (2), $err(0, w) \leq err(0, M^e) \leq \epsilon$. Further, for $i \in J'_C$,

$$\|est_2(e_i) - e_i\|_1 \leq \|est_2(e_i)\|_1 + \|e_i\|_1 < C + 1$$

since, $i \in J'_C$ and $est_2(e_i) = V_1 V_1^T e_i$. Combining, for $i \in J'_C$

$$err(est_2(e_i), e_i) < \epsilon(C+1) < 3\epsilon$$

by choosing $C = 2$. By Lemma 9 it follows that $|J'_C| = n - |J_C| \geq n - n/2$. □

The following theorem is needed in the next proof.

Theorem 1 (Alon [1,2]). *There exists a positive constant c so that the following holds. Let B be an n by n real matrix with $b_{i,i} \geq \frac{1}{2}$ for all i and $|b_{i,j}| \leq \epsilon$ for all $i \neq j$, where, $\frac{1}{2\sqrt{n}} \leq \epsilon < \frac{1}{4}$. Then* $\text{rank}(B) \geq \frac{c \log n}{\epsilon^2 \log(1/\epsilon)}$. □

Lemma 11. *Let $\frac{1}{2\sqrt{n}} \leq \epsilon < \frac{1}{12}$ and let A_n be a free stream automaton that solves* ApproxFreq(ϵ). *Then, $n - \dim(M^e) = \Omega(\frac{\log n}{\epsilon^2 \log(\epsilon^{-1})})$.*

Proof. By Lemma 10, there exists $J' \subset \{1, 2, \ldots, n\}$ such that $|J'| \geq \lceil n/2 \rceil$ and $err_2(e_i) \leq 3\epsilon$ for $i \in J'$. Let V_1 be the $n \times r$ orthonormal matrix given by Lemma 6 spanning $\mathbb{R}^n/M^e$. Define Y to be the $|J'| \times r$ sub-matrix of V_1 that includes the ith row of V_1 for each $i \in J'$. Let $U = Y^T$ which is an $r \times |J'|$ matrix. For $i \in J'$,

$$err(est_2(e_i), e_i) \leq 3\epsilon, \text{ or, } \frac{\|V_1 V_1^T e_i - e_i\|_\infty}{\|e_i\|_1} = \|V_1 V_1^T e_i - e_i\|_\infty \leq 3\epsilon \ .$$

Therefore, for $j, k \in \{1, 2, \ldots, |J'|\}$, $|U_j^T U_k| \leq 3\epsilon$ if $j \neq k$ and $|U_j^T U_j - 1| \leq 3\epsilon$. The matrix UU^T satisfies the premises of Theorem 1. Therefore,

$$n - \dim(M^e) = \text{rank}(V_1) \geq \text{rank}(U) = \text{rank}(UU^T) = \Omega\left(\frac{\log n}{\epsilon^2 \log \epsilon^{-1}}\right) \ . \qquad \square$$

Lemma 12. *Let $\frac{1}{2\sqrt{n}} \leq \epsilon < \frac{1}{12}$. Suppose A_n be a free automaton that uses $s(n,m)$ bits on the work-tape to solve* ApproxFreq(ϵ). *Then, $s(n,m) = \Omega(\frac{(\log n)(\log m)}{\epsilon^2 \log \epsilon^{-1}})$.*

Proof. Let $M =$ kernel of A_n. By Lemma 11, $n - \dim M^e = \Omega\left(\frac{\log n}{\epsilon^2 (\log \epsilon^{-1})}\right)$. By Lemma 4, $s(n,m) = \Omega((n - \dim M) \log m)$. Since, $\dim M = \dim M^e$, the result follows. □

4.2 General Path Independent Automata

We now show that for the problem ApproxFreq(ϵ), it is sufficient to consider free automata. Let A_n be a path-independent automaton that solves ApproxFreq(ϵ) and has kernel M. Suppose that $\mathbb{Z}^n/M$ is not free. Let M' be the module that removes the torsion from $\mathbb{Z}^n/M$, that is, $M' = \{x \in \mathbb{Z}^n \mid \exists a \in \mathbb{Z}, a \neq 0 \text{ and } ax \in M\}$.

Lemma 13. *$\mathbb{Z}^n/M'$ is torsion-free.* □

Fact 14. *Let $b_1, b_2, \ldots, b_r$ be a basis of M'. Then, $\exists\ \alpha_1, \ldots, \alpha_r \in \mathbb{Z} - \{0\}$ such that $\alpha_1 b_1, \ldots, \alpha_r b_r$ is a basis for M. Hence, $M^e = (M')^e$.* □

We show that if a path independent automaton with kernel M can solve ApproxFreq(ϵ), then a free automaton with kernel $M' \supset M$ can solve ApproxFreq(4ϵ).

Lemma 15. *Suppose A_n is a path independent automaton for solving* ApproxFreq(ϵ) *and has kernel M. Then, there exists a free automaton B_n with kernel M' such that $M' \supset M$, $\mathbb{Z}^n/M'$ is free, and $err(\min_{\ell_1}(x + M'), x) \leq 4\epsilon$.*

Lemma 16. *Suppose $\frac{1}{2\sqrt{n}} \leq \epsilon < \frac{1}{48}$. Let A_n be a path independent automaton that solves* ApproxFreq (ϵ). *If A_n has kernel M, then, $n - \dim M = \Omega\left(\frac{\log n}{\epsilon^2 \log(1/\epsilon)}\right)$.*

Proof. By Lemma 15, there exists a free automaton A'_n with kernel $M' \supset M$ that solves ApproxFreq(4ϵ). Therefore, $n - \dim M \geq n - \dim M' = \Omega\left(\frac{\log n}{\epsilon^2 \log \epsilon^{-1}}\right)$, by Lemma 12. □

5 Path Reversible Automata

In this section, we show that given a path reversible automaton A_n, one can construct a path independent automaton B_n that is an output restriction of A_n and Space$(B_n, m) \leq$ Space$(A_n, m) + O(\log n)$. Let A_n be a path reversible automaton. For $f \in \mathbb{Z}^n$, define $\phi_{A_n}(f) = \{s \mid \exists \sigma \text{ s.t. } o \oplus \sigma = s \text{ and } \mathsf{freq}\ \sigma = f\}$. The kernel of A_n is defined as follows: $M = M_{A_n} = \{f \mid o \in \phi_{A_n}(f)\}$. Let $C = C(A_n)$ be the set of reachable configurations from the initial state o of A_n and let $C_m = C_m(A_n)$ denote the subset of $C(A_n)$ that are reachable from the initial state o on input streams σ with $|\sigma| \leq m$. Define a binary relation over C as follows: $s \sim t$ if there exists $f \in \mathbb{Z}^n$ such that $s, t \in \phi_{A_n}(f)$.

Lemma 17. *1. M is a sub-module of $\mathbb{Z}^n$.*
2. If $f - g \in M$ then $\phi_{A_n}(f) = \phi_{A_n}(g)$, and, if $\phi_{A_n}(f) \cap \phi_{A_n}(g)$ is non-empty, then, $f - g \in M$.
3. The relation $\sim$ over C is an equivalence relation.
4. The map $[s] \mapsto f + M$, for $s \in \phi_{A_n}(f)$, is well-defined, 1-1 and onto.

Let B_n be a path independent stream automaton whose configurations are the set of cosets of M and whose transition is defined as by the sum of the cosets, that is, $f + (x + M) = (f + x) + M$, constructed using Lemma 5. Its output on an input stream σ is defined as:

$$\text{output}_{B_n}(\sigma) = \text{choice}\ \{\text{output of } A_n \text{ in configuration } s \mid s \in \phi_{A_n}(\mathsf{freq}\ \sigma)\}$$

where, choice S returns some element from its argument set S.

Lemma 18. *B_n is an output restriction of A_n.*

We can now prove the main lemma of the section.

Lemma 19. *Let A_n be a path reversible automaton with kernel M. Then, there exists a path independent automaton B_n with kernel M that is an output restriction of A_n such that $\log|C_m(A_n)| + O(\log n) \geq Space(B_n, m)$, for $m \geq 1$.*

Proof. Let B_n be constructed in the manner described above. By Lemma 18, is an output-restriction of A_n. Since the map $[s] \rightarrow f + M$, for $s \in \phi_{A_n}(f)$ is 1-1 and onto (Lemma 17, part 4), therefore, for every m, each reachable configuration of B_n after processing streams σ with $\mathsf{freq}\ \sigma \in [-m \ldots m]^n$ can be associated with a disjoint aggregate of configurations of A_n. The number of reachable configurations of B_n after processing streams with frequency in $[-m \ldots m]^n$ is $|\{x + M \mid x \in [-m \ldots m]^n\}|$. Thus, $|C(A_n)| \geq |\{x + M \mid x \in [-m \ldots m]^n\}|$. By Lemma 5, $\text{Space}(B_n, m) = \log\ |\{x + M \mid x \in [-m \ldots m]^n\}| + O(\log n)$. Combining, we obtain the statement of the lemma. □

Remarks. The above procedure transforms a path reversible automaton A_n to a path-independent automaton B_n such that $\log|C_m(A_n)| + O(\log n) \geq$ Space (B_n, m), for all $m \geq 1$. However, the arguments only use the property that the transition function $\oplus_{A_n}$ is path reversible, and the fact that the subset of reachable configurations $C_m(A_n)$ on streams of size at most m is finite. The argument is more general and also applies to computation performed by an infinite-state deterministic automaton in the classical sense that returns an output after it sees the end of its input, with set of states C, initial state o and a path-reversible transition function $\oplus'_{A_n}$. The above argument shows that such an automaton A_n can be simulated by a path-independent stream automaton B_n with finite control and additional space overhead of $O(\log n)$ bits, such that B_n is an output-restriction of A_n. We will use this observation in the next section.

6 Path Non-reversible Automata

In this section, we show that corresponding to every general stream automaton A_n, there exists a path reversible automaton A'_n that is an output-restriction of A'_n, such that $\text{Space}(A_n, m) \geq \log|C_m(A'_n)|$. By Lemma 19, corresponding to any path reversible automaton A'_n, there exists an output-restricted and path independent automaton B_n, such that $\log|C_m(A'_n)| \geq \text{Space}(B_n, m) - O(\log n)$. Together, this proves a basic property of stream automata, namely, that, for every stream automaton A_n, there exists a path-independent stream automaton B_n that is an output-restriction of A_n and $\text{Space}(B_n, m) \leq \text{Space}(A_n, m) + O(\log n)$. We construct the path-reversible automaton A'_n only to the extent of designing a path-reversible transition function $\oplus_{A'_n}$, a set of configurations $C(A'_n)$ and specifying the output of A'_n if the end of the stream is met while at any $s \in C(A'_n)$. As per the remarks at the end of the previous section, this is sufficient to enable the construction of the path-independent automaton B_n

from A'_n. We explain the basic idea here; a formal presentation is given in the full version [12].

We construct a transition function $\oplus'$ from $\oplus$ over a space $\{\alpha(s) \mid s \in C\}$, where, $\alpha(s)$ is an equivalence class over C (see [12] for details; informally, $s \sim_\alpha t$, if there exist infinitely many pairs of streams σ, σ' such that $s \oplus \sigma = t \oplus \sigma'$ and $\mathsf{freq}\ \sigma = \mathsf{freq}\ \sigma' = 0$). A transition function $\oplus'$ is now constructed such that (a) $s \oplus' \sigma = \alpha(s \oplus \sigma)$, and, (b) $\oplus'$ is path reversible, stated in Lemma 20.

Lemma 20. *For $s \in C$, $\alpha(s) \oplus' e_i \circ -e_i = \alpha(s)$ and $\alpha(s) \oplus' -e_i \circ e_i = \alpha(s)$.* □

A path reversible automaton A'_n is defined as follows. Initially A'_n is in the state $\alpha(o)$. After reading a stream record (one of the e_i's or $-e_i$'s), A'_n uses the transition function $\oplus'$ instead of $\oplus$ to process its input. However, $s \oplus' \sigma = \alpha(s \oplus \sigma)$, where, $\alpha(t)$ is a set (possibly infinite) of states that cause A_n to transit from configuration t on some input σ', with $\mathsf{freq}\ \sigma' = 0$. *Equivalently, this can be interpreted as if σ' has been inserted into the input tape just after A_n reaches the configuration s and before it processes the next symbol–hence, A'_n is an output-restriction of A_n and is equally correct for frequency-dependent computations. This is the main idea of this construction.* Thus, transitions of $\oplus'$ are equivalent to inserting some specifically chosen strings $\sigma_1, \sigma_2, \ldots$, each having $\mathsf{freq}\ = 0$, after reading each letter (i.e., $\pm e_i$) of the input. The output of A'_n on input stream σ is identical to the output of A_n on the stream σ', where, σ' is obtained by inserting zero frequency sub-streams into it. Therefore, $\mathsf{freq}\ (\sigma') = \mathsf{freq}\ (\sigma)$ and A'_n is an output restriction of A_n. By Lemma 20, the transition function $\oplus'$ is path reversible. Let $C' = C(A'_n)$ and $C'_m = C_m(A'_n)$. Since, $\alpha(s)$ is an equivalence class over $C(A_n)$, the map $s \mapsto \alpha(s)$ implies that $|C'_m| = |\{\alpha(s) \mid s \in C_m\}| \leq |C_m|$. Starting from A'_n, one can construct a path independent automaton B_n as per the discussion in Section 5. Theorem 2 summarizes this discussion.

Theorem 2 (Basic property of computations using stream automata). *For every stream automaton A_n, $\exists$ a path-independent stream automaton B_n that is an output-restriction of A_n and Space$(B_n, m) \leq$ Space$(A_n, m) + O(\log n)$.*

Proof. Let $\oplus'$ be the transition function of the path-reversible automaton constructed as described above and let B_n be the path-independent automaton obtained by translating $\oplus'$ using the procedure of Section 5. Let C_m and C'_m denote the number of reachable configurations of A_n and A'_n, respectively, over streams with frequency vector in $[-m \ldots m]^n$. Let $s_A = s_A(n, m)$. Let M be the kernel of B_n. Then,

$$|Q_A| s_A 2^{s_A} \geq |C_m| \geq |C'_m| \geq |\{x + M \mid x \in [-m \ldots m]^n\}| \geq (2m+1)^{n - \dim M}$$

where, the last two inequalities follow from Lemma 19. Taking logarithms, Space $(A_n, m) \geq \log |\{x + M \mid x \in [-m \ldots m]^n\}| \geq$ Space$(B_n, m) - O(\log n)$, by Lemma 5. □

Theorem 3. *Suppose that $\frac{1}{2\sqrt{n}} \leq \epsilon < \frac{1}{48}$ and let A_n be a stream automaton that solves* ApproxFreq(ϵ). *Then, Space$(A_n, m) = \Omega\left(\frac{\log m \log n}{\epsilon^2 \log(1/\epsilon)}\right)$.*

Proof. By Theorem 2, there exists a path independent automaton B_n that is an output-restriction of A_n and $\text{Space}(A_n, m) \geq \text{Space}(B_n, m) - O(\log n)$. By Lemma 1, B_n solves ApproxFreq(ϵ). If M is the kernel of B_n, then by Lemma 4, $\text{Space}(B_n) = \Omega((n - \dim M)(\log(2m+1)))$. By Lemma 16, $n - \dim M = \Omega\left(\epsilon^{-2}(\log(1/\epsilon))^{-1}\log n\right)$. Thus,

$$\text{Space}(A_n, m) = \Omega((n - \dim M)\log m) - O(\log n) = \Omega\left(\frac{(\log m)(\log n)}{\epsilon^2 \log(1/\epsilon)}\right) . \qquad \square$$

Since, any path-independent automaton is arbitrarily mergeable (see text before Lemma 5), Theorem 2 also establishes the claim regarding arbitrary mergeability made in Section 1.

Acknowledgements. The author thanks Piotr Indyk and Kranti Kumar Potanapalli for useful discussions.

References

1. Alon, N.: Problems and results in extremal combinatorics I. Discr. Math. 273(1-3), 31–53 (2003)
2. Alon, N.: Perturbed identity matrices have high rank: proof and applications (2006), `http://www.math.tau.ac.il/~nogaa/identity.pdf`
3. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating frequency moments. J. Comp. Sys. and Sc. 58(1), 137–147 (1998)
4. Bose, P., Kranakis, E., Morin, P., Tang, Y.: Bounds for Frequency Estimation of Packet Streams. In: Proc. SIROCCO, pp. 33–42 (2003)
5. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)
6. Cormode, G., Muthukrishnan, S.: An Improved Data Stream Summary: The Count-Min Sketch and its Applications. J. Algorithms 55(1)
7. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)
8. Flajolet, P., Martin, G.N.: Probabilistic Counting Algorithms for Database Applications. J. Comp. Sys. and Sc. 31(2), 182–209 (1985)
9. Ganguly, S., Majumder, A.: CR-precis: A Deterministic Summary Structure for Update Streams. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614. Springer, Heidelberg (2007)
10. Ganguly, S., Majumder, A.: Deterministic K-set Structure. In: Proc. ACM PODS, pp. 280–289 (2006), `www.cse.iitk.ac.in/users/sganguly`
11. Ganguly, S.: Distributed deterministic approximation of vector sums (November 2007) (manuscript)
12. Ganguly, S.: Lower bounds on frequency estimation of data streams. arXiv:cs/0701004v3 (February 2008)
13. Gilbert, A., Kotidis, Y., Muthukrishnan, S., Strauss, M.: How to Summarize the Universe: Dynamic Maintenance of Quantiles. In: Proc. VLDB, Hong Kong, August 2002, pp. 454–465 (2002)
14. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags. ACM TODS 28(1), 51–55 (2003)
15. Misra, J., Gries, D.: Finding repeated elements. Sci. Comput. Programm. 2, 143–152 (1982)

From Invariants to Canonization in Parallel

Johannes Köbler[1] and Oleg Verbitsky[2,⋆]

[1] Institut für Informatik, Humboldt Universität zu Berlin, D-10099 Berlin, Germany
koebler@informatik.hu-berlin.de
[2] Institute for Applied Problems of Mechanics and Mathematics, Naukova 3b, 79060 Lviv, Ukraine
verbitsky@informatik.hu-berlin.de

Abstract. A function f of a graph is called a *complete graph invariant* if two given graphs G and H are isomorphic exactly when $f(G) = f(H)$. If additionally, $f(G)$ is a graph isomorphic to G, then f is called a *canonical form* for graphs. Gurevich [9] proves that any polynomial-time computable complete invariant can be transformed into a polynomial-time computable canonical form. We extend this equivalence to the polylogarithmic-time model of parallel computation for classes of graphs having either bounded rigidity index or small separators. In particular, our results apply to three representative classes of graphs embeddable into a fixed surface, namely, to 3-connected graphs admitting either a polyhedral or a large-edge-width embedding as well as to all embeddable 5-connected graphs. Another application covers graphs with treewidth bounded by a constant k. Since for the latter class of graphs a complete invariant is computable in NC, it follows that graphs of bounded treewidth have a canonical form (and even a canonical labeling) computable in NC.

1 Introduction

We write $G \cong H$ to indicate that G and H are isomorphic graphs. A *complete invariant* is a function f on graphs such that $f(G) = f(H)$ if and only if $G \cong H$. If, in addition, $f(G)$ is a graph isomorphic to G, then f is called a *canonical form* for graphs. For a given graph G and a one-to-one map σ on the vertices of G, we use G^σ to denote the isomorphic image of G under σ. A *canonical labeling* assigns to each graph G a map σ so that the function f defined as $f(G) = G^\sigma$ is a complete invariant. Note that f is even a canonical form. Thus, the notion of a canonical labeling is formally stronger than that of a canonical form which in turn is formally stronger than that of a complete invariant.

Obviously, a polynomial-time computable complete invariant can be used to decide in polynomial time whether two given graphs are isomorphic. Conversely, it is not known whether a polynomial-time decision algorithm for graph isomorphism implies the existence of a polynomial-time complete invariant (cf. the discussion in [2, Sect. 5]). However, for many classes of graphs for which we

⋆ Supported by an Alexander von Humboldt fellowship.

have an efficient isomorphism test, we also have a canonical labeling algorithm of comparable complexity (see, e.g., [17,14]); but this often requires substantial additional efforts (cf., e.g., [3,1]).

Gurevich [9] proves that a polynomial-time computable complete graph invariant can be used to compute a canonical labeling in polynomial time. This result is really enlightening because there are approaches to the graph isomorphism problem which are based on computing a graph invariant and, without an extra work, do not provide us with a canonical form. An important example is the *k-dimensional Weisfeiler-Lehman algorithm* WL^k. Given an input graph G, the algorithm outputs a coloring of its vertices in polynomial time (where the degree of the polynomial bounding the running time depends on k). WL^k always produces the same output for isomorphic input graphs. Whether the algorithm is able to distinguish G from every non-isomorphic input graph H depends on whether the dimension k is chosen large enough for G. In particular, $k = 1$ suffices for all trees T. However, notice that the coloring computed by WL^1 on input T partitions the vertex set of T into the orbits of the automorphism group of T and hence WL^1 does not provide a canonical labeling unless T is rigid (i.e., T has only the trivial automorphism). We mention that an appropriate modification of the 1-dimensional Weisfeiler-Lehman algorithm to a canonical labeling algorithm is suggested in [11].

The reduction of a canonical labeling to a complete invariant presented in [9] (as well as in [11]) is inherently sequential and thus leaves open the following question.

Question 1. Suppose that for the graphs in a certain class C we are able to compute a complete invariant in NC. Is it then possible to compute also a canonical labeling for these graphs in NC?

For several classes of graphs, NC algorithms for computing a complete invariant are known (see, e.g., [4,17,14,10]). For example, in [10] it is shown that a k-dimensional Weisfeiler-Lehman algorithm making logarithmically many rounds can be implemented in $\mathrm{TC}^1 \subseteq \mathrm{NC}^2$ and that such an algorithm succeeds for graphs of bounded treewidth. Similar techniques apply also to planar graphs but for this class a canonical labeling algorithm in AC^1 is known from an earlier work [17]. Nevertheless, also in this case it is an interesting question whether the approach to the planar graph isomorphism problem suggested in [10], which is different from the approach of [17], can be adapted for finding a canonical labeling. Finally, Question 1 even makes sense for classes C for which we don't know of any NC-computable complete invariant since such an invariant may be found in the future.

We notice that a positive answer to Question 1 also implies that the search problem of computing an isomorphism between two given graphs in C, if it exists, is solvable in NC whenever for C we have a complete invariant in NC (notice that the known polynomial-time reduction of this search problem to the decision version of the graph isomorphism problem is very sequential in nature, see [12]).

As our main result we give an affirmative answer to Question 1 for any class of graphs having either small *separators* (Theorem 2) or bounded *rigidity index* (Theorem 5). A quite general example for a class of graphs having small separators is the class of graphs whose treewidth is bounded by a constant. Since, as mentioned above, a complete invariant for these graphs is computable in TC^1 [10], Theorem 2 immediately provides us with an NC (in fact TC^2) canonical labeling algorithm for such graphs (Corollary 3). As a further application we also get a TC^2 algorithm for solving the search problem for pairs of graphs in this class (Corollary 4).

Regarding the second condition we mention the following representative classes of graphs with bounded rigidity index:

– 3-connected graphs having a *large-edge-width* embedding into a fixed surface S (Corollary 7).[1]

– 3-connected graphs having a *polyhedral* embedding into a fixed surface S (Corollary 9).

– 5-connected graphs embeddable into a fixed surface S (Corollary 10).

As shown by Miller and Reif [17], the canonization problem for any hereditary class of graphs C (meaning that C is closed under induced subgraphs) AC^1 reduces to the canonization problem for the class of all 3-connected graphs in C. Thus, with respect to the canonization problem, the 3-connected case is of major interest.

The rest of the paper is organized as follows. In Sect. 2 we provide the necessary notions and fix notation. Graphs with small separators are considered in Sect. 3 and graphs with bounded rigidity index are considered in Sect. 4. Section 5 summarizes our results and discusses remaining open problems.

2 Preliminaries

The concept of polylogarithmic parallel time is captured by the hierarchy of complexity classes $\mathrm{NC} = \bigcup_{i\geq 1} \mathrm{NC}^i$, where NC^i consists of functions computable by DLOGTIME-constructible boolean circuits of polynomial size and depth $O(\log^i n)$. The class AC^i is the extension of NC^i to circuits with unbounded fan-in and TC^i is a further extension allowing threshold gates as well. Recall also that $\mathrm{AC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1$ and $\mathrm{NC}^i \subseteq \mathrm{AC}^i \subseteq \mathrm{TC}^i \subseteq \mathrm{NC}^{i+1}$, where L (resp. NL) is the set of languages accepted by (non)deterministic Turing machines using logarithmic space. Alternatively, the AC^i level of the NC hierarchy can be characterized as the class of all functions computable by a CRCW PRAM with polynomially many processors in time $O(\log^i n)$.

The vertex set of a graph G is denoted by $V(G)$. The set of all vertices adjacent to a vertex $v \in V(G)$ is called the *neighborhood* of v and denoted by $\Gamma(v)$.

A *colored graph* G, besides the binary adjacency relation, has unary relations $U_1, \ldots, U_n$ defined on $V(G)$. If a vertex v satisfies U_i, we say that v has color i. A vertex is allowed to have more than one color or none. It is supposed that the number of colors is equal to the number of vertices in a graph, though some

[1] This result is actually stated in a stronger form, without referring to a parameter S.

of the color relations may be empty. A colored graph $\langle G, U_1, \ldots, U_n \rangle$ will be called a *coloring* of the underlying graph G. An *isomorphism* between colored graphs must preserve the adjacency relation as well as the color relations. Thus, different colorings of the same underlying graph need not be isomorphic.

We consider only classes of graphs that are closed under isomorphism. For a given class of graphs C we use C^* to denote the class containing all colorings of any graph in C.

Let C be a class of graphs and let f be a function mapping graphs to strings over a finite alphabet. We call f a *complete invariant for C* if for any pair of graphs G and H in C we have $G \cong H$ exactly when $f(G) = f(H)$. A *canonical labeling for C* assigns to each graph G on n vertices a one-to-one map $\sigma : V(G) \to \{1, \ldots, n\}$ such that $f(G) = G^\sigma$ is a complete invariant for C. Note that a complete invariant f originating from a canonical labeling has an advantageous additional property: $f(G) \neq f(H)$ whenever G is in C and H is not. Moreover, it provides us with an isomorphism between G and H whenever $f(G) = f(H)$.

The notions of a complete invariant and of a canonical labeling are easily extensible to colored graphs. In our proofs, extending these notions to colored graphs will be technically beneficial and, at the same time, will not restrict the applicability of our results. In fact, any available complete-invariant algorithm for some class of graphs C can be easily extended to C^* without increasing the required computational resources. In particular, this is true for the parallelized version of the multi-dimensional Weisfeiler-Lehman algorithm suggested in [10].

3 Small Separators

For a given graph G and a set X of vertices in G, let $G - X$ denote the graph obtained by removing all vertices in X from G. A set X is called a *separator* if every connected component of $G - X$ has at most $n/2$ vertices, where n denotes the number of vertices of G. A class of graphs C is called *hereditary* if for every $G \in C$, every induced subgraph of G also belongs to C.

Theorem 2. *Let C be a hereditary class of graphs such that for a constant r, every graph $G \in C$ has an r-vertex separator. Suppose that C^* has a complete invariant f computable in* TC^k *(resp.* AC^k*) for some $k \geq 1$. Then C has a canonical labeling in* TC^{k+1} *(resp.* AC^{k+1}*).*

Proof. Having f in our disposal, we design a canonical labeling algorithm for C. Let G be an input graph with vertex set $V(G) = \{1, \ldots, n\}$ and assume that G has an r-vertex separator. We describe a recursive algorithm for finding a canonical renumbering $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$. In the following, the parameter d refers to the recursion depth. Initially $d = 1$. Further, set $R = 2^r + r$.

For a given sequence $s = (v_1, \ldots, v_r)$ of vertices, let G_s denote the coloring of G in which v_i receives color $(d-1)R + i$.

For each sequence $s = (v_1, \ldots, v_r)$ in parallel we do the following. First of all, we check if the set $\{v_1, \ldots, v_r\}$ is a separator. We are able to do this in AC^1 since checking if two vertices are in the same connected component reduces to the s-t-connectivity problem, which is easily solvable in NL, and the remaining job can be easily organized in TC^0. If the verification is positive, we mark the sequence s as *separating*. If no such sequence s is separating, i.e., G has no r-vertex separator, we terminate and output the identity permutation. Otherwise, for each separating sequence s in parallel we compute $f(G_s)$. Then in AC^1 we find a sequence $s = (v_1, \ldots, v_r)$ for which the value $f(G_s)$ is lexicographically minimum. For this purpose we use the fact that lexicographic comparison can be done in AC^0 and employ a known TC^0 sorting algorithm.

At this stage we are able to determine the renumbering σ only in a few points. Namely, we set $\sigma(v_i) = (d-1)R + i$ for each $i \le r$.

To proceed further, let $F_1, \ldots, F_m$ be the connected components of $G - X$ where $X = \{v_1, \ldots, v_r\}$. We color each $v \notin X$ by its adjacency pattern to X, that is, by the set of all neighbors of v in X, encoding this set by a number in the range between $(d-1)R + r + 1$ and dR. Each F_j, regarded as a colored graph, will be called an *X-flap*. For each X-flap F_j in parallel, we now compute $f(F_j)$ and establish the lexicographic order between these values. At this stage we fix the following partial information about the renumbering σ under construction: $\sigma(u) < \sigma(v)$ whenever we have $f(F_l) < f(F_j)$ for the two flaps F_l and F_j containing u and v, respectively. Thus, we split $V(G) \setminus X$ into blocks $V(F_1), \ldots, V(F_m)$ and determine the renumbering σ first between the blocks. It may happen that for some flaps we have $f(F_l) = f(F_j)$. We fix the σ-order between the corresponding blocks arbitrarily. Note that the output will not depend on a particular choice made at this point.

It remains to determine σ inside each block $V(F_j)$. We do this in parallel. For F_j with more than r vertices we repeat the same procedure as above with the value of d increased by 1. If $F = F_j$ has $t \le r$ vertices, we proceed as follows. Let a be the largest color present in F. We choose a bijection $\tau : V(F) \to \{1, \ldots, t\}$ and define σ on $V(F)$ by $\sigma(u) < \sigma(v)$ if and only if $\tau(u) < \tau(v)$. To make the choice, with each such τ we associate the colored graph F_τ obtained from F by adding new colors, namely, by coloring each $v \in V(F)$ with color $a + \tau(v)$. For each τ we compute $f(F_\tau)$ and finally choose the τ minimizing $f(F_\tau)$ in the lexicographic order. Note that, if the minimum is attained by more than one τ, the output will not depend on a particular choice.

Finally, we have to estimate the depth of the TC (resp. AC) circuit implementing the described algorithm. At the recursive step of depth d we deal with graphs having at most $n/2^{d-1}$ vertices. It follows that the circuit depth does not exceed $\log_2^k n + \log_2^k(n/2) + \log_2^k(n/4) + \cdots + \log_2^k(r) \le \log_2^{k+1} n$. $\square$

It is well known that all graphs of treewidth t have a $(t+1)$-vertex separator [21]. By [10], this class has a complete invariant computable in TC^1 and therefore is in the scope of Theorem 2.

Corollary 3. *For each constant t, a canonical labeling for graphs of treewidth at most t can be computed in* TC^2.

Theorem 2 also has relevance to the complexity-theoretic *decision-versus-search* paradigm. Let C be a class of graphs. It is well known (see, e.g., [12]) that, if we are able to test isomorphism of graphs in C^* in polynomial time, we are also able to find an isomorphism between two given isomorphic graphs in C in polynomial time. As the standard reduction is very sequential in nature, it is questionable if this implication stays true in the model of parallel computation. Nevertheless, a canonical labeling immediately provides us with an isomorphism between two isomorphic graphs.

Corollary 4. *For each constant t, an isomorphism between isomorphic graphs of treewidth at most t can be computed in* TC^2.

4 Bounded Rigidity Index

In this section we show that the canonization problem for any class of graphs with bounded rigidity index NC reduces to the corresponding complete invariant problem. Further we show that certain embeddability properties of a given class of graphs C imply a bound on the rigidity index of the graphs in C.

4.1 Canonizing Rigid Graphs

A set $S \subseteq V(G)$ of vertices is called *fixing* if every non-trivial automorphism of G moves at least one vertex in S. The *rigidity index* of a graph G is defined to be the minimum cardinality of a fixing set in G and denoted by $rig(G)$.

Theorem 5. *Let C be a class of graphs such that for a constant r, we have $rig(G) \leq r$ for all $G \in C$. Suppose that C^* has a complete invariant f computable in AC^k, for some $k \geq 1$. Then C has a canonical labeling also in AC^k.*

Proof. Let an input graph G with vertex set $V(G) = \{1, \ldots, n\}$ be given. We describe an algorithm that uses f as a subroutine in order to find a canonical renumbering $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ for G, provided that $G \in C$.

For a given sequence $s = (v_1, \ldots, v_r)$ of vertices, let G_s denote the coloring of G in which v_i receives color i. If v is another vertex, $G_{s,v}$ denotes the coloring where vertex v additionally gets color $r + 1$.

For each such sequence s in parallel we do the following. For each v in parallel we compute $f(G_{s,v})$. If all the values $f(G_{s,v})$, $v \in V(G)$, are pairwise distinct, which is decidable in AC^0, mark s as *fixing*. If no fixing sequence s of length r exists, which implies $G \notin C$, we terminate and output the identity permutation. Otherwise, for each fixing sequence s in parallel, we compute $f(G_s)$ and determine a sequence $s = (v_1, \ldots, v_r)$ for which $f(G_s)$ is lexicographically minimum (as was already mentioned in the proof of Theorem 2, this can be done in TC^0). The output permutation σ is now computed as follows. For each $i \leq r$, we set $\sigma(v_i) = i$. To determine σ everywhere else, we sort the values $f(G_{s,v})$ for all $v \in V(G) \setminus \{v_1, \ldots, v_r\}$ lexicographically and set $\sigma(v)$ to be the number of v in this order increased by r. $\square$

4.2 Basics of Topological Graph Theory

A detailed exposition of the concepts discussed in this section can be found in [19]. We are interested in embeddability of an abstract graph G into a surface S. All surfaces are supposed to be 2-dimensional, connected, and closed.

In an *embedding* Π of G into S, each vertex v of G is represented by a point on S (labeled by v and called *vertex of the* Π*-embedded graph* G) and each edge uv of G is drawn on S as a continuous curve with endpoints u and v. The curves are supposed to be non-self-crossing and any two such curves either have no common point or share a common endpoint. A *face* of Π is a connected component of the space obtained from S by removing the curves. We consider only *cellular* embeddings meaning that every face is homeomorphic to an open disc. A *closed walk* in a graph is a sequence of vertices $v_1v_2\cdots v_k$ such that v_i and v_{i+1} are adjacent for any $i < k$, and v_1 and v_k are adjacent as well. Notice that some of the vertices may coincide. We will not distinguish between a closed walk $v_1v_2\cdots v_k$ and any cyclic shift of it or of its reversal $v_kv_{k-1}\cdots v_1$. A closed walk $v_1v_2\cdots v_k$ is called Π*-facial*, if there exists a face F of Π, such that the vertices $v_1, v_2, \ldots, v_k$ occur in this order as labels along the boundary of F.

Two embeddings Π and Π' of G into S are called *equivalent* if they can be obtained from each other by a homeomorphism of S onto itself (respecting vertex labels). Since such a homeomorphism takes faces of one embedding to faces of the other embedding, we see that equivalent embeddings have equal sets of facial walks. In fact, the converse is also true: if the set of the Π-facial walks is equal to the set of the Π'-facial walks, then Π and Π' are equivalent. This follows from the fact that up to homeomorphism, the surface S is reconstructible from the set of facial walks by attaching an open disc along each facial walk.

A closed walk $v_1v_2\cdots v_k$ can be alternatively thought of as the sequence of edges $e_1e_2\cdots e_k$ where $e_i = v_iv_{i+1}$ $(i < k)$ and $e_k = v_1v_k$. Every edge either appears in two Π-facial walks (exactly once in each) or has exactly two occurrences in a single Π-facial walk. An embedding Π is called *polyhedral* if every Π-facial walk is a cycle (i.e., contains at most one occurrence of any vertex) and every two Π-facial walks either have at most one vertex in common or share exactly one edge (and no other vertex).

Let $\mathrm{Aut}(G)$ denote the automorphism group of G. For a given automorphism $\alpha \in \mathrm{Aut}(G)$, let Π^α denote the embedding of G obtained from Π by relabeling the vertices according to α. Note that Π^α and Π are not necessarily equivalent (they are *topologically isomorphic*, that is, obtainable from one another by a surface homeomorphism which is allowed to ignore the vertex labeling). An embedding Π is called *faithful* if Π^α is equivalent to Π for every automorphism $\alpha \in \mathrm{Aut}(G)$.

Recall that a graph G is k*-connected* if it has at least $k+1$ vertices and stays connected after removing any set of at most $k-1$ vertices. We now summarize known results showing that, for $k \geq 3$, the flexibility of embedding a k-connected graph into certain surfaces is fairly restricted.

The Whitney Theorem. [23] *Up to equivalence, every 3-connected planar graph has a unique embedding into the sphere.*

The Mohar-Robertson Theorem. [18] *Up to equivalence, every connected*[2] *graph has at most c_S polyhedral embeddings into a surface S, where c_S is a constant depending only on S.*

A closed curve in a surface is *contractible* if it is homotopic to a point. The *edge-width* of an embedding Π is the minimum length of a non-contractible cycle in the Π-embedded graph. Π is called a *large-edge-width embedding* (abbreviated as *LEW embedding*) if its edge-width is larger than the maximum length of a Π-facial walk.

The Thomassen Theorem. [22] (see also [19, Corollary 5.1.6]) *Every 3-connected graph having a LEW embedding into a surface S has, up to equivalence, a unique embedding into S. Moreover, such a surface S is unique.*

Note that if a graph has a unique embedding into a surface (as in the Whitney Theorem or the Thomassen Theorem), then this embedding is faithful.

As we have seen, an embedding is determined by its set of facial walks (up to equivalence). We will need yet another combinatorial specification of an embedding. To simplify the current exposition, we restrict ourselves to the case of orientable surfaces.

Let G be a graph and let T be a ternary relation on the vertex set $V(G)$ of G. We call $R = \langle G, T\rangle$ a *rotation system* of G if T fulfills the following two conditions:

(1) If $T(a, b, c)$ holds, then b and c are in $\Gamma(a)$, the neighborhood of a in G.

(2) For every vertex a, the binary relation $T(a, \cdot, \cdot)$ is a directed cycle on $\Gamma(a)$ (i.e., for every b there is exactly one c such that $T(a, b, c)$, for every c there is exactly one b such that $T(a, b, c)$, and the digraph $T(a, \cdot, \cdot)$ is connected on $\Gamma(a)$).

An embedding Π of a graph G into an orientable surface S determines a rotation system $R_\Pi = \langle G, T_\Pi\rangle$ in a natural geometric way. Namely, for $a \in V(G)$ and $b, c \in \Gamma(a)$ we set $T_\Pi(a, b, c) = 1$ if, looking at the neighborhood of a in the Π-embedded graph G from the outside of S, b is followed by c in the clockwise order.

The *conjugate* of a rotation system $R = \langle G, T\rangle$, denoted by R^*, is the rotation system $\langle G, T^*\rangle$, where T^* is defined as $T^*(a, b, c) = T(a, c, b)$. This notion has two geometric interpretations. First, $(R_\Pi)^*$ is a variant of R_Π where we look at the Π-embedded graph from the inside rather than from the outside of the surface (or, staying outside, just change the clockwise order to the counter-clockwise order). Second, $(R_\Pi)^* = R_{\Pi^*}$ where Π^* is a mirror image of Π.

It can be shown that two embeddings Π and Π' of G into S are equivalent if and only if $R_\Pi = R_{\Pi'}$ or $R_\Pi = R^*_{\Pi'}$ (see [19, Corollary 3.2.5]).

Further, for a given rotation system $R = \langle G, T\rangle$ and automorphism $\alpha \in \mathrm{Aut}(G)$, we define another rotation system $R^\alpha = \langle G, T^\alpha\rangle$ by $T^\alpha(a, b, c) = T(\alpha^{-1}(a), \alpha^{-1}(b), \alpha^{-1}(c))$. It is not hard to see that $R^\alpha_\Pi = R_{\Pi^\alpha}$. If $R = \langle G, T\rangle$ and $R' = \langle G, T'\rangle$ are two rotation systems of the same graph G and $R' = R^\alpha$ for some $\alpha \in \mathrm{Aut}(G)$, then this equality means that α is an isomorphism from R' onto R (respecting not only the binary adjacency relation but also the ternary relations of these structures).

[2] It is known that only 3-connected graphs have polyhedral embeddings.

4.3 Rigidity from Non-flexible Embeddability

Let α be a mapping defined on a set V. We say that α *fixes an element* $x \in V$ if $\alpha(x) = x$. Furthermore, we say that α *fixes a set* $X \subseteq V$ if α fixes every element of X.

Lemma 6. *If a graph G has a faithful embedding Π into some surface S, then $rig(G) \le 3$.*

Proof. Clearly, G is connected as disconnected graphs don't have a cellular embedding. If G is a path or a cycle, then $rig(G) \le 2$. Otherwise, G contains some vertex v with at least 3 neighbors. Notice that a facial walk cannot contain a segment of the form uvu. Therefore, some facial walk W contains a segment uvw, where u and w are two different neighbors of v. As v has at least one further neighbor that is distinct from u and w, uvw cannot be a segment of any other facial walk than W.

We now show that $\{u, v, w\}$ is a fixing set. Assume that α is an automorphism of G that fixes the vertices u, v and w. We have to prove that α is the identity.

Note that $v_1 v_2 \cdots v_k$ is a Π-facial walk if and only if $\alpha(v_1)\alpha(v_2)\cdots\alpha(v_k)$ is a Π^α-facial walk. Since Π and Π^α are equivalent and hence, have the same facial walks, α takes each Π-facial walk to a Π-facial walk. It follows that α takes W onto itself. Since α fixes two consecutive vertices of W, it actually fixes W.

Call two Π-facial walks W_1 and W_2 *adjacent* if they share an edge. Suppose that adjacent facial walks W_1 and W_2 share an edge $u_1 u_2$ and that α fixes W_1. Since $u_1 u_2$ cannot participate in any third facial walk, α takes W_2 onto itself. Since u_1 and u_2 are fixed, α fixes W_2, too.

Now consider the graph whose vertices are the Π-facial walks with the adjacency relation defined as above. It is not hard to see that this graph is connected, implying that α is the identity on the whole vertex set $V(G)$. □

By the Thomassen Theorem and by Lemma 6 it follows that every 3-connected LEW embeddable graph has rigidity index at most 3. Hence we can apply Theorem 5 to obtain the following result.

Corollary 7. *Let C be any class consisting only of 3-connected LEW embeddable graphs. If C^* has a complete invariant computable in AC^k, $k \ge 1$, then C has a canonical labeling in AC^k.*

Noteworthy, the class of all 3-connected LEW embeddable graphs is recognizable in polynomial time [19, Theorem 5.1.8].

Lemma 8. *If a connected graph G has a polyhedral embedding into a surface S, then we have $rig(G) \le 4c$, where c is the total number of non-equivalent polyhedral embeddings of G into S.*

Proof. To simplify the current exposition, we prove the lemma only for the case that S is orientable. Let $a \in V(G)$ be a vertex in G. We call two rotation systems $R = \langle G, T\rangle$ and $R' = \langle G, T'\rangle$ of G *a-coherent* if the binary relations $T(a, \cdot, \cdot)$ and $T'(a, \cdot, \cdot)$ coincide.

Claim 1. Let ab be an edge in G. Then any isomorphism α between two a-coherent rotation systems $R = \langle G, T\rangle$ and $R' = \langle G, T'\rangle$ of G that fixes both a and b, fixes also $\Gamma(a)$.

Proof of Claim. Since α fixes a, it takes $\Gamma(a)$ onto itself. Since $T(a,\cdot,\cdot) = T'(a,\cdot,\cdot)$, α is an automorphism of this binary relation. The latter is a directed cycle and α must be a shift thereof. Since α fixes b, it has to fix the whole cycle. $\triangleleft$

Let $R_1, \ldots, R_{2c}$ (where $R_i = \langle G, T_i\rangle$) be the rotation systems representing all polyhedral embeddings of G into S (i.e., each of the c embeddings is represented by two mutually conjugated rotation systems). Pick an arbitrary edge xy in G. For each i, $1 < i \le 2c$, select a vertex x_i so that R_i and R_1 are not x_i-coherent and the distance between x and x_i is minimum (it may happen that $x_i = x$). Furthermore, select y_i and z_i in $\Gamma(x_i)$ so that $T_1(x_i, y_i, z_i) \ne T_i(x_i, y_i, z_i)$. We will show that $\{x, y, y_2, z_2, \ldots, y_{2c}, z_{2c}\}$ is a fixing set. Assume that $\alpha \in \mathrm{Aut}(G)$ fixes all these vertices. We have to show that α is the identity.

Notice that R_1^α is a polyhedral embedding of G into S because so is R_1. Therefore $R_1^\alpha = R_k$ for some $k \le 2c$. Suppose first that R_k and R_1 are x-coherent. We will apply Claim 1 repeatedly to $R = R_k$ and $R' = R_1$. We first put $a = x$ and $b = y$ and see that α fixes $\Gamma(x)$. If the distance between x and x_k is more than 1, we apply Claim 1 once again for xx' being the first edge of a shortest path P from x to x_k (now $a = x'$ and $b = x$; we have $\alpha(x') = x'$ as $x' \in \Gamma(x)$, and R_k and R_1 are x'-coherent by our choice of x_k). Applying Claim 1 successively for all edges along P except the last one, we arrive at the conclusion that $\alpha(x_k) = x_k$. This also applies for the case that R_k and R_1 are not x-coherent, when we have $x_k = x$ by definition.

It follows that α is an isomorphism between the cycles $T_k(x_k,\cdot,\cdot)$ and $T_1(x_k,\cdot,\cdot)$. Our choice of y_k and z_k rules out the possibility that $k \ge 2$ and we conclude that $k = 1$. In other words, R and R' are coherent everywhere. Therefore, we are able to apply Claim 1 along any path starting from the edge $ab = xy$. Since G is connected, we see that α is the identity permutation on $V(G)$. $\square$

By the Mohar-Robertson Theorem and Lemma 8 it follows that every connected graph having a polyhedral embedding into a surface S has rigidity index bounded by a constant depending only on S.[3] Applying Theorem 5, we obtain the following result.

Corollary 9. *Let C be any class containing only graphs having a polyhedral embedding into a fixed surface S. If C^* has a complete invariant computable in* AC^k*, $k \ge 1$, then C has a canonical labeling in* AC^k.

We conclude this section by applying a ready-to-use result on the rigidity index of 5-connected graphs that are embeddable into a fixed surface S.

The Fijavž-Mohar Theorem. [7] *The rigidity index of 5-connected graphs embeddable into a surface S is bounded by a constant depending only on S.*

[3] As we recently learned, this result has been independently obtained in [7] by using a different argument.

Corollary 10. *Let C be the class of 5-connected graphs embeddable into a fixed surface S. If C^* has a complete invariant computable in* AC^k, $k \geq 1$, *then C has a canonical labeling in* AC^k.

5 Conclusion and Open Problems

For several important classes of graphs, we provide NC Turing-reductions of canonical labeling to computing a complete invariant. As a consequence, we get a canonical labeling NC algorithm for graphs with bounded treewidth by using a known [10] NC-computable complete invariant for such graphs.

We also consider classes of graphs embeddable into a fixed surface. Though we currently cannot cover this case in full extent, we provide NC reductions between the canonical labeling and complete invariant problems for some representative subclasses (namely, 3-connected graphs with either a polyhedral or an LEW embedding as well as all embeddable 5-connected graphs).

To the best of our knowledge, complete invariants (even isomorphism tests) in NC are only known for the sphere but not for any other surface. The known isomorphism tests and complete-invariant algorithms designed in [5,13,15,16,8] run in sequential polynomial time. Nevertheless, the hypothesis that the complexity of some of these algorithms can be improved from P to NC seems rather plausible. By this reason it would be desirable to extend the reductions proved in the present paper to the whole class of graphs embeddable into S, for any fixed surface S. As a first step in this direction one could consider the class of 4-connected toroidal graphs.[4]

A more ambitious research project is to find an NC-reduction of the canonical labeling problem to computing a complete invariant for classes of graphs that are defined by excluding certain graphs as minors or, equivalently, for classes of graphs closed under minors. A polynomial-time canonization algorithm for such classes has been worked out by Ponomarenko [20]. Note that any class of graphs with bounded treewidth as well as any class consisting of all graphs embeddable into a fixed surface is closed under minors.

Acknowledgement. We thank Gašper Fijavž and Bojan Mohar for sending us their manuscript [7].

References

1. Arvind, V., Das, B., Mukhopadhyay, P.: On isomorphism and canonization of tournaments and hypertournaments. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 449–459. Springer, Heidelberg (2006)
2. Arvind, V., Torán, J.: Isomorphism testing: Pespective and open problems. Bulletin of the European Association of Theoretical Computer Science 86, 66–84 (2005)
3. Babai, L., Luks, E.: Canonical labeling of graphs. In: Proc. 15th ACM Symposium on Theory of Computing, pp. 171–183. ACM Press, New York (1983)

[4] As shown in [7], graphs in this class can have arbitrarily large rigidity index.

4. Chlebus, B.S., Diks, K., Radzik, T.: Testing isomorphism of outerplanar graphs in parallel. In: Koubek, V., Janiga, L., Chytil, M.P. (eds.) MFCS 1988. LNCS, vol. 324, pp. 220–230. Springer, Heidelberg (1988)
5. Filotti, I.S., Mayer, J.N.: A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In: Proc. 12th ACM Symposium on Theory of Computing, pp. 236–243. ACM Press, New York (1980)
6. Fijavž, G., Mohar, B.: Rigidity and separation indices of Paley graphs. Discrete Mathematics 289, 157–161 (2004)
7. Fijavž, G., Mohar, B.: Rigidity and separation indices of graphs in surfaces. A manuscript in preparation, cited in [6]
8. Grohe, M.: Isomorphism testing for embeddable graphs through definability. In: Proc. 32th ACM Symposium on Theory of Computing, pp. 63–72. ACM Press, New York (2000)
9. Gurevich, Y.: From invariants to canonization. Bulletin of the European Association of Theoretical Computer Science 63, 115–119 (1997)
10. Grohe, M., Verbitsky, O.: Testing graph isomorphism in parallel by playing a game. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 3–14. Springer, Heidelberg (2006)
11. Immerman, N., Lander, E.: Describing graphs: a first order approach to graph canonization. In: Selman, A.L. (ed.) Complexity Theory Retrospective, pp. 59–81. Springer, Heidelberg (1990)
12. Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: Its Structural Complexity. Birkhäuser, Boston (1993)
13. Lichtenstein, D.: Isomorphism for graphs embeddable on the projective plane. In: Proc. 12th ACM Symposium on Theory of Computing, pp. 218–224. ACM Press, New York (1980)
14. Lindell, S.: A logspace algorithm for tree canonization. In: Proc. 24th ACM Symposium on Theory of Computing, pp. 400–404. ACM Press, New York (1992)
15. Miller, G.L.: Isomorphism testing for graphs of bounded genus. In: Proc. 12th ACM Symposium on Theory of Computing, pp. 225–235. ACM Press, New York (1980)
16. Miller, G.L.: Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus. Information and Computation 56, 1–20 (1983)
17. Miller, G.L., Reif, J.H.: Parallel tree contraction. Part 2: Further applications. SIAM Journal on Computing 20, 1128–1147 (1991)
18. Mohar, B., Robertson, N.: Flexibility of polyhedral embeddings of graphs in surfaces. J. Combin. Theory, Ser. B 83, 38–57 (2001)
19. Mohar, B., Thomassen, C.: Graphs on surfaces. The John Hopkins University Press (2001)
20. Ponomarenko, I.: The isomorphism problem for classes of graphs closed under contraction. In: Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov (in Russian), vol. 174, pp. 147–177 (1988); English translation in: Journal of Mathematical Sciences 55, 1621–1643 (1991)
21. Robertson, N., Seymour, P.D.: Graph minors II. Algorithmic aspects of tree-width. J. Algorithms 7, 309–322 (1986)
22. Thomassen, C.: Embeddings of graphs with no short noncontractible cycles. J. Combin. Theory, Ser. B 48, 155–177 (1990)
23. Whitney, H.: 2-isomorphic graphs. Amer. Math. J. 55, 245–254 (1933)

Self-referentiality of Justified Knowledge

Roman Kuznets

Ph.D. Program in Computer Science
CUNY Graduate Center
365 Fifth Avenue, New York, NY 10016, USA
kuznets@gmail.com

Abstract. The principal result of Justification Logic is the Realization Theorem, which states that behind major epistemic modal logics there are corresponding systems of evidence/justification terms sufficient for reading all provable knowledge assertions as statements about justifications. A knowledge/belief modality is self-referential if there are modal sentences that cannot be realized without using self-referential evidence of type "t is a proof of $A(t)$." Building on an earlier result that S4 and its justification counterpart LP describe knowledge that is self-referential, we show that the same is true for $\mathsf{K4}$, $\mathsf{D4}$, and T with their justification counterparts whereas for K and D self-referentiality can be avoided. Therefore, no single modal axiom from the standard axiomatizations of these logics is responsible for self-referentiality.

1 Introduction

The modality in GL corresponds to provability in the formal arithmetic, which is known to be self-referential. But it is not clear how to formulate this property by means of the modal language itself.

By contrast, the language of justification logic (see [3]) provides a natural way to formulate what it means for the modality in a modal logic to be self-referential. Instead of using existential statements $\Box F$, read as "there exists a proof of F," justification logics employ an explicit justification construct $t\colon F$, read "term t serves as a justification for F." In this setting, self-referentiality clearly occurs when a term t proves something about itself:

$$\vdash t\colon F(t) \ . \tag{1}$$

Not only are such constructions allowed by the language, but there are also many theorems of this type, notably with $t = c$ being an atomic justification, a constant.

Definition 1. *Let F be a justification formula. The* forgetful projection $^\circ$ *turns it into a modal formula by replacing each occurrence of justification terms in F by $\Box$, $(t\colon G)^\circ = \Box\,(G^\circ)$, while leaving all Boolean connectives and sentence letters intact.*

The forgetful projection of a set X of justification formulas is a set of modal formulas $X^\circ = \{F^\circ \mid F \in X\}$.

A logic L can be viewed as a set of L-theorems. Then

Definition 2. *A modal logic* ML *is said to be a* forgetful projection *of a justification logic* JL *if* $\mathsf{JL}^\circ = \mathsf{ML}$.

It was shown in [1] that the forgetful projection of the first justification logic, LP, is exactly $\mathsf{S4}$, i.e., $\mathsf{LP}^\circ = \mathsf{S4}$. This statement is typically called the Realization Theorem because this equality essentially states two things:

1. Replacing each justification term in an LP-theorem by $\Box$ yields an $\mathsf{S4}$-theorem.
2. Vice versa, it is possible to *realize* all occurrences of $\Box$ in an $\mathsf{S4}$-theorem by justification terms in such a way that the resulting justification formula is valid. This process of restoring terms hidden in $\Box$'s is called *Realization*.

For each of the modal logics $\mathsf{K}, \mathsf{D}, \mathsf{T}, \mathsf{K4}, \mathsf{D4}, \mathsf{S4}, \mathsf{K5}, \mathsf{K45}, \mathsf{KD45}, \mathsf{S5}$ a justification counterpart was developed, so that its forgetful projection is exactly this modal logic (see [1, 3, 4, 8, 9]).

In particular, since $\vdash \Box A$ for any axiom A in a modal logic ML, there must be some term t in its justification counterpart JL such that $\vdash t\!:\!(A^r)$, where A^r is a realization of A. In most cases, an axiom of ML is realized by an axiom of JL. Justifications for axioms are called *justification constants*, and, unless we have a reason to track or restrict their use, we typically postulate that each constant justifies all axioms. Thus, $\vdash c : A(c)$, where $A(c)$ is an axiom that contains at least one occurrence of c.

A natural question to ask is **whether such self-referential constants are necessary for the Realization Theorem to hold**. Apart from being direct as in $\vdash c\!:\!A(c)$, self-referentiality may also occur as a result of a cycle of references:

$$\vdash c_2\!:\!A_1(c_1), \quad \ldots \quad , \quad \vdash c_n\!:\!A_{n-1}(c_{n-1}), \quad \vdash c_1\!:\!A_n(c_n) \ . \tag{2}$$

If direct self-referentiality is expendable, we should ask whether such self-referential cycles are still needed for the Realization.

It was shown in [5] that the realization of $\mathsf{S4}$ in LP does require direct self-referentiality of constants. In this paper, we prove the following:

– Realization of $\mathsf{K4}$ in $\mathsf{J4}$, of $\mathsf{D4}$ in $\mathsf{JD4}$, and of T in JT requires direct self-referentiality;
– Realization of K in J and of D in JD can be performed without any self-referential cycles.

Sect. 2 describes several justification logics and their forgetful projections. At the end of the section, we propose a precise definition of self-referentiality in modal logics. Epistemic semantics for the justification logics from Sect. 2 is described in Sect. 3. Using this semantics, in Sect. 4, we prove that the Realization Theorem for $\mathsf{K4}$, $\mathsf{D4}$, and T requires self-referentiality. Sect. 5 demonstrates how to avoid self-referentiality while realizing logics K and D. Sect. 6 analyzes the significance of these results and outlines directions for future research.

2 Justification Logics and Self-referentiality Defined

The first justification logic, LP, was introduced in [1], where its forgetful projection was shown to be S4 (see also [2]). Justification counterparts for K, D, T, K4, and D4 were developed and the Realization Theorem for them was proven in [4]. Realizations of several modal logics with the Negative Introspection Axiom were considered in [3, 8, 9], but their self-referential properties are outside the scope of this paper, which focuses on the modal logics

$$\mathsf{K},\quad \mathsf{D},\quad \mathsf{T},\quad \mathsf{K4},\quad \mathsf{D4},\quad \mathsf{S4} \tag{3}$$

and their respective justification counterparts

$$\mathsf{J},\quad \mathsf{JD},\quad \mathsf{JT},\quad \mathsf{J4},\quad \mathsf{JD4},\quad \mathsf{LP}\ . \tag{4}$$

We will show that for the first two pairs of the modal logic with its justification counterpart self-referentiality can be avoided whereas the last four pairs require direct self-referentiality.

The language of justification logic is that of propositional logic enriched by a new construct $t : F$, where F is any formula and t is a justification term. Justification terms are built from justification constants $a, b, c, \dots$ and justification variables $x, y, z, \dots$ by means of three operations: a unary operation ! and two binary operations + and $\cdot$.[1]

All six justification logics from (4) share the following axioms and rules

A1. Classical propositional axioms and rule *modus ponens*
A2. *Application Axiom* $s{:}(F \to G) \to (t{:}F \to (s \cdot t){:}G)$
A3. *Monotonicity Axiom* $s{:}F \to (s + t){:}F, \qquad t{:}F \to (s + t){:}F$
R4. *Axiom Internalization Rule*: for each axiom A and each justification constant c, formula $c{:}A$ is again an axiom.

These axioms and rules alone yield the basic justification logic J whose forgetful projection is K, the weakest normal modal logic. It is easy to see that the forgetful projection of axioms of J yields theorems of K. Just as other modal logics from (3) are obtained by adding axiom schemes to K, so their justification counterparts from (4) can be obtained by adding corresponding justification schemes to J. In each case, the added modal axiom scheme is the forgetful projection of the respective justification scheme[2]:

Modal Scheme	Justification Scheme	Name of Justification Scheme	To Be Added in Logics
$\Box F \to F$	$t{:}F \to F$	A4. Factivity	JT, LP
$\Box F \to \Box\Box F$	$t{:}F \to\, !t{:}(t{:}F)$	A5. Positive Introspection	J4, JD4, LP
$\Box\bot \to \bot$	$t{:}\bot \to \bot$	A7. Consistency	JD, JD4

[1] Operation ! is only used in J4, JD4, and LP.
[2] Axiom numbering is mostly inherited from [3].

It is important to note that the modal Seriality Axiom in the last row of the table is a single axiom, whereas its realization requires an axiom scheme A7.

Theorem 1 (Realization Theorem, [1, 4])

$$\mathsf{J}^\circ = \mathsf{K} \qquad \mathsf{JD}^\circ = \mathsf{D} \qquad \mathsf{JT}^\circ = \mathsf{T}$$
$$\mathsf{J4}^\circ = \mathsf{K4} \qquad \mathsf{JD4}^\circ = \mathsf{D4} \qquad \mathsf{LP}^\circ = \mathsf{S4}$$

For each justification logic, a family of weaker logics is defined with a supervised use of rule R4. Note that this rule has a different scope in different justification logics because they have different axiom sets. Thus, the following definition of a constant specification depends on the respective logic. In particular, a constant specification for LP may not be a constant specification for J.

Definition 3. *A* constant specification $\mathcal{CS}$ for a justification logic L *is any set of formulas $c{:}A$ that can be introduced by the Axiom Internalization Rule* R4 *of this logic. The only requirement is for such a set to be downward closed, i.e., if $c_1{:}c_2{:}A \in \mathcal{CS}$, then $c_2{:}A \in \mathcal{CS}$.*

Definition 4. *Let $\mathcal{CS}$ be a constant specification for a justification logic L. By $\mathsf{L}_{\mathcal{CS}}$ we understand the logic obtained by replacing* R4 *in logic L by the rule*

$$\text{R4}_{\mathcal{CS}}. \qquad \vdash c{:}A \quad \textit{where } c{:}A \in \mathcal{CS}\ .$$

Each logic L from (4) is essentially $\mathsf{L}_{\mathcal{TCS}}$ with the *total constant specification*, i.e., with every constant justifying all axioms.

Definition 5. *A constant specification $\mathcal{CS}$ for a justification logic is called* self-referential *if*

$$\{c_2{:}A_1(c_1), \dots, c_n{:}A_{n-1}(c_{n-1}), c_1{:}A_n(c_n)\} \subset \mathcal{CS} \tag{5}$$

for some constants c_i and axioms $A_i(c_i)$ with at least one occurrence of c_i.

A constant specification $\mathcal{CS}$ is directly self-referential *if $c{:}A(c) \in \mathcal{CS}$.*

A constant specification is axiomatically appropriate *if every axiom A of the logic has at least one constant c such that $c{:}A \in \mathcal{CS}$.*

The total constant specification is always directly self-referential. Therefore, the standard proofs of the Realization Theorem only show that Realization is possible when direct self-referentiality is used. Our task is to determine whether Realization can be achieved without self-referentiality.

Definition 6. *Let a modal logic ML be the forgetful projection of a justification logic JL, i.e., $\mathsf{JL}^\circ = \mathsf{ML}$. We call the modal logic ML* directly self-referential *if $(\mathsf{JL}_{\mathcal{CS}})^\circ \neq \mathsf{ML}$ for any $\mathcal{CS}$ that is not directly self-referential.*

We call ML self-referential *if $(\mathsf{JL}_{\mathcal{CS}})^\circ \neq \mathsf{ML}$ for any $\mathcal{CS}$ that is not self-referential.*

$\mathsf{S4}$ was shown in [5] to be directly self-referential.[3] In this paper, we will prove that $\mathsf{K4}$, $\mathsf{D4}$, and T are also directly self-referential whereas K and D are not self-referential.

[3] The term “directly” was not used in [5].

3 Epistemic Models for Justification Logics

Self-referentiality of $\mathsf{K4}$, $\mathsf{D4}$, and T will be established by a semantic argument. Unlike [5], where M-models were used, here we will employ more general F-models, which are based on Kripke models and thus are closer to the standard epistemic semantics. These F-models were first developed for LP; soundness and completeness of LP w.r.t. them can be found in [6]. The adaptation of these models to J, JT, and $\mathsf{J4}$ first appeared in [6]. Soundness and completeness arguments for J and JD can be found in [8], for JT and $\mathsf{J4}$ in [3]. The F-models for $\mathsf{JD4}$ are, perhaps, first developed in this paper.

Definition 7 (F-models for $\mathsf{J}_{\mathcal{CS}}$). *An* F-model for $\mathsf{J}_{\mathcal{CS}}$ *is a quadruple* $\mathcal{M} = \langle W, R, \mathcal{A}, v\rangle$, *where* $W \neq \emptyset$ *is a set of worlds;* $R \subseteq W \times W$ *is an accessibility relation; valuation* $v : SLet \to 2^W$ *assigns to a sentence letter* P *a set* $v(P) \subseteq W$ *of all worlds where this sentence letter is deemed true; finally, the* admissible evidence function $\mathcal{A} : Tm \times Fm \to 2^W$ *assigns to a pair of a term* t *and a formula* F *a set* $\mathcal{A}(t, F) \subseteq W$ *of all worlds where* t *is deemed admissible evidence for* F. *The admissible evidence function* $\mathcal{A}$ *must satisfy several closure conditions:*

C2. $\mathcal{A}(t, F \to G) \cap \mathcal{A}(s, F) \subseteq \mathcal{A}(t \cdot s, G)$
C3. $\mathcal{A}(t, F) \cup \mathcal{A}(s, F) \subseteq \mathcal{A}(t + s, F)$
$\mathcal{CS}$. $\mathcal{A}(c, A) = W$ *for every* $c{:}A \in \mathcal{CS}$.

The forcing relation $\Vdash$ *is defined as follows:*

- $\mathcal{M}, w \Vdash P$ *iff* $w \in v(P)$ *where* P *is a sentence letter;*
- *Boolean cases are standard;*
- $\mathcal{M}, w \Vdash t{:}F$ *iff 1)* $\mathcal{M}, u \Vdash F$ *for all* wRu *and 2)* $w \in \mathcal{A}(t, F)$.

The closure conditions C2 and C3 are required to validate axioms A2 and A3 respectively, which is reflected in the numbering. Note that $w \in \mathcal{A}(t, F)$ in no way implies that F itself is true. Rather $w \in \mathcal{A}(t, F)$ means that at world w term t is acceptable, although not necessarily conclusive, evidence for F.

Definition 8 (F-models for $\mathsf{JD}_{\mathcal{CS}}, \mathsf{JT}_{\mathcal{CS}}, \mathsf{J4}_{\mathcal{CS}}, \mathsf{JD4}_{\mathcal{CS}}, \mathsf{LP}_{\mathcal{CS}}$). *An F-model for these logics must satisfy all conditions for an F-model for* $\mathsf{J}_{\mathcal{CS}}$ *plus additional requirements that depend on the additional axioms of the respective logic:*

- *For* $\mathsf{JT}_{\mathcal{CS}}$ *and* $\mathsf{LP}_{\mathcal{CS}}$, *axiom* $t{:}F \to F$ *requires* R *to be* ***reflexive***.
- *For* $\mathsf{JD}_{\mathcal{CS}}$ *and* $\mathsf{JD4}_{\mathcal{CS}}$, *axiom* $t{:}\bot \to \bot$ *requires* R *to be* ***serial***.
- *For* $\mathsf{J4}_{\mathcal{CS}}$, $\mathsf{JD4}_{\mathcal{CS}}$, *and* $\mathsf{LP}_{\mathcal{CS}}$, *axiom* $t{:}F \to {!t}{:}t{:}F$ *requires* R *to be* ***transitive***. *In addition, two more closure conditions are imposed on* $\mathcal{A}$*:*
  C5. $\mathcal{A}(t, F) \subseteq \mathcal{A}(!t,\ t{:}F)$
  Monotonicity. wRu *and* $w \in \mathcal{A}(t, F)$ *imply* $u \in \mathcal{A}(t, F)$

Theorem 2 (Completeness Theorem, [3, 6], RK). $\mathsf{J}_{\mathcal{CS}}$, $\mathsf{JT}_{\mathcal{CS}}$, $\mathsf{J4}_{\mathcal{CS}}$, *and* $\mathsf{LP}_{\mathcal{CS}}$ *are sound and complete w.r.t. their F-models.* $\mathsf{JD}_{\mathcal{CS}}$ *and* $\mathsf{JD4}_{\mathcal{CS}}$ *are sound w.r.t. their F-models; completeness also holds provided* $\mathcal{CS}$ *is axiomatically appropriate.*

Proof. The cases of $\mathsf{J}_{\mathcal{CS}}$, $\mathsf{JT}_{\mathcal{CS}}$, $\mathsf{J4}_{\mathcal{CS}}$, and $\mathsf{LP}_{\mathcal{CS}}$ are covered in [3]. The proof for $\mathsf{JD}_{\mathcal{CS}}$ and $\mathsf{JD4}_{\mathcal{CS}}$ can be found in [7]. $\Box$

4 Self-referential Cases: S4, D4, T, and K4

In [5], direct self-referentiality of knowledge encompassed by $\mathsf{S4}$ and LP was proven by constructing an $\mathsf{LP}_{\mathcal{CS}}$-counter-model for any potential realization of $\mathsf{S4} \vdash \Diamond(P \to \Box P)$, or equivalently, of $\mathsf{S4} \vdash \neg\Box\neg(P \to \Box P)$, where $\mathcal{CS}$ was the maximal constant specification for LP without directly self-referential constants.

We will employ a similar argument for weaker logics using F-models instead of M-models.

Theorem 3. *Realization of* $\mathsf{D4}$ *in* $\mathsf{JD4}$ *and of* T *in* JT *requires direct self-referentiality.*

Proof. Note that $\Phi = \neg\Box\neg(P \to \Box P)$ is derivable in both $\mathsf{D4}$ and T.[4] Therefore, we can use the same argument, namely show that no potential realization of Φ is valid in $\mathsf{JD4}_{\mathcal{CS}}$- or $\mathsf{JT}_{\mathcal{CS}}$-models respectively for the respective maximal $\mathcal{CS}$ without directly self-referential constants. The proof for these two logics is uniform (and can, in fact, be applied to $\mathsf{S4}/\mathsf{LP}$ too).

Let $\mathsf{L} \in \{\mathsf{JD4}, \mathsf{JT}\}$ and $\mathcal{CS}$ be the maximal constant specification for L without directly self-referential constants. For any pair of terms t and t' used in place of the two $\Box$'s in Φ, we will construct an F-model for $\mathsf{L}_{\mathcal{CS}}$ that falsifies $\neg t\!:\![\neg(P \to t'\!:\!P)]$, thus showing that no realization of Φ is $\mathsf{L}_{\mathcal{CS}}$-valid. (Note that only soundness is used in this argument.)

Given t and t', consider the following F-model for $\mathsf{L}_{\mathcal{CS}}$: $\mathcal{M} = \langle W, R, \mathcal{A}, v\rangle$ with the Kripke frame $\langle W, R\rangle$ that consists of a single reflexive world w. Such R is obviously serial, reflexive, and transitive, thus making the frame suitable for $\mathsf{JD4}$, JT, and LP alike. Let $v(P) = W = \{w\}$, i.e., $\mathcal{M}, w \Vdash P$. The truth values of other sentence letters are not important.

Since w is the only world in the model, we can write $\Vdash F$ instead of $\mathcal{M}, w \Vdash F$; $\mathcal{A}(s, F)$ instead of $w \in \mathcal{A}(s, F)$; $\neg\mathcal{A}(s, F)$ instead of $w \notin \mathcal{A}(s, F)$.

The admissible evidence function $\mathcal{A}$ depends on terms t and t'. We require $\mathcal{A}(t,\ \neg(P \to t'\!:\!P))$. An admissible evidence function for either logic must satisfy closure conditions C2, C3, and $\mathcal{CS}$-closure; additionally for $\mathsf{JD4}_{\mathcal{CS}}$ and $\mathsf{LP}_{\mathcal{CS}}$, Monotonicity and C5 must hold. Monotonicity is trivially satisfied. Let $\mathcal{A}$ be the minimal admissible evidence function with $\mathcal{A}(t,\ \neg(P \to t'\!:\!P))$ that satisfies all the necessary closure conditions. Minimality here means that $\mathcal{A}(s, F)$ only if it can be derived from $\mathcal{A}(t,\ \neg(P \to t' : P))$ using the closure conditions for the logic.

It suffices to show $\neg\mathcal{A}(t', P)$ to falsify $\neg t\!:\![\neg(P \to t' : P)]$. Indeed, $\nVdash t' : P$ if $\neg\mathcal{A}(t', P)$. Given $\Vdash P$, it yields $\Vdash \neg(P \to t'\!:\!P)$. Finally, with this formula true at the only world and with $\mathcal{A}(t,\ \neg(P \to t'\!:\!P))$, we will have $\Vdash t\!:\![\neg(P \to t'\!:\!P)]$.

$\neg\mathcal{A}(t', P)$ follows from the following technical lemma. Let $\mathcal{A}_0$ be the minimal admissible evidence function for the logic (without the $\mathcal{A}(t,\ \neg(P \to t'\!:\!P))$ requirement). Clearly, $\mathcal{A}_0(s, F)$ implies $\mathcal{A}(s, F)$ as the closure conditions used are the same, with $\mathcal{A}$ having one additional *ad hoc* requirement.

[4] The idea to use this formula for these logics is due to Melvin Fitting.

Lemma 1. *For any subterm s of term t':*

1. *If $\mathcal{A}_0(s,F)$, then $\mathsf{L}_{\mathcal{CS}} \vdash F$ and F does not contain occurrences of t'.*
2. *If $\mathcal{A}(s,F)$, but $\neg\mathcal{A}_0(s,F)$, then F has at least one occurrence of t'. Moreover, the only such implication is $F = \neg(P \to t' {:} P)$.*[5]

Proof (Sketch). The proof is by induction on the size of s. Essentially, we show that all the closures due to C2, an analog of *modus ponens*, happen within $\mathcal{A}_0$, so that outside of it the closure derivation is, in a sense, "cut-free."

The fact that $\mathcal{CS}$ has no directly self-referential constants is used in the proof of Claim 1 of the lemma: whenever $\mathcal{A}_0(c,A)$, we have $c{:}A \in \mathcal{CS}$; thus, neither c nor term t', whose subterm c is, can occur in the axiom A.

The full proof can be found in the Appendix. $\square$

It remains to apply Lemma 1 to term t' itself. $\mathsf{L}_{\mathcal{CS}} \nvdash P$, so by Lemma 1.1, $\neg\mathcal{A}_0(t',P)$. But then, since t' does not occur in P, by Lemma 1.2, $\neg\mathcal{A}(t',P)$. $\square$

Theorem 4. *Realization of* $\mathsf{K4}$ *in* $\mathsf{J4}$ *requires direct self-referentiality.*

Proof. The Hilbert formulation of $\mathsf{D4}$ is obtained from that of $\mathsf{K4}$ by adding the Seriality Axiom. Therefore, $\mathsf{K4} \vdash \Diamond T \to \Diamond(P \to \Box P)$,[6] or equivalently, $\Psi = \Box\neg(P \to \Box P) \to \Box\bot$ is derivable in $\mathsf{K4}$.

For any potential realization $\Psi^r = t : [\neg(P \to t' : P)] \to k : \bot$, we construct an F-model for $\mathsf{J4}_{\mathcal{CS}}$ that falsifies Ψ^r, thus showing that no realization of Ψ is $\mathsf{J4}_{\mathcal{CS}}$-valid. Like in the cases of $\mathsf{JD4}_{\mathcal{CS}}$ and $\mathsf{JT}_{\mathcal{CS}}$ from Theorem 3, here $\mathcal{CS}$ is the maximal constant specification for $\mathsf{J4}$ without directly self-referential constants.

By contrast, the falsifying model here consists of a single irreflexive world. As in such a model any F is vacuously true at all accessible worlds, $\Vdash s : F$ iff $\mathcal{A}(s,F)$. Again, $\mathcal{A}$ is taken to be the minimal one with $\mathcal{A}(t,\ \neg(P \to t' : P))$. Valuation v is unimportant. We need to show $\neg\mathcal{A}(k,\bot)$.

Lemma 2. *Let $\mathcal{A}$ be the minimal admissible evidence function with $\mathcal{A}(r,B)$ in a single-world F-model for $\mathsf{J4}_{\mathcal{CS}}$. If $\mathcal{A}(s,G)$, then $B,\ r{:}B \vdash_{\mathsf{J4}_{\mathcal{CS}}} G$.*

Proof (Sketch). The proof is by induction on the closure derivation of $\mathcal{A}(s,G)$ from $\mathcal{A}(r,B)$. It can be easily restored by an interested reader.

The intuition might tell you that $r : B$ is not necessary as an additional hypothesis. The following example due to Vladimir Krupski shows otherwise: if $\mathcal{A}(x,P)$ then $\mathcal{A}(!\,x, x{:}P)$, but surely $P \nvdash_{\mathsf{J4}_{\mathcal{CS}}} x{:}P$. $\square$

If $\mathcal{A}(k,\bot)$, then, by Lemma 2, $\neg(P \to t' : P),\quad t : [\neg(P \to t' : P)] \quad \vdash_{\mathsf{J4}_{\mathcal{CS}}} \quad \bot$. But this cannot be the case since in the proof of Theorem 3 we constructed an F-model with both hypotheses being true. It was a $\mathsf{JD4}_{\mathcal{CS}}$-model, so it must also be a $\mathsf{J4}_{\mathcal{CS}}$-model since fewer restrictions are imposed on the latter and the $\mathcal{CS}$ for the latter is a subset of the $\mathcal{CS}$ for the former. A contradiction. $\square$

[5] We consider $\neg G$ to be an abbreviation of $G \to \bot$.

[6] The idea to use this formula for $\mathsf{K4}$ is due to Melvin Fitting.

5 Non-self-referential Cases: D and K

In this section, we will show that $(\mathsf{JD}_{\mathcal{CS}})^\circ = \mathsf{D}$ and $(\mathsf{J}_{\mathcal{CS}})^\circ = \mathsf{K}$ for some non-self-referential constant specifications $\mathcal{CS}$.

To construct such constant specifications, we will divide the set of constants into levels indexed by non-negative integers, with each level consisting of countably many constants. Let $\ell(c)$ denote the level of constant c. For either logic, let

$$\mathcal{CS} = \{c\colon A \in \mathcal{TCS} \mid \text{for all constants } a \text{ that occur in } A,\ \ell(a) < \ell(c)\}\ . \tag{6}$$

This constant specification is axiomatically appropriate.

Lemma 3 (Internalization Property). *Let $\mathsf{L}_{\mathcal{CS}}$ be a justification logic with an axiomatically appropriate $\mathcal{CS}$. Then, for any derivation $F_1, \dots, F_n \vdash_{\mathsf{L}_{\mathcal{CS}}} B$ there exists an evidence term $t(x_1, \dots, x_n)$ such that*

$$x_1\colon F_1, \dots, x_n\colon F_n \vdash_{\mathsf{L}_{\mathcal{CS}}} t(x_1, \dots, x_n)\colon B\ . \tag{7}$$

Proof. A step-by-step translation from the given derivation into the target one.

A	$\rightsquigarrow$	$c\colon A$	where A is an axiom or $A = c'\colon A'$
F_i	$\rightsquigarrow$	$x_i\colon F_i$	hypotheses
$\dfrac{D \to G \quad D}{G}$	$\rightsquigarrow$	$\dfrac{s_1\colon(D \to G) \quad s_2\colon D}{(s_1 \cdot s_2)\colon G}$	by A2 and *modus ponens* twice

□

Since the constant specification (6) has infinitely many constants on each level, it is always possible to choose a fresh constant c in the second line of the proof.

Theorem 5. *It is possible to realize* D *in* JD *and* K *in* J *without self-referentiality.*

Proof. We will prove that $(\mathsf{JD}_{\mathcal{CS}})^\circ = \mathsf{D}$ and $(\mathsf{J}_{\mathcal{CS}})^\circ = \mathsf{K}$ for the $\mathcal{CS}$ from (6). Since $\mathsf{L}_{\mathcal{CS}} \subseteq \mathsf{L}$, we have $(\mathsf{JD}_{\mathcal{CS}})^\circ \subseteq \mathsf{JD}^\circ = \mathsf{D}$ and $(\mathsf{J}_{\mathcal{CS}})^\circ \subseteq \mathsf{J}^\circ = \mathsf{K}$.

To show the other inclusion, we will reprove the Realization Theorem using the $\mathcal{CS}$ from (6). One of the ways to prove Realization is by step-by-step transformation of a cut-free Gentzen derivation of a modal theorem F into a Hilbert derivation of its realization F^r. Here $\vdash \Gamma \Rightarrow \Delta$ is being transformed into $\Gamma^r \vdash \bigvee \Delta^r$.[7] A detailed description can be found in [2, 4, 5]. Axioms of the Gentzen modal system are restricted to $\bot \Rightarrow$ and $P \Rightarrow P$ for sentence letters P to have a better control over where and how $\Box$'s are introduced. All occurrences of $\Box$ in the Gentzen modal derivation are divided into families of related occurrences. A cut-free derivation preserves polarity of formulas, so there are positive and negative families of $\Box$'s. We realize each negative family by a fresh justification variable. A positive family is realized by a sum of auxiliary variables $v_1 + \ldots + v_n$, one variable per each use of the modal rules to introduce a $\Box$ from this family. If all $\Box$'s from a positive family are introduced by Weakening, the family is instantiated by a fresh justification variable. The transformation is done by induction on the depth of the Gentzen derivation.

[7] As always, the empty disjunction is interpreted as $\bot$.

The Gentzen axioms, propositional rules, and Contraction can be translated using the standard propositional translation from Gentzen into Hilbert. Since the reasoning involved is purely propositional, neither Axiom Internalization is used, nor are new constants introduced. Weakening does not require Axiom Internalization either; it may bring constants from other branches, but never a fresh constant. Thus, new constants are introduced by Axiom Internalization only to translate modal rules. The only modal rule for logic K is $\dfrac{C_1,\dots,C_n \Rightarrow B}{\Box C_1,\dots,\Box C_n \Rightarrow \Box B}$. In addition, logic D has $\dfrac{C_1,\dots,C_n, D \Rightarrow}{\Box C_1,\dots,\Box C_n, \Box D \Rightarrow}$ (see, for instance, [10]). To translate both rules we use the Internalization Property (Lemma 3).

Consider the K-rule first. By IH, we already have a Hilbert derivation of $C_1^r,\dots,C_n^r \vdash B^r$. By Lemma 3, $x_1 : C_1^r,\dots,x_n : C_n^r \vdash t : B^r$ for some t, where each x_i is the chosen realization of the negative $\Box$ in front of C_i. We then substitute t for the auxiliary variable that corresponds to this modal rule in the sum realization of the $\Box$ in front of B throughout the Hilbert proof.

The D-rule is similar. Here $x_1 : C_1^r,\dots,x_n : C_n^r, x_{n+1} : D^r \vdash t : \bot$ is obtained after Internalization. Using axiom A7, $t : \bot \to \bot$, and *modus ponens*, we can derive $\bot$. Since no positive $\Box$ is introduced, there is no global substitution of auxiliary variables.

The proof of Lemma 3 shows that the Axiom Internalization Rule in the internalized derivation appears only where axioms or Axiom Internalization Rule instances were in the original derivation. We are free to pick a fresh constant every time. So how can a self-referential cycle appear if we always pick fresh constants? Where does it appear for stronger modal logics? When a term t substitutes for an auxiliary variable v, which appears in an Axiom Internalization instance $c : A(v)$, the constant c can *a priori* occur in t. As shown in Sect. 4 and [5], this cannot be avoided in many logics with other modal Gentzen rules.

We show how to avoid such occurrences of c in t for K and D while staying within (6). Let us define the *depth of an occurrence of $\Box$ in a modal formula F* by induction on the size of F: the outer $\Box$ in $\Box G$ has depth 0 in $\Box G$; for any occurrence of $\Box$ inside G, its depth in $\Box G$ is obtained by adding 1 to its depth in G.

Let us also define the *level of an occurrence of $\Box$ in a Gentzen derivation* as its depth in the formula in which it occurs plus the number of modal rules used on its branch after this occurrence. It is easy to prove that

Lemma 4. *In a Gentzen K or D derivation of $\Rightarrow G$, levels of all occurrences of $\Box$ from a given family are equal to the depth of the family's occurrence in G.*

Let N be the largest level of $\Box$'s in the given cut-free derivation. As we showed, a new constant can be introduced only during Internalization while translating a modal rule. For all rules of level i, let us always use constants of level $N - i$. When constants introduced later on a branch refer to constants introduced on this branch earlier, the former have larger levels because the levels of modal rules decrease toward the root of the derivation. It remains to show that the substitution of terms for auxiliary variables does not violate the level structure of (6).

Indeed, every time a modal rule is used on a branch, all $\Box$'s it introduces have the level of this rule, say m, which is strictly smaller than the levels of all $\Box$'s already on the branch. Suppose the Internalization used to translate this modal rule introduced an Axiom Internalization $c\!:\!A(v)$ with an auxiliary variable v. This v corresponds to a family of $\Box$'s already present on the branch, which must have a larger level $l > m$. Wherever the modal rule corresponding to v occurs, by Lemma 4, it has the same level l. Therefore, when a term t substitutes for v, all the constants in t will have level $N - l < N - m = \ell(c)$. Thus, substitutions do not violate the conditions of our constant specification. $\Box$

6 Conclusions and Future Research

Further studies of self-referentiality can develop in various directions. We still do not know an example when self-referentiality is required, but direct self-referentiality can be avoided.

Self-referentiality results can be used to prove structural properties of Gentzen modal derivations, e.g., the unavoidability of double introduction of the same family of $\Box$'s on the same branch for directly self-referential modal logics.

It remains to see what triggers self-referentiality. It appears that self-referentiality is tied to the ability to mix levels of $\Box$'s in a Gentzen derivation, but we need a larger sample set to make any definite conclusions. We conjecture that the statement of Lemma 4 can be viewed as a purely modal formulation of a sufficient criterion for non-self-referentiality. It would be interesting to see whether it is also necessary.

Acknowledgements. The author is greatly indebted to Sergei Artemov, Melvin Fitting, and Vladimir Krupski, whose advice helped to shape this paper. Many thanks to Galina Savukova for editing this text.

References

[1] Artemov, S.N.: Operational modal logic. Technical Report MSI 95–29, Cornell University (1995)

[2] Artemov, S.N.: Explicit provability and constructive semantics. Bulletin of Symbolic Logic 7(1), 1–36 (2001)

[3] Artemov, S.N.: Justification logic. Technical Report TR-2007019, CUNY Ph.D. Program in Computer Science (2007)

[4] Brezhnev, V.N.: On explicit counterparts of modal logics. Technical Report CFIS 2000–05, Cornell University (2000)

[5] Brezhnev, V.N., Kuznets, R.: Making knowledge explicit: How hard it is. Theoretical Computer Science 357(1–3), 23–34 (2006)

[6] Fitting, M.: The logic of proofs, semantically. Annals of Pure and Applied Logic 132(1), 1–25 (2005)

[7] Kuznets, R.: Complexity Issues in Justification Logic. PhD thesis, CUNY Graduate Center (2008)

[8] Pacuit, E.: A note on some explicit modal logics. In: Proceedings of the 5th Panhellenic Logic Symposium, Athens, Greece, July 25–28, 2005, University of Athens (2005)
[9] Rubtsova, N.: Evidence reconstruction of epistemic modal logic $\mathsf{S5}$. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 313–321. Springer, Heidelberg (2006)
[10] Wansing, H.: Sequent calculi for normal modal propositional logics. Journal of Logic and Computation 4(2), 125–142 (1994)

Appendix

Lemma 1

For any subterm s of term t':

1. If $\mathcal{A}_0(s,F)$, then $\mathsf{L}_{\mathcal{CS}} \vdash F$ and F does not contain occurrences of t'.
2. If $\mathcal{A}(s,F)$, but $\neg\mathcal{A}_0(s,F)$, then F has at least one occurrence of t'. Moreover, the only such implication is $F = \neg(P \rightarrow t' : P)$.

Proof. The proof is by induction on the size of s.

(A) **Case $\mathbf{s = x}$**, a justification variable.
1. For any F, we have $\neg\mathcal{A}_0(x,F)$, so Claim 1 is vacuously true.
2. $\mathcal{A}(x,F)$ only if $t = x$ and $F = \neg(P \rightarrow t' : P)$, which does contain t' and is the only allowed implication.

(B) **Case $\mathbf{s = c}$**, a justification constant.
1. If $\mathcal{A}_0(c,F)$, formula F must be either an axiom or an instance of the Axiom Internalization Rule. In either case, F is derivable. At the same time, $\mathcal{CS}$ is not directly self-referential, so F cannot contain occurrences of c, a subterm of t'. Thus, F cannot contain t' either.
2. $\mathcal{A}(c,F)$, but $\neg\mathcal{A}_0(c,F)$ only if $t = c$ and $F = \neg(P \rightarrow t' : P)$, which does contain t' and is the only allowed implication.

(C) **Case $\mathbf{s = s_1 + s_2}$** .
1. If $\mathcal{A}_0(s_1{+}s_2,\ F)$, then, by the closure condition C3, $\mathcal{A}_0(s_i,F)$ for some $i = 1,2$. By IH, F is a theorem that does not contain t'.
2. If $\mathcal{A}(s_1{+}s_2,\ F)$, but $\neg\mathcal{A}_0(s_1{+}s_2,\ F)$, then either
(α) $t = s_1 + s_2$ and $F = \neg(P \rightarrow t' : P)$, which satisfies Claim 2, or else
(β) by C3, $\mathcal{A}(s_i,F)$, but $\neg\mathcal{A}_0(s_i,F)$ for some $i = 1,2$. By IH, F contains t', and, if an implication, is $\neg(P \rightarrow t' : P)$.

(D) **Case $\mathbf{s = s_1 \cdot s_2}$** .
1. If $\mathcal{A}_0(s_1 \cdot s_2,\ F)$, by C2, there must exist a formula G such that $\mathcal{A}_0(s_1,\ G \rightarrow F)$ and $\mathcal{A}_0(s_2,G)$. By IH, both $G \rightarrow F$ and G are derivable, hence F is derivable by *modus ponens*. By IH, $G \rightarrow F$ does not contain t', thus neither can F.
2. If $\mathcal{A}(s_1{\cdot}s_2,\ F)$, but $\neg\mathcal{A}_0(s_1{\cdot}s_2,\ F)$, there are several possibilities:
(α) $t = s_1 \cdot s_2$ and $F = \neg(P \rightarrow t' : P)$, which satisfies Claim 2; or else
by C2, there should exist a G such that either

(β) $\mathcal{A}(s_1,\ G \to F)$ and $\mathcal{A}(s_2, G)$ while $\neg\mathcal{A}_0(s_1,\ G \to F)$ or

(γ) $\mathcal{A}(s_1,\ G \to F)$ and $\mathcal{A}(s_2, G)$ while $\neg\mathcal{A}_0(s_2, G)$.

We will show that both subcases (β) and (γ) are inconsistent.

In subcase (β), by IH, Claim 2 for subterm s_1, $G \to F = \neg(P \to t' : P) = (P \to t' : P) \to \bot$. So $G = P \to t' : P$, which is another implication. Hence, by IH, Claim 2 for s_2, we should have $\mathcal{A}_0(s_2, G)$, which contradicts the IH, Claim 1 for s_2 since $P \to t' : P$ contains t'. The contradiction shows impossibility of subcase (β).

In subcase (γ), by IH, Claim 2 for s_2, formula G should contain t'. Then $G \to F$ would also contain t'. Hence, by IH, Claim 1 for s_1, we should have $\neg\mathcal{A}_0(s_1,\ G \to F)$, and we are back in the impossible subcase (β). So subcase (γ) is also impossible.

(E) ***Case*** $\mathbf{s = !\,s_1}$ (only for logics $\mathsf{J4}_{\mathcal{CS}}$, $\mathsf{JD4}_{\mathcal{CS}}$, and $\mathsf{LP}_{\mathcal{CS}}$).

1. If $\mathcal{A}_0(!s_1,\ F)$, then, by C5, $F = s_1 : G$ for some G such that $\mathcal{A}_0(s_1, G)$. By IH, Claim 1, G is a theorem that does not contain t'. $\mathcal{A}_0(s_1, G)$ implies that $\mathcal{A}'(s_1, G) = W$ in any model $\mathcal{M}' = \langle W', R', \mathcal{A}', v' \rangle$ for $\mathsf{L}_{\mathcal{CS}}$. In any such model, $\mathcal{M}', w' \Vdash G$ for all $w' \in W'$ by the Soundness part of Theorem 2. By definition of $\Vdash$, it follows that $\mathcal{M}', w' \Vdash s_1 : G$ for any world $w' \in W'$ in any $\mathcal{M}'$. By the Completeness part of Theorem 2, $s_1 : G$ is derivable.

Since G does not contain t' and s_1 is a proper subterm of t', formula $s_1 : G$ cannot contain t' either.

2. If $\mathcal{A}(!s_1,\ F)$, but $\neg\mathcal{A}_0(!s_1,\ F)$, then either

(α) $t = !\,s_1$ and $F = \neg(P \to t' : P)$, which satisfies Claim 2, or else

(β) by C5, $F = s_1 : G$ for some G such that $\mathcal{A}(s_1, G)$, but $\neg\mathcal{A}_0(s_1, G)$. By IH, Claim 2, G contains t', thus so does $s_1 : G$, which is not an implication. $\square$

On the Complexity of Membership and Counting in Height-Deterministic Pushdown Automata

Nutan Limaye[1], Meena Mahajan[1], and Antoine Meyer[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113, India
{nutan,meena}@imsc.res.in
[2] LIAFA, Université Paris Diderot – Paris 7, Case 7014,
75205 Paris Cedex 13, France
ameyer@liafa.jussieu.fr

Abstract. While visibly pushdown languages properly generalise regular languages and are properly contained in deterministic context-free languages, the complexity of their membership problem is equivalent to that of regular languages. However, the corresponding counting problem could be harder than counting paths in a non-deterministic finite automaton: it is only known to be in LogDCFL.

We investigate the membership and counting problems for generalisations of visibly pushdown automata, defined using the notion of height-determinism. We show that, when the stack-height of a given PDA can be computed using a finite transducer, both problems have the same complexity as for visibly pushdown languages. We also show that when allowing pushdown transducers instead of finite-state ones, both problems become LogDCFL-complete; this uses the fact that pushdown transducers are sufficient to compute the stack heights of all real-time height-deterministic pushdown automata, and yields a candidate arithmetization of LogDCFL that is no harder than LogDCFL(our main result).

1 Introduction

There is a close connection between complexity classes and formal language theory. Over the years, various language classes have been studied from the complexity theoretic perspective. Capturing the complexity of membership has been the goal in this approach. The study of language classes and their complexity under meaningful closures was first started by Sudborough [1,2]. In [1], he showed that the *nondeterministic linear context-free languages* or LIN are complete for the complexity class NL (nondeterministic log-space). In [2], he defined two interesting complexity classes, namely LogCFL and LogDCFL, as the log-space closures of context-free languages (CFLs) and their deterministic counterparts (DCFLs) respectively. Ibarra, Jiang and Ravikumar [3] further studied subclasses of CFL such as $\mathsf{DLIN}_{LL(1)}$ (the deterministic counterpart of LIN defined by $LL(1)$ linear grammars), *Dyck2* and *bracketed expressions* and showed that they are contained in NC^1. Holzer and Lange [4] showed that deterministic linear context-free languages (DLIN), as defined via $LR(1)$ linear grammars, are equivalent to those accepted by deterministic 1-turn automata $\mathsf{DPDA}_{\text{1-turn}}$ (deterministic pushdown automata that never push after a pop move). They showed that deciding membership in a DLIN language is complete for L, in contrast to the result of [3]. Barrington made an important contribution to the above study [5], showing that the class of regular languages, REG, is complete for the circuit complexity class NC^1 comprising of polynomial size log depth bounded fan-in AND-OR circuits. (As the classes get smaller, completeness is via not L -reductions but appropriate weaker notions such as AC^0-many-one-reductions.) See [6] for an overview of these results.

Visibly pushdown automata (VPA) are real-time pushdown automata whose stack behaviour is dictated solely by the input letter under consideration. They are also referred to as input-driven PDA. The membership problem for VPL was considered in [7,8,9]; [9] shows that languages accepted by such PDA are in NC^1. A rigorous language-theoretic study of VPA was done in [10], where it is shown that they can be determinised. Thus they lie properly between REG and DCFLs, and their membership problem is complete for NC^1.

A related line of study is understanding the power of counting. It is easy to see from the proof of [1] that counting the number of parse trees in a linear grammar, #LIN, is equivalent to counting accepting paths in an NL machine. At the lower end, however, though Barrington's result showed that deciding membership in REG (and hence in the language of a nondeterministic finite-state automaton or NFA) is equivalent to NC^1, counting the number of accepting paths in an NFA (#NFA) is not yet known to be equivalent to arithmetic NC^1, $\#\mathsf{NC}^1$. In [11], a one-way containment is shown: $\#\mathsf{NFA} \subseteq \#\mathsf{NC}^1$, but to this day the converse reduction remains open[1]. A natural question to ask is what generalisation of NFA can capture $\#\mathsf{NC}^1$ in this setting. In [12], it was claimed that the generalisation to VPA adds no power, #VPA is equivalent to #NFA. However, this claim was later retracted in [13], where it is shown, however, that #VPA functions can be computed in LogDCFL (and are hence presumably weaker than #PDA functions).

Our starting point in this note is a careful examination of what makes membership and path-counting easier in VPA than in general PDA. The intention is to identify a largest possible class of PDA for which the technique used for VPA can still be applied. This technique exploits the fact that, despite nondeterminism, all paths on a given input word have the same stack-profile, and furthermore, this profile is very easily computable. One can view the partitioning of the input alphabet as height advice being provided to an algorithm for deciding membership. This naturally leads to the conjecture that PDA possessing easy-to-compute height advice functions should be easier than general PDA. The *real-time height-deterministic* PDA defined by Nowotka and Srba ([14]), rhPDA, are a natural candidate: they are defined as precisely those PDA that possess height advice functions. They also very naturally generalise a subclass of the *synchronised* PDA defined by Caucal ([15]), namely the subclass where the synchronisation function is the stack-height, and, as in general synchronised PDA, is computable

[1] [11] shows a weaker converse: every $\#\mathsf{NC}^1$ function can be expressed as the difference of two #NFA functions.

by a finite-state transducer. We provide a parameterised definition of rhPDA that captures this generalisation: the complexity of the transducer computing the height advice function is the parameter. We then examine the complexity of membership and path-counting at either end of the parameterisation.

A related model equivalent to VPA is that of nested word automata (NWA), defined in [16] with a view to applications in software verification and XML document processing. [17] defines motley word automata (MWAs) as a generalisation of NWAs. Our techniques can be used to show that the equivalence is indeed very strong: deciding membership and counting accepting paths in NWA and MWA are NC^1-equivalent to the same problems over VPA.

2 Preliminaries

A *pushdown automaton* (PDA) over Σ is a tuple $P = (Q, q_0, F, \Gamma, \Sigma, \delta)$ where Q is a finite set of control states, $q_0 \in Q$ the initial state, $F \subseteq Q$ a set of accepting states, Γ a finite alphabet of stack symbols, Σ a finite alphabet of labels and δ a finite set of transition rules $pU \xrightarrow{a} qV$ with $p, q \in Q$, $U, V \in \Gamma^*$ and $a \in \Sigma$. In the following, we will only consider weak PDA, whose rules are such that $|UV| \leq 1$.[2]

A *configuration* of P is a word of the form pW with $p \in Q$ and $W \in \Gamma^*$, where p is the current control state and W is the current stack content read from top to bottom. The stack height at configuration pW is $|W|$.

The semantics of P are defined with respect to its transition graph $G_P = \{pUW \xrightarrow{a} qVW \mid pU \xrightarrow{a} qV \in \delta,\ W \in \Gamma^*\}$. A *run* of P on input word $w \in \Sigma^*$ from configuration pW is a path in G_P between vertex pW and some vertex qW', written $pW \xrightarrow{w} qW'$. Such a run is successful (or accepting) if $pW = q_0$ and q belongs to the set F of accepting states of P. By $L(G_P, S, T)$ where S, T are sets of vertices of G_P, we mean all words $w \in \Sigma^*$ such that $c \xrightarrow{w} c'$ for some configurations $c \in S$, $c' \in T$. The *language* of P is the set of all words w over which there exists an accepting run; i.e. it is the language $L(G_P, \{q_0\}, F\Gamma^*)$.

A *visibly* pushdown automaton (VPA) over Σ is a weak PDA P where Σ is partitioned as $\Sigma_c \cup \Sigma_r \cup \Sigma_i$, and for all $p \in Q$, $a \in \Sigma$, $p \xrightarrow{a} q\gamma \Rightarrow a \in \Sigma_c$ (a push move/*call*), $p\gamma \xrightarrow{a} q \Rightarrow a \in \Sigma_r$ (a pop move/*return*) and $p \xrightarrow{a} q \Rightarrow a \in \Sigma_i$ (an internal move). VPA can be nondeterministic, i.e. δ need not be a function. VPA are equivalent to the earlier notion of input-driven automata when run on well-matched strings, and their languages can be easily reduced to input-driven languages, as observed in [12,13].

For any class $\mathcal{C}$ of automata, its arithmetic version $\#\mathcal{C}$ is defined as follows:

$$\#\mathcal{C} = \{f : \Sigma^* \to \mathbb{N} \mid \text{ for some } M \in \mathcal{C},\ f(x) = \#\mathsf{acc}_M(x) \text{ for all } x \in \Sigma^*\}$$

Proposition 1 ([11,13]). *The following inclusions hold:*

$$\#\mathsf{NFA} \subseteq \#\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{LogDCFL} \qquad \#\mathsf{NFA} \subseteq \#\mathsf{VPA} \subseteq \mathsf{LogDCFL}$$

[2] This is only for simplicity, as all results go through for arbitrary $U, V \in \Gamma^*$.

(A containment $\mathcal{F} \subseteq \mathcal{C}$ involving both a function class $\mathcal{F}$ and a language class $\mathcal{C}$ means: $\forall f \in \mathcal{F}, L^f \in \mathcal{C}$, where $L^f = \{\langle x, i, b\rangle \mid \text{the } i\text{th bit of } f(x) \text{ is } b\}$.)

3 Revisiting the VPL in NC^1 Proof

In [9], Dymond proved that the membership problem for VPL is in NC^1. Dymond's proof transforms the problem of recognition/membership to efficiently evaluating an expression whose values are binary relations on a finite set and whose operations are functional compositions and certain unary operations depending on the inputs. This transformation is done in NC^1. Containment in NC^1 follows from the result, due to Buss [18], that the evaluation of formulae involving expressions as k-ary functions over a finite domain is in NC^1.

For a VPA, on an input w, the stack height after processing i letters, $h(i, w)$ (or simply $h(w)$ if $i = |w|$), is the same across any run. Define a set of binary relations, denoted $\Rightarrow^{i,j}$ for $1 \leq i \leq j \leq |w|$, on surface configurations $(q, \gamma) \in Q \times \Gamma$ (state and stack-top pair). These relations are expected to capture all cases where surface configurations are reachable from one another without accessing the previous stack profiles. A unary operation, and a composition operation, are defined on these relations. Given a string w, the main work is to figure out the correct indices for the relations and then the appropriate operations. But that can be accomplished essentially by computing stack heights for various configurations, which is easy for VPL.

As pointed out in [9], the above transformation works for more than VPAs.

Remark 1 ([9]). Dymond's NC^1 membership algorithm works for any pushdown automaton M satisfying the following three conditions.

- There should be no ϵ-moves.
- Accepting runs should end with an empty stack (and a final state).
- There should exist an NC^1-computable function h such that for $w \in \Sigma^*$ and $0 \leq i \leq |w|$, $h(i, w)$ is the height of the stack after processing the first i symbols of w. If M is non-deterministic, then $h(i, w)$ should be consistent with some run ρ of M on w; further, if M accepts w, then ρ should be an accepting run.

Clearly, VPA satisfy these conditions. By definition, they have no ϵ-moves. Though they may not end with an empty stack, this can be achieved by appropriate padding that is computable in TC^0, see for instance [12,13]. (TC^0 is a subclass of NC^1 consisting of polynomial size constant depth circuits where each gate is allowed unbounded fan-in, and gates compute MAJORITY and NOT.) Though VPA may be nondeterministic, all runs have the same height profile, and the function $h(i, w)$ can in fact be computed in TC^0.

Since any computation up to NC^1 can be allowed for Dymond's proof to go through, VPL do not fully exploit Dymond's argument. We explore a natural generalisation of VPA allowing us to define natural classes for which Dymond's scheme or its precursor from [8] may work for deciding membership, and then examine the power of counting in these models.

4 More General Height Functions: Height-Determinism

Adding a pushdown stack to an NFA significantly increases the complexity of both membership (regular to context-free) and path-counting (#NFA to #PDA). However, if stack operations are restricted to an input-driven discipline, as in VPA, then membership is no harder than for NFA, and path-counting seems easier than over general PDA. What is being exploited is that, despite nondeterminism, all paths on a given input word have the same stack-profile, and this profile is computable in NC^1 (and even in TC^0). One can view the partitioning of the input alphabet as height advice being provided to an algorithm for deciding membership. This naturally leads to the question: what can be deduced from the existence of such height advice, independently of how this function is computed?

The term *height-determinism*, coined by [14], captures precisely this idea. A PDA is height-deterministic if the stack height reached after any partial run depends only on the input word w which has been read so far, and not on non-deterministic choices performed by the automaton. Consequently, in any (real-time) height-deterministic pushdown automaton (rhPDA), all runs on a given input word have the same stack profile. Another way to put it is that for any rhPDA P, there should exist a *height-advice function* h from Σ^* to integers, such that $h(w)$ is the stack-height reached by P on any run over w.

Any rhPDA that accepts on an empty stack and whose height-advice function h is computable in NC^1 directly satisfies the conditions in Remark 1, and hence its membership problem lies in NC^1. In this section, we explore some sub-classes of rhPDA and discuss the complexity of their membership and counting problems.

Let us first give a formal definition of rhPDA.

Definition 1 (rhPDA, [14]). *A real-time (weak) pushdown automaton*[3] $P = (Q, q_0, F, \Gamma, \Sigma, \delta)$ *is called height-deterministic if it is complete (does not get stuck on any run), and* $\forall w \in \Sigma^*$, $q_0 \xrightarrow{w} q\alpha$ *and* $q_0 \xrightarrow{w} q\beta$ *imply* $|\alpha| = |\beta|$.

The robustness of this notion is illustrated by the fact that rhPDA retain most good properties of VPA, even when the actual nature of the height-advice function is left unspecified. This had already been obtained in [15] for a slightly different class (which the authors of [14] admittedly used as a starting point in the elaboration of their paper).

Proposition 2 ([14,15]). *Any* rhPDA *can be determinised. Consequently, for a fixed h, the class of languages accepted by* rhPDA *and whose height advice function is h forms a boolean algebra (and properly includes regular languages). Moreover, language equivalence between two* rhPDA *with the same height-advice function is decidable.*

All these results are effective as soon as h is computable. Since any deterministic real-time PDA is also height-deterministic, another consequence of the fact that rhPDA can be determinised is that the whole class rhPDA accepts precisely the class of real-time DCFL.

[3] In [14], the definition involves rules of the form $pX \xrightarrow{a} q\alpha$ where $\alpha \in \{\epsilon, X\} \cup \{YX | Y \in \Gamma\}$. This is not an essential requirement for the results presented here.

4.1 Instances of Height-Deterministic PDA

The definition of a rhPDA leaves the exact nature of the height-advice function h unspecified. This is troublesome, since h could be arbitrarily complex. We consider some classes of specific height-advice functions, the simplest being VPA.

Following the framework developed by Caucal [15], we consider classes $\mathcal{T}$ of transducers mapping words to integers. A *transducer* T over Σ and $\mathbb{Z}$ is a transition system $(C, c_0, F, (\Sigma \times \mathbb{Z}), \delta)$, where c_0 denotes the initial configuration and F a set of final configurations, and whose transitions described by δ are labelled with pairs (a, k), where a is a letter and k an integer. The first component of any such label is considered as an input, and the second component as an output. A run $c_0(a_1, k_1)c_1 \ldots c_{n-1}(a_n, k_n)c_n$ is associated to the pair $(w, k) = (a_1 \ldots a_n, k_1 + \ldots + k_n)$. Such a transducer defines a relation $g_T \subseteq \Sigma^* \times \mathbb{Z}$ defined as the set of all pairs (w, k) labelling an accepting run in T.

In our setting, we only consider both input-complete and input-deterministic transducers (i.e. transducers whose underlying Σ-labelled transition system is deterministic and complete), in which all configurations are final (in which case we omit F in the definition). Consequently, for any such transducer T the relation g_T is actually a function, and is defined over the whole set Σ^*. The transition graph G_P of a PDA P is said to be *compatible* with a transducer T if for every vertex s of G_P, if $u, v \in L(G_P, \{q_0\}, \{s\})$ then $g_T(u) = g_T(v)$.

One may consider several kinds of transducers. The simplest class is *finite-state transducers* (FST), where the configuration space C is simply a finite set of control states (often written Q). One may also consider *pushdown transducers* (PDT) whose underlying Σ-labelled transition system is a PDA transition graph, or even more complex transducers (for instance defined using Turing machines).

Definition 2. *For any class $\mathcal{T}$ of complete deterministic transducers,* rhPDA($\mathcal{T}$) *is the class of* rhPDA *whose height function h can be computed by a transducer T in $\mathcal{T}$, in the sense that $h(w) = |g_T(w)|$ (absolute value of $g_T(w)$) for all w.*

The height-advice function of any VPA running on well-matched strings can be computed by a single-state transducer, that reads letters and outputs $+1$ or -1 or 0 depending on whether the letter is in Σ_c or Σ_r or Σ_i. However, note that such single-state transducers can also compute stack-heights for languages that are provably not in VPL. Also, allowing more than one state in a FST provably enlarges the class of languages.

Example 1. The language $EQ(a, b) = \{w \mid |w|_a = |w|_b\}$ is not accepted by any VPA for any partition of $\{a, b\}$. But a single-state transducer can compute the stack-height of the obvious DPDA acceptor: it outputs $+1$ on a and -1 on b.

The language $REV = \{wcw^R \mid w \in \{a, b\}^*\}$ is not a VPL. The obvious DPDA acceptor has a height function computable by a two-state transducer: one state has $+1$ on a and b, the other has -1 on a and b, and moves to second state on c. It is easy to see that for any PDA accepting REV, two states in the transducer are essential for computing stack height.

Further, [14] provided a separating example in rhPDA but not in rhPDA(FST); from Proposition 3 below, it follows that this language is in fact in rhPDA(PDT). In the remainder of this section, we will focus on the classes rhPDA(FST) and rhPDA(PDT), and also to some extent on the class rhPDA($\mathsf{rDPDA}_{\mathsf{1\text{-}turn}}$), where the transducer is a 1-turn PDT.

The class we define as rhPDA(FST) is a restricted (and simpler) subclass of the synchronised pushdown automata considered by Caucal in [15]. Even though Caucal's results require, for a PDA to be synchronised by a transducer T, that the transition graph of P' satisfy some additional geometric properties with respect to g_T, these properties are always satisfied when only considering stack-height[4]. One can thus see rhPDA(FST) as the intersection of rhPDA with synchronised PDA. As an aside, we note that [14] considers the class rhPDA(FST) as equivalent to synchronised PDA. This is not guaranteed to be true and has to be proved, since [15] also permits synchronisation by norms other than stack-height.

Finally, we note that since, by definition, rhPDA are complete, it is in fact unnecessary to consider more complex transducers than deterministic and complete PDTs. Formally:

Proposition 3. *For any* rhPDA P *whose height-advice function is* h*, there exists a deterministic and complete pushdown transducer* T *such that* $h(w) = g_T(w)$ *for all* $w \in \Sigma^*$*. That is, every* rhPDA *is in* rhPDA(PDT)*.*

4.2 Complexity of the Membership Problem

As we already mentioned, rhPDA have exactly the same power as real-time DPDA in terms of accepted languages. This settles the complexity of the membership question for the whole class rhPDA (and thus also for rhPDA(PDT)): it is in LogDCFL, and since the hardest DCFL ([2]) is hard for the class LogDCFL and is accepted by a real-time DPDA, it is also hard for LogDCFL.

We observe easy bounds on the complexity of the height-advice function.

Lemma 1. *For a complete deterministic transducer* T *computing function* g_T*,*

1. *If* T *is a* FST*, then* g_T *is computable in* NC^1*.*
2. *If* T *is a* $\mathsf{rDPDA}_{1\text{-}turn}$*, then* g_T *is computable in* L*.*
3. *If* T *is a* PDT*, then* g_T *is computable in* LogDCFL*.*

This allows us to apply Dymond's algorithm for rhPDA(FST).

Lemma 2. *For any fixed* rhPDA(FST)*, the membership problem is in* NC^1*.*

This membership algorithm exploits Dymond's construction better than VPA, as the height function requires a possibly NC^1-complete computation (predicting states of the transducer). Recall that for VPA, the height function is computable in TC^0, a subclass of NC^1.

[4] In the terminology of [15], this is due to the fact that all transition graphs of pushdown automata are regular by stack height.

In [8], the membership problem for VPLs is shown to be in L. We observe that their algorithm can be more explicitly implemented as L^g where g is the height function of the VPL. In this form, it can be generalised to any rhPDA having height function g, as stated in Theorem 1 below. The proof follows from Lemmas 3 and 4, and the result, along with Lemma 1, yields the next corollary.

Theorem 1. *For any fixed* rhPDA P *with height function* g*, the membership problem is in* L^g*.*

Corollary 1

1. *The membership problem for* rhPDA($\mathsf{rDPDA}_{\mathsf{1\text{-}turn}}$) *is in* L*.*
2. *The membership problem for* rhPDA *is in* LogDCFL*.*

The class rhPDA($\mathsf{rDPDA}_{\mathsf{1\text{-}turn}}$) referred to here contains languages accepted by real-time $\mathsf{DPDA}_{\mathsf{1\text{-}turn}}$ as well as languages accepted by rhPDA(FST). It is contained in DCFLs. The upper bound for rhPDA follows from [14], where it is shown that rhPDA as a language class equals the DCFLs accepted by DPDA with no ϵ-moves, and so is a proper subclass of DCFLs.

Lemma 4 uses the algorithm from [8] to establish the L^{g_T} bound for well-matched inputs, and Lemma 3 brings the input in that form.

Lemma 3. *For every* rhPDA(T) P *over an alphabet* Σ*, there is a corresponding* rhPDA(T') P' *over an alphabet* Σ' *and a* $\mathsf{L}^{g_{T'}}$ *many-one reduction* f *such that for every* $x \in \Sigma^*$*,* $\#\mathsf{acc}_P(x) = \#\mathsf{acc}_{P'}(f(x))$*, and* $f(x)$ *is well-matched.*

Lemma 4 (Algorithm 2 of [8], stated differently). *Let* $P = (Q, \Sigma, Q_{in}, \Gamma, \delta, Q_F)$ *be a* rhPDA(T) *accepting well-matched strings. Given an input string* x*, checking if* $x \in L(P)$ *(membership test for* $L(P)$*) can be done in* L^{g_T}*.*

4.3 Complexity of the Counting Problem

The aspect of rhPDA which interests us in this study is that it is a nondeterministic model capturing the deterministic class LogDCFL. It thus provides a way of arithmetising LogDCFL, simply by counting the number of accepting paths on each word in a rhPDA. We call the class of such functions #rhPDA. In particular, we consider the classes #rhPDA(FST) and #rhPDA(PDT).

We have seen that although rhPDA(FST) properly generalises VPA, the membership problem has the same complexity as that over VPA. It turns out that even the path-counting problem has the same complexity.

Theorem 2. #rhPDA(FST) $\equiv$ #VPA *(via* NC^1 *many-one reductions).*

Proof Sketch. VPA are contained in rhPDA(FST), so we only need to show that computing #rhPDA(FST) functions reduces to computing #VPA functions.

Let P be an rhPDA with height-advice computed by FST T. A naive approach would be to construct a single PDA P' that simulates (P, T) by running PDA P along with transducer T. However, such a PDA P' will not necessarily be a VPA. Now consider the string rewritten using an enriched alphabet which consists of the input letter along with a tag indicating whether P should push or pop. On this enriched alphabet, if the tags are correct, then a PDA that simulates the original PDA P (i.e. ignores the tags) behaves like a VPA. But by Lemma 1, the correct tags for any word can be computed in NC^1. □

Theorem 3 shows that membership and counting for rhPDA have the same complexity, a situation rather unusual for nondeterministic complexity classes.

Theorem 3. *#rhPDA is in LogDCFL.*

The proof of this theorem proceeds in several stages. To compute a #rhPDA function f on input x, we first compute $f(x)$ modulo several small (logarithmic) primes, and then reconstruct $f(x)$ from these residues. This is the standard Chinese remainder technique (see for instance [19]), stated formally below.

Lemma 5 (folklore). *Let P be a fixed rhPDA. There is a constant $c \geq 0$, depending only on P, such that given input x, the number of accepting paths of P on input x can be computed in logarithmic space with oracle access to the language L_{res} defined below. (Here p_i denotes the ith prime number.)*

$$L_{res} = \{\langle x, i, j, b\rangle | 1 \leq i \leq |x|^c, \text{ the } j\text{th bit of } \#\mathsf{acc}_P(x) \bmod p_i \text{ is } b\ \}$$

We now show that L_{res} can be computed by a polynomial time DAuxPDA machine – a deterministic polynomial time PDA with $O(\log n)$ auxiliary space, this model characterises LogDCFL – making oracle queries to the height-advice function g_T. This follows from the technique of [8] as used in [13] to show that #VPA functions are in LogDCFL.

Lemma 6. *If P is any rhPDA and T a PDT computing its height-advice function, then L_{res} is in $\mathsf{LogDCFL}^{g_T}$.*

Lemmas 1 and 6 together imply that L_{res} is in LogDCFL(LogDCFL). This is not adequate for us, since it is not known whether LogDCFL(LogDCFL) $\subseteq$ LogDCFL. (Relativising a space-bounded class is always tricky. Here, we have a pushdown class with auxiliary space, making the relativisation even more sensitive.) However, we further note that the $\mathsf{LogDCFL}^{g_T}$ machine accepting L_{res} makes oracle queries which all have short representations: each query can be written in logarithmic space. (Strictly speaking, the input x is also part of the query. But for eliminating the oracle, this plays no role.) In such a case, we can establish a better bound, which may be of independent interest:

Lemma 7. *Let $L(M^A)$ be the language accepted by a poly-time DAuxPDA M which makes $O(\log n)$-bits oracle queries to a language $A \in$ LogDCFL. Then $L(M^A) \in$ LogDCFL.*

Combining these lemmas proves Theorem 3, since L(LogDCFL) equals LogDCFL.

5 Related Models: Nested and Motley Words

In [16], Alur and Madhusudan defined nested word automata (NWA) as an equivalent model for VPA, motivated by applications in software verification and XML document processing. In [17], Blass and Gurevich defined motley word automata (MWAs) as a generalisation of NWAs. The definitions of models of NWA and MWA are orthogonal to the notion of height-determinism. However, we observe that their complexity bounds are the same as that of VPL for both membership and counting problems.

We begin with definitions of NWA and MWA.

A *nested relation* ν of width n, for $n \geq 0$, is a binary relation over $[1, n]$ such that (1) if $\nu(i, j)$ then $i < j$; (2) if $\nu(i, j)$ and $\nu(i', j')$ then either $\{i, j\} = \{i', j'\}$ or $\{i, j\} \cap \{i', j'\} = \emptyset$, and (3) if $\nu(i, j)$ and $\nu(i', j')$ and $i < i'$ then either $j < i'$ or $j' < j$.

If ν is a nested relation with $\nu(i, j)$, then i is the *call-predecessor* of j and j is the *return-successor* of i. The definition requires that each position has at most one call-predecessor or at most one return-successor but not both.

A nested word over an alphabet Σ is a pair (w, ν) such that $w \in \Sigma^*$, and ν is a nested relation of width $|w|$. A position $k \in [1, |w|]$ of w is a *call position* if $(k, j) \in \nu$ for some j, a *return position* if $(i, k) \in \nu$ for some i, and an *internal position* otherwise.

Definition 3 (NWA). *A nested word automaton (*NWA*) A over an alphabet Σ is a tuple $(Q, q_0, F, \Sigma, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states , $\delta = \langle \delta_c, \delta_i, \delta_r \rangle$ is a set of transitions such that $\delta_c \subseteq Q \times \Sigma \times Q$, $\delta_i \subseteq Q \times \Sigma \times Q$ and $\delta_r \subseteq Q \times Q \times \Sigma \times Q$ are the transitions for call, internal and return positions respectively.*

A starts in state q_0 and reads the word left to right. At a call or internal position, the next state is determined by the current state and input symbol, while at a return position, the next state can also depend on the state just before the matching call-predecessor. A run ρ of the automaton A over a nested word $nw = (a_1 \dots a_n, \nu)$ is a sequence $q_0, \dots, q_n$ over Q such that for each $1 \leq j \leq n$,

- if j is a call position, then $(q_{j-1}, a_j, q_j) \in \delta_c$
- if j is a internal position, then $(q_{j-1}, a_j, q_j) \in \delta_i$
- if j is a return position with call-predecessor k, then $(q_{j-1}, q_{k-1}, a_j, q_j) \in \delta_r$.

A accepts the nested word nw if $q_n \in F$. The language $L(A)$ of a nested-word automaton A is the set of nested words it accepts.

A *motley word* mw of dimension d over Σ is a tuple $(w, \nu_1, \dots, \nu_d)$, where $w \in \Sigma^*$ and $\nu_1, \dots, \nu_d$ are nested relations of width $|w|$.

Definition 4 (MWA). *A motley word automaton (*MWA*) A of dimension d is a direct product $A_1 \times \dots \times A_d$ of d* NWA *$A_1, \dots, A_d$.*[5]

A *run* of A on dimension d motley word $mw = (w, \nu_1, \dots, \nu_d)$ with $|w| = n$ is a sequence $(q_0^1, \dots, q_0^d), \dots, (q_n^1, \dots, q_n^d)$ of states of A such that every $(q_0^k, \dots, q_n^k)$ is a run of A_k on the nested word (w, ν_k). A run of A on mw is *accepting* (or mw is accepted by A) if each of the d constituent runs is. $L(A)$ is defined as usual.

[5] As NWA are in general non-deterministic, so are motley automata. A MWA $A_1 \times \dots \times A_d$ is deterministic if every nested word automata A_k is so.

The languages of nested/motley words accepted by NWA or MWA are called *regular* nested/motley languages. Regular motley languages strictly generalise regular nested languages [17], since for some $i \neq j$, the same position can be a call-position for ν_i and a return position for ν_j.

It is shown in [16] (Theorem 6) that for a fixed NWA, the membership question is in NC^1. The analogous question for a fixed MWA is easily seen to have the same complexity, since it involves answering membership questions for d different, but fixed, NWAs, where d is the dimension of the MWA.

In both models, NWA and MWA, non-determinism is allowed in the definition. We show that path-counting in NWA and MWA is equivalent to that in VPA. This does not follow from the equivalence of membership testing; rather, it requires that the equivalence be demonstrated by a parsimonious reduction.

Theorem 4. *Deciding membership and counting accepting paths in* NWA *and* MWA *are equivalent, via* NC^1*-many-one reductions, to the corresponding problems over* VPA*.*

6 Conclusion

We have studied a range of real-time height-deterministic pushdown automata lying between visibly and real-time deterministic pushdown automata. Fig. 1 depicts the relations between language classes, Fig. 2 shows their closures under appropriate reductions, and Fig. 3 shows the corresponding counting classes. (Dashed arrows indicate incomparability, dotted arrows containment, and solid arrows proper containment.)

Some open questions remain. First, it would be interesting to investigate additional classes lying between rhPDA(FST) and rhPDA(PDT). Also, the only known upper bound for #VPL, #rhPDA(FST) and #rhPDA(rDLIN), is LogDCFL. It would be interesting to refine this bound. Finally, all of our results concern height-deterministic PDA. However, [15] allows PDA to be synchronised by functions other than stack-height. It is not clear how many of our proofs carry over.

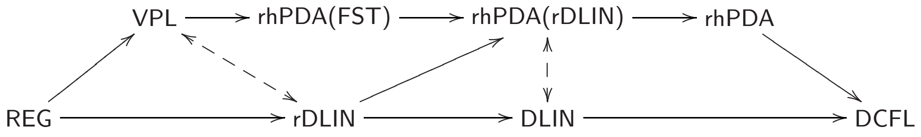


Fig. 1. Summary of language classes

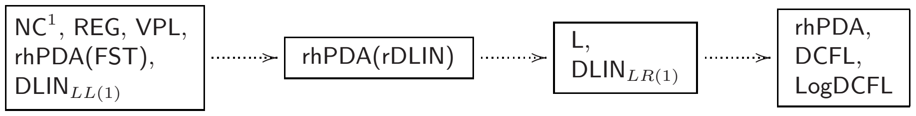


Fig. 2. Summary of language classes closures

$$\begin{tikzcd}
\mathsf{NC}^1 \arrow[r, dotted] & \#\mathsf{NFA} \arrow[r, dotted] & \boxed{\begin{array}{l}\#\mathsf{VPL},\\ \#\mathsf{rhPDA(FST)}\end{array}} \arrow[r, dotted] & \#\mathsf{rhPDA(rDLIN)} \arrow[r, dotted] & \#\mathsf{rhPDA} \arrow[r, dotted] & \mathsf{LogDCFL}
\end{tikzcd}$$

Fig. 3. Summary of counting classes

References

1. Sudborough, I.H.: A note on tape-bounded complexity classes and linear context-free languages. JACM 22(4), 499–500 (1975)
2. Sudborough, I.: On the tape complexity of deterministic context-free language. JACM 25(3), 405–414 (1978)
3. Ibarra, O., Jiang, T., Ravikumar, B.: Some subclasses of context-free languages in NC^1. IPL 29, 111–117 (1988)
4. Holzer, M., Lange, K.J.: On the complexities of linear LL(1) and LR(1) grammars. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 299–308. Springer, Heidelberg (1993)
5. Barrington, D.: Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. JCSS 38(1), 150–164 (1989)
6. Lange, K.J.: Complexity and structure in formal language theory. In: 8th CoCo, pp. 224–238. IEEE Computer Society, Los Alamitos (1993)
7. 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. 422–432. Springer, Heidelberg (1980)
8. Braunmuhl, B.V., 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)
9. Dymond, P.: Input-driven languages are in $\log n$ depth. IPL 26, 247–250 (1988)
10. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th STOC, pp. 202–211. ACM, New York (2004)
11. Caussinus, H., McKenzie, P., Thérien, D., Vollmer, H.: Nondeterministic NC^1 computation. JCSS 57(2), 200–212 (1998)
12. Limaye, N., Mahajan, M., Rao, B.V.R.: Arithmetizing classes arround NC^1 and L. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 477–488. Springer, Heidelberg (2007)
13. Limaye, N., Mahajan, M., Rao, B.V.R.: Arithmetizing classes arround NC^1 and L. Technical Report ECCC TR07- (2007) submitted to TCS (spl.issue for STACS 2007) (2007)
14. 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)
15. Caucal, D.: Synchronization of pushdown automata. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 120–132. Springer, Heidelberg (2006)
16. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
17. Blass, A., Gurevich, Y.: A note on nested words. Technical Report MSR-TR-2006-139, Microsoft Research (October 2006)
18. Buss, S.: The Boolean formula value problem is in ALOGTIME. In: 19th STOC, pp. 123–131. ACM, New York (1987)
19. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, Heidelberg (1999)

Public Key Encryption and Encryption Emulation Attacks

Denis Osin* and Vladimir Shpilrain

Department of Mathematics, The City College of New York, New York, NY 10031
denis.osin@gmail.com, shpil@groups.sci.ccny.cuny.edu
http://www.sci.ccny.cuny.edu/~osin/
http://www.sci.ccny.cuny.edu/~shpil/

Abstract. The main purpose of this paper is to suggest that public key encryption can be secure against the "encryption emulation" attack (on the sender's encryption) by computationally unbounded adversary, with one reservation: a legitimate receiver decrypts correctly with probability that can be made arbitrarily close to 1, but not equal to 1.

1 Summary of Our Claims

We thought it would make sense to summarize, for the reader's convenience, our two main claims in a separate section, before proceeding to a narrative introduction.

In Section 3, we describe a public-key encryption protocol that allows Bob (the sender) to send secret information to Alice (the receiver), encrypted one bit at a time, so that:

1. Assuming that Eve (the adversary):
   (a) is computationally unbounded,
   (b) knows everything about Bob's encryption algorithm and hardware,
   (c) does not know Alice's algorithm for creating public key, then:
   she cannot decrypt any single bit correctly with probability $> \frac{3}{4}$ by emulating Bob's encryption algorithm.
   We note that the assumption (c) is not in line with what is called "Kerckhoffs' assumptions", or "Kerckhoffs' principle" (see e.g. [3]), which is considered mandatory in practical cryptography. However, the assumption (a) is not encountered in real life either, so this claim of ours should be considered from a purely theoretical point of view, although we speculate in the end of the Introduction that this claim may be of interest in real-life non-commercial cryptography.

   The second claim *is* in line with Kerckhoffs' assumptions.

* Research of the first author was partially supported by the NSF grant DMS-0605093. Research of the second author was partially supported by the NSF grant DMS-0405105.

2. Assuming that Eve:
   (a) knows everything about Alice's and Bob's algorithms and hardware (Kerckhoffs' assumptions),
   (b) is exponentially, but not superexponentially, computationally superior to Alice and Bob, then:
   she cannot decrypt any single bit correctly with probability significantly higher than $\frac{3}{4}$ by using the encryption emulation attack in the broad sense (i.e., where she can emulate both the sender's and the receiver's algorithms).
   We do not want to be too formal here about what "exponentially, but not superexponentially, computationally superior" means. Intuitively, the reader can think of the following interpretation: if Alice and Bob are capable of performing at most n operations per second, then Eve can perform at most C^n operations per second for some fixed constant C independent of n.
   More specifically, we show that there is a k-step algorithm for the receiver (Alice) to obtain her public key that takes time $O(k^3)$, whereas the adversary who would like to emulate this algorithm with all possible randomness would have to take time $O(k^k)$.

Our focus in this paper is on claim (1) because, in our opinion, it is more interesting from the theoretical point of view.

2 Introduction

In this paper we continue to explore how non-recursiveness of a decision problem (as opposed to computational hardness of a search problem) can be used in public key cryptography. This line of research was started in [6] (note that the earlier protocol of Magyarik and Wagner [2] was *not* based on non-recursiveness of a decision problem, contrary to what the title of their paper may suggest; this was recently pointed out in [1]). One of the problems with the paper [6] is that it is somewhat too heavy on combinatorial group theory, at least for a non-expert. Most of that group theory is needed to separate the receiver (Alice) and the adversary (Eve) in power. This is, indeed, a very non-trivial problem that opens several interesting research avenues.

Here our primary focus is on the "encryption emulation" attack on the sender's (Bob's) transmissions. We suggest that Bob's encryption can be made reasonably secure against the "encryption emulation" attack by computationally unbounded adversary, with one reservation: a legitimate receiver decrypts correctly with probability that can be made arbitrarily close to 1, but not equal to 1.

First we recall what the "encryption emulation" attack on the sender's encryption is:

> Eve emulates the encryption algorithm over and over again, each time with fresh randomness, until the transmission to be attacked is obtained; this will happen eventually with overwhelming probability. The correctness of the scheme then guarantees that the corresponding secret key (as obtained by the adversary performing key generation) allows to decrypt illegitimately.

This attack would indeed work fine (at least, for computationally unbounded adversary) if the correctness of the scheme was perfect. However, if there is a gap, no matter how small (it can be easily made on the order of 10^{-200}, see [6]), between 1 and the probability of correct decryption by a legitimate receiver, then this gap can be very substantially "amplified" for the adversary, thus making the probability of correct illegitimate decryption anything but overwhelming. To explain how and why this is possible, we do not really need to introduce any serious group theory.

We emphasize at this point that when we talk about security against "computationally unbounded adversary" in this paper, we do not claim security against the "encryption emulation" (or any other) attack on the receiver's public key or decryption algorithm, but we only claim security against the encryption emulation attack *on the sender's transmission*. It seems that the problem of security of the sender's encryption algorithm is of independent interest. Of course, in commercial applications to, say, Internet shopping or banking, both the sender's and the receiver's algorithms are assumed to be known to the adversary ("Kerckhoffs' assumptions"), and the receiver's decryption algorithms (or algorithms for obtaining public keys) are usually more vulnerable to attacks. However, in some other applications, say, to electronic signatures (not to mention non-commercial, e.g. military applications), decryption algorithms or algorithms for generating public keys (by the receiver) need not be public, whereas encryption algorithms (of the sender) always are. It is therefore important to have a consensus in the cryptographic community on the first of the two security claims in Section 1 of the present paper.

Thus, in Section 3, we present a protocol which is reasonably secure against the encryption emulation attack on the sender's transmission by a computationally unbounded adversary who has complete information on the algorithm(s) and hardware that the sender uses for encryption. More precisely, in our protocol the sender transmits his private bit sequence by encrypting one bit at a time, and the receiver decrypts each bit correctly with probability that can be made arbitrarily close to 1, but not equal to 1. *At the same time, the (computationally unbounded) adversary decrypts each bit (by emulating the sender's encryption algorithm) correctly with probability at most* $\frac{3}{4}$.

There are essentially no requirements on the sender's computational abilities; in fact, encryption can be done by hand, which can be a big advantage in some situations; for example, a field operative can receive a public key from a command center and transmit encrypted information over the phone, without even using a computer. In the same scenario, there is nothing about the receiver's algorithms (or even about the general setup) that has to be known to the public. The only public information that comes from the receiver in this scenario is the encrypting instructions that she transmits to the sender (along with the public key).

In Section 5, we use the same protocol to address the second claim from our Section 1; in particular, we allow the adversary to emulate both the encryption and decryption algorithms. More precisely, here we allow the adversary to emulate the receiver's algorithm for obtaining the public key. If such an attack is successful, the adversary recovers the receiver's private key used for decryption, and then the encryption emulation attack on the sender's encryption will allow the adversary to decrypt correctly with the same probability as the receiver would. We show however that there is an algorithm for the receiver to obtain her public key that takes time $O(k^3)$ (where k is the complexity of the public key), whereas the adversary who would like to emulate this algorithm with all possible randomness would have to take time $O(k^k)$.

3 Encryption Protocol

In this section, we describe an encryption protocol with the following features:

(F1) Bob encrypts his secret bit by a word in a public alphabet X.
(F2) Alice (the receiver) decrypts Bob's transmission correctly with probability that can be made arbitrarily close to 1, but not equal to 1.
(F3) The adversary, Eve, is assumed to have no bound on the speed of computation or on the storage space.
(F4) Eve is assumed to have complete information on the algorithm(s) and hardware that Bob uses for encryption. However, Eve cannot predict outputs of Bob's random numbers generator (the latter could be just coin tossing, say). Neither does she know Alice's algorithm for obtaining public keys.
(F5) Eve cannot decrypt Bob's secret bit correctly with probability $> \frac{3}{4}$ by emulating Bob's encryption algorithm.

Once again: in this section, we only claim security against the "encryption emulation" attack (by computationally unbounded adversary) on the sender's transmissions. This does not mean that the receiver's private keys in our protocol are insecure against real-life (i.e., computationally bounded) adversaries, but this is the subject of Section 5. Here we prefer to focus on what is secure against computationally unbounded adversary since this paradigm shift looks important to us (at least, from the theoretical point of view).

We also have to reiterate that the encryption protocol which is presented in this section is probably not very suitable for commercial applications (such as Internet shopping or banking) due to a large amount of work required from Alice to receive just one bit from Bob. Bob, on the other hand, may not even need a computer for encryption.

Now we are getting to the protocol description. In one round of this protocol, Bob transmits a single bit, i.e., Alice generates a new public key for each bit transmission.

(P0) Alice publishes two group presentations by generators and defining relators:

$$\Gamma_1 = \langle x_1, x_2, \ldots, x_n \mid r_1, r_2, \ldots, r_k \rangle$$

$$\Gamma_2 = \langle x_1, x_2, \ldots, x_n \mid s_1, s_2, \ldots, s_m \rangle.$$

One of them defines the trivial group, whereas the other one defines an infinite group, but only Alice knows which one is which. In the group that is infinite, Alice should be able to efficiently solve the word problem, i.e., given a word $w = w(x_1, x_2, \ldots, x_n)$, she should be able to determine whether or not $w = 1$ in that group. There is a large and easily accessible pool of such groups (called *small cancellation groups*), see [6] for discussion.

Bob is instructed to transmit his private bit to Alice as follows:

(P1) In place of "1", Bob transmits a pair of words (w_1, w_2) in the alphabet $X = \{x_1, x_2, \ldots, x_n, x_1^{-1}, \ldots, x_n^{-1}\}$, where w_1 is selected randomly, while w_2 is selected to be equal to 1 in the group G_2 defined by Γ_2.

(P2) In place of "0", Bob transmits a pair of words (w_1, w_2), where w_2 is selected randomly, while w_1 is selected to be equal to 1 in the group G_1 defined by Γ_1.

Now we have to specify the algorithms that Bob should use to select his words.

Algorithm "0" (for selecting a word $v = v(x_1, \ldots, x_n)$ not equal to 1 in a Γ_i) is quite simple: Bob just selects a random word by building it letter-by-letter, selecting each letter uniformly from the set $X = \{x_1, \ldots, x_n, x_1^{-1}, \ldots, x_n^{-1}\}$. The length of such a word should be a random integer from an interval that Bob selects up front, based on his computational abilities. In the end, Bob should cancel out all subwords of the form $x_i x_i^{-1}$ or $x_i^{-1} x_i$.

Algorithm "1" (for selecting a word $u = u(x_1, \ldots, x_n)$ equal to 1 in a Γ_i) is slightly more complex. It amounts to applying a random sequence of operations of the following two kinds, starting with the empty word:

1. Inserting into a random place in the current word a pair hh^{-1} for a random word h.
2. Inserting into a random place in the current word a random conjugate $g^{-1} r_i g$ of a random defining relator r_i.

In the end, Bob should cancel out all subwords of the form $x_i x_i^{-1}$ or $x_i^{-1} x_i$. The length of the resulting word should be in the same range as the length of the output of Algorithm "0". We do not go into more details here because all claims in this section remain valid no matter what algorithm for producing words equal to 1 is chosen, as long as it returns a word whose length is in the same range as that of the output of Algorithm "0".

Now let us explain why the legitimate receiver (Alice) decrypts correctly with overwhelming probability. Suppose, without loss of generality, that the group G_1 is trivial, and G_2 is infinite. Then, if Alice receives a pair of words (w_1, w_2) such that $w_1 = 1$ in G_1 and $w_2 \neq 1$ in G_2, she concludes that Bob intended to transmit a "0". This conclusion is correct with probability 1. If Alice receives (w_1, w_2) such that $w_1 = 1$ in G_1 and $w_2 = 1$ in G_2, she concludes that Bob intended to transmit a "1". This conclusion is correct with probability which is close to 1, but not equal to 1 because it may happen, with probability $\epsilon > 0$, that the random word w_2 selected by Bob is equal to 1 in G_2. The point here is that, if G_2 is infinite, this ϵ is negligible and, moreover, for "most" groups G_2 this ϵ tends to 0 exponentially fast as the length of w_2 increases. For more precise statements, see [6]; here we just say that it is easy for Alice to make sure that G_2 is one of those groups.

4 Emulating Encryption

Now we are going to discuss Eve's attack on Bob's transmission. Under our assumptions (F3), (F4) Eve can identify the word(s) in the transmitted pair which is/are equal to 1 in the corresponding group(s), as well as the word, if any, which is not equal to 1. Indeed, for any particular transmitted word w she can use the "encryption emulation" attack, as described in our Introduction: she emulates algorithms '0" and "1" over and over again, each time with fresh randomness, until the word w is obtained. Thus, Eve is building up two lists, corresponding to two algorithms above. Our first observation is that the list that corresponds to the Algorithm "0" is useless to Eve because it is eventually going to contain *all* words in the alphabet $X = \{x_1, \ldots, x_n, x_1^{-1}, \ldots, x_n^{-1}\}$, with overwhelming probability. Therefore, Eve may just as well forget about this list and concentrate on the other one, that corresponds to the Algorithm "1". Now the situation boils down to the following: if the word w appears on the list, then it is equal to 1 in the corresponding group G_i. If not, then not.

It may seem that Eve should encounter a problem detecting $w \neq 1$: how can she conclude that w does *not* appear on the list if the list is infinite (more precisely, of a priori unbounded length) ? This is where our condition (F4) plays a role: if Eve has complete information on the algorithm(s) and hardware that Bob uses for encryption, then she does know a bound on the size of the list.

Thus, Eve can identify the word(s) in the transmitted pair which is/are equal to 1 in the corresponding group(s), as well as the word, if any, which is not equal to 1. There are the following possibilities now:

1. $w_1 = 1$ in G_1, $w_2 = 1$ in G_2;
2. $w_1 = 1$ in G_1, $w_2 \neq 1$ in G_2;
3. $w_1 \neq 1$ in G_1, $w_2 = 1$ in G_2.

It is easy to see that one of the possibilities (2) or (3) cannot actually occur, depending on which group G_i is trivial. Then, the possibility (1) occurs with probability $\frac{1}{2}$ (either when Bob wants to transmit "1" and G_1 is trivial, or when Bob wants to transmit "0" and G_2 is trivial). If this possibility occurs, Eve cannot decrypt Bob's bit correctly with probability $> \frac{1}{2}$ because she does not know which group G_i is trivial. If Eve knew Alice's algorithm for generating the public key as well as Alice's hardware capabilities, then Eve would be able to find out which G_i is trivial, but we specifically consider attacks on the sender's encryption in this paper. We just note, in passing, that for a real-life (i.e., computationally bounded) adversary to find out which presentation Γ_i defines the trivial group is by no means easy and deserves to be a subject of separate investigation; we discuss this in the next section. Here we just say that there are many different ways to efficiently construct very complex presentations of the trivial group, some of them involving a lot of random choices. See e.g. [5] for a survey on the subject.

In any case, our claim (F5) was that Eve cannot decrypt Bob's bit correctly with probability $> \frac{3}{4}$ by emulating Bob's encryption algorithm, which is obviously true in this scheme since the probability for Eve to decrypt correctly is, in fact, precisely $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot 1 = \frac{3}{4}$. (Note that Eve decrypts correctly with probability 1 if either of the possibilities (2) or (3) above occurs.)

Someone may say that $\frac{3}{4}$ is a rather high probability of illegitimate decryption, even though this is just for one bit. Recall however that we are dealing with computationally unbounded adversary, while Bob can essentially do his encryption by hand! All he needs is a generator of uniformly distributed random integers in the interval between 1 and $2n$ (the latter is the cardinality of the alphabet X). Besides, note that with the probability of correctly decrypting one bit equal to $\frac{3}{4}$, the probability to correctly decrypt, say, a credit card number of 16 decimal digits would be on the order of 10^{-7}, which is comparable to the chance of winning the jackpot in a lottery. Of course, there are many tricks that can make this probability much smaller, but we think we better stop here because, as we have pointed out before, our focus here is on the new paradigm itself.

5 Encryption/Decryption Emulation Attack by a Computationally Superior Yet Bounded Adversary

In this section, we show that Eve would need a serious computational power (superexponential compared to that of Alice) to run an emulation attack on Alice's algorithm for generating a public key if this algorithm is sophisticated enough. This attack is similar to the emulation attack on Bob's encryption that we considered before:

> Eve emulates Alice's algorithm for generating a public key over and over again, each time with fresh randomness, until the actual public key is obtained; this will happen eventually with overwhelming probability.

Obviously, if this attack is successful, then the encryption emulation attack on the sender's encryption, as described in the Introduction, will allow Eve to decrypt Bob's bit correctly with overwhelming probability because Eve would know, just like Alice does, which public presentation Γ_i is a presentation of the trivial group.

Thus, we are going to focus on the above attack and describe a particular algorithm that Alice can use to generate a presentation of the trivial group, such that emulating this algorithm with all possible randomness would entail going over superexponentially many possibilities.

The algorithm itself is quite simple, and it produces special kinds of presentations of the trivial group. It is known (see e.g. [4], [5]) that the following presentations define the trivial group:

$$\langle x, y |\ x^{-1}y^n x = y^{n+1},\ w = 1\rangle,$$

where $n \geq 1$ and w is any word in x and y with exponent sum 1 on x. Let us assume that the length of w is k, a sufficiently large integer selected by Alice, which can be considered a measure of complexity of the above presentation. Technically, the integer n, too, influences complexity of the presentation, but it is less important to us, so we will consider n fixed (and rather small) in what follows.

Now we are going to describe a k-step algorithm that Alice can use to obtain a presentation of complexity $O(k^3)$ that would define the trivial group.

1. At the first step, Alice selects a presentation of the form

$$\langle x_1, y_1 |\ x_1^{-1}y_1^n x_1 = y_1^{n+1},\ w_1 = 1\rangle,$$

   with a random word w_1 in x_1 and y_1 of length k, having exponent sum 1 on x_1.
2. At the ith step, $1 < i < k$, Alice adds two new generators, x_i and y_i, and a new relator w_i, which is a random word in x_i and y_i of length k, having exponent sum 1 on x_i. Also, she adds the relation $x_i^{-1}y_i^n x_i = y_i^{n+1}$. The resulting presentation still defines the trivial group; in particular, $x_i = 1$ and $y_i = 1$ in this group. After that, Alice "mixes" new generators with old ones by inserting $x_i^{\pm 1}$ and $y_i^{\pm 1}$ in k random places in each old relator. The idea is to have roughly k generators with index i in each old relator.
3. The kth step is special. Alice adds two new generators, x_k and y_k, and two relators, $x_k^{-1}y_k x_k y_k^{-2}$ and w_k of small length (say, between 3 and 6) having exponent sum 1 on x_k. The resulting group is therefore still trivial; in particular, all generators are equal to 1 in this group. Then Alice adds roughly k more relators to this presentation, where each relator is a random word of length 1, 2, or 3 in the generators $x_1, y_1, \ldots, x_k, y_k$. The only thing Alice takes care of is that each generator occurs in at least one of these new relators. Then Alice mixes these new relators with each other and with the two relators $x_k^{-1}y_k x_k y_k^{-2}$ and w_k by using operations of the following kinds: $r_i \to r_i r_j$, or $r_i \to r_i r_j^{-1}$, or $r_i \to r_j r_i$, or $r_i \to r_j^{-1} r_i$ for different relators r_i, r_j, until all relators will have roughly k occurrences of each generator.
4. Finally, Alice randomly renames the generators to hide the order of steps.

It is fairly obvious that the complexity of the resulting presentation is $O(k^3)$, whereas to emulate all possible randomness in the above algorithm Eve would have to take time $O(k^k)$ because she would have to explore, in particular, random (sub)words of length k on $2k$ letters (and their inverses).

6 A Challenge

Here we offer a computational challenge to illustrate one of our points, namely, that Eve might need a serious computational power to detect a presentation of the trivial group among two given presentations.

Let

$\Gamma_1 = \langle x, y, z \mid x^{-1}zy^{-1}x^{-1}yxzx^{-1}y^{-1}xyz^{-1},\ y^{-1}x^{-1}yxz^{-1}x^{-1}y^{-1}xyxzx^{-1}y^{-1}x,$ $xy^{-1}z^2x^{-1}y^{-2}x^3z^{-1}x^{-1}y \rangle,$

$\Gamma_2 = \langle x, y, z \mid x^{-1}zy^{-1}x^{-1}yxzx^{-1}y^{-1}xyz^{-1},\ y^{-1}x^{-1}yxz^{-1}x^{-1}y^{-1}xyx^{-1}zxy^{-1}x,$ $xy^{-1}z^2x^{-1}y^{-2}x^3z^{-1}x^{-1}y \rangle.$

The question is: which Γ_i is a presentation of the trivial group?

References

1. Birget, J.-C., Magliveras, S., Sramka, M.: On public-key cryptosystems based on combinatorial group theory. Tatra Mountains Mathematical Publications 33, 137–148 (2006)
2. Magyarik, M.R., Wagner, N.R.: A Public Key Cryptosystem Based on the Word Problem. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 19–36. Springer, Heidelberg (1985)
3. Menezes, A.J.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
4. Miller, C.F., Schupp, P.: Some presentations of the trivial group. In: Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Geometric Group Theory and Computer Science. Contemp. Math., Amer. Math. Soc., vol. 250, pp. 113–115 (1999)
5. Myasnikov, A.D., Myasnikov, A.G., Shpilrain, V.: On the Andrews-Curtis equivalence. Contemp. Math., Amer. Math. Soc. 296, 183–198 (2002)
6. Shpilrain, V., Zapata, G.: Using decision problems in public key cryptography, (preprint) `http://www.sci.ccny.cuny.edu/~shpil/wppkc.pdf`

A Uniform Lower Bound on Weights of Perceptrons

Vladimir V. Podolskii[*]

Moscow State University
podolskii@lpcs.math.msu.su

Abstract. A threshold gate is a linear function of input variables with integer coefficients (weights). It outputs 1 if the value of the function is positive. The sum of absolute values of coefficients is called the total weight of the threshold gate. A perceptron of order d is a circuit of depth 2 having a threshold gate on the top level and conjunctions of fan-in at most d on the remaining level.

For every n and $d \leq D \leq \varepsilon n^{1/6}$ we construct a function computable by a perceptron of order d but not computable by any perceptron of order D with total weight $2^{o(n^d/D^{4d})}$. In particular, if D is a constant, our function is not computable by any perceptron of order D with total weight $2^{o(n^d)}$. Previously functions with this properties were known only for $d = 1$ (and arbitrary D) [2] and for $D = d$ [12].

1 Introduction

A threshold gate with input Boolean variables $x_1, \ldots, x_n$ is a Boolean function of the form $sgn(\sum_{i=1}^{n} w_i x_i - t)$, where $w_1, w_2, \ldots, w_n, t$ are integers, called the weights and the threshold, respectively, and sgn stands for the sign function: $sgn(x) = 1$ if x is positive, $sgn(x) = 0$ otherwise.

The sum of absolute values of all coefficients, $\sum_i |w_i| + |t|$, is called the total weight of the threshold gate.

In this paper we consider circuits composed of threshold gates (threshold circuits). More specifically we prove exponential lower bounds for the size of threshold circuits of a special kind (called perceptrons) computing certain explicit functions. There are two ways to define the size of a threshold circuit: we take weights of wires into account or we do not. In our lower bounds, we adopt the first way. That is, we define the size of a threshold circuit as the sum of total weights of its gates. This is equivalent to allowing only weights ± 1 on wires and defining the size as the total number of wires in the circuit.

The threshold circuits considered in this paper are of constant depth. The first exponential lower bound for such circuits is the bound for circuits composed of NOTs, ANDs and ORs (of unbounded fan-in) from [15,7]. (Note that AND and OR are indeed a special kind of threshold gates.) The next exponential bound

[*] Work is partially supported by grant 06-01-00122 from Russian Federation Basic Research Fund.

was obtained for bounded depth circuits containing ANDs, ORs, NOTs and MOD p gates (where p is a prime number)) [13,14].[1] (Note that every MOD m gate can be computed by a threshold circuits of constant depth and polynomial size.) For MOD m gates with a composite m no super-polynomial bounds are known, even for depth-2 circuits.

For general depth-2 threshold circuits, there is an exponential lower bound for the size of any circuit computing the inner product of n bit strings [6] (recall that we take weights into account). Later this result was improved in [11]: the exponential lower bound holds even if we do not count the weights on wires from the inputs (but count the weight of the top threshold gate). There are no known super-polynomial lower bounds for the number of gates in depth-2 threshold circuits (i.e. if we do not count weights at all).

In this paper we consider perceptrons, which are depth-2 threshold circuits of a special kind. A perceptron of order d is a circuit of depth 2 having a threshold gate on the top level and conjunctions of fan-in at most d on the remaining level. A perceptron of order 1 is just a threshold gate. The total weight of a perceptron is the total weight of its threshold gate. Since the conjunction of Boolean variables corresponds to the product of them, a perceptron of order d is the sign of degree d integer polynomial. Perceptrons were studied intensively in sixties in the artificial intelligence community (see [9]), as a simplest model of a neuron. On the other hand, they arise quite naturally in Computational Complexity Theory [3].

We will prove lower bounds for the total weights of perceptrons in terms of their order d and the number of input variables n. As size is bigger than total weight, our bounds hold for size as well. Note that there is a Boolean function of n variables that is not computable by any perceptron of order less than n (for example, the parity function, see [9]). In this paper we are interested only in functions that are computable by perceptrons of small order d and such that any perceptron of order d or a bigger order computing the function has large total weight. For example, we are not interested in the inner product (studied in [6,11])—that function is not computable by perceptrons of order less than n, as parity reduces to it.

We know three results of this kind. Håstad in [8] constructed a function of n variables, that is computable by a threshold gate, but any such threshold gate has total weight at least $n^{\Omega(n)}$. Note that by result of Muroga [10], if a Boolean function is computed by a threshold gate, it can be computed by a threshold gate of total weight $n^{O(n)}$ (see also [8]). Thus Håstad's bound is tight up to a factor in the exponent. The second result is due to Beigel [2]. He exhibited a function of n variables that is computable by a threshold gate with total weight 2^n, and for every D any perceptron of order D computing that function has total weight $2^{\Omega(n/D^3)}$. His lower bound is not tight: it differs from Muroga's upper bound in that the base of the power is 2 instead of n. On the other hand, he proved a lower bound for any order D and not only for $D = 1$. The third result

[1] The gate MOD m on inputs $x_1, \ldots, x_n$ outputs 1 if the number of 1s among $x_1, \ldots, x_n$ is divisible by m.

is from our previous paper [12]. We have generalized there Håstad result to any d: for every $d \geqslant 1$ there is a perceptron of order d such that every perceptron of order d that computes the same function f has total weight at least $n^{\Omega(n^d)}$ (the constant hidden in Ω-notation depends on d). This bound is also tight up to a factor in the exponent. Indeed, Muroga's upper bound for $d = 1$ easily generalizes to all d. That is, every perceptron of order d with n input variables is equivalent to a perceptron of order d and total weight $n^{O(n^d)}$, where the constant hidden in O-notation depends on d. (Assume that conjunctions on the bottom level are different and every variable appears in each conjunction at most once. Thus there are at most $O(n^d)$ functions on the bottom level of a perceptron of order d. Consider them as independent variables. In this way the upper bound for the total weight of threshold gates translates to the upper bound $n^{O(dn^d)}$ for perceptrons.)

Is it possible to generalize Beigel's result to arbitrary d? Namely, is it true that for all d and n there is a function f computable by a perceptron of order d and such that for all D any perceptron of order D computing f has total weight $n^{\Omega(n^d)}$ (or $2^{\Omega(n^d)}$, as in Beigel's result)? The constant in $\Omega(n^d)$ does not depend on n, but may depend on D and d? This question is open for every $d > 1$.

In this paper, we make a step towards answering this question in positive. For all n and $d \leqslant D \leqslant \varepsilon n^{1/6}$ we construct a function computable by a perceptron of order d, but not computable by a perceptron of order at most D with total weight $2^{o(n^d/D^{4d})}$. This result is still not optimal for two reasons: (1) the base of the power is constant, and not n, and (2) the constructed function depends on D. But there is an essential progress compared to the results of [2], [12]. Indeed, we obtain lower bound $2^{\Omega(n^d)}$ for all d, and not only for $d = 1$, as in [2], and our lower bound is true for perceptrons whose order is in a rather wide range, and not only for perceptrons of order d, as in [12].

The constant in $\Omega(n^d)$ in our result is of the order $\Omega(D^{-4d})$. This makes our bound valuable for non-constant D. For example, we can fix arbitrary d and let $D = \log n$. In this case we obtain a sequence f_n of functions computable by a perceptron of order d and such that every perceptron of order at most $\log n$ computing f_n has total weight $2^{\Omega(n^d/\log^{4d} n)}$. In particular, this lower bound holds for perceptrons of any constant order.

2 The Result

Let $[n]$ denote the set of first n natural numbers, $\{1, 2, \ldots, n\}$.

Our functions are generalizations of the function used by Beigel [2] for similar purposes:

Definition 1 ([2]). *For an $x \in \{0,1\}^n$ the Boolean function* ODD-MAX-BIT(x) *is* 1 *if the rightmost 1 in x has an odd index. (If x has no ones then let, say,* ODD-MAX-BIT$(x) = 0$.*)*

Now we are going to define our function. The function depends on three natural parameters n, d and D, where $d \leqslant D$. The first two parameters indicate the number of variables (more specifically, it has nd variables). The meaning of d and D is the following: the function is computed be a perceptron of order d (of large weight) and is not computable by any perceptron of order at most D that has small total weight (we will explain further what is "small" and "large" in this context).

We partition dn input variables into d groups of equal size n: $(x^1, x^2, \ldots, x^d)$ where $x^i = (x^i_1, x^i_2, \ldots, x^i_n)$. Each evaluation of x^i identifies a set of natural numbers $X_i = \{j | x^i_j = 1\}$. And conversely, each subset X_i of $[n]$ defines a unique assignment of Boolean values to x^i. Therefore we will assume in the sequel that our function maps d-tuples $(X_1, \ldots, X_d)$ of subsets of $[n]$ to $\{0, 1\}$.

Similar to ODD-MAX-BIT(x), the value of our function on a tuple $(X_1, \ldots, X_d)$ depends only on the largest tuple $(\alpha_1, \ldots, \alpha_d)$ in the set $X_1 \times \cdots \times X_d$, with respect to some total order on $[n]^d$. The ordering relation we will use is quite complicated, it plays the key role in our proof.

First we define $D+1$ different total orders on the set $[n]$. The first one, $<_1$, is the normal order: $1, 2, \ldots, n$. Other orders are cyclic shifts of this one. More specifically, partition the set $[n]$ into $D+1$ contiguous blocks $u_1, \ldots, u_{D+1}$ having equal sizes: $u_i = (i-1)\frac{n}{D+1} + 1, \ldots, i\frac{n}{D+1}$ (if n is not a multiple of $D+1$, then the sizes of blocks may differ by 1). For any $i = 2, \ldots, D+1$, the order $<_i$ is obtained from the normal order by exchanging the first $i-1$ blocks and the last $D+2-i$ blocks: $u_i, \ldots, u_{D+1}, u_1, \ldots, u_{i-1}$[2]. Let $\mathrm{num}_i(t)$ denote the ordinal number of t w.r.t. the order $<_i$ (here $i = 1, \ldots, D+1$ and $t \in [n]$, the ordinal number of the least element is 1).

The order on $[n] \times \ldots \times [n]$ (d times) we will use is essentially the lexicographical order. However, there is an important difference. In the lexicographical order, all the components of two given d-tuples in $[n]^d$ are compared w.r.t. a fixed order. In contrast, we will compare kth components of given tuples w.r.t. an order from the list $<_1, \ldots, <_{D+1}$ which depends on the compared tuples and on k. More specifically, it depends on the ordinal number of $k-1$st component of the given tuples in the order which is used to compare their $k-1$st components.

The first components α_1 and β_1 of two given d-tuples, $(\alpha_1, \ldots, \alpha_d)$ and $(\beta_1, \ldots, \beta_d)$, are compared w.r.t. the order $<_1$. If they are not equal, we have already compared the tuples. Otherwise, we compare the second components. The recursive rule to choose the next order is as follows.

Assume that the order $<_{i_k}$ (to compare kth components) is already defined and it happens that $\alpha_k = \beta_k$. The order to compare $(k+1)$st is determined by the ordinal number of α_k (which coincides with β_k by the assumption) in the order $<_{i_k}$. Namely,

$$i_{k+1} \equiv \left\lceil \frac{\mathrm{num}_{i_k}(\alpha_k)}{2} \right\rceil (mod(D+1)).$$

[2] Actually, it only matters that the largest $n/(D+1)$ elements are different in all orders. We prefer to fix a particular easily described set of orders having this property.

In other words, as α_k increases from the minimum to the maximum (w.r.t. $<_{i_k}$), the number i_{k+1} takes all the values

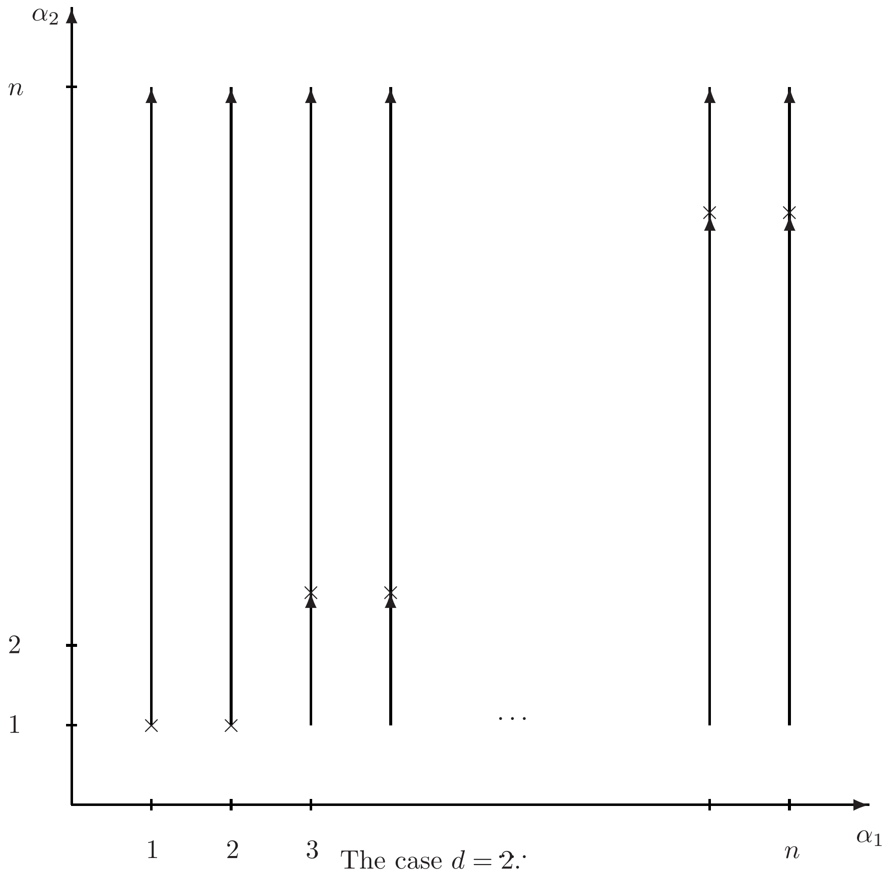
$$1, 1, 2, 2, \ldots, (D+1), (D+1), 1, 1, \ldots, (D+1), (D+1), \ldots \tag{1}$$

in the specified order.

It is important that, for every tuple α, the orders used to compare α's component to components of other tuples β depend only on α and on the number of the component.

By construction, the defined binary relation on $[n]^d$ is asymmetric and total. It is easy to verify that it is transitive. Thus it is a strict total order.

It is instructive to analyze the case $d = 2$. Represent pairs (α_1, α_2) by points on the plane.



The case $d = 2$.

The first component α_1 is associated with the horizontal axis, and the second component α_2 with the vertical axis. We have two rules to compare the pairs:

Rule 1. The more left the pair is, the smaller it is.

Rule 2. In every column the minimal pair is marked by "×" and the arrows indicate the direction from small pairs to large ones. (In the two leftmost columns the minimal pairs are in the bottom row. In the following two columns they are in the rows number $\frac{n}{D+1}$, in the next two columns they are in the row number $\frac{2n}{D+1}$ and so on.) Above the minimal pair, the higher the pair is, the larger it is. The same holds for the pairs below it. The pair at the bottom follows the topmost pair.

In the case $d > 2$ the order is not so easy. However, as we will reason by induction, we will always consider only two consecutive coordinates. Thus we will be in the situation that is similar to the case $d = 2$. The only difference is that the order on the first coordinate might be different from $<_1$.

To define our function we need one more notation. Let $\alpha = (\alpha_1, \ldots, \alpha_d)$ be a d-tuple from $[n]^d$ and $k \in [d]$. Note that we have assigned to each tuple $\alpha = (\alpha_1, \ldots, \alpha_d)$ certain orders $<_{i_1}, \ldots, <_{i_d}$. Let $\mathrm{odd}_k(\alpha) = 1$ if $\mathrm{num}_{i_k}(\alpha_k)$ is odd and $\mathrm{odd}_k(\alpha) = 0$ otherwise.

For $X \subseteq [n]$ let $\max_i(X)$ denote the ordinal number of the maximal element X w.r.t. the order $<_i$, that is $\max_i(X) = \max_{t \in X} \mathrm{num}_i(t)$.

We are going to define our function now. Its value on the tuple $(X_1, \ldots, X_d)$ depends only on the maximal tuple in $X_1 \times \ldots \times X_d$.

Definition 2. *Let $X = (X_1, \ldots, X_d)$ be a tuple of non-empty subsets of $[n]$. Let $\alpha = (\alpha_1, \ldots, \alpha_d)$ the largest tuple in $X_1 \times \ldots \times X_d$ (w.r.t. above defined order on $[n]^d$). We define our function as follows:*

$$\text{ODD-MAX-BIT}_d^D(X) = \mathrm{odd}_1(\alpha) \oplus \mathrm{odd}_2(\alpha) \oplus \ldots \oplus \mathrm{odd}_d(\alpha).$$

If at least one X_i is empty, we let $\text{ODD-MAX-BIT}_d^D(X) = 0$.

Lemma 1. *The function* ODD-MAX-BIT_d^D *is computed by a perceptron of order d.*

Proof. First note that a tuple $\alpha = (\alpha_1, \ldots, \alpha_d)$ is in $X_1 \times \ldots \times X_d$ iff the AND of all variables $x^1_{\alpha_1}, \ldots, x^d_{\alpha_d}$ evaluates to 1. We will denote this AND by $\mathrm{AND}(\alpha)$.

Let $t_1, t_2, \ldots, t_{n^d}$ be the enumeration of all tuples in $[n^d]$ in the increasing order. Then $\text{ODD-MAX-BIT}_d^D(X)$ is equal to the sign of the sum

$$\sum_j (-1)^{\mathrm{odd}_1(t_j)+\ldots+\mathrm{odd}_d(t_j)} \cdot 2^j \cdot \mathrm{AND}(t_j).$$

Indeed, for all $t_j \notin X_1 \times \ldots \times X_d$ we have $\mathrm{AND}(t_j) \equiv 0$. Thus only those t_j in $X_1 \times \ldots \times X_d$ make a contribution to the displayed sum. The coefficients 2^j are chosen so that the contribution of the largest tuple is greater than the total contribution of all other tuples.

The next theorem is our main result.

Theorem 1. *The function* $\text{ODD-MAX-BIT}_d^D(X)$ *is not computed be any perceptron of order at most D and of total weight w provided*

$$D^6 + D^5 \log(n+1) < \delta n, \tag{2}$$

$$w < 2^{\left(\frac{\varepsilon n}{D^4}\right)^d}. \tag{3}$$

Here, ε, δ are small positive constants.

Proof. Assume that a perceptron of order at most D and of total weight w computes our function and the inequality (2) is fulfilled. We will prove that the inequality (3) is false.

Consider the polynomial C representing the perceptron. In what follows we will substitute only 0 and 1 for its variables. Thus we will identify its input values with d-tuples of subsets of $[n]$ and we will write $C(X_1, \dots, X_d)$. W.l.o.g. we assume that C is a multi-linear polynomial (i.e. of degree at most 1 in each variable). The value of C on any Boolean assignment does not exceed the total weight of the perceptron. Thus it suffices to find $X_1, \dots, X_d$ such that

$$|C(X_1, \dots, X_d)| \geqslant 2^{\left(\frac{\varepsilon n}{D^4}\right)^d}.$$

We present first a sketch of the construction of such $X_1, \dots, X_d$ (Section 2.1) and then a formal proof (Section 2.2).

2.1 A Sketch of the Construction

Our plan is the following. We will build a sequence of $\Omega\left(\left(\frac{\varepsilon n}{D^4}\right)^d\right)$ d-tuples of subsets of $[n]$. The absolute value of C on almost every tuple in that sequence will exceed at least 2 times its absolute value on the preceding one. For the remaining few tuples, C's value may decrease compared to the C's value on the preceding tuple. The total decrease in C's value due to "bad" tuples will not affect the total increase in C's value due to "good" tuples. As the absolute value of C on the first tuple is at least 1, its value on the last assignment will be exponential in the length of the sequence.

How will we obtain a tuple on which C's value is twice more than that on the preceding one? Let us fix values of all $X_1, \dots, X_{d-1}$ (and X_d remains free). The function ODD-MAX-BIT$_d^D(X_1, \dots, X_d)$ becomes essentially the function ODD-MAX-BIT(X_d) and the polynomial C becomes a polynomial in $x_1^d, \dots, x_n^d$ of degree at most D. The only difference between ODD-MAX-BIT$_d^D(X_1, \dots, X_d)$ and ODD-MAX-BIT(X_d) is that the order $<_i$ on $[n]$ used in the definition of ODD-MAX-BIT$_d^D$ might be different from the order $<_1$, which is used in the definition of ODD-MAX-BIT(X_d). A lemma of Beigel from [2] (Lemma 2) states that if $\max(X)$ is not very close to n then it is possible to change the set X so that the absolute value of the polynomial sign representing ODD-MAX-BIT(X) increases at least 2 times and $\max(X)$ increases only a little. In this result, it is important that the same order on $[n]$ is used in the definition of ODD-MAX-BIT(X) and in the definition of $\max(X)$. Applying Beigel's result to the order $<_i$, we obtain a way to increase C's value.

Here is a more detailed exposition of our plan in the case $d = 2$. Let us start with $X_1^0 = \{1\}, X_2^0 = \{1\}$. Fix the first coordinate X_1^0. Beigel's result (Lemma 2) implies that if the maximal element in X_2^0 is small enough (as in our case), then it is possible to change $\Omega(n)$ times the value X_2 so that each change doubles the absolute value of $C(X_1^0, X_2)$. Hence there is X_2^1 such that

$$|C(X_1^0, X_2^1)| \geqslant 2^{\Omega(n)}|C(X_1^0, X_2^0)|.$$

Then we fix the second coordinate X_2^1 and change X_1 so that (a) the value of C decreases only a little and (b) w.r.t. the new order on the second coordinate, the maximal element in X_2^1 is again small. Thus we can again use Lemma 2 to obtain X_2^2 such that

$$|C(X_1^1, X_2^2)| \geqslant 2^{\Omega(n)}|C(X_1^1, X_2^1)|.$$

We will show that this procedure can be repeated $\Omega(n)$ times. Therefore at the end we will get $X_1^{\Omega(n)}, X_2^{\Omega(n)}$ such that

$$|C(X_1^{\Omega(n)}, X_2^{\Omega(n)})| \geqslant 2^{\Omega(n^2)}|C(X_1^0, X_2^0)| \geqslant 2^{\Omega(n^2)}.$$

An analysis will show that the constant in $\Omega(n^2)$ is of order $\Omega(D^{-8})$ and thus we obtain $w \geqslant |C(X_1^{\Omega(n)}, X_2^{\Omega(n)})| \geqslant 2^{\Omega(n^2/D^8)}$.

How does the argument change in the case $d = 3$? We first fix the first coordinate $X_1^0 = \{1\}$. The function ODD-MAX-BIT$_d^D(X_1^0, X_2, X_3)$ essentially coincides with the similar function of X_2, X_3, that is, with the function ODD-MAX-BIT$_d^D(X_2, X_3)$ Therefore it is possible to use the inequality proved in the case $d = 2$. Then we fix the second and the third coordinates and change the first one so that (a) the value of C decreases only a little and (b) we can again use the inequality proved in the case $d = 2$. We will prove that this procedure can be repeated $2^{\Omega(n)}$ times.

Thus our proof will proceed by induction on d (called k in Lemma 4).

2.2 A Formal Proof

First we state a Lemma which was essentially proved in [2]. Let $f(X)$ be a polynomial with integer coefficients in variables $x_1, \ldots, x_n$ of degree at most D. Recall that we identify assignments of Boolean values to variables $x_1, \ldots, x_n$ and subsets X of $[n]$. Thus we use the notation $f(X)$ instead of $f(x_1, \ldots, x_n)$.

Lemma 2 ([2]). *There is a positive constant γ such that the following holds. Let A, B be disjoint subsets of $[n]$ such that $|B| \geqslant \gamma D^2$. Assume that for every non-empty $M \subseteq B$ we have $sgn(f(A)) \neq sgn(f(A \cup M))$. Then there is $M \subseteq B$ with*

$$|f(A \cup M)| \geqslant 2|f(A)|.$$

This lemma will enable to increase the value of C at the expense of increasing $X_1, \ldots, X_d$. And the next one will enable to decrease the size of X_i so that $C(X)$ decreases not very much.

Lemma 3. *Let $f(X)$ be a polynomial of n variables of degree at most D with integer coefficients. Let $X^0 \subseteq [n]$ be an input to f. Then there exists an input X^1 such that $|X^1| \leqslant D$ and*

$$|f(X^1)| \geqslant \frac{|f(X^0)|}{2^{D(D+1)}(n+1)^D}.$$

Proof. Every monomial of f has the form $w(S) \cdot \prod_{i \in S} x_i$, where S is a subset of $[n]$ and $w(S)$ is an integer number.[3] Using this notation we can express the value $f(X^0)$ as $f(X^0) = \sum_{S \subseteq X^0} w(S)$.

There are at most $(n+1)^D$ monomials in f. Hence there exists a set $S \subseteq X^0$ such that $|S| \leqslant D$ and $|w(S)| \geqslant \frac{|f(X^0)|}{(n+1)^D}$.

Consider the minimal S with such properties and distinguish two cases:

1. For any proper subset $S_1 \subsetneq S$ we have $|w(S_1)| \leqslant \frac{|w(S)|}{2^{D+1}}$. Then

$$\sum_{S_1} |w(S_1)| \leqslant 2^D \frac{|w(S)|}{2^{D+1}} = \frac{|w(S)|}{2}.$$

   Hence $|f(S)| \geqslant |w(S)| - \sum_{S_1} |w(S_1)| \geqslant \frac{|w(S)|}{2} \geqslant \frac{|f(X^0)|}{2(n+1)^D}$.
2. There exists proper subset $S_1 \subsetneq S$ such that $|w(S_1)| > \frac{|w(S)|}{2^{D+1}}$. In this case we replace S by S_1 and repeat the construction. Since the cardinality of S decreases after each repetition, we will have at most D iterations.

At the end we will obtain set X^1 such that $|f(X^1)| \geqslant \frac{|f(X^0)|}{2^{D(D+1)}(n+1)^D}$.

With these lemmas at hand we are able to prove the key lemma.

Lemma 4. *Let $X_1^0, \dots, X_d^0$ be a tuple of subsets of $[n]$ and let $\alpha = (\alpha_1, \dots, \alpha_d)$ denote the maximal tuple in $X_1^0 \times \dots \times X_d^0$. Let $<_l$ be the order used to compare the kth component of α. Assume that $\mathrm{num}_l(\alpha_k) \leqslant \frac{Dn}{D+1}$. Then there are $X_k', X_{k+1}', \dots, X_d'$ such that*

$$|C(X_1^0, \dots, X_k', \dots, X_d')| \geqslant 2^{\left(\frac{\varepsilon n}{D^4}\right)^{d-k+1}} |C(X_1^0, \dots, X_k^0, \dots, X_d^0)|.$$

Here ϵ is a positive constant.

Proof. The proof goes by downward induction on k. The base of induction is $k = d$. For $k = d$ the inequality is obtained by $\varepsilon n / D^4$ applications of Lemma 2 to the polynomial $f(X) = C(X_1^0, \dots, X_{d-1}^0, X)$.

More specifically, the function ODD-MAX-BIT$_d^D(X_1^0, \dots, X_{d-1}^0, X)$ either coincides with ODD-MAX-BIT(X), or coincides with its negation. Here we assume that in the definition of ODD-MAX-BIT the elements in $[n]$ are arranged w.r.t. the order $<_l$. Thus the polynomial $f(X)$ sign-represents the function ODD-MAX-BIT(X). Let $A = X_d^0$ and let B stand for the set of γD^2 numbers in $[n]$ obtained as follows. Let s be the ordinal number of $\max_l(A)$ w.r.t. the order $<_l$. Include in B all the elements with ordinal numbers $s+1, s+3, \dots, s+2\gamma D^2-1$ w.r.t. $<_l$. For every non-empty $M \subset B$ we have ODD-MAX-BIT$(X_d^0 \cup M) \neq$ ODD-MAX-BIT(X_d^0). Therefore the assumptions of Lemma 2 are fulfilled and there is $M \subseteq B$ with $|f(X_d^0 \cup M)| \geqslant 2|f(X_d^0)|$. Change X_d^0 to $X_d^0 \cup M$ and repeat the argument. Each iteration increases X_d^0 by at most $2\gamma D^2$, thus we

[3] W.l.o.g. we may assume that f is a multilinear polynomial.

can make $n/(D+1)2\gamma D^2$ iterations. Thus we obtain X'_d such that $|f(X'_d)| \geqslant 2^{\Omega(n/D^3)}|f(X_0^d)|$ (even with a smaller exponent 3 in D^3 than claimed 4).

To show the lemma for $k < d$ we repeat several times the following three steps.

Step 1. Consider the polynomial $f(X) = C(X_1^0, \ldots, X_k^0, X, \ldots, X_d^0)$ (we fix all inputs except for X_{k+1}). Apply Lemma 3 to this polynomial and to $X^0 = X_{k+1}^0$. The resulting set X^1 has at most D elements and

$$|C(X_1^0, \ldots, X^1, \ldots, X_d^1)| \geqslant 2^{-(D(D+1)+D\log(n+1))}|C(X_1^0, \ldots, X_k^0, \ldots, X_d^0)|.$$

Replace X_{k+1}^0 by X^1.

Step 2. The goal of this step is to change X_k^0 so that the ordinal number of X_{k+1}^0 be at most $\frac{Dn}{D+1}$ in the order on $k+1$st component associated with the largest element of $X_1^0 \times \ldots \times X_d^0$.

Since the cardinality of X_{k+1}^0 is at most D, there is an order $<_i$ such that the ordinal number of the largest element in X_{k+1}^0 w.r.t. this order is at most $nD/(D+1)$. Indeed, the latter means that X_{k+1}^0 is disjoint with the set of D largest elements in that order. And our orders are defined so that the sets of D largest elements are pair-wise disjoint for all $D+1$ orders. Thus no D-element set can intersect all of them.

Adding to X_k^0 only one element among $2(D+1)$ elements following its largest element w.r.t. the order $<_l$, we can change the order on $k+1$st component so that the new order be the order $<_i$ from the previous paragraph. However such change of X_k^0 may decrease $C(X_1^0, \ldots, X_d^0)$ very much, and we do not want that.

So we need again Lemma 2. This time we apply it to the polynomial

$$f(X) = C(X_1^0, \ldots, X, X_{k+1}^0, \ldots, X_d^0)$$

and to $A = X_k^0$. We also have to specify the set B from the conditions of the lemma. We have to choose B carefully, as we need that after replacing A by $A \cup M$ the new order on the $k+1$st component be $<_i$.

Let U stand for the set of $2\gamma D^2(D+1)$ numbers that follow the maximal element of X_k^0 w.r.t. the order $<_l$:

$$U = \{x | x \in [n],\ \mathrm{num}_l(\alpha_k) + 1 \leqslant \mathrm{num}_l(x) \leqslant \mathrm{num}_l(\alpha_k) + 2\gamma D^2(D+1)\}.$$

Note that the rule (1) to define the order on $k+1$st component is periodic with period $2(D+1)$. In every period, every order occurs twice: it occurs once with an odd index and once with an even index. Therefore U has exactly $2\gamma D^2$ elements that will define the order $<_i$ on $k+1$st component, call U_i the set of such numbers. Exactly half of numbers in U_i occur with an odd index. Let B_0 stand for the set of such numbers and let $B_1 = U_i \setminus B_0$.

We claim that the conditions of Lemma 2 are fulfilled for $A = X_k^0$ and for B equal either to B_0 or to B_1. Indeed, all elements in U_i are greater than $\max_l(A)$. Thus for every non-empty $M \subseteq U_i$ we have $\max_l(A \cup M) = \max_l(M)$. Moreover, all the components of the maximal element of $X_1^0 \times \cdots \times (A \cup M) \times \cdots \times X_d^0$, except for the kth one, do not depend on M. Indeed, its first $k-1$ component coincides with corresponding components of the maximal element in $X_1^0 \times \cdots \times A \times \cdots \times X_d^0$ and all elements of U_i define the same order on the $k+1$th component. This implies that the function $\text{ODD-MAX-BIT}_d^D(X_1^0, \ldots, A \cup M, \ldots, X_d^0)$ is equal either to ODD-MAX-BIT$(A \cup M)$ or to its negation (where ODD-MAX-BIT is defined w.r.t. the order $<_l$). W.l.o.g. assume that the first case holds. Then for all non-empty $M \subseteq U_i$ the sign of $f(A \cup M)$ is equal to ODD-MAX-BIT$(A \cup M)$. The function ODD-MAX-BIT$(A \cup M)$ depends only of the parity of the index of maximal element of M (w.r.t. $<_l$). Thus the sign of $f(A \cup M)$ is constant for all non-empty $M \subseteq B_0$ and the same holds for subsets of B_1. And for subsets of B_1 the sign of $f(A \cup M)$ is different from that for subsets of B_0. Let $B = B_0$ if the sign of $f(A \cup M)$ is different from that of $f(A)$ for non-empty $M \subseteq B_0$ and let $B = B_1$ otherwise.

Now we can apply Lemma 2 to obtain $M \subseteq B$ with $|f(A \cup M)| \geqslant 2|f(A)|$. Replace X_k^0 by $X_k^0 \cup M$ (we have a little bit more than needed, as the absolute value of C is even doubled).

Step 3. Note that after Step 2 the tuple $X_1^0, \ldots, X_d^0$ satisfies the conditions of Lemma 4 for $k+1$. Applying the induction hypothesis, we obtain $X_{k+1}^1, X_{k+2}^1, \ldots, X_d^1$ with

$$|C(X_1^0, \ldots, X_k^1, \ldots, X_d^1)| \geqslant 2^{\left(\frac{\varepsilon n}{D^4}\right)^{d-k}} |C(X_1^0, \ldots, X_k^1, X_{k+1}^0 \ldots, X_d^0)|. \tag{4}$$

How the first k coordinates of the maximal element of $X_1 \times \cdots \times X_d$ have been changed due to making Steps 1, 2 and 3? The first $k-1$ components have remain intact and the kth component has increased by at most $2\gamma D^2(D+1)$ w.r.t. $<_l$. At the start, by conditions of the lemma, the ordinal number of kth component was less than $\frac{nD}{D+1}$. Thus we can repeat these three steps $\frac{n}{2\gamma D^2(D+1)^2}$ times. Choose ε so that $\frac{n}{2\gamma D^2(D+1)^2} \geqslant \frac{2\varepsilon n}{D^4}$. Then the number of iterations is at least $\frac{2\varepsilon n}{D^4}$.

During each iteration the absolute value of C first decreases by a factor of at most $2^{D(D+1)+D\log(n+1)}$ on Step 1 and then increases by a factor of $2^{\left(\frac{\varepsilon n}{D^4}\right)^{d-k}}$ on Step 3. We can assume that the constant δ in the inequality (2) is so small that the exponent in the first factor is less than $\varepsilon n/2D^4$. Thus, after each iteration, the absolute value of C increases by a factor of

$$2^{\left(\frac{\varepsilon n}{D^4}\right)^{d-k} - \frac{\varepsilon n}{2D^4}} \geqslant 2^{\left(\frac{\varepsilon n}{D^4}\right)^{d-k}/2}.$$

Raising this factor to the power of the number of iterations, we obtain the desired factor $2^{\left(\frac{\varepsilon n}{D^4}\right)^{d-k+1}}$.

Let us finish the proof of the theorem. Let in Lemma 4 $X_1^0 = \ldots = X_d^0 = \{1\}$ and $k = 1$. The conditions of the lemma are fulfilled and hence there are $X_1', \ldots, X_d'$, with

$$|C(X_1', \ldots, X_d')| \geqslant 2^{\left(\frac{\varepsilon n}{D^4}\right)^d} |C(\{1\}, \ldots, \{1\})| \geqslant 2^{\left(\frac{\varepsilon n}{D^4}\right)^d}.$$

Now we can fix parameters in Theorem 1 and obtain the following results.

Theorem 2. *Let $D(n) = n^{o(1)}$. Then for any $\epsilon > 0$ the function* $\text{ODD-MAX-BIT}_d^{D(n)}(X)$ *is computable by a perceptron of order d, but is not computable by any perceptron of order at most $D(n)$ with total weight $2^{n^{d-\epsilon}}$.*

Theorem 3. *Let d and D be constants. Then the function* $\text{ODD-MAX-BIT}_d^{D}(X)$ *is computable by a perceptron of order d, but is not computable by a perceptron of order at most D with total weight $2^{o(n^d)}$.*

Acknowledgements. I am grateful to Nikolay Vereshchagin and to Harry Buhrman for drawing my attention to this problem.

References

1. Allender, E.: Circuit complexity before the dawn of the new Millennium. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 1–18. Springer, Heidelberg (1996)
2. Beigel, R.: Perceptrons, PP and the polynomial hierarchy. Computational Complexity 4, 339–349 (1994)
3. Beigel, R., Reingold, N., Spielman, D.A.: The perceptron strikes back. In: Proceedings of Structure in Complexity Theory Conference, pp. 286–291 (1991)
4. Chandra, A., Stockmeyer, L., Vishkin, U.: Constant depth reducibility. SIAM Journal on Computing 13, 423–439 (1984)
5. Goldmann, M., Håstad, J., Razborov, A.A.: Majority gates vs. general weighted threshold gates. Computational Complexity 2, 277–300 (1992)
6. Hajnal, A., Maass, W., Pudlák, P., Szegedy, M., Turán, G.: Threshold circuits of bounded depth. Journal on Computer and System Science 46, 129–154 (1993)
7. Håstad, J.: Almost optimal lower bounds for small depth circuits. In: Micali, S. (ed.) Randomness and Computation, Advances in Computing Research, vol. 5, pp. 143–170. JAI Press Inc. (1989)
8. Håstad, J.: On the size of weights for threshold gates. SIAM Journal on Discrete Mathematics 7(3), 484–492 (1994)
9. Minsky, M.L., Papert, S.A.: Perceptrons. MIT Press, Cambridge (1968)
10. Muroga, S.: Threshold logic and its applications. Wiley-Interscience, Chichester (1971)
11. Nisan, N.: The communication complexity of threshold gates. In: Miklós, D., Sós, V.T., Szönyi, T. (eds.) Combinatorics, Paul Erdös is Eighty, vol. 1, pp. 301–315. Jason Bolyai Math. Society, Budapest, Hungary (1993)
12. Podolskii, V.V.: Perceptrons of large weight. In: Proceedings, Second International Symposium on Computer Science in Russia, pp. 328–336 (2007)
13. Razborov, A.: Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. Matematicheskie Zametki 41(4), 598–607 (1987); English translation in Mathematical Notes of the Academy of Sci. of the USSR 41(4), 333–338 (1987)
14. Smolensky, R.: Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In: Proceedings, 19th ACM Symposium of Theory of Computing, pp. 77–82 (1987)
15. Yao, A.: Separating the polynomial-time hierarchy by oracles. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 1–10 (1985)

Lambek Grammars with One Division Are Decidable in Polynomial Time

Yury Savateev*

Department of Mathematical Logic, Faculty of Mechanics and Mathematics,
Moscow State University, Moscow, 119991, Russia

Abstract. Lambek grammars provide a useful tool for studying formal and natural languages. The generative power of unidirectional Lambek grammars equals that of context-free grammars. However, no feasible algorithm was known for deciding membership in the corresponding formal languages. In this paper we present a polynomial algorithm for deciding whether a given word belongs to a language generated by a given unidirectional Lambek grammar.

1 Introduction

Lambek calculus and Lambek categorial grammars were first introduced in [4]. Lambek calculus uses syntactic types that are built from primitive types using three binary connectives: multiplication, left division, and right division. Natural fragments of Lambek calculus are the product-free Lambek calculus, which does not use multiplication, and the unidirectional Lambek calculus, which has only one connective left: a division (left or right).

The Lambek calculus is intended for specifying formal languages (sets of finite words over a finite alphabet); this is done in the framework of categorial grammars, where all language-specific information is put in a lexicon and the derivation rules are the same for all languages.

Lambek categorial grammars have a designated type (usually primitive) and assign different types to elements of an alphabet (to one element one can assign several types). The grammar accepts a word if there are assignments for the elements of the alphabet in the word such that the sequent with a string of these assignments as the antecedent and the designated type as the succedent is derivable in Lambek calculus. Depending on which fragment of Lambek calculus is used, one can study product-free Lambek grammars and unidirectional Lambek grammars.

Categorial grammars based on the original Lambek calculus recognize exactly the context-free languages that do not contain the empty word (this was proved by M. Pentus in [7]). The same holds for both of the product-free and the

* This research was supported in part by the Russian Foundation for Basic Research, grant 08-01-00399, and by the Dutch-Russian cooperation program NWO/RFBR, project No. 047.017.014.

unidirectional Lambek calculus (the last result was proved by W. Buszkowski in [2]).

Of course, for linguistic applications it is important to have efficient algorithms that find out whether a sequence of symbols is accepted by a given grammar. In fact, for a given calculus in the framework of categorial grammars there are two important algorithmic problems. The first one concerns derivability in the calculus, and the second one is about acceptability by a grammar (the second problem has an additional degree of nondeterminism, since in categorial grammar an element of the alphabet may be associated with several syntactic types; here the input consists of a grammar and a word). For every variant of the calculus these problems obviously lie in NP. For the original Lambek calculus both problems are NP-complete (see [6]). A polynomial algorithm for determining derivability in the unidirectional calculus was presented in [8].

The algorithm presented in [8] goes as follows. At first, one decomposes the given types into atomic building blocks, labels them with natural numbers (which indicate how deeply the atomic building block is nested in denominators of division operators), and puts them in a certain order (which reflects the group-theoretic interpretation of the division operator). Next, one evaluates an auxiliary predicate of 'acceptability' for all substrings of the string of labelled atomic building blocks. Doing this in the manner of dynamic programming leads to a straightforward cubic algortihm for deciding derivability in the unidirectional Lambek calculus.

One might expect that the acceptability problem is harder, since in general one word corresponds to exponentially many Lambek calculus sequents, whose derivability determines whether the word is accepted (at least one of the sequents must be derivable). Fortunately, the algorithm from [8] can be extended to provide a polynomial-time solution for this more general problem.

For each of these variants of Lambek calculus we assume that antecedents of sequents are just strings of types without any additional structure (i.e., we have full associativity). However, non-associative variants of Lambek calculus (see [5]) can be studied too. In [1] it was proven that the grammar acceptability problem for non-associative product-free Lambek grammars can be solved in polynomial time. In [3] the polynomiality was proven for the derivability problem for full non-associative Lambek calculus. There are open questions concerning variants that have restricted associativity, for example only left associativity.

In this paper we prove that for the unidirectional Lambek grammar the acceptability problem is decidable in deterministic polynomial time and present an algorithm for it. Concerning the three associative fragments, now only the case of the product-free Lambek calculus remains open.

2 Lambek Calculus L

The original Lambek calculus can be constructed as follows. Let $\mathbf{P} = \{p_0, p_1, \ldots\}$ be a countable set of what we call *primitive types*. Let Tp be the set of *types* constructed from primitive types with three binary connectives $/$, $\backslash$, and $\cdot$. We will denote primitive types by small letters (p, q, $r, \ldots$) and types by capital letters (A, B, $C, \ldots$). By capital greek letters (Π, Γ, $\Delta, \ldots$) we will denote finite (possibly empty) sequences of types. Expressions like $\Pi \to A$ where Π is not empty are called sequents.

Axioms and rules of L:

$$A \to A$$

$$(\to /)\frac{\Pi A \to B}{\Pi \to B/A} \qquad (\to \backslash)\frac{A\Pi \to B}{\Pi \to A\backslash B}$$

$$(/ \to)\frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma (B/A) \Pi \Delta \to C} \qquad (\backslash \to)\frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma \Pi (A\backslash B) \Delta \to C}$$

$$(\cdot \to)\frac{\Gamma A B \Delta \to C}{\Gamma (A \cdot B) \Delta \to C} \qquad (\to \cdot)\frac{\Gamma \to A \quad \Delta \to B}{\Gamma \Delta \to A \cdot B}\,.$$

(Here Γ and Δ can be empty, but Π must be non-empty).

In this paper we will only consider the unidirectional Lambek calculus $\mathrm{L}^{\backslash}$, which is a similar calculus but with only one connective $\backslash$ (so it has a smaller set of types, $\mathrm{Tp}(\backslash)$). It only has the axiom and two rules, $(\to \backslash)$ and $(\backslash \to)$. The choice of $\backslash$ over $/$ is arbitrary: all the constructions and proofs can be rewritten for $\mathrm{L}^{/}$.

3 Unidirectional Lambek Grammars

We assume that a finite alphabet Σ and a distinguished type $B \in \mathrm{Tp}(\backslash)$ are given (usually when Lambek grammars are defined the distinguished type is said to be in $\mathbf{P}$, but the algorithm we will present works for all $B \in \mathrm{Tp}(\backslash)$). An unidirectional Lambek grammar is a mapping f such that, for all $t \in \Sigma$, $f(t) \subset \mathrm{Tp}(\backslash)$ and $f(t)$ is finite.

The language generated by the Lambek grammar is defined as the set of all words $x_1 \ldots x_n$ over the alphabet Σ for which there exists a sequent $A_1 \ldots A_n \to B$ such that $A_i \in f(x_i)$ for all $i \leqslant n$ and $\mathrm{L}^{\backslash} \vdash A_1 \ldots A_n \to B$. We shall denote this language by $L(\Sigma, B, f)$.

In [2] it was proven that the languages generated by unidirectional Lambek grammars are exactly all context-free languages without the empty word.

4 Representation of Sequents

We will use the representation of sequents as strings of atoms used in [8].

Let Atn be the set of *atoms* or *primitive types with superscripts*, $\{p^{(i)} | p \in \mathbf{P}, i \in \mathbb{N}\}$. We will consider the set of *strings of atoms* Atn^*. We will denote strings by $\mathbb{A}$, $\mathbb{B}$, and so on; ε will denote the empty string. Let $(\cdot)^{+2}$ be the operation on Atn^* satisfying the following conditions:

$$(p^{(i)})^{+2} = p^{(i+2)}$$

$$(\mathbb{A}\mathbb{B})^{+2} = (\mathbb{A})^{+2}(\mathbb{B})^{+2}\,.$$

Consider $\gamma, \overline{\gamma} : \mathrm{Tp}(\backslash) \to \mathrm{Atn}^*$, two mappings from types to strings of atoms defined by

$$\begin{aligned} \gamma(p) &= p^{(1)} & \overline{\gamma}(p) &= p^{(2)} \\ \gamma(A\backslash B) &= \overline{\gamma}(A)\gamma(B) & \overline{\gamma}(A\backslash B) &= \overline{\gamma}(B)(\gamma(A))^{+2}. \end{aligned}$$

Also define $\gamma(\Pi)$ for $\Pi \in \mathrm{Tp}(\backslash)^*$ as follows: if $\Pi = A_1 A_2 \dots A_n$, then $\gamma(\Pi) = \gamma(A_1)\gamma(A_2)\dots\gamma(A_n)$. It is readily seen that for any $n > 0$ the string $\gamma(A_1 \dots A_n)$ ends in $p^{(1)}$ and the total number of atoms with superscript 1 in it equals n.

Here are some examples of how γ and $\overline{\gamma}$ work:

$$\gamma\big((p\backslash q)\big) = p^{(2)}q^{(1)}$$

$$\overline{\gamma}\big((p\backslash q)\big) = q^{(2)}p^{(3)}$$

$$\gamma\big(((s\backslash q)\backslash(r\backslash p))\big) = q^{(2)}s^{(3)}r^{(2)}p^{(1)}$$

$$\overline{\gamma}\big(((s\backslash q)\backslash(r\backslash p))\big) = p^{(2)}r^{(3)}s^{(4)}q^{(3)}$$

$$\gamma\Big(\big(p\backslash(q\backslash(r\backslash(s\backslash(t\backslash u)))))\big)\big(((((p\backslash q)\backslash r)\backslash s)\backslash t)\backslash u)\big)\Big) = \\ = p^{(2)}q^{(2)}r^{(2)}s^{(2)}t^{(2)}u^{(1)}t^{(2)}r^{(4)}p^{(6)}q^{(5)}s^{(3)}u^{(1)}$$

Now we define two subsets of Atn^*: *plus-strings* and *minus-strings*. We will write $\mathbb{A}^+$ and $\mathbb{B}^-$ to denote that $\mathbb{A}$ is a plus-string and $\mathbb{B}$ is a minus-string. These two sets are the smallest sets that satisfy the following recursive conditions:

$$(\varepsilon)^+$$
$$(p^{(1)})^+$$
$$(p^{(2)})^-$$
$$\mathbb{A}^+, \mathbb{B}^+ \Rightarrow (\mathbb{A}\mathbb{B})^+$$
$$\mathbb{A}^-, \mathbb{B}^+, \mathbb{B} \neq \varepsilon \Rightarrow (\mathbb{A}\mathbb{B})^+$$
$$\mathbb{A}^-, \mathbb{B}^+ \Rightarrow (\mathbb{A}(\mathbb{B})^{+2})^- .$$

We will use the following notational convention: $^+$ and $^-$ can relate to parts of a string. For example when we write $\mathbb{A}^+\mathbb{B}\mathbb{C}^-$, this means "we have a string of the form $\mathbb{A}\mathbb{B}\mathbb{C}$ such that $\mathbb{A}^+$ and $\mathbb{C}^-$".

The mapping γ is a bijection between sequences from $\mathrm{Tp}(\backslash)^*$ and non-empty plus-strings. And $\overline{\gamma}$ is a bijection between $\mathrm{Tp}(\backslash)$ and minus-strings.

In the calculus S the derivable objects are of the form $\to \mathbb{A}$, where $\mathbb{A}$ is from Atn^*. Axioms and rules of S are the following:

$$\to p^{(1)}p^{(2)}$$

$$(\mathrm{S1})\,\frac{\to \mathbb{A}^+\mathbb{B}^+p^{(2)}}{\to \mathbb{B}p^{(2)}(\mathbb{A})^{+2}}$$

$$(\mathrm{S2})\,\frac{\to \mathbb{A}^+\mathbb{B}^- \qquad \to \mathbb{C}^+\mathbb{D}^+p^{(2)}}{\to \mathbb{C}\mathbb{A}\mathbb{B}\mathbb{D}p^{(2)}}\,.$$

Here $\mathbb{A}, \mathbb{B}$, and $\mathbb{D}$ must be non-empty.

Lemma 1 (Equivalence between $\mathrm{L}^{\backslash}$ **and** S**).** *For every* $A \in \mathrm{Tp}(\backslash)$ *and* $\Pi \in \mathrm{Tp}(\backslash)^*$ *we have*

$$\mathrm{L}^{\backslash} \vdash \Pi \to A \iff \mathrm{S} \vdash\to \gamma(\Pi)\overline{\gamma}(A)\,.$$

And conversely for $\mathbb{B}, \mathbb{C} \in \mathrm{Atn}^*$ *such that* $\mathbb{B}^+$ *and* $\mathbb{C}^-$ *we have*

$$\mathrm{S} \vdash\to \mathbb{B}\mathbb{C} \iff \mathrm{L}^{\backslash} \vdash \gamma^{-1}(\mathbb{B}) \to \overline{\gamma}^{-1}(\mathbb{C})\,.$$

The proof of Lemma 1 can be found in [8].

Lemma 2 (Derivability Criterion for S**).** *Suppose* $\mathbb{A}, \mathbb{B} \in \mathrm{Atn}^*, \mathbb{A}^+$, *and* $\mathbb{B}^-$. *Then* $\mathrm{S} \vdash\to \mathbb{A}\mathbb{B}$ *if and only if all atoms in* $\mathbb{A}\mathbb{B}$ *can be divided into pairs satisfying the following conditions:*

1. *A pair consists of* $p^{(i)}$ *and* $p^{(i+1)}$. *In other words, atoms in a pair correspond to the same primitive type, their superscripts differ by 1, and the atom with lesser superscript stays to the left (the string is thought to be written from left to right).*
2. *If atoms of the string are mapped to different points on a straight line with the same order (the atom lies between two others on a string iff the corresponding point for that atom lies between corresponding points for the other two), then lines connecting points corresponding to atoms in a pair, can be drawn in the upper semiplane simultaneously without intersections. In other words, atoms in any two pairs can only follow each other in one of the following orders:*

$$\ldots p^{(i)} \ldots p^{(i+1)} \ldots q^{(j)} \ldots q^{(j+1)} \ldots$$

$$\ldots p^{(i)} \ldots q^{(j)} \ldots q^{(j+1)} \ldots p^{(i+1)} \ldots$$

3. *If the superscript of the leftmost atom in a pair equals* $2l$ *(is even), then there is an atom with superscript less than* $2l$ *between the elements of the pair.*

Proof. ($\Rightarrow$). Consider a derivation of $\to \mathbb{A}\mathbb{B}$ in S. Divide the atoms into pairs throughout all the strings used in the derivation as follows: the atoms in one axiom form a pair, and the string resulting from an application of a rule has the same pairing as the premises (all atoms from the premises go over to the result).

By induction on the rules of S one can easily prove that such a pairing satisfies all necessary conditions.

($\Leftarrow$) We will use induction on the length of the string $\mathbb{A}\mathbb{B}$. For strings of length 2 the proof is trivial.

Suppose that $\mathbb{A}^+$, $\mathbb{B}^-$, and $\mathbb{A}\mathbb{B}$ has a pairing satisfying all necessary conditions. Since $\mathbb{B}$ is a minus-string, it can be either $p^{(2)}$ or $p^{(2)}(\mathbb{C})^{+2}$ for some plus-string $\mathbb{C}$. In the second case $\mathrm{S} \vdash\rightarrow \mathbb{C}\mathbb{A}p^{(2)} \Leftrightarrow \mathrm{S} \vdash\rightarrow \mathbb{A}\mathbb{B}$ (there is only one variant for the last step in the derivation of the string $\rightarrow \mathbb{A}\mathbb{B}$ for long $\mathbb{B}$), so we can prove the derivability of strings of length n only for those that have short $\mathbb{B}$.

The string $\mathbb{A}p^{(2)}$ is of the form $\mathbb{D}q^{(1)}r^{(2)}\mathbb{E}s^{(1)}p^{(2)}$, where $\mathbb{E} \succ 1$. It is readily seen that if a proper pairing exists, then $s = p$, $r = q$, and these atoms form two pairs from the proper pairing.

There are two possibilities: either $\mathbb{E} \succ 2$ or there is an atom with superscript 2 in $\mathbb{E}$.

If $\mathbb{E} \succ 2$, then $\mathbb{D}q^{(1)}q^{(2)}\mathbb{E}$ is a shorter string having a proper pairing, $(\mathbb{D}q^{(1)})^+$, and $(q^{(2)}\mathbb{E})^-$. By the induction assumption it means that $\mathrm{S} \vdash\rightarrow \mathbb{D}q^{(1)}q^{(2)}\mathbb{E}$. By applying rule S2 to this string and $\rightarrow p^{(1)}p^{(2)}$ we get $\rightarrow \mathbb{D}q^{(1)}q^{(2)}\mathbb{E}p^{(1)}p^{(2)}$.

Now let us suppose that there is an atom with superscript 2 in $\mathbb{E}$. Let $\mathbb{E}$ be of the form $\mathbb{E}''t^{(2)}\mathbb{E}'$, where $\mathbb{E}'' \succ 2$. The paired atom for this $t^{(2)}$ is some $t^{(1)}$ standing to the left of $q^{(1)}$. Then the string is of the following form:

$$\mathbb{D}t^{(1)}\mathbb{D}''q^{(1)}q^{(2)}\mathbb{E}''t^{(2)}\mathbb{E}'p^{(1)}p^{(2)}\,.$$

Note that $(\mathbb{D}'t^{(1)}t^{(2)}\mathbb{E}'p^{(1)})^+$, $(p^{(2)})^-$, $(\mathbb{D}''q^{(1)})^+$, $(q^{(2)}\mathbb{E}'')^-$, and the pairings for $\mathbb{D}t^{(1)}t^{(2)}\mathbb{E}'p^{(1)}p^{(2)}$ and $\mathbb{D}''q^{(1)}q^{(2)}\mathbb{E}''$ taken from the pairing for the whole string $\mathbb{D}t^{(1)}\mathbb{D}''q^{(1)}q^{(2)}\mathbb{E}''t^{(2)}\mathbb{E}'p^{(1)}p^{(2)}$ satisfy all necessary conditions. By the induction assumption it means that $\mathrm{S} \vdash\rightarrow \mathbb{D}t^{(1)}t^{(2)}\mathbb{E}'p^{(1)}p^{(2)}$ and $\mathrm{S} \vdash\rightarrow \mathbb{D}''q^{(1)}q^{(2)}\mathbb{E}''$. Then by applying rule S2 to these strings we can get $\rightarrow \mathbb{D}t^{(1)}\mathbb{D}''q^{(1)}q^{(2)}\mathbb{E}''t^{(2)}\mathbb{E}'p^{(1)}p^{(2)}$. This exactly means that $\mathrm{S} \vdash\rightarrow \mathbb{A}p^{(2)}$.

Thus in both cases we have proved that $\mathrm{S} \vdash\rightarrow \mathbb{A}p^{(2)}$. The proof is now complete. □

For example, the string

$$p^{(1)}p^{(2)}q^{(2)}r^{(1)}r^{(2)}s^{(3)}s^{(4)}q^{(3)}$$

has a proper pairing, thus $\mathrm{S} \vdash\rightarrow p^{(1)}p^{(2)}q^{(2)}r^{(1)}r^{(2)}s^{(3)}s^{(4)}q^{(3)}$ and by Lemma 2

$$\mathrm{L}^{\backslash} \vdash p(p\backslash(q\backslash r)) \rightarrow ((s\backslash q)\backslash(s\backslash r))\,.$$

The strings

$$p^{(1)}q^{(2)}$$
$$p^{(1)}q^{(2)}p^{(2)}r^{(1)}r^{(2)}q^{(3)}$$
$$p^{(1)}p^{(2)}p^{(2)}p^{(3)}q^{(2)}r^{(1)}r^{(2)}q^{(3)}$$

do not have proper pairings, thus

$$\mathrm{S} \not\vdash\!\!\to p^{(1)}q^{(2)}$$
$$\mathrm{S} \not\vdash\!\!\to p^{(1)}q^{(2)}p^{(2)}r^{(1)}r^{(2)}q^{(3)}$$
$$\mathrm{S} \not\vdash\!\!\to p^{(1)}p^{(2)}p^{(2)}p^{(3)}q^{(2)}r^{(1)}r^{(2)}q^{(3)}\,.$$

By Lemma 1 this also means that

$$\mathrm{L}^{\backslash} \not\vdash p \to q$$
$$\mathrm{L}^{\backslash} \not\vdash p(q\backslash(p\backslash r)) \to (q\backslash r)$$
$$\mathrm{L}^{\backslash} \not\vdash p(p\backslash((p\backslash p)\backslash(q\backslash r))) \to (q\backslash r)\,.$$

5 The Algorithm for Grammar Acceptance

Let (Σ, B, f) be a Lambek grammar. For every letter $a \in \Sigma$, let k^a denote $|f(a)|$, and for $j \leqslant k^a$ let A^a_j be the j-th possible type assignment for a. In other words, $f(a) = \{A^a_1, \ldots, A^a_{k^a}\}$ for every $a \in \Sigma$. Suppose we are given a word $x = x_1 \ldots x_n \in \Sigma^+$. Our goal is to check whether there are numbers $m_i \leqslant k^{x_i}$ such that $\mathrm{L}^{\backslash} \vdash A^{x_1}_{m_1} \ldots A^{x_n}_{m_n} \to B$. Since $\mathrm{L}^{\backslash}$ and S are equivalent (Lemma 1), we can seek m_i that satisfy $\mathrm{S} \vdash\!\!\to \gamma(A^{x_1}_{m_1}) \ldots \gamma(A^{x_n}_{m_n})\overline{\gamma}(B)$ instead.

For every letter $a \in \Sigma$ and $i \leqslant k^a$, let $\mathbb{A}^a_i = \gamma(A^a_i)$. Let Q^a be the following string of elements of the set $\mathrm{Atn} \cup \{*, \langle, \rangle\}$:

$$\langle * \mathbb{A}^a_1 * \ldots * \mathbb{A}^a_{k^a} * \rangle\,.$$

In other words, Q^a consists of Atn strings for every possible type assignment for a, separated by asterisks and enclosed in angle brackets. Also let $\mathbb{B} = \overline{\gamma}(B)$.

Let $W^x \in (\mathrm{Atn} \cup \{*, \langle, \rangle\})^*$ be the following string:

$$Q^{x_1} \ldots Q^{x_n} \langle * \mathbb{B}\,.$$

(Note: angle brackets in strings used in this construction are not assumed to be balanced.)

For $1 \leqslant i \leqslant |W^x|$, let W^x_i denote the i-th symbol of W^x. For $1 \leqslant i < j \leqslant |W^x|$, let $W^x_{[i,j]}$ denote the substring of W^x that starts at the i-th symbol and ends at the j-th. (Symbols can be atoms, asterisks, or angle brackets, $W^x_{[1,|W^x|]} = W^x$).

Let $M(i, j)$, where $1 \leqslant i < j \leqslant |W^x|$, be a function with the following properties: $M(i, j) = 1$ if one of the following holds:

1. $W^x_{[i,j]}$ lies in Atn^* and has a pairing satisfying all the conditions from the derivability criterion (a proper pairing).
2. $W^x_{[i,j]}$ is of the form $\langle \ldots \rangle \ldots \langle V * \mathbb{C}$, where $\mathbb{C} \in \mathrm{Atn}^*$, V contains no angle brackets, there are g pairs of matched angle brackets, for the h-th pair of them there is a substring of the form $*\mathbb{D}_h*$ in between them such that $\mathbb{D}_h \in \mathrm{Atn}^*$, and the string $\mathbb{D}_1 \ldots \mathbb{D}_g\mathbb{C}$ has a proper pairing.

3. $W^x_{[i,j]}$ is of the form $\mathbb{D} * U\rangle \dots \langle V * \mathbb{C}$ where $\mathbb{C}, \mathbb{D} \in \mathrm{Atn}^*$, U and V contain no angle brackets, there are g pairs of matched angle brackets, for the h-th pair of them there is a substring of the form $*\mathbb{E}_h*$ in between them such that $\mathbb{E}_h \in \mathrm{Atn}^*$, and the string $\mathbb{D}\mathbb{E}_1 \dots \mathbb{E}_g\mathbb{C}$ has a proper pairing.

In all other cases $M(i, j) = 0$.

The whole string W^x is of the second form. So, if $M(1, |W|) = 1$, then in each of the n pairs of angle brackets there is a substring of the form $*\mathbb{D}_j*$ such that $\mathbb{D}_1 \dots \mathbb{D}_n\mathbb{B}$ has a proper pairing. All of the substrings of the said form in j-th pair of angle brackets are Atn strings for different type assignments for x_j. This means that they can be taken as the required $\mathbb{A}^{x_j}_{m_i}$ such that $\mathrm{S} \vdash\!\rightarrow \gamma(A^{x_1}_{m_1}) \dots \gamma(A^{x_n}_{m_n})\overline{\gamma}(B)$ (Obviously $(\mathbb{D}_1 \dots \mathbb{D}_n)^+$ and $\mathbb{B}^-$). This exactly means that $x \in L(\Sigma, B, f)$. Also if $M(1, |W|) = 0$, then $x \notin L(\Sigma, B, f)$.

We will calculate $M(i, j)$ dynamically. First we declare that for all $i < n$ $M(i, i + 1) = 1$ only if $W^x_i = p^{(2k+1)}$ and $W^x_{i+1} = p^{(2k+2)}$ for some $p \in \mathbf{P}$ and $k \in \mathbb{N}$ (and 0 in all other cases).

How do we calculate $M(i, j)$, if we already know all $M(g, h)$ for all g and h such that $1 \leqslant g < h \leqslant n$ and $h - g < i - j$? There are several cases:

1. $W^x_i, W^x_j \in \mathrm{Atn}$. If there exists g such that $i < g < j - 1$, $W^x_g \in \mathrm{Atn}$, $W^x_{g+1} \in \mathrm{Atn}$, $M(i, g) = 1$, and $M(g + 1, j) = 1$, then we put $M(i, j) = 1$.
2. $W^x_i = p^{(m)}$, $W^x_j = p^{(m+1)}$ for some $p \in \mathbf{P}$ and $m \in \mathbb{N}$. If $M(i+1, j-1) = 1$, m is odd or m is even and there exists g such that $i < g < j$ and the superscript of W^x_g is less then m, then we put $M(i, j) = 1$.
3. $W^x_{[i,j]}$ is of the form $p^{(1)} * U\rangle\langle V * p^{(2)}$ where U and V contain no angle brackets. Then we put $M(i, j) = 1$.
4. $W^x_{[i,j]}$ is of the form $\langle \dots\rangle \dots \langle \dots p^{(j)}$. If there exists g such that $i < g < j$, $W^x_g = *$, $W^x_{[i+1,g]}$ contains no angle brackets, and $M(g + 1, j) = 1$, then we put $M(i, j) = 1$.
5. $W^x_{[i,j]}$ is of the form $p^{(1)} * \dots\rangle \dots \langle \dots p^{(2)}$. If $M(g, j - 1) = 1$, where g is the position in W^x of the first left angle bracket in $W^x_{[i,j]}$, then we put $M(i, j) = 1$.

In all other cases we put $M(i, j) = 0$.

Lemma 3 (Correctness of the algorithm). *The described algorithm correctly computes $M(i; j)$.*

Proof. Let us call substrings $W^x_{[i,j]}$ for which $M(i, j) = 1$ 'accepted'. We will prove that the algorithm correctly finds all accepted substrings by induction on the length of the substring. We will refer to 'forms' of the accepted substrings: these are the forms used in definition of the function $M(i, j)$. Every accepted substring ends with an atom that participate in the pairing for it. There are no accepted substrings that contain asterisks but no angle brackets.

For substrings of length 2 it is obvious: accepted substrings of the length 2 can only be of the first form and can only have the form $p^{(2k+1)}p^{(2k+2)}$ for some $p \in \mathbf{P}$ and $k \in \mathbb{N}$. The algorithm finds all of them correctly.

Let us now consider substrings $W^x_{[i,j]}$ of length $l > 2$ such that for all $l' < l$ the algorithm finds accepted strings of the length l' correctly.

1. The substring $W^x_{[i,j]}$ is of the first form, i.e., lies in Atn^*. There is a pairing for all atoms of the substring that satisfy all conditions of the derivability criterion. There are two possible cases depending on wether leftmost and rightmost atoms form a pair. If they do not, then $W^x_{[i,j]}$ can be divided in two shorter substrings each of them having a proper pairing. If they do then $W^x_{[i+1,j-1]}$ is a shorter substring with a proper pairing. By the induction assumption these shorter substrings are already found by the algorithm, and thus it will accept $W^x_{[i,j]}$ (case 1 or 2 of the algorithm description).
2. The substring $W^x_{[i,j]}$ is of the second form. If it is accepted, there exists a substring of the form $*\mathbb{D}_1*$ in between the first pair of angle brackets such that $\mathbb{D}_1$ participates in the pairing. Let i' be the position of the first atom of $\mathbb{D}_1$. The substring $W^x_{[i',j]}$ is a substring of the third form, it is shorter than $W^x_{[i,j]}$, and it is accepted. Therefore by the induction assumption it is already found by the algorithm, and thus it will accept $W^x_{[i,j]}$ (case 4 of the algorithm description).
3. The substring $W^x_{[i,j]}$ is of the third form. The atom W^x_i participates in the pairing for the substring $W^x_{[i,j]}$. Let the pair for W^x_i be some atom $W^x_{i'}$. First, let us consider the case $i' \neq j$. Then obviously $i' < j - 1$, or there will be no possible pair for W^x_j. The atom $W^x_{i'}$ has a superscript bigger than 1. This means that $W^x_{i'+1}$ is also an atom, because asterisks can go only after atoms with superscript one, since W^x is constructed of Atn strings for Lambek types. This means that $W^x_{[i,i']}$ and $W^x_{[i'+1,j]}$ are both of the first or the third form, are shorter, and accepted. Therefore by the induction assumption they are already found by the algorithm, and thus it will accept $W^x_{[i,j]}$ (case 1 of the algorithm description).

   Now let us suppose that W^x_i and W^x_j form a pair. Suppose that superscript of W^x_i is bigger then 1. This means that superscript of W^x_j is more than 2, and it cannot be the leftmost in an Atn string for a Lambek type. Therefore both W^x_{i+1} and W^x_{j-1} are atoms. Thus substring $W^x_{[i+1,j-1]}$ is of the third form, is shorter, and accepted. Therefore by the induction assumption it is already found by the algorithm, and thus it will accept $W^x_{[i,j]}$ (case 2 of the algorithm description). The only case left is when the superscript of W^x_i is 1 and consequently the superscript of W^x_j is 2. If there are no matched angle brackets in $W^x_{[i,j]}$, the substring $W^x_{[i,j]}$ is accepted only if it is of the form $p^{(1)} * U\rangle\langle V * p^{(2)}$, because if W^x_{j-1} is an atom, there is atom with superscript 2 in $W^x_{[i,j]}$ that must participate in the pairing, but there is no possible pair for it. If W^x_j is of the form $p^{(1)} * U\rangle\langle V * p^{(2)}$ it is accepted by case 3 of the algorithm description. Now suppose, that W^x_j is of the form $p^{(1)} * \ldots\rangle \ldots \langle \ldots p^{(2)}$. Let us consider the part of the string that participates in the pairing and lies in between the rightmost pair of matched angle brackets. Its rightmost atom has superscript 1, so it's pair must stay to the right of it, and therefore it stays to right of the last left angle bracket. This means that there are atoms besides W^x_j that participate in the pairing and stay to right of the last left angle bracket. Therefore W^x_{j-1} is also an atom. This means that the substring $W^x_{[j',j-1]}$, where j' is the position in W^x of the first left angle bracket in $W^x_{[i,j]}$, is of the second form, is shorter and accepted. Therefore by the induction assumption it is already found by the algorithm, and thus it will accept $W^x_{[i,j]}$ (case 5 of the algorithm description). $\square$

We can check all the conditions for $W^x_{[i,j]}$ for each case in $O(j-i)$ steps. There are $O(|W|^2)$ substrings. Thus we can calculate $M(1,n)$ in $O(|W^x|^3)$ time.

Acknowledgements. I am most grateful to Professor M. Pentus for his constant attention to my work.

References

1. Aarts, E., Trautwein, K.: Non-associative Lambek categorial grammar in polynomial time. Mathematical Logic Quarterly 41, 476–484 (1995)
2. Buszkowski, W.: The equivalence of unidirectional Lambek categorial grammars and context-free grammars. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 31(4), 369–384 (1985)
3. de Groote, P.: The non-associative Lambek calculus with product in polynomial time. In: Murray, N.V. (ed.) Automated Reasoning with Analytic Tableaux and Related Methods, pp. 128–139. Springer, Heidelberg (1999)
4. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65(3), 154–170 (1958)
5. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) Structure of Language and Its Mathematical Aspects, Proc. Symposia Appl. Math., vol. 12, pp. 166–178. Amer. Math. Soc, Providence, RI (1961)
6. Pentus, M.: Lambek calculus is NP-complete. Theoretical Computer Science 357(1–3), 186–201 (2006)
7. Pentus, M.: Lambek grammars are context free. In: Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science, pp. 429–433 (1993)
8. Savateev, Y.: The derivability problem for lambek calculus with one division. Technical report, Utrecht University, Artificial Intelligence Preprint Series no. 56 (2006)

Cryptanalysis of Stickel's Key Exchange Scheme

Vladimir Shpilrain[⋆]

Department of Mathematics, The City College of New York, New York, NY 10031
`shpil@groups.sci.ccny.cuny.edu`
`http://www.sci.ccny.cuny.edu/~shpil`

Abstract. We offer cryptanalysis of a key exchange scheme due to Stickel [11], which was inspired by the well-known Diffie-Hellman protocol. We show that Stickel's choice of platform (the group of invertible matrices over a finite field) makes the scheme vulnerable to linear algebra attacks with very high success rate in recovering the shared secret key (100% in our experiments). We also show that obtaining the shared secret key in Stickel's scheme is not harder for the adversary than solving the *decomposition search problem* in the platform (semi)group.

1 Introduction

In this paper, we offer cryptanalysis of a key exchange scheme due to Stickel [11]. His protocol is reminiscent of the well-known Diffie-Hellman protocol (see e.g. [3]), although formally it is not a generalization of the latter. We show in our Section 4 that Stickel's choice of platform (the group of invertible matrices over a finite field) makes the protocol vulnerable to linear algebra attacks. It appears that even such a seemingly minor improvement as using non-invertible matrices instead of invertible ones would already make Stickel's protocol significantly less vulnerable, at least to linear algebra attacks.

Perhaps more importantly, we show in Section 3 that to obtain the shared secret key in Stickel's scheme, the adversary does not have to solve any discrete logarithm-type problem; instead, he/she can solve the apparently easier *decomposition search problem* in the platform (semi)group G which is:

> Given a recursively presented (semi)group G, two recursively generated sub(semi)groups $A, B \leq G$, and two elements $u, w \in G$, find two elements $x \in A$ and $y \in B$ that would satisfy $x \cdot w \cdot y = u$, provided at least one such pair of elements exists.

We give some background on this problem in Section 3, and in Section 4 we describe a platform-specific attack on Stickel's scheme which boils down to solving a system of 1922 linear equations with 961 unknowns over a finite field $\mathbf{F}_{2^m}$. Finally, in Section 5 we give a couple of simple suggestions on improving Stickel's scheme.

[⋆] Partially supported by the NSF grant DMS-0405105 and by an NSA grant.

2 Stickel's Protocol

Let G be a public non-abelian finite group, $a, b \in G$ public elements such that $ab \neq ba$. The key exchange protocol goes as follows. Let N and M be the orders of a and b, respectively.

1. Alice picks two random natural numbers $n < N, m < M$ and sends $u = a^n b^m$ to Bob.
2. Bob picks two random natural numbers $r < N, s < M$ and sends $v = a^r b^s$ to Alice.
3. Alice computes $K_A = a^n v b^m = a^{n+r} b^{m+s}$.
4. Bob computes $K_B = a^r u b^s = a^{n+r} b^{m+s}$.

Thus, Alice and Bob end up with the same group element $K = K_A = K_B$ which can serve as the shared secret key.

When it comes to implementation details, the exposition in [11] becomes somewhat foggy. In particular, it seems that the author actually prefers the following more general version of the above protocol.

Let $w \in G$ be public.

1. Alice picks two random natural numbers $n < N, m < M$, an element c_1 from the center of the group G, and sends $u = c_1 a^n w b^m$ to Bob.
2. Bob picks two random natural numbers $r < N, s < M$, an element c_2 from the center of the group G, and sends $v = c_2 a^r w b^s$ to Alice.
3. Alice computes $K_A = c_1 a^n v b^m = c_1 c_2 a^{n+r} w b^{m+s}$.
4. Bob computes $K_B = c_2 a^r u b^s = c_1 c_2 a^{n+r} w b^{m+s}$.

Thus, Alice and Bob end up with the same group element $K = K_A = K_B$.

We note that for this protocol to work, G does not have to be a group; a semigroup would do just as well (in fact, even better, as we argue in Section 5).

In [11], it was suggested that the group of invertible $k \times k$ matrices over a finite field F_{2^l} is used as the platform group G. We show in Section 4 that this choice of platform makes the protocol vulnerable to linear algebra attacks, but first, in Section 3, we discuss a general (i.e., not platform-specific) approach to attacking Stickel's protocol. We emphasize that this general approach works if G is any semigroup, whereas the attack in Section 4 is platform-specific; in particular, it only works if G is a group, but may not work for arbitrary semigroups.

3 Preliminary Cryptanalysis of Stickel's Protocol

Recall that Alice transmits $u = c_1 a^n w b^m$ to Bob.

Our first observation is: to get a hold of the shared secret key K in the end, it is sufficient for the adversary (Eve) to find any elements $x, y \in G$ such that $xa = ax,\ yb = by,\ u = xwy$. Indeed, having found such x, y, Eve can use Bob's transmission $v = c_2 a^r w b^s$ to compute:

$$xvy = xc_2 a^r w b^s y = c_2 a^r xwy b^s = c_2 a^r u b^s = K.$$

This implies, in particular, that multiplying by c_i does not enhance the security of the protocol. More importantly, this also implies that it is not necessary for Eve to recover any of the exponents n, m, r, s; instead, she can just solve a system of equations $xa = ax,\ yb = by,\ u = xwy$, where a, b, u, w are known and x, y unknown elements of the platform (semi)group G. This shows that, in fact, Stickel's protocol departs from the Diffie-Hellman protocol farther than it seems. Moreover, solving the above system of equations in G is actually nothing else but solving the (subsemigroup-restricted) *decomposition search problem* which is:

> Given a recursively presented (semi)group G, two recursively generated sub(semi)groups $A, B \leq G$, and two elements $u, w \in G$, find two elements $x \in A$ and $y \in B$ that would satisfy $x \cdot w \cdot y = u$, provided at least one such pair of elements exists.

In reference to Stickel's scheme, the sub(semi)groups A and B are the *centralizers* of the elements a and b, respectively. The centralizer of an element $g \in G$ is the set of all elements $c \in G$ such that $gc = cg$. This set is a subsemigroup of G; if G is a group, then this set is a subgroup.

There are several key exchange protocols that directly use the alleged hardness of the decomposition search problem in various (semi)groups, see e.g. [2], [7], [8], [9]. So far, no particular (semi)group has been recognized as providing a secure platform for any of those protocols. Several attacks on the decomposition search problem in various "abstract" groups (i.e., in groups given by generators and defining relators) were reported, see e.g. [1], [4], [5]. It appears likely that semigroups of matrices over specific rings can generally make good platforms, as we argue in [6]. Stickel, too, used matrices in his paper [11], but he has made several poor choices, as we are about to see in the next Section 4. Also, Stickel's scheme is *at most* as secure as those schemes that are directly based on the alleged hardness of the decomposition search problem, because there are ways to attack Stickel's scheme without attacking the relevant decomposition search problem; for instance, Sramka [10] has offered an attack aimed at recovering one of the exponents n, m, r, s in Stickel's protocol. Our attack that we describe in Section 4 is more efficient, but on the other hand, it is aimed at recovering the shared secret key only, whereas Sramka's attack is aimed at recovering a private key.

4 Linear Algebra Attack

Now we are going to focus on the particular platform group G suggested by Stickel in [11]. In his paper, G is the group of invertible $k \times k$ matrices over a finite field $\mathbf{F}_{2^l}$, where $k = 31$. The parameter l was not specified in [11], but from what is written there, one can reasonably guess that $2 \leq l \leq k$. The choice of matrices a, b, w is not so important for our attack; what is important is that a and b are invertible. We note however that the choice of matrices a and b in

[11] (more specifically, the fact that the entries of these matrices are either 0 or 1) provides an extra weakness to the scheme as we will see at the end of this section.

Recall from our Section 3 that it is sufficient for Eve to find at least one solution of the system of equations $xa = ax,\ yb = by,\ u = xwy$, where a, b, u, w are known and x, y unknown $k \times k$ matrices over $\mathbf{F}_{2^l}$. Each of the first two equations in this system translates into a system of k^2 linear equations for the (unknown) entries of the matrices x and y. However, the equation $u = xwy$ does not translate into a system of linear equations for the entries because it has a product of two unknown matrices. We therefore have to use the following trick: multiply both sides of the equation $u = xwy$ by x^{-1} on the left (here is where we use the fact that x is invertible!) to get

$$x^{-1}u = wy.$$

Now, since $xa = ax$ if and only if $x^{-1}a = ax^{-1}$, we denote $x_1 = x^{-1}$ and replace the system of equations mentioned in the previous paragraph by the following one:

$$x_1a = ax_1,\ yb = by,\ x_1u = wy.$$

Now each equation in this system translates into a system of k^2 linear equations for the (unknown) entries of the matrices x_1 and y. Thus, we have a total of $3k^2$ linear equations with $2k^2$ unknowns. Note however that a solution of the displayed system will yield the shared key K if and only if x_1 is invertible because $K = xvy$, where $x = x_1^{-1}$.

Since u is a known invertible matrix, we can multiply both sides of the equation $x_1u = wy$ by u^{-1} on the right to get $x_1 = wyu^{-1}$, and then eliminate x_1 from the system:

$$wyu^{-1}a = awyu^{-1},\ yb = by.$$

Now we have just one unknown matrix y, so we have $2k^2$ linear equations for k^2 entries of y. Thus, we have a heavily overdetermined system of linear equations (recall that in Stickel's paper, $k = 31$, so $k^2 = 961$). We know that this system must have at least one non-trivial (i.e., non-zero) solution; therefore, if we reduce the matrix of this system to an echelon form, there should be at least one free variable. On the other hand, since the system is heavily overdetermined, we can expect the number of free variables to be not too big, so that it is feasible to go over possible values of free variables one at a time, until we find some values that yield an invertible matrix y. (Recall that all entries of y are either 0 or 1; this is an extra weakness of Stickel's scheme that we mentioned before.) Note that checking the invertibility of a given matrix is easy because it is equivalent to reducing the matrix to an echelon form. In fact, in all our experiments there was just one free variable, so the last step (checking the invertibility) was not needed because if there is a unique non-zero solution of the above system, then the corresponding matrix y should be invertible.

5 Suggestions on Improving Stickel's Scheme

The most obvious suggestion on improving Stickel's scheme is, as we mentioned before, to use non-invertible elements a, b, w; this implies, in particular, that the platform should be a semigroup with (a lot of) non-invertible elements. If one is to use matrices, then it makes sense to use the semigroup of *all* $k \times k$ matrices over a finite ring (not necessarily a field!). Such a semigroup typically has a lot of non-invertible elements, so it should be easy to choose a, b, w non-invertible, in which case the linear algebra attack from the previous section would not work. One more advantage of not restricting the pool to invertible matrices is that one can use not just powers a^j of a given public matrix in Stickel's protocol, but arbitrary expressions of the form $\sum_{i=1}^{p} c_i \cdot a^i$, where c_i are constants, i.e. elements of the ground ring.

Of course, there is no compelling reason why matrices should be employed in Stickel's scheme, but as we have explained in Section 3, with an abstract platform (semi)group G, Stickel's scheme is broken if the relevant decomposition search problem is solved, and so far, no particular abstract (semi)group has been recognized as resistant to known attacks on the decomposition search problem.

6 Conclusions

1. We have shown that obtaining the shared secret key K in Stickel's scheme is not harder for the adversary than solving the decomposition search problem in the platform (semi)group G.
2. We have described an efficient linear algebra attack, with 100% success rate (according to our experiments), on Stickel's scheme with parameters suggested in [11].
3. We have suggested possible improvements of Stickel's scheme related to the choice of platform; our main suggestion is to use a semigroup with a lot of non-invertible elements instead of a group.

References

1. Garber, D., Kaplan, S., Teicher, M., Tsaban, B., Vishne, U.: Probabilistic solutions of equations in the braid group. Advances in Applied Mathematics 35, 323–334 (2005)
2. Ko, K.H., Lee, S.J., Cheon, J.H., Han, J.W., Kang, J., Park, C.: New public-key cryptosystem using braid groups. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 166–183. Springer, Heidelberg (2000)
3. Menezes, A.J.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
4. Myasnikov, A.G., Shpilrain, V., Ushakov, A.: A practical attack on some braid group based cryptographic protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 86–96. Springer, Heidelberg (2005)
5. Ruinskiy, D., Shamir, A., Tsaban, B.: Cryptanalysis of group-based key agreement protocols using subgroup distance functions. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 61–75. Springer, Heidelberg (2007)

6. Shpilrain, V.: Hashing with polynomials. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 22–28. Springer, Heidelberg (2006)
7. Shpilrain, V., Ushakov, A.: Thompson's group and public key cryptography. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 151–164. Springer, Heidelberg (2005)
8. Shpilrain, V., Ushakov, A.: A new key exchange protocol based on the decomposition problem. Contemp. Math., Amer. Math. Soc. 418, 161–167 (2006)
9. Sidelnikov, V.M., Cherepnev, M.A., Yashcenko, V.Y.: Systems of open distribution of keys on the basis of noncommutative semigroups. Ross. Acad. Nauk Dokl. 332, (1993); English translation: Russian Acad. Sci. Dokl. Math. 48, 384–386 (1994)
10. Sramka, M.: On the Security of Stickel's Key Exchange Scheme (preprint)
11. Stickel, E.: A New Method for Exchanging Secret Keys. In: Proc. of the Third International Conference on Information Technology and Applications (ICITA 2005), vol. 2, pp. 426–430 (2005)

Combinatorial Complexity of Regular Languages

Arseny M. Shur

Ural State University

Abstract. We study combinatorial complexity (or *counting function*) of regular languages, describing these functions in three ways. First, we classify all possible asymptotically tight upper bounds of these functions up to a multiplicative constant, relating each particular bound to certain parameters of recognizing automata. Second, we show that combinatorial complexity equals, up to an exponentially small term, to a function constructed from a finite number of polynomials and exponentials. Third, we describe oscillations of combinatorial complexity for factorial, prefix-closed, and arbitrary regular languages. Finally, we construct a fast algorithm for calculating the growth rate of complexity for regular languages, and apply this algorithm to approximate growth rates of complexity of power-free languages, improving all known upper bounds for growth rates of such languages.

1 Introduction

The combinatorial complexity $C_L(n)$ of a language L is the number of words (strings) of length n in L. This is the most natural counting function associated with a language. Along with other functions of this kind appeared in the literature, such as subword/factor, palindromic, arithmetical, pattern, and maximal pattern complexities, it characterizes the richness of the language. In particular, the study of the behaviour of combinatorial complexity helps to understand the structure of languages and to discover the lack of some important properties such as the properties to be regular, context-free, generated by morphism, or freely generated as a subset of a free semigroup.

The combinatorial complexity (under different names) has been intensively studied for many languages and particular classes of languages. Probably the first results in this direction were obtained by Morse and Hedlund [20]. A systematic study of combinatorial complexity was initiated by Ehrenfeucht and Rozenberg in [12]; they focused mostly on an important, but narrow class of D0L-languages. A representative selection of results on complexity can be found in Sect. 9 of [6]. A massive related research on growth functions of algebraic structures, such as groups, semigroups, rings and graded algebras (see, e.g., [19,15,28,29]) should also be mentioned here.

Chomsky's hierarchy and its refinements provide very natural classes to study from the complexity point of view. In particular, combinatorial complexity of regular languages was studied already by Chomsky and Miller [7]. It was discovered independently by several authors that for context-free languages (and hence, for regular ones) a dichotomy theorem holds: the complexity of such a language grows either exponentially or polynomially (the earliest reference is [27]).

All possible *polynomial* complexity functions of context-free languages were described in [1]. Another result of [1] states that each of these functions is also the combinatorial complexity of some regular language. However, in the general case no satisfactory description is known even for regular languages. A coarse estimation of the behaviour of the combinatorial complexity for regular languages was made in Sect. 4 of [16]. The main goal of this paper is to obtain a complete description of the combinatorial complexity of regular languages.

A handy classification of combinatorial complexity in this particular case seems very important. Another reason is the study of factorial (that is, closed under taking factors) languages. Most of the complexity studies concentrates on factorial languages, which naturally arise from "negative" properties of words, such as the property "to avoid some regularity". The growth rate of the combinatorial complexity of an *arbitrary* recursive factorial language can be estimated with any precision by means of regular languages. In this paper we use the obtained description to develop a simple and efficient algorithm for such estimation. Hopefully, our results will make it possible to process some non-factorial languages in a similar way.

Our description of the combinatorial complexity of regular languages consists of two parts. The "coarse" part is given by Theorem 2. Combinatorial complexity is not necessary monotone, so Theorem 2 classifies all possible *supremum* asymptotic behaviours up to a multiplicative constant and relates these behaviours to certain parameters of recognizing automata. Theorem 3, which is the "finer" part of our description, reveals that the combinatorial complexity of a regular language is exponentially close to a function constructed from a finite number of polynomials and exponentials.

Using this description, we classify the behaviour of combinatorial complexity in terms of oscillations and show the main distinctions in such behaviour for factorial, prefix-closed, and arbitrary regular languages (Theorem 4).

In the last section, we provide a fast algorithm estimating the growth rate of a given regular language with any prescribed precision. We show how to apply this algorithm to approximate growth rates of factorial languages and briefly discuss the results of computational experiments. In particular, we improve all previously known upper bounds of the growth rates for power-free languages.

2 Preliminaries

We recall necessary notation and definitions. For more background, see [9,14,18].

1. Languages and automata. *Words* and *languages* over a finite *alphabet* Σ are considered. A word u is a *factor* (respectively *prefix*, *suffix*) of the word w if w can be represented as $\bar{v}u\hat{v}$ (respectively $u\hat{v}$, $\bar{v}u$) for some (possibly empty) words $\bar{v}$ and $\hat{v}$. As usual, we write Σ^n for the set of all n-letter words and Σ^* for the set of all words over Σ. The languages, obtained from the language $L \in \Sigma^*$ by closing under taking prefixes (suffixes, factors) are denoted by $\mathrm{PR}(L)$

(respectively, $\textsc{suf}(L)$ and $\textsc{fact}(L)$). A language is *prefix-closed* (*suffix-closed*, *factorial*) if it coincides with its corresponding closure.

A word w is *forbidden* for the language L if it is a factor of no element of L. The set of all minimal (with respect to the factorization order) forbidden words for a language is called the *antidictionary* of this language. If a factorial language L over the alphabet Σ has the antidictionary M, then the following equalities holds:

$$L = \Sigma^* - \Sigma^* M \Sigma^*, \qquad M = \Sigma L \cap L\Sigma \cap (\Sigma^* - L).$$

We see that any antidictionary determines a unique factorial language, which is regular if the antidictionary is also regular (in particular, finite).

We consider *deterministic finite automata* (dfa's) with *partial* transition function. An automaton (deterministic or nondeterministic) is *consistent* if each of its vertices is contained in some accepting path. We view dfa as a digraph, sometimes even omitting the labels of edges and vertices. For a dfa, the number of words of length n in the recognized language obviously equals the number of *accepting paths* of length n in the automaton. So, to calculate combinatorial complexity we count paths rather than words. For a fixed automaton, we denote the number of (u, v)-paths of length n by $P_{uv}(n)$ and the number of paths of length n starting at the vertex u by $P_u(n)$.

2. Complexity. For an arbitrary language L, we are mostly interested in the asymptotic behaviour of its combinatorial complexity. To compare functions, we use the standard O-, Ω-, and Θ-notation. To study the supremum asymptotic behaviour of non-monotonic complexity functions, we also use the notation $f(n) = \bar{\Omega}(g(n))$ iff there exist a sequence $\{n_i\}_1^\infty$ and a constant $C > 0$ such that $f(n_i) \geq Cg(n_i)$ for all i, and $f(n) = \bar{\Theta}(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \bar{\Omega}(g(n))$. As usual, we call a complexity function *polynomial* if it is $\bar{\Omega}(n^p)$ and $O(n^{\hat{p}})$ for some $p, \hat{p} \geq 0$, and *exponential* if it is $\bar{\Omega}(\alpha^n)$ and $O(\hat{\alpha}^n)$ for some $\alpha, \hat{\alpha} > 1$. For short, we often use expressions like "polynomial language".

Another useful characteristic of the asymptotic behaviour of a nonnegative-valued function is its *growth rate* $\limsup_{n\to\infty}(f(n))^{1/n}$. The growth rate sometimes gives less precision than a Θ-expression, but usually is easier to determine. We usually refer to *growth rate of a language* rather than the growth rate of its complexity function.

3. Digraphs and linear algebra. A *strongly connected component* (scc) of a digraph G is a maximal w.r.t. inclusion subgraph G' such that there exists a path from any vertex of G' to any other vertex of G'. A digraph is *strongly connected*, if it consists of a unique scc. A *component digraph* $\hat{G}$ of a digraph G is the acyclic digraph obtained from G by contracting all edges within each scc. Thus, the vertices of $\hat{G}$ can be considered as scc's of G. There is a natural *projective map* from G onto $\hat{G}$, mapping any vertex to its scc. This map allows one to define the *projection* of a path in G, which is a path in $\hat{G}$.

The adjacency matrix of a digraph is nonnegative, hence the classical Perron-Frobenius Theorem is applicable to this matrix. This theorem is formulated below for *irreducible* matrices. We mention that among the adjacency matrices of digraphs exactly those of strongly connected digraphs are irreducible.

Theorem 1 (Perron, Frobenius). *Let M be an irreducible nonnegative matrix, and α be the maximum absolute value of an eigenvalue of M. Then (1) α is itself an eigenvalue of M; (2) all eigenvalues of M with the absolute value α are simple; (3) if there are exactly k such eigenvalues $\lambda_1, \ldots, \lambda_k$, then $\lambda_j = \alpha\varepsilon_k^j$, where $j = 1, \ldots, k$, and $\varepsilon_k = e^{(2\pi i/k)}$ is the first kth degree root of unit; (4) M has a (strictly) positive eigenvector corresponding to α.*

The mentioned eigenvalue α is referred to as the *Frobenius root* of M. If M is the adjacency matrix of a digraph G, then α is called the *index* of G. Note that α is an algebraic number. A well-known result states that the index of a digraph equals maximum of the indices of its scc's. The scc's of index 0 are singletons and that of index 1 are simple cycles. The indices of other scc's are strictly greater than 1. The number k of eigenvalues with maximum absolute value is called the *imprimitivity number* of the matrix (or of the digraph). For digraphs, this number equals the greatest common divisor of lengths of all cycles. We are ready to state our first result, which classifies combinatorial complexity of regular languages w.r.t. $\bar{\Theta}$-notation.

Theorem 2. *Let a regular language L be recognized by a consistent dfa $\mathcal{A}$. Then $C_L(n) = \bar{\Theta}(n^m\alpha^n)$, where α is the index of $\mathcal{A}$, and $m{+}1$ is the maximum number of scc's of index α connected by a single path in $\mathcal{A}$.*

Now turn to the "finer" description of combinatorial complexity of regular languages. For polynomial languages the following result was proved in [1]:

> *for an arbitrary polynomial context-free language L there exist positive integers N and r, and polynomials $p_0(n), \ldots, p_{r-1}(n)$ with rational coefficients such that for all $n > N$ the equality $C_L(n) = p_{n \bmod r}(n)$ holds.*

The main difference between the polynomial case and the general one is the following. In the polynomial case, any complexity function is *asymptotically equal* to a function constructed of a finite number of given polynomials. In the general case, due to the presence of exponentials, such precision is impossible. Nevertheless, any complexity function still *asymptotically behaves* as a function, "finitely constructed" in a similar way.

We start with the key definition. As usual, $\mathbb{N}_0$ stands for the set of nonnegative integers, $\mathbb{R}^+$ denotes the set of nonnegative real numbers, and "asymptotically" means "for all values of n greater than some constant".

Definition 1. *Let $F(n) : \mathbb{N}_0 \to \mathbb{R}^+$ be an arbitrary function, r be a positive integer. The set $\{f_0(n), \ldots, f_{r-1}(n)\}$ of (non-strictly) increasing functions from $\mathbb{N}_0$ to $\mathbb{R}^+$ is called an asymptotic set for $F(n)$, if for each set $N_i = \{n \in \mathbb{N}_0 \mid n \bmod r = i\}$ either $f_i(n) \equiv 0$ and $F(n)$ is asymptotically zero on the set N_i or $f_i(n) \not\equiv 0$ and there exists a real constant $\gamma_i > 1$ such that the inequality $\left|\frac{F(n)}{f_i(n)} - 1\right| < \gamma_i^{-n}$ holds asymptotically on N_i.*

Each function $f_i(n)$ is called an asymptotic function for $F(n)$ on the set N_i.

We point out that condition (2) is stronger, than the equivalence of F and f_i (on the set N_i). Indeed, we require not only that the ratio $\frac{F(n)}{f_i(n)}$ tends to 1 as n approaches infinity along N_i, but also that it tends to 1 *with exponential rate.* Now our second result can be formulated.

Theorem 3. *For any regular language L there exist a positive integer r, algebraic numbers $\alpha_0, \ldots, \alpha_{r-1} \in [1, \infty)$, and polynomials $p_0(n), \ldots, p_{r-1}(n)$ with algebraic real coefficients such that $\{p_0(n)\alpha_0^n, \ldots, p_{r-1}(n)\alpha_{r-1}^n\}$ is an asymptotic set for the complexity function $C_L(n)$.*

3 Theorem 2 and Its Corollaries

Sketch of the proof. Two lemmas on digraphs are needed.

Lemma 1. *Let s, u and v be three vertices of a digraph. If $P_{su}(n) = \bar{\Theta}(f(n))$, and there exists an (u, v)-path, $P_{sv}(n) = \bar{\Omega}(f(n))$. If, moreover, a (v, u)-path also exists, then $P_{sv}(n) = \bar{\Theta}(f(n))$.*

Lemma 2. *Let G be a non-singleton strongly connected digraph, α be its index, and r be its imprimitivity number. Then $P_{uv}(n) = \bar{\Theta}(\alpha^n)$ for arbitrary vertices u, v of G. Moreover, $P_{uv}(n)$ admits an asymptotic set $\{\mu_0\alpha^n, \ldots, \mu_{r-1}\alpha^n\}$, where $\mu_0, \ldots, \mu_{r-1}$ are nonnegative constants.*

To count paths, we fix a consistent automaton $\mathcal{A}$ recognizing the language L. Let s be the initial vertex of $\mathcal{A}$, and D be the component digraph of $\mathcal{A}$. By the consistency of $\mathcal{A}$, D has a single source, which is the scc containing the initial vertex of $\mathcal{A}$, and each sink of D contains a terminal vertex of $\mathcal{A}$.

Suppose that $P_s(n) = \bar{\Theta}(f(n))$. For any path $(s \to u_1 \to \ldots \to u_r)$ we build its projection. Since D is acyclic, there are only finitely many different projections. Hence, we can show that there exists a *maximal* path d in D with $\bar{\Theta}(f(n))$ paths from s projected onto it. Using Lemma 1, we then get $C_L(n) = \bar{\Theta}(f(n))$.

Thus, it remains to estimate the number of paths from s projected onto d. This can be done by induction on the length of d. The inductive base is provided by Lemma 2. To verify the inductive step, some (not very extensive) calculations are needed.

Remark 1. It follows readily from the proof of Theorem 2, that if two consistent dfa's are isomorphic as unlabeled digraphs, then the combinatorial complexities of languages they recognize both are $\bar{\Theta}(f(n))$ for some function $f(n)$.

Corollary 1. *There exists a function $f(n)$ such that $C_L(n) = \bar{\Theta}(f(n))$ for some context-free language L and $C_R(n) \neq \bar{\Theta}(f(n))$ for any regular language R.*

Proof. Take the context-free language D of all strings of properly placed parentheses (binary Dyck language). We get $C_D(n) = \bar{\Theta}\left(\frac{2^n}{n^{3/2}}\right)$, whence the result.

Thus, the coincidence of combinatorial complexity of context-free and regular languages holds only for polynomial languages.

Now, what can be said about asymptotic behaviour of factorial regular languages? It appears that there are no additional resctrictions.

Proposition 1. *If a regular language L has the complexity $\bar{\Theta}(f(n))$ for some function f, then the languages* $\mathrm{PR}(L)$, $\mathrm{SUF}(L)$, *and* $\mathrm{FACT}(L)$ *have the complexity $\bar{\Theta}(f(n))$ as well.*

Remark 2. In the general case, the gap between $C_L(n)$ and $C_{\mathrm{FACT}(L)}(n)$ can be arbitrarily large. To see this, take an infinite word and consider the language of all its prefixes and the language of all its finite factors.

Proposition 2. *For an integer $m \geq 0$ and an algebraic number $\alpha \geq 1$, a factorial regular language over a k-letter alphabet with complexity $\bar{\Theta}(n^m\alpha^n)$ exists iff there exists a digraph with maximal outdegree k and $\bar{\Theta}(n^m\alpha^n)$ paths of length n.*

4 Theorem 3 and Its Corollaries

Sketch of the proof. Suppose that the language L is recognized by a consistent dfa $\mathcal{A}$ with the adjacency matrix A, and $\chi(\lambda) = \lambda^k + a_{k-1}\lambda^{k-1} + \ldots + a_0$ is the characteristic polynomial of A. By the textbook Hamilton-Cayley theorem, $\chi(A)$ is zero matrix, hence $\mathbf{x}\cdot\chi(A) = \mathbf{0}$ for any vector $\mathbf{x}$. We fix a vector $\mathbf{x}$ and consider the vector function $\mathbf{F}(n) = \mathbf{x}A^n$. This function satisfies the linear homogeneous recurrence relation with constant coefficients

$$\mathbf{F}(n) = -a_{k-1}\mathbf{F}(n{-}1) - \ldots - a_0\mathbf{F}(n{-}k),$$

By the main theorem on such relations, the function $\mathbf{F}(n)$ can be represented in the closed form using the roots of $\chi(\lambda)$. More precisely, let $\lambda_1, \ldots, \lambda_s$ be all eigenvalues of A and $d_1, \ldots, d_s$ be their multiplicities. Then

$$\mathbf{F}(n) = (\mathbf{b}_{11}{+}\mathbf{b}_{12}n{+}\ldots{+}\mathbf{b}_{1d_1}n^{d_1-1})\lambda_1^n + \ldots + (\mathbf{b}_{s1}{+}\ldots{+}\mathbf{b}_{sd_s}n^{d_s-1})\lambda_s^n,$$

where the constant vectors $\mathbf{b}_{11}, \ldots, \mathbf{b}_{sd_s}$ can be uniquely determined.

Next we observe that for $\mathbf{x} = (1, 0, \ldots, 0)$ the function $C_L(n)$ equals the sum of the functions $f_u(n) = (\mathbf{F}(n))_u$ over the set of terminal vertices. By Theorem 2, $C_L(n) = \bar{\Omega}(\alpha^n)$, where α is the index of $\mathcal{A}$. As the number of summands is finite, $f_u(n) = \bar{\Omega}(\alpha^n)$ for some u. By some calculations heavily using Theorem 1 we find an asymptotic set of the required form for $f_u(n)$. Finally we show that if the functions $f_u(n)$ and $f_v(n)$ have such asymptotic sets, so does the function $f_u(n) + f_v(n)$. Thus, the required asymtotic set can be easily obtained.

The following corollaries are used in Sect. 6 to justify a method of calculating the growth rates of arbitrary regular languages. Note that the finiteness of all the β's in Corollary 2 yields the equality of all exponentials in the asymptotic set for $C_L(n)$, as well as the equality of degrees of all polynomials in this set.

Corollary 2. *Let a factorial language L be recognized by a consistent automaton $\mathcal{A}$ with the imprimitivity number k. Then each of the sequences $\left\{\frac{C_L(nk+i+1)}{C_L(nk+i)}\right\}_1^\infty$, where $i = 0, \ldots, k{-}1$, converges to a nonnegative (possibly infinite) limit β_i. If all these limits are finite, then the number $(\beta_0\cdot\ldots\cdot\beta_{k-1})^{1/k}$ is the growth rate of L. In particular, if $k = 1$, then the whole sequence $\left\{\frac{C_L(n+1)}{C_L(n)}\right\}_1^\infty$ converges to the growth rate of L.*

Corollary 3. *Suppose that, in terms of Corollary 2, all the numbers $\beta_0, \ldots, \beta_{k-1}$ are finite. Let d be the degree of all polynomials in the asymptotic set for the complexity function $C_L(n)$. Then*

1) $\left|\frac{C_L(nk+i+1)}{C_L(nk+i)} - \beta_i\right| = O(\gamma^n)$ *for any* $i = 0, \ldots, k-1$ *and some* $\gamma < 1$, *if* $d = 0$*;*

2) $\left|\frac{C_L(nk+i+1)}{C_L(nk+i)} - \beta_i\right| = \Theta(\frac{1}{n})$ *for any* $i = 0, \ldots, k-1$, *if* $d > 0$.

5 Oscillation of Complexity Functions

The complexity functions of languages behave in a very different manner, from straightforward $f(n) = 2^n$ to somewhat complicated and irregular. The "irregularity" is usually expressed by oscillation between relatively high and relatively low values. We are going to study the oscillation property for regular languages on the basis of the results of two previous sections.

There are only a few papers studying the behaviour of complexity functions. They deal with *subword complexity* of infinite words (combinatorial complexity of the language of finite factors of such words). Cassaigne [5] proved that for any infinite word of linear subword complexity, this complexity has bounded variation. Thus, no "big" oscillation occur in this case. On the other hand, Balogh and Bollobas [2] constructed a binary infinite word whose subword complexity oscillates between linear and "almost exponential" (of type $\bar{\Theta}(2^{n/\log n})$) growth.

It is convinient to express the oscillation property of the function f in terms of the ratio $\frac{f(n+1)}{f(n)}$. The function f is called *oscillating* (resp. *non-oscillating*), if the limit $\lim\limits_{n\to\infty} \frac{f(n+1)}{f(n)}$ does not exist (resp. exists). We say that an oscillating function f is *wild*, if $\limsup\limits_{n\to\infty} \frac{f(n+1)}{f(n)} = \infty$ or $\liminf\limits_{n\to\infty} \frac{f(n+1)}{f(n)} = 0$. One can prove that the limit $\lim\limits_{n\to\infty} \frac{f(n+1)}{f(n)}$, if exists, coincides with the growth rate of f.

Theorem 4. *All possible types of combinatorial complexity for arbitrary, prefix-closed, and factorial regular languages w.r.t. oscillation property are listed in the following table (W=wild, O=oscillating, N=non-oscillating):*

Regular languages	*Bounded*	*Polynomial*	*Exponential*
Arbitrary	*W,O,N*	*W,O,N*	*W,O,N*
Prefix-closed	*O,N*	*O,N*	*O,N*
Factorial	*O,N*	*N*	*O,N*

Proof. Except for three non-trivial examples, the statements of the theorem are either obvious or easily follow from Theorem 3. The examples follow.

1. Bounded factorial languages with oscillating complexity. Balogh and Bollobas [2] constructed an infinite sequence $\{B_s\}$ of languages over the binary alphabet such that $C_{B_s}(n)$ oscillates between the constants $(2s-1)$ and s^2. We prove that these languages are regular.

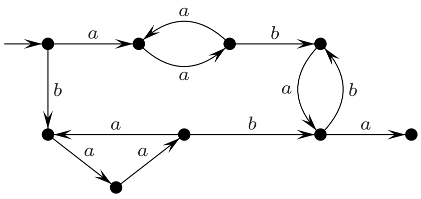


Fig. 1. Consistent dfa recognizing prefix-closed language K. All vertices are terminal.

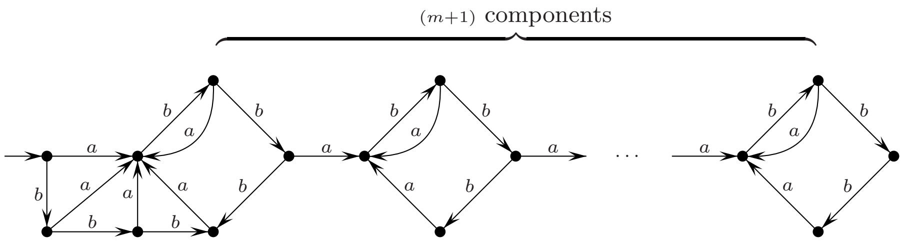


Fig. 2. Consistent dfa recognizing factorial language E_m with oscillating complexity of type $\bar{\Theta}(n^m\alpha^n)$. All vertices are terminal.

2. Polynomial prefix-closed languages with oscillating complexity. Fig. 1 exhibits a dfa recognizing a prefix-closed regular language K such that $C_K(n) = \frac{3}{2}n + O(1)$ for odd n and $C_K(n) = n + O(1)$ for even n.
3. Exponential factorial languages with oscillating complexity. Consider the language $E_m \subseteq \{a,b\}^*$ $(m \geq 0)$ defined by the following rule: any word from E_m has no factors a^2, b^4, and at most m occurences of the factor *abba* (these occurences can overlap). Such a language is obviously factorial, and is recognized by the dfa in Fig. 2. In view of Lemma 2, an imprimitivity number greater than 1 is required for some scc of the automaton as a necessary condition for oscillation. Non-singleton scc's of the given dfa have imprimitivity number 2, and the oscillation of combinatorial complexity can be proved.

Combining Theorems 3 and 4 we get the following important corollary.

Corollary 4. *An asymptotic set for the complexity of a prefix-closed (in particular, factorial) regular language has the form* $\{p_0(n)\alpha^n, \ldots, p_{r-1}(n)\alpha^n\}$, *where the degrees of all polynomials* $p_i(n)$ *coincide.*

6 Application to Upper Bounds

In the last section of this paper we use Corollaries 2 and 3 to obtain an efficient general method of finding upper bounds to the growth rates of factorial languages. The idea to use languages with finite antidictionary for this purpose is by no means new. It is attributed to Brandenburg [3] and was used by many others. We show how this idea can be made much more efficient.

Recall how to use the languages with finite antidictionary to estimate the complexity of a given factorial language $L \subseteq \Sigma^*$ (see [25] for more properties of such estimations). Let M be the antidictionary of L. Consider a family $\{M_i\}$ of finite subsets of M such that

$$M_1 \subseteq M_2 \subseteq \ldots \subseteq M_n \subseteq \ldots \subseteq M, \qquad M_1 \cup M_2 \cup \ldots \cup M_n \cup \ldots = M.$$

(We shall take for M_i the set of all words of M with lengths at most i.) Denote by L_i the factorial language over Σ having the antidictionary M_i. One has

$$L \subseteq \ldots \subseteq L_i \subseteq \ldots \subseteq L_1, \qquad L_1 \cap L_2 \cap \ldots \cap L_n \cap \ldots = L,$$

and for any n, there is i such that $L \cap \Sigma^n = L_i \cap \Sigma^n$. Then

$$C_L(n) = \ldots = C_{L_i}(n) \leq \ldots \leq C_{L_1}(n).$$

Hence the sequence $C_{L_i}(n)$ converges to $C_L(n)$ from above. Thus, we can choose i as big as we can operate with, and estimate the complexity of the language L_i. Since L_i is regular, its growth rate equals the Frobenius root of some nonnegative matrix, and can be calculated in finite time with any fixed precision. But this time can be pretty big, thus preventing us from getting good bounds.

For efficient calculation of the growth rates we provide an algorithm `GRate`. Its idea is simple: we use the automaton $\mathcal{A}$ recognizing a language L to count consequently $C_L(1), \ldots, C_L(N)$ for some N and return $\alpha_N = C_L(N)/C_L(N-1)$ as an approximation of the growth rate (Corollary 2). If $\mathcal{A}$ is strongly connected with the imprimitivity number 1, then we actually follow a well-known iterative method of finding the Frobenius root (see, e.g., [13]), but without explicit use of the matrix. The difference $(\alpha_N - \alpha_{N-1})$ in this case decreases at the same rate as the approximation error (this rate is exponential by Corollary 3(1)), so one can easily regulate the number N of iterations required to achieve the desired precision (some actual values of N in our experiments, taken for $\delta = 10^{-10}$, are shown in Table 1). Furthermore, only a little gadget is needed to operate the case of arbitrary imprimitivity number. Splitting $\mathcal{A}$ into scc's and processing them separately we satisfy the first condition. Finally we get

Theorem 5. *The growth rate of a regular language given by a consistent dfa $\mathcal{A}$ can be calculated with any precision δ, $0 < \delta < 1$, in time $\Theta(-\log \delta \cdot |\mathcal{A}|)$ using $\Theta(|\mathcal{A}|)$ additional space.*

We have implemented algorithm `GRate` to estimate the growth rates of *languages of bounded exponent.* To introduce this well-known in combinatorics of words class of languages we need a few definitions. If a word $w \in \Sigma^*$ is viewed as a function $\{1, \ldots, n\} \to \Sigma$, then a *period* of w is any period of this function. The *exponent* of the word w is the ratio of the length of w to its least period. A factorial language L is said to be of bounded exponent, if the exponents of all its words are bounded from above by some constant.

The word is *β-free*, if the exponents of all its factors are strictly less than the number β, and *β^+-free*, if these exponents are less than or equal to β. By β-free (β^+-free) languages we mean the languages of *all* β-free (respectively β^+-free) words over some alphabet. Such languages are the most intensively studied languages of bounded exponent. The study of the binary 3-free (cube-free) and ternary 2-free (square-free) languages can be traced back to the celebrated work of Thue [26], who prove their infiniteness. The first estimations of growth rates of these languages were made by Brandenburg [3] and then the bounds with increased precision were given in a series of papers, see [11,22,21], for example. The binary 2^+-free (overlap-free) language was also introduced by Thue and has appeared in numerous works since then. The binary $(7/3)^+$-free language recently attracted attention as it lies on the borderline between polynomial and exponential β-free languages in the binary case [17]. And this list would not be complete without an infinite series of *threshold* languages, first studied by Dejean [10]. The threshold language for a k-letter alphabet is the minimal infinite β- or β^+-free language for this alphabet. It is known that binary 2^+-free, ternary $(7/4)^+$-free, and 4-ary $(7/5)^+$-free languages are threshold ones. For $k \geq 5$ Dejean conjectured in 1972 that the k-ary $(k/(k{-}1))^+$-free language is threshold; this conjecture has been proved in most but not all cases, see, e.g. [4].

For a β-free (or β^+-free) language L over a k-letter alphabet we construct a finite antidictionary M_i by some optimized search, build a consistent dfa $\mathcal{A}_i$ for the language L_i applying the algorithm of [8] to M_i, and then apply `GRate` to $\mathcal{A}_i$, getting the approximated growth rate of L. We also note that the symmetry of L allows to modify the mentioned algorithm of [8] such that the size of the obtained automaton will be approximately $k!$ times less than the size of $\mathcal{A}_i$.

Finally, we present Table 1 with our main numerical results on upper bounds for growth rates of β-free and β^+-free languages. Most of the obtained bounds are quite close to the growth rates of target languages. This conclusion is justified by the fast convergence of growth rates of approximating languages (see the "Delta" column) and confirmed by some independent studies. For example, Richard and Grimm [22] used the method of differential approximants to suggest 1,301762 as the growth rate for ternary 2-free language with the admissible error $\pm\, 2{\cdot}10^{-6}$, and our last approximation is well inside the range.

From Table 1 some conclusions can be made about the behaviour of the growth rate as a function in β over a fixed alphabet. "Big" jumps of the growth rate appear between β-free and β^+-free languages for a rational β with small denominator. This jumps are due to "allowance" of short factors with the exponent β (like the factor a^3 when we move from 3-free to 3^+-free binary language). In the binary case, Table 1 gives a very representative picture of this property. The point of the first big jump of the growth rate is of certain interest.

Observation 1. *For $k = 3, 4, 5$ growth rate jumps by more than a unit in the point $\beta_k = (k{-}1)/(k{-}2)$.*

This effect is certainly caused by the following fact. For $(k{-}1)/(k{-}2)$-free words, any consecutive $(k{-}1)$ letters should be distinct, thus leaving only one "degree of freedom" for a letter in the word: it either is distinct from $(k{-}1)$ previous letters, or coincide with the first of them. When the exponent $(k{-}1)/(k{-}2)$ is allowed, a letter gets a second degree of freedom, since it may coincide also with the second of the previous $(k{-}1)$ letters. We conjecture that this "jumping" property holds for all $k \geq 3$.

Table 1. Upper bounds for growth rates. The columns from left to right contain: size of the alphabet, exponent, the best previously known bound; then the parameters of our approximation follow: maximum length of forbidden words, number of words in the antidictionary, number of vertices in the "reduced" dfa, number of iterations needed for the precision 10^{-10}, the growth rate, and the variation "Delta" of the obtained growth rate w.r.t. the growth rate of the approximation by a dfa of size $\approx |\mathcal{A}_i|/2k!$.

k	β	Previous	i	$\lvert M_i\rvert$	$\lvert\mathcal{A}_i\rvert/k!$	N	Growth rate	Delta
2	$(7/3)^+$	1,2299 [17]	127	269684	9195979	210	1,22064487	$7{\cdot}10^{-8}$
2	$5/2$	-	115	70066	2224353	200	1,22950218	$1{\cdot}10^{-7}$
2	$(5/2)^+$	-	93	387068	10069765	115	1,366301114	$7{\cdot}10^{-9}$
2	$8/3$	-	83	73016	1725995	115	1,3762704	$2{\cdot}10^{-8}$
2	$(8/3)^+$	-	75	108500	2328649	105	1,45086113	$1{\cdot}10^{-8}$
2	3	1,4576 [22]	93	375576	10676055	100	1,45757728694	$5{\cdot}10^{-11}$
2	3^+	-	49	26094	384650	50	1,795126410	$3{\cdot}10^{-9}$
3	$(7/4)^+$	-	99	1224483	9161729	210	1,2456148	$4{\cdot}10^{-6}$
3	2	1,3017886 [21]	96	1377033	10970070	160	1,30176215	$3{\cdot}10^{-7}$
3	2^+	-	29	1086837	2564335	40	2,6058795	$6{\cdot}10^{-7}$
4	$(7/5)^+$	-	191	347080	1105491	2500	1,069721	$2{\cdot}10^{-4}$
4	$(3/2)$	-	150	125440	358485	900	1,09697	$2{\cdot}10^{-4}$
4	$(3/2)^+$	-	28	5594032	2379474	50	2,280727	$8{\cdot}10^{-5}$
5	$(5/4)^+$	-	99	1087765	365963	210	1,158901	$1{\cdot}10^{-4}$
5	$(4/3)$	-	83	138565	42368	200	1,16508	$1{\cdot}10^{-3}$
5	$(4/3)^+$	-	27	27777685	1920804	50	2,250161	$5{\cdot}10^{-4}$
6	$(6/5)^+$	-	67	1012116	38796	200	1,225441	$2{\cdot}10^{-4}$

We conclude with another interesting result of our experiments with approximations of β-free and β^+-free languages. This result is not numerical.

Observation 2. *Some approximating languages are oscillating. Namely, polynomial binary β-free or β^+-free languages have oscillating approximations.*

The discovered oscillations were predictable. In [24], we gave a description of all approximating languages L_i for the famous Thue-Morse language, which forms a proper subset of the 2^+-free binary language. A precise form of the non-singleton component in the automaton for any language L_i was found. This component has big imprimitivity number, and hence, a "strong potential" for oscillation. On the other hand, if a binary word w of exponent less than $(7/3)$ can be extended to both sides to an arbitrarily long word, the exponent of which is also less than $(7/3)$, then w must belong to the Thue-Morse language [23]. Hence, the approximating languages for the binary languages from 2^+-free to $(7/3)$-free are expected to behave in a similar way as the approximating languages for Thue-Morse. For the first few approximations, the non-singleton scc's are the same for Thue-Morse, 2^+-free, and $(7/3)$-free languages. By a careful analysis, this result probably can be extended to all approximations of these languages.

References

1. D'Alessandro, F., Intrigila, B., Varricchio, S.: On the structure of counting function of sparse context-free languages. Theor. Comp. Sci. 356, 104–117 (2006)
2. Balogh, J., Bollobas, B.: Hereditary properties of words. RAIRO – Inf. Theor. Appl. 39, 49–65 (2005)
3. Brandenburg, F.-J.: Uniformly growing k-th power free homomorphisms. Theor. Comput. Sci. 23, 69–82 (1983)
4. Carpi, A.: On the repetition threshold for large alphabets. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 226–237. Springer, Heidelberg (2006)
5. Cassaigne, J.: Special factors of sequences with linear subword complexity. In: Dassow, J., Rozenberg, G., Salomaa, A. (eds.) Developments in Language Theory, II, pp. 25–34. World Scientific, Singapore (1996)
6. Choffrut, C., Karhumäki, J.: Combinatorics of words, ch. 6. In: Rosenberg, G., Salomaa, A. (eds.) Handbook of formal languages, vol. 1, pp. 329–438. Springer, Berlin (1997)
7. Chomsky, N., Miller, G.A.: Finite state languages. Inf. and Control 1(2), 91–112 (1958)
8. Crochemore, M., Mignosi, F., Restivo, A.: Automata and forbidden words. Inform. Processing Letters 67(3), 111–117 (1998)
9. Cvetković, D.M., Doob, M., Sachs, H.: Spectra of graphs. Theory and applications, 3rd edn. Johann Ambrosius Barth, Heidelberg (1995)
10. Dejean, F.: Sur un Theoreme de Thue. J. Comb. Theory, Ser. A 13(1), 90–99 (1972)
11. Edlin, A.: The number of binary cube-free words of length up to 47 and their numerical analysis. J. Diff. Eq. and Appl. 5, 153–154 (1999)
12. Ehrenfeucht, A., Rozenberg, G.: On subword complexities of homomorphic images of languages. RAIRO Inform. Theor. 16, 303–316 (1982)
13. Franklin, J.N.: Matrix theory. Prentice-Hall Inc., Englewood Cliffs NJ (1968)
14. Gantmacher, F.R.: Application of the theory of matrices. Interscience, New York (1959)
15. Govorov, V.E.: Graded algebras. Mat. Zametki 12, 197–204 (1972) (Russian)
16. Ibarra, O., Ravikumar, B.: On sparseness, ambiguity and other decision problems for acceptors and transducers. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 1986. LNCS, vol. 210, pp. 171–179. Springer, Heidelberg (1986)
17. Karhumäki, J., Shallit, J.: Polynomial versus exponential growth in repetition-free binary words. J. Combin. Theory. Ser. A 105, 335–347 (2004)
18. Lothaire, M.: Combinatorics on words. Addison-Wesley, Reading (1983)
19. Milnor, J.: Growth of finitely generated solvable groups. J. Diff. Geom. 2, 447–450 (1968)
20. Morse, M., Hedlund, G.A.: Symbolic dynamics. Amer. J. Math. 60, 815–866 (1938)
21. Ochem, P., Reix, T.: Upper bound on the number of ternary square-free words. In: Electronic proceedings of Workshop on words and automata (WOWA 2006), S.-Petersburg, p. 8 (2006)
22. Richard, C., Grimm, U.: On the entropy and letter frequencies of ternary square-free words. Electronic J. Combinatorics 11(1), p. 14 (2004)
23. Shur, A.M.: The structure of the set of cube-free Z-words in a two-letter alphabet. Izv. Ross. Akad. Nauk Ser. Mat. 64, 201–224 (2000) (Russian); English translation in Izv. Math. 64, 847–871 (2000)

24. Shur, A.M.: Combinatorial complexity of rational languages. Discr. Anal. and Oper. Research, Ser. 1, 12(2), 78–99 (2005)
25. Shur, A.M.: Rational approximations of polynomial factorial languages. Int. J. Found. Comput. Sci. 18, 655–665 (2007)
26. Thue, A.: Über unendliche Zeichenreihen. Kra. Vidensk. Selsk. Skrifter. I. Mat.-Nat. Kl., Christiana 7, 1–22 (1906)
27. Trofimov, V.I.: Growth functions of some classes of languages. Cybernetics 6, 9–12 (1981) (Russian)
28. Trofimov, V.I.: Growth functions of finitely generated semigroups. Semigroup Forum 21, 351–360 (1980)
29. Ufnarovsky, V.A.: On the growth of algebras. Proceedings of Moscow University 1(4), 59–65 (1978) (Russian)

On Sequences with Non-learnable Subsequences

Vladimir V. V'yugin

Institute for Information Transmission Problems, Russian Academy of Sciences,
Bol'shoi Karetnyi per. 19, Moscow GSP-4, 127994, Russia
vyugin@iitp.ru

Abstract. The remarkable results of Foster and Vohra was a starting point for a series of papers which show that any sequence of outcomes can be learned (with no prior knowledge) using some universal randomized forecasting algorithm and forecast-dependent checking rules. We show that for the class of all computationally efficient outcome-forecast-based checking rules, this property is violated. Moreover, we present a probabilistic algorithm generating with probability close to one a sequence with a subsequence which simultaneously miscalibrates all partially weakly computable randomized forecasting algorithms.

According to the Dawid's prequential framework we consider partial recursive randomized algorithms.

1 Introduction

Let a binary sequence $\omega_1, \omega_2, \ldots, \omega_{n-1}$ of outcomes is observed by a forecaster whose task is to give a probability p_n of a future event $\omega_n = 1$. The evaluation of probability forecasts is based on a method called *calibration*: informally, following Dawid [1] forecaster is said to be well-calibrated if for any p^* the event $\omega_n = 1$ holds in $100p^*\%$ of moments of time as he choose $p_n \approx p^*$. (see also [2]).

Let us give some notations. Let Ω be the set of all infinite binary sequences, Ξ be the set of all finite binary sequences and λ be the empty sequence. For any finite or an infinite sequence $\omega = \omega_1 \ldots \omega_n \ldots$, we write $\omega^n = \omega_1 \ldots \omega_n$ (we put $\omega_0 = \omega^0 = \lambda$). Also, $l(\omega^n) = n$ denotes the length of the sequence ω^n. If x is a finite sequence and ω is a finite or infinite sequence then $x\omega$ denotes the concatenation of these sequences, $x \sqsubseteq \omega$ means that $x = \omega^n$ for some n.

In the measure-theoretic framework we expect that the forecaster has a method for assigning probabilities p_n of a future event $\omega_n = 1$ for all possible finite sequences $\omega_1, \omega_2, \ldots, \omega_{n-1}$. In other words, all conditional probabilities

$$p_n = P(\omega_n = 1 | \omega_1, \omega_2, \ldots, \omega_{n-1})$$

must be specified and the overall probability distribution in the space Ω of all infinite binary sequences will be defined. But in reality, we should recognize that we have only individual sequence $\omega_1, \omega_2, \ldots, \omega_{n-1}$ of events and that the corresponding forecasts p_n whose testing is considered may fall short of defining a full probability distribution in the whole space Ω. This is the point of *the prequential principle* proposed by Dawid [1]. This principle says that the evaluation of a probability forecaster should depend only on his actual probability forecasts and the corresponding outcomes. The additional information contained in a probability measure that has these probability forecasts as conditional probabilities should not enter in the evaluation. According to Dawid's prequential framework we do not consider numbers p_n as conditional probabilities generated by some overall probability distribution defined for all possible events. In such a way, *a deterministic forecasting system* is *a partial recursive* function $f : \Xi \rightarrow [0, 1]$. We suppose that a valid forecasting system f is defined on all finite initial fragments $\omega_1, \dots, \omega_{n-1}, \dots$ of an analyzed individual sequence of outcomes.

First examples of individual sequences for which well-calibrated deterministic forecasting is impossible (non-calibrable sequences) were presented by Oakes [6] (see also Shervish [9]). Unfortunately, the methods used in these papers, and in Dawid [1], [2], do not comply with prequential principle; they depend on some mild assumptions about the measure from which probability forecasts are derived as conditional probabilities. The method of generation the non-calibrable sequences with probability arbitrary close to one presented in V'yugin [11] also is based on the same assumptions. In this paper we modify construction from [11] for the case of partial deterministic and randomized forecasting systems do not corresponding to any overall probability distributions.

Oakes [6] showed that any everywhere defined forecasting system f is not calibrated for a sequence $\omega = \omega_1\omega_2 \dots$ defined

$$\omega_i = \begin{cases} 1 \text{ if } p_i < 0.5 \\ 0 \text{ otherwise} \end{cases}$$

and $p_i = f(\omega_1 \dots \omega_{i-1})$, $i = 1, 2, \dots$.

Foster and Vohra [3] showed that the well-calibrated forecasts are possible if these forecasts are randomized. By *a randomized* forecasting system they mean a random variable $f(\alpha; x)$ defined on some probability space Ω_x supplied by some probability distribution Pr_x, where $x \in \Xi$ is a parameter. As usual, we omit the argument α. For any infinite ω, these probability distributions $Pr_{\omega^{i-1}}$ generate the *overall probability distribution* Pr on the direct product of probability spaces $\Omega_{\omega^{i-1}}$, $i = 1, 2, \dots$.

It was shown in [3], [4] that any sequence can be learned: for any $\Delta > 0$, a universal randomized forecasting system f was constructed such that for any sequence $\omega = \omega_1\omega_2 \dots$ the overall probability Pr of the event

$$\left| \frac{1}{n} \sum_{i=1}^{n} I(\tilde{p}_i)(\omega_i - \tilde{p}_i) \right| \leq \Delta \tag{1}$$

tends to one as $n \rightarrow \infty$, where $\tilde{p}_i = f(\omega^{n-1})$ is the random variable, $I(p)$ is the characteristic function of an arbitrary subinterval of $[0, 1]$; we call this function a *forecast-based* checking rule.

Lehrer [5] and Sandrony et al. [8] extended the class of checking rules to combination of *forecast-* and *outcome-based* checking rules: a checking rule is a function $c(\omega^{i-1}, p) = \delta(\omega^{i-1})I(p)$, where $\delta : \Xi \rightarrow \{0, 1\}$ is an outcome-based checking rule, and $I(p)$ is a characteristic function of a subinterval of $[0, 1]$. They also considered a more general class of randomized forecasting systems - random variables $\tilde{p}_i = f(\alpha; \omega^{i-1}, p^{i-1})$, where $p^{i-1} = p_1, \ldots, p_{i-1}$ is the sequence of past realized forecasts.

For $k = 1, 2, \ldots$, let $\{\delta_k\}$ be any sequence of outcome-based checkng rules and $\{I_k\}$ be any sequence of characteristic functions of subintervals of $[0, 1]$. Sandrony et al. [8] defined a randomized universal forecasting system which calibrates all checking rules $\{\delta_k I_k\}$, $k = 1, 2, \ldots$, i.e., such that for any $\Delta > 0$ and for any sequence $\omega = \omega_1\omega_2\ldots$, the overall probability of the event (1) tends to one as $n \to \infty$, where $\tilde{p}_i = f(\omega^{n-1}, p^{i-1})$ and $I(\tilde{p}_i)$ is replaced on $\delta_k(\omega^{i-1})I_k(\tilde{p}_i)$ for all $k = 1, 2, \ldots$.

In this paper we consider the class of all computable (partial recursive) outcome-based checking rules $\{\delta_k\}$ and a slightly different class of randomized forecasting systems: our forecasting systems are random variables $\tilde{p}_i = f(\alpha; \omega^{i-1})$ do not depending on past realized forecasts (this take a place for the universal forecasting systems defined in [3] and [10] [1]). Concurrently, such a function can be undefined outside ω, it requires that any well defined forecasting system must be defined on all initial fragments of an analyzed sequence of outcomes. This peculiarity is important, since we consider forecasting systems possessing some computational properties: there is an algorithm computing the probability distribution function of such forecasting system. This algorithm when fed to some input can never finish its work, and so, is undefined on this input.

In this case, a universal randomized forecasting algorithm which calibrates all computationally efficient outcome-forecast-based checking rules does not exist. Moreover, we construct *a probabilistic generator* (or probabilistic algorithm) of *non-learnable* (in this way) sequences. This generator outputs with probability close to one an infinite sequence such that for each randomized forecasting system $\tilde{p}_i = f(\alpha; \omega^{i-1})$ some computable outcome-based checking rule δ selects an infinite subsequence of ω on which the property (1) fails for some characteristic function I with the overall probability one, where the overall probability is associated with the forecasting system f.

2 Miscalibrating the Forecasts

We use standard notions of the theory of algorithms. This theory is systematically treated in, for example, Rogers [7]. We fix some effective one-to-one enumeration of all pairs (triples, and so on) of nonnegative integer numbers. We identify any pair (t, s) and its number $\langle t, s\rangle$; let $p(\langle t, s\rangle) = t$.

A function ϕ: $A \to \mathcal{R}$ is called (lower) semicomputable if $\{(r, x) : r < \phi(x)\}$ (r is a rational number) is a recursively enumerable set. A function ϕ is upper semicomputable if $-\phi$ is lower semicomputable. Standard argument based on

[1] Note that the algorithm from [8] can be modified in a fashion of [3], i.e., such that at any step of the construction past forecasts can be replaced on measures with finite supports defined on previous steps. Since these measures are defined recursively in the process of the construction, they can be eliminated from the condition of the universal forecasting algorithm.

the recursion theory shows that there exist the lower and upper semicomputable real functions $\phi^-(j,x)$ and $\phi^+(k,x)$ universal for all lower semicomputable and upper semicomputable functions from $x \in \Xi$; in particular every computable real function $\phi(x)$ can be represented as $\phi(x) = \phi^-(j,x) = \phi^+(k,x)$ for all x, for some j and k. Let $\phi^-_s(j,x)$ be equal to the maximal rational number r such that the triple (r,j,x) is enumerated in s steps in the process of enumerating of the set $\{(r,j,x) : r < \phi(j,x),\ r \text{ is rational}\}$ and equals $-\infty$, otherwise. Any such function $\phi^-_s(j,x)$ takes only finite number of rational values distinct from $-\infty$. By definition, $\phi^-_s(j,x) \le \phi^-_{s+1}(j,x)$ for all j,s,x, and $\phi^-(j,x) = \lim\limits_{s\to\infty} \phi^-_s(j,x)$. An analogous non-increasing sequence of functions $\phi^+_s(k,x)$ exists for any upper semicomputable function.

Let $i = \langle t,k\rangle$. We say that a real function $\phi_i(x)$ is *defined on* x if given any degree of precision - positive rational number $\kappa > 0$, it holds $|\phi^+_s(t,x) - \phi^-_s(k,x)| \le \kappa$ for some s; $\phi_i(x)$ undefined, otherwise. If any such s exists then for minimal such s, $\phi_{i,\kappa}(x) = \phi^-_s(k,x)$ is called the rational approximation (from below) of $\phi_i(x)$ up to κ; $\phi_{i,\kappa}(x)$ undefined, otherwise.

To define a measure P on Ω, we define values $P(z) = P(\Gamma_z)$ for all intervals $\Gamma_z = \{\omega \in \Omega : z \sqsubseteq \omega\}$, where $z \in \Xi$, and extend this function on all Borel subsets of Ω in a standard way.

We use also a concept of *computable operation* on $\Xi \bigcup \Omega$ (see [12]). Let $\hat{F}$ be a recursively enumerable set of ordered pairs of finite sequences satisfying the following properties: (i) $(x,\lambda) \in \hat{F}$ for each x; (ii) if $(x,y) \in \hat{F}$, $(x',y') \in \hat{F}$ and $x \sqsubseteq x'$ then $y \sqsubseteq y'$ or $y' \sqsubseteq y$ for all finite binary sequences x,x',y,y'. A computable operation F is defined as follows

$$F(\omega) = \sup\{y \mid x \sqsubseteq \omega \text{ and } (x,y) \in \hat{F} \text{ for some } x\},$$

where $\omega \in \Omega \bigcup \Xi$ and sup is in the sense of the partial order $\sqsubseteq$ on Ξ.

A probabilistic algorithm is a pair (L,F), where $L(x) = L(\Gamma_x) = 2^{-l(x)}$ is the uniform measure on Ω and F is a computable operation. For any probabilistic algorithm (L,F) and a set $A \subseteq \Omega$, we consider the probability $L\{\omega : F(\omega) \in A\}$ of generating by means of F a sequence from A given a uniformly distributed sequence ω.

A partial randomized forecasting system f is *weakly computable* if its *weak probability distribution function* $\varphi_n(\omega^{n-1}) = Pr_n\{f(\omega^{n-1}) < \frac{1}{2}\}$ is a partial recursive function from ω^{n-1}.

Any function $\delta : \Xi \to \{0,1\}$ is called an outcome-based selection (or checking) rule. For any sequence $\omega = \omega_1\omega_2\ldots$, the selection rule δ selects a sequence of indices n_i such that $\delta(\omega^{n_i-1}) = 1$, $i = 1,2,\ldots$, and the corresponding subsequence $\omega_{n_1}\omega_{n_2}\ldots$ of ω.

The following theorem is the main result of this paper. In particular, it shows that the construction of the universal forecasting algorithm from Sandrony et al. [8] is computationally non-efficient in a case when the class of all partial recursive outcome-based checking rules $\{\delta_k\}$ is used.

Theorem 1. *For any $\epsilon > 0$ a probabilistic algorithm (L,F) can be constructed, which with probability $\ge 1-\epsilon$ outputs an infinite binary sequence $\omega = \omega_1\omega_2\ldots$*

such that for every partial weakly computable randomized forecasting system f defined on all initial fragments of the sequence ω there exists a computable selection rule δ defined on all these fragments and such that for $\nu = 0$ or for $\nu = 1$ the overall probability of the event

$$\limsup_{n\to\infty} \left| \frac{1}{n} \sum_{i=1}^{n} \delta(\omega^{i-1}) I_\nu(\tilde{p}_i)(\omega_i - \tilde{p}_i) \right| \geq 1/16 \qquad (2)$$

equals one, where I_0 and I_1 are the characteristic functions of the intervals $[0, \frac{1}{2})$ and $[\frac{1}{2}, 1]$, $\tilde{p}_i = f(\omega^{i-1})$ is a random variable, $i = 1, 2, \ldots$, and the overall probability distribution is associated with f.

Proof. For any probabilistic algorithm (L, F), we consider the function

$$Q(x) = L\{\omega : x \sqsubseteq F(\omega)\}. \qquad (3)$$

It is easy to verify that this function is lower semicomputable and satisfies: $Q(\lambda) \leq 1$; $Q(x0) + Q(x1) \leq Q(x)$ for all x. Any function satisfying these properties is called semicomputable semimeasure. For any semicomputable semimeasure Q a probabilistic algorithm (L, F) exists such that (3) holds. Though the semimeasure Q is not a measure, we consider the corresponding measure on the set Ω

$$\bar{Q}(\Gamma_x) = \inf_n \sum_{l(y)=n, x\sqsubseteq y} Q(y).$$

We will construct a semicomputable semimeasure Q as a some sort of network flow. We define an infinite network on the base of the infinite binary tree. Any $x \in \Xi$ defines two edges $(x, x0)$ and $(x, x1)$ of length one. In the construction below we will mount to the network extra edges (x, y) of length > 1, where $x, y \in \Xi$, $x \sqsubseteq y$ and $y \neq x0, x1$. By the length of the edge (x, y) we mean the number $l(y) - l(x)$. For any edge $\sigma = (x, y)$ we denote by $\sigma_1 = x$ its starting vertex and by $\sigma_2 = y$ its terminal vertex. A computable function $q(\sigma)$ defined on all edges of length one and on all extra edges and taking rational values is called *a network* if for all $x \in \Xi$

$$\sum_{\sigma:\sigma_1=x} q(\sigma) \leq 1.$$

Let G be the set of all extra edges of the network q (it is a part of the domain of q). By *q-flow* we mean the minimal semimeasure P such that $P \geq R$, where the function R is defined by the following recursive equations $R(\lambda) = 1$ and

$$R(y) = \sum_{\sigma:\sigma_2=y} q(\sigma) R(\sigma_1) \qquad (4)$$

for $y \neq \lambda$. A network q is called *elementary* if the set of extra edges is finite and $q(\sigma) = 1/2$ for almost all edges of unit length. For any network q, we define the *network flow delay* function (q-delay function)

$$d(x) = 1 - q(x, x0) - q(x, x1).$$

The construction below works with all computable real functions $\phi_t(x)$, $x \in \Xi$, $t = 1, 2, \ldots$. We suppose that for any computable function ϕ there exist infinitely many programs t such that $\phi_t = \phi$. [2] Any pair $i = \langle t, s \rangle$ is considered as a program for computing the rational approximation $\phi_{t,\kappa_s}(\omega^{n-1})$ of ϕ_t from below up to $\kappa_s = 1/s$.

By the construction below we visit any function ϕ_t on infinitely many steps n. To do this, we use the function $p(n)$: for any positive integer number i we have $p(n) = i$ for infinitely many n.

Let β be a finite sequence and $1 \le k < l(\beta)$. A bit β_k of the sequence β is called *hardly predictable* by a program $i = \langle t, s \rangle$ if $\phi_{t,\kappa_s}(\beta^{k-1})$ is defined and

$$\beta_k = \begin{cases} 0 \text{ if } \phi_{t,\kappa_s}(\beta^{k-1}) \ge \frac{1}{2} \\ 1 \text{ otherwise} \end{cases}$$

Lemma 1. *Let $i = \langle t, s \rangle$ be a program and μ be an arbitrary sufficiently small positive real number. Then for any binary sequence x of length n the portion of all sequences γ of length $K = \lceil (2+\mu)i \rceil n$ (in the set of all finite sequences of length K) such that*

1) $\phi_{t,\kappa_s}(x\gamma^k)$ is defined for all $0 \le k < K$,

2) the number of hardly predictable bits of γ by the forecasting program i is less than in,

is $\le 2^{-2\mu^2 in + O(\log(in))}$ for all sufficiently large n.

Proof. Any function $\sigma(x)$, where $x \in \Xi$ and $\sigma(x) \in \{A, B\}$, is called *labelling* if $\sigma(x0) \ne \sigma(x1)$ for all $x \in \Xi$. For any γ of length K and for any k such that $1 \le k < K$, define $\sigma(\gamma^{k+1}) = A$ and $\sigma(\gamma^k \bar{\gamma}_{k+1}) = B$ if the bit γ_{k+1} of the sequence $x\gamma$ is hardly predictable, where we denote $\bar{\theta} = 1 - \theta$ for any binary bit θ. Since $\phi_{t,\kappa_s}(x\gamma^k)$ is defined for all $0 \le k < K$, then $\sigma(\gamma^{k+1})$ is also defined for all these k. This partial labelling σ can be easily extended on the set of all binary sequences of length K in many different ways. We fix some such extension. Then the total number of all γ satisfying 1)-2) does not exceed the total number of all binary sequences of length K with $\le in$ labels A. Therefore, for all sufficiently large n, the portion of these γ does not exceed

$$\sum_{i \le in} \binom{K}{i} 2^{-K} \le 2^{-(1-H(1/2-\mu))2in + O(\log(in))} \le 2^{-2\mu^2 in + O(\log(in))},$$

where $H(r) = -r \log r - (1-r)\log(1-r)$. □

In the following we put $\mu = 1/\log(i+1)$.

We define an auxiliary relation $B(i, q^{n-1}, \sigma, n)$ and a function $\beta(x, q^{n-1}, n)$. Let $x, \beta \in \Xi$. The value of $B(i, q^{n-1}, (x, \beta), n)$ is *true* if the following conditions hold:

- $n \ge (1 + \lceil (2 + \log^{-1}(i+1))i \rceil) l(x)$;
- $l(\beta) = n$ and $x \sqsubseteq \beta$;

[2] To obtain this property, we can replace the sequence $\phi_t(x)$ on a sequence $\phi'_{\langle t,s \rangle}(x) = \phi_t(x)$ for all s.

- $d^{n-1}(\beta^j) < 1$ for all j such that $1 \le j < n$;
- for all j, $l(x) < j \le (1 + \lceil (2 + \log^{-1}(i+1))i \rceil) l(x)$, the value $\phi_{t,\kappa_s}(\beta^{j-1})$ is computed in $\le n$ steps, and for at least $il(x)$ of these j the bit β_j is hardly predictable by the program $i = \langle t, s \rangle$.

The value of $B(i, q^{n-1}, (x, \beta), n)$ is *false*, otherwise. Define

$$\beta(x, q^{n-1}, n) = \min\{y : p(l(y)) = p(l(x)), B(p(l(x)), q^{n-1}, (x, y), n)\}.$$

Here min is considered for lexicographical ordering of strings; we suppose that $\min \emptyset$ is undefined.

Construction. Let $\rho(n) = (n + n_0)^2$ for some sufficiently large n_0 (the value n_0 will be specified below in the proof of Lemma 5).

Using the mathematical induction by n, we define a sequence q^n of elementary networks. Put $q^0(\sigma) = 1/2$ for all edges σ of length one.

Let $n > 0$ and a network q^{n-1} is defined. Let d^{n-1} be the q^{n-1}-delay function and let G^{n-1} be the set of all extra edges. We suppose also that $l(\sigma_2) < n$ for all $\sigma \in G^{n-1}$.

Let us define a network q^n. At first, we define a network flow delay function d^n and a set G^n. The construction can be split up into two cases.

Let $w(i, q^{n-1})$ be equal to the minimal m such that $p(m) = i$ and $m > l(\sigma_2)$ for each extra edge $\sigma \in G^{n-1}$ such that $p(l(\sigma_1))) < i$.

The inequality $w(i, q^m) \ne w(i, q^{m-1})$ can be induced by some task $j < i$ that mounts an extra edge $\sigma = (x, y)$ such that $l(x) > w(i, q^{m-1})$ and $p(l(x)) = p(l(y)) = j$. Lemma 2 (below) will show that this can happen only at finitely many steps of the construction.

Case 1. $w(p(n), q^{n-1}) = n$ (the goal of this part is to start a new task $i = p(n)$ or to restart the existing task $i = p(n)$ if it was destroyed by some task $j < i$ at some preceding step).

Put $d^n(y) = 1/\rho(n)$ for $l(y) = n$ and define $d^n(y) = d^{n-1}(y)$ for all other y. Put also $G^n = G^{n-1}$.

Case 2. $w(p(n), q^{n-1}) < n$ (the goal of this part is to process the task $i = p(n)$). Let C_n be the set of all x such that $w(i, q^{n-1}) \le l(x) < n$, $0 < d^{n-1}(x) < 1$, the function $\beta(x, q^{n-1}, n)$ is defined [3] and there is no extra edge $\sigma \in G^{n-1}$ such that $\sigma_1 = x$.

In this case for each $x \in C_n$ define $d^n(\beta(x, q^{n-1}, n)) = 0$, and for all other y of length n such that $x \sqsubset y$ define

$$d^n(y) = \frac{d^{n-1}(x)}{1 - d^{n-1}(x)}.$$

Define $d^n(y) = d^{n-1}(y)$ for all other y. We add an extra edge to G^{n-1}, namely, define

$$G^n = G^{n-1} \cup \{(x, \beta(x, q^{n-1}, n)) : x \in C_n\}.$$

[3] In particular, $p(l(x)) = i$ and $l(\beta(x, q^{n-1}, n)) = n$.

We say that the task $i = p(n)$ *mounts* the extra edge $(x, \beta(x, q^{n-1}, n))$ to the network and that all existing tasks $j > i$ are destroyed by the task i.

After Case 1 and Case 2, define for any edge σ of unit length

$$q^n(\sigma) = \frac{1}{2}(1 - d^n(\sigma_1))$$

and $q^n(\sigma) = d^n(\sigma_1)$ for each extra edge $\sigma \in G^n$.

Case 3. Cases 1 and 2 do not hold. Define $d^n = d^{n-1}$, $q^n = q^{n-1}$, $G^n = G^{n-1}$.

As the result of the construction we define the network $q = \lim\limits_{n\to\infty} q^n$, the network flow delay function $d = \lim\limits_{n\to\infty} d^n$ and the set of extra edges $G = \cup_n G^n$.

The functions q and d are computable and the set G is recursive by their definitions. Let Q denotes the q-flow.

The following lemma shows that any task can mount new extra edges only at finite number of steps. Let $G(i)$ be the set of all extra edges mounted by the task i, $w(i, q) = \lim_{n\to\infty} w(i, q^n)$.

Lemma 2. *The set $G(i)$ is finite, $w(i, q)$ exists and $w(i, q) < \infty$ for all i.*

Proof. Note that if $G(j)$ is finite for all $j < i$, then $w(i, q) < \infty$. Hence, we must prove that the set $G(i)$ is finite for any i. Suppose that the opposite assertion holds. Let i be the minimal such that $G(i)$ is infinite. By choice of i the sets $G(j)$ for all $j < i$ are finite. Then $w(i, q) < \infty$.

For any x such that $l(x) \geq w(i, q)$, consider the maximal m such that for some initial fragment $x^m \sqsubseteq x$ there exists an extra edge $\sigma = (x^m, y) \in G(i)$. If no such extra edge exists define $m = w(i, q)$. By definition, if $d(x^m) \neq 0$ then $1/d(x^m)$ is an integer number. Define

$$u(x) = \begin{cases} 1/d(x^m) \text{ if } d(x^m) \neq 0, l(x) \geq w(i, q) \\ \rho(w(i, q)) \text{ if } l(x) < w(i, q) \\ 0 \text{ otherwise} \end{cases}$$

By construction the integer valued function $u(x)$ has the property: $u(x) \geq u(y)$ if $x \sqsubseteq y$. Besides, if $u(x) > u(y)$ then $u(x) > u(z)$ for all z such that $x \sqsubseteq z$ and $l(z) = l(y)$. Then the function

$$\hat{u}(\omega) = \min\{n : u(\omega^i) = u(\omega^n) \text{ for all } i \geq n\}$$

is defined for all $\omega \in \Omega$. It is easy to see that this function is continuous. Since Ω is compact space in the topology generated by intervals Γ_x, this function is bounded by some number m. Then $u(x) = u(x^m)$ for all $l(x) \geq m$. By the construction, if any extra edge of ith type was mounted to $G(i)$ at some step then $u(y) < u(x)$ holds for some new pair (x, y) such that $x \sqsubseteq y$. This is contradiction with the existence of the number m. □

An infinite sequence $\alpha \in \Omega$ is called an *i-extension* of a finite sequence x if $x \sqsubseteq \alpha$ and $B(i, q^{n-1}, x, \alpha^n, n)$ is true for almost all n.

A sequence $\alpha \in \Omega$ is called *i-closed* if $d(\alpha^n) = 1$ for some n such that $p(n) = i$, where d is the q-delay function. Note that if $\sigma \in G(i)$ is some extra edge (i.e. an edge of ith type) then $B(i, q^{n-1}, \sigma, n)$ is true, where $n = l(\sigma_2)$.

Lemma 3. *Let for any initial fragment ω^n of an infinite sequence ω some i-extension exists. Then either the sequence ω will be i-closed in the process of the construction or ω contains an extra edge of ith type (i.e. $\sigma_2 \sqsubseteq \omega$ for some $\sigma \in G(i)$).*

Proof. Let a sequence ω is not i-closed. By Lemma 2 the maximal m exists such that $p(m) = i$ and $d(\omega^m) > 0$. Since the sequence ω^m has an i-extension and $d(\omega^m) < 1$, by Case 2 of the construction a new extra edge (ω^m, y) of ith type must be mounted to the binary tree. By the construction $d(y) = 0$ and $d(z) \neq 0$ for all z such that $\omega^m \sqsubseteq z$, $l(z) = l(y)$, and $z \neq y$. By the choice of m we have $y \sqsubseteq \omega$. □

Lemma 4. *It holds $Q(y) = 0$ if and only if $q(\sigma) = 0$ for some edge σ of unit length located on y (this edge satisfies $\sigma_2 \sqsubseteq y$).*

Proof. The necessary condition is obvious. To prove that this condition is sufficient, let us suppose that $q(y^n, y^{n+1}) = 0$ for some $n < l(y)$ but $Q(y) \neq 0$. Then by definition $d(y^n) = 1$. Since $Q(y) \neq 0$ an extra edge $(x, z) \in G$ exists such that $x \sqsubseteq y^n$ and $y^{n+1} \sqsubseteq z$. But, by the construction, this extra edge can not be mounted to the network $q^{l(z)-1}$ since $d(z^n) = 1$. This contradiction proves the lemma. □

For any semimeasure P define $E_P = \{\omega \in \Omega : \forall n(P(\omega^n) \neq 0)\}$ - the support set of P. It is easy to see that $\bar{P}(E_P) = \bar{P}(\Omega)$. By Lemma 4 $E_Q = \Omega \setminus \cup_{d(x)=1} \Gamma_x$.

Lemma 5. *It holds $\bar{Q}(E_Q) > 1 - \frac{1}{2}\epsilon$.*

Proof. We bound $\bar{Q}(\Omega)$ from below. Let R be defined by (4). By definition of the network flow delay function, we have

$$\sum_{u:l(u)=n+1} R(u) = \sum_{u:l(u)=n} (1 - d(u))R(u) + \sum_{\sigma:\sigma\in G, l(\sigma_2)=n+1} q(\sigma)R(\sigma_1). \quad (5)$$

Define an auxiliary sequence $S_n = \sum\limits_{u:l(u)=n} R(u) - \sum\limits_{\sigma:\sigma\in G, l(\sigma_2)=n} q(\sigma)R(\sigma_1)$. At first, we consider the case $w(p(n), q^{n-1}) < n$. If there is no edge $\sigma \in G$ such that $l(\sigma_2) = n$ then $S_{n+1} \geq S_n$. Suppose that some such edge exists. Define

$$P(u, \sigma) \Longleftrightarrow l(u) = l(\sigma_2) \& \sigma_1 \sqsubseteq u \& u \neq \sigma_2 \& \sigma \in G.$$

By definition of the network flow delay function, we have

$$\sum_{u:l(u)=n} d(u)R(u) = \sum_{\sigma:\sigma\in G, l(\sigma_2)=n} d(\sigma_2) \sum_{u:P(u,\sigma)} R(u) =$$
$$= \sum_{\sigma:\sigma\in G, l(\sigma_2)=n} \frac{d(\sigma_1)}{1 - d(\sigma_1)} \sum_{u:P(u,\sigma)} R(u) \leq \sum_{\sigma:\sigma\in G, l(\sigma_2)=n} d(\sigma_1)R(\sigma_1) =$$
$$= \sum_{\sigma:\sigma\in G, l(\sigma_2)=n} q(\sigma)R(\sigma_1). \quad (6)$$

Here we used the inequality $\sum\limits_{u:P(u,\sigma)} R(u) \le R(\sigma_1) - d(\sigma_1)R(\sigma_1)$ for all $\sigma \in G$ such that $l(\sigma_2) = n$. Combining this bound with (5) we obtain $S_{n+1} \ge S_n$.

Let us consider the case $w(p(n), q^{n-1}) = n$. Then $\sum\limits_{u:l(u)=n} d(u)R(u) \le \rho(n) = (n+n_0)^{-2}$. Combining (5) and (6) we obtain $S_{n+1} \ge S_n - (n+n_0)^{-2}$ for all n. Since $S_0 = 1$, this implies $S_n \ge 1 - \sum\limits_{i=1}^{\infty}(i+n_0)^{-2} \ge 1 - \frac{1}{2}\epsilon$ for some sufficiently large constant n_0. Since $Q \ge R$, it holds

$$\bar{Q}(\Omega) = \inf_n \sum_{l(u)=n} Q(u) \ge \inf_n S_n \ge 1 - \frac{1}{2}\epsilon.$$

Lemma is proved. □

Lemma 6. *There exists a set U of infinite binary sequences such that $\bar{Q}(U) \le \epsilon/2$ and for any sequence $\omega \in E_Q \setminus U$ for each partial computable forecasting system the condition (2) holds.*

Proof. Let ω be an infinite sequence and let f be a partial computable forecasting system such that the corresponding $\phi_t(\omega^{n-1})$ is defined for all n. Let $i = \langle t, s\rangle$ be a program for computing the rational approximation ϕ_{t,κ_s} from below up to $\kappa_s = 1/s$.

If $d(\omega^m) = 1$ for some m such that $p(m) = i$ then for every β of length $(1 + \lceil (2 + \log^{-1}(i+1)\rceil i)m$ such that $\omega^m \sqsubseteq \beta$ there are $< im$ bits hardly predictable by the forecasting program i.

We show that $\bar{Q}$-measure of all intervals generated by such β becomes arbitrary small for all sufficiently large i. Since there are no extra edges σ such that $\omega^m \sqsubseteq \sigma_1$, the measure $\bar{Q}$ when restricted on interval Γ_{ω^m} is proportional to the uniform measure. Then by Lemma 1, where $\mu = \log^{-1}(i+1)$, $\bar{Q}$-measure of all such β decreases exponentially by im. Therefore, for each j there exists a number m_j such that $\bar{Q}(U_j) \le 2^{-(j+1)}$, where U_j is the union of all intervals Γ_β defined by all β of length $(1 + \lceil (2 + \log^{-1}(i+1))i\rceil)m$ for $m \ge m_j$ containing $< im$ bits hardly predictable by the forecasting program $i = p(m)$. Define $U = \cup_{j>k} U_j$, where $k = \lceil -\log_2 \epsilon - 1\rceil$. We have $\bar{Q}(U) < \epsilon/2$.

Define a selection rule γ as follows:

– define $\gamma(\omega^{j-1}) = 1$ if $\sigma_1 \sqsubseteq \omega^{j-1} \sqsubseteq \sigma_2$ for some $\sigma \in G(i)$ and the jth bit of σ_2 is hardly predictable by the forecasting program i;
– define $\gamma(\omega^{j-1}) = 0$ otherwise.

We also define two selection rules J_ν, where $\nu = 0, 1$,

$$J_\nu(\omega^{j-1}) = \begin{cases} 1 - \nu \text{ if } \phi_{t,\kappa_s}(\omega^{j-1}) < \frac{1}{2} \\ \nu \text{ if } \phi_{t,\kappa_s}(\omega^{j-1}) \ge \frac{1}{2} \end{cases}$$

Suppose that $\omega \notin U$ and $\phi_t(\omega^n)$ is defined for all n. Then ω is an i-extension of ω^n for each n. Since for each n the sequence ω^n is not i-closed, by Lemma 3 there exists an extra edge $\sigma \in G(i)$ such that $\sigma_2 \sqsubseteq \omega$. In the following, let $m = l(\sigma_1)$, $n = (1 + \lceil (2 + \log^{-1}(i+1))i \rceil)m$.

Then by the construction the selection rule $\delta_\nu(\omega^{j-1}) = \gamma(\omega^{j-1})J_\nu(\omega^{j-1})$, for $\nu = 0$ or for $\nu = 1$, selects from a fragment of ω of length n a subsequence $\omega_{t_1}, \dots, \omega_{t_l}$ of length $l \ge im/2$. Since by definition these bits are hardly predictable, we have $\omega_{t_j} = 1$ for all j such that $1 \le j \le l$ if $\nu = 0$, and $\omega_{t_j} = 0$ for all these j if $\nu = 1$.

Let $\tilde{p}_j = f(\omega^{j-1})$, $j = 1, 2, \dots$, be an arbitrary computable randomizing forecasting system (it is a random variable) defined on all initial fragments of $\omega = \omega_1\omega_2 \dots$. Then $\phi(\omega^{j-1}) = Pr\{\tilde{p}_j \ge \frac{1}{2}\}$ is a computable real function. By definition $\phi = \phi_t$ for infinitely many t and

$$\phi_{t,\kappa_s}(\omega^{j-1}) \le \phi_t(\omega^{j-1}) \le \phi_{t,\kappa_s}(\omega^{j-1}) + \kappa_s. \tag{7}$$

for all s and j. Consider two random variables, for $\nu = 0$ and for $\nu = 1$,

$$\vartheta_{n,\nu} = \sum_{j=1}^{n} \delta_\nu(\omega^{j-1}) I_\nu(\tilde{p}_j)(\omega_j - \tilde{p}_j).$$

Suppose that $l \ge im/2$ holds for $\nu = 0$. Then using (7) we obtain

$$\begin{aligned} E(\vartheta_{n,0}) \ge \sum_{j=m+1}^{n} \delta_0(\omega^{j-1}) Pr\{\tilde{p}_j < \frac{1}{2}\}\frac{1}{2} - m \ge \\ \ge \frac{im}{4}(\frac{1}{2} - \kappa_s) - m \end{aligned} \tag{8}$$

Since $n = (1 + \lceil (2 + \log^{-1}(i+1))i \rceil)m$, i can be arbitrary large and we visit any pair $i = \langle t, s \rangle$ infinitely often, we obtain from (8)

$$\limsup_{n\to\infty} \frac{1}{n} E(\vartheta_{n,0}) \ge 1/16. \tag{9}$$

Analogously, if $\nu = 1$ we obtain

$$\liminf_{n\to\infty} \frac{1}{n} E(\vartheta_{n,1}) \le -1/16. \tag{10}$$

The martingale strong law of large numbers says that for $\nu = 0, 1$ with Pr-probability one

$$\frac{1}{n}\sum_{j=1}^{n} \delta_\nu(\omega^{j-1}) I_\nu(\tilde{p}_j)(\omega_j - \tilde{p}_j) - \frac{1}{n}E(\vartheta_{n,\nu}) \to 0 \tag{11}$$

as $n \to \infty$. Combining (9), (10) and (11) we obtain (2).

Lemma 6 and Theorem 1 are proved. □

The following theorem is a generalization of the result from V'yugin [11] for partial defined computable deterministic forecasting systems.

Theorem 2. *For any $\epsilon > 0$ a probabilistic algorithm (L, F) can be constructed, which with probability $\geq 1 - \epsilon$ outputs an infinite binary sequence $\omega = \omega_1\omega_2\ldots$ such that for every partial deterministic forecasting algorithm f defined on all initial fragments of the sequence ω a computable outcome-based selection rule δ exists defined on all these fragments such that*

$$\limsup_{n\to\infty} \left| \frac{1}{n} \sum_{i=1}^{n} \delta(\omega^{i-1})(\omega_i - f(\omega^{i-1})) \right| \geq 1/8. \tag{12}$$

The proof of this theorem is based on the same construction.

Acknowledgements

Author thanks an anonymous referee pointing out the connection of the results of this paper with works [5] and [8].

This research was partially supported by Russian foundation for fundamental research: 06-01-00122-a.

References

1. Dawid, A.P.: The Well-Calibrated Bayesian [with discussion], J. Am. Statist. Assoc. 77, 605–613 (1982)
2. Dawid, A.P.: Calibration-Based Empirical Probability [with discussion], Ann. Statist 13, 1251–1285 (1985)
3. Foster, D.P., Vohra, R.: Asymptotic Calibration. Biometrika 85, 379–390 (1998)
4. Kakade, S.M., Foster, D.P.: Deterministic Calibration and Nash Equilibrium. In: Shawe-Taylor, J., Singer, Y. (eds.) COLT 2004. LNCS (LNAI), vol. 3120, pp. 33–48. Springer, Heidelberg (2004)
5. Lehrer, E.: Any Inspection Rule is Manipulable. Econometrica 69(5), 1333–1347 (2001)
6. Oakes, D.: Self-Calibrating Priors Do not Exists [with discussion]. J. Am. Statist. Assoc. 80, 339–342 (1985)
7. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)
8. Sandroni, A., Smorodinsky, R., Vohra, R.: Calibration with Many Checking Rules. Mathematics of Operations Research 28(1), 141–153 (2003)
9. Schervish, V.: Comment [to Oakes, 1985]. J. Am. Statist. Assoc. 80, 341–342 (1985)
10. Vovk, V.: Defensive forecasting for optimal prediction with expert advice, arXiv:0708.1503v1 (2007)
11. V'yugin, V.V.: Non-Stochastic Infinite and Finite Sequences. Theor. Comp. Science 207, 363–382 (1998)
12. Zvonkin, A.K., Levin, L.A.: The Complexity of Finite Objects and the Algorithmic Concepts of Information and Randomness. Russ. Math. Surv. 25, 83–124 (1970)

Algorithms for Multiterminal Cuts

Mingyu Xiao

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong SAR, China
myxiao@cse.cuhk.edu.hk

Abstract. Given a graph $G = (V, E)$ with n vertices and m edges, and a subset T of l vertices called terminals, the *Edge* (respectively, *Vertex*) *Multiterminal Cut* problem is to find a set of k edges (non-terminal vertices), whose removal from G separates each terminal from all the others. These two problems are NP-hard for $l \geq 3$ but well-known to be polynomial-time solvable for $l = 2$ by the flow technique. In this paper, we show that Edge Multiterminal Cut is polynomial-time solvable for $k = O(\log n)$ by presenting an $O(2^k lT(n, m))$ algorithm, where $T(n, m) = O(\min(n^{2/3}, m^{1/2})m)$ is the running time of finding a minimum (s, t) cut in an unweighted graph. We also give two algorithms for Vertex Multiterminal Cut that run in $O(l^k T(n, m))$ time and $O((k!)^2 T(n, m))$ time respectively. The former one indicates that Vertex Multiterminal Cut is solvable in polynomial time for l being a constant and $k = O(\log n)$, and the latter one improves the best known algorithm of running time $O(4^{k^3} n^{O(1)})$. When $l = 3$, we show that the running times can be improved to $O(1.415^k T(n, m))$ for Edge Multiterminal Cut and $O(2.059^k T(n, m))$ for Vertex Multiterminal Cut. Furthermore, we present a simple idea to solve another important problem *Multicut* by finding minimum multiterminal cuts. Our algorithms for Multicuts are also faster than the previously best algorithm.

Based on a notion *farthest minimum isolating cut*, we present some properties for Multiterminal Cuts, which help shed light on the structure of optimal cut problems, and enables us to design efficient algorithms for Multiterminal Cuts, as well as some other related cut problems.

Keywords: Graph Algorithm, Multiterminal Cut, Multicut, Fixed Parameter Tractability.

1 Introduction

Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, positive integers k and l, and a set $T \subset V$ of l vertices, called *terminals*, the Edge (respectively, Vertex) Multiterminal Cut problem is to find a subset S of k edges (non-terminal vertices) such that all terminals in T are in different components of $G - S$. These two Multiterminal Cut problems are fundamental network design problems and have applications in VLSI system design, parallel computing, distributed computing, clustering, and many others [5], [18], [4].

Dahlhaus *et al.* [5] initiated the study of the Multiterminal Cut problems by establishing their NP-hardness for each fixed $l \geq 3$ and giving a $(2 - 2/l)$-approximation algorithm for Edge Multiterminal Cut. For Edge Multiterminal Cut, Calinescu *et al.* [2] used a novel geometric relaxation to obtain a $(1.5-1/l)$-approximation algorithm. And the current best approximation algorithm is the $(1.3438 - \epsilon_l)$-approximation algorithm developed by Karger *et al.* [14]. Garg *et al.* [9] obtained a $(2 - 2/l)$-approximation algorithm for Vertex Multiterminal Cut, and Naor and Zosin [17] gave a 2-approximation algorithm for the Multiterminal Cut problems in directed graphs. In terms of exact algorithms, the Multiterminal Cut problems for $l = 2$ degenerate to the well known Minimum Edge/Vertex Cut problems and can be solved in polynomial time by flow techniques and some other techniques. For $l \geq 3$, Dahlhaus *et al.* [5] gave an $O((4l)^l n^{(2l-1)} \log n)$ algorithm for solving Edge Multiterminal Cut in planar graphs, which was later improved to $O(l^l (n-l)^{2l} n \log(n-l))$ by Yeh [18]. Marx [16] obtained an $O(4^{k^3} n^{O(1)})$ algorithm for Vertex Multiterminal Cut in general graphs.

In this paper, we revisit the exact algorithms for Multiterminal Cuts. To make them tractable, we put much attention to the scenarios where the size k of the multiterminal cut is small. This view of point has drawn much attention in NP-hard problems recently, such as Feige and Mahdian's work [8] on finding small balanced cuts, Marx's work [15] on the closest substring problem with small distances, and so on. Those who are familiar with Parameterized Complexity [7] will find our results positive on the *fixed parameterized tractability* of finding the minimum multiterminal cuts.

We show that Edge Multiterminal Cut can be solved in $O(c^k |I|^{O(1)})$ time (c is a constant and $|I|$ is the input size of the problem), which implies that when the size of the solution is $O(\log |I|)$, Edge Multiterminal Cut can be solved in polynomial time. When the number l of terminals is a constant, Vertex Multiterminal Cut has a similar result. We prove that Vertex Multiterminal Cut can be solved in $O(l^k T(n,m))$ time. We also present an alternative algorithm for Vertex Multiterminal Cut with running time $O(\frac{(k!)^2}{2^k} T(n,m))$, which directly improves the previously known result of $O(4^{k^3} n^{O(1)})$ [16]. The running times of our algorithms are in the form of $O(f(k)|I|^{O(1)})$, where $f(k)$ is a computable function. It shows that Edge Multiterminal Cut and Vertex Multiterminal Cut are *fixed parameter tractable* when k is taken as a parameter, which is a positive result in Parameterized Complexity.

Most of our algorithms are based on a simple observation: There are two cases for each element (an edge, or a vertex for the vertex version) in the *farthest minimum isolating cut* for a terminal (See definition in Section 2): in the solution set or not. We arbitrarily choose one element and branch at it by either including it in the solution set or excluding it from the solution set. In the former branch, the size of the current solution set increases by 1; in the latter branch, we can get an equivalent graph, in which the size of the farthest minimum isolating cut for the same terminal will increase. Since the sizes of the solution set and all the minimum isolating cuts are bounded by k if a solution exists, we can build a bounded search tree to solve Multiterminal Cuts. The rest of this paper is organized as follows:

In Section 2, we first introduce some basic techniques and present some structural properties of optimal cut problems. *Farthest minimum isolating cut* is a frequently used notion in our algorithms. Based on it, we can identify a small set of edges (vertices) in various ways to branch upon in enumerating possible minimum multiterminal cuts. In Section 3, we present an $O(2^{\frac{l-2}{l-1}k}lT(n,m))$ algorithm for Edge Multiterminal Cut (See Theorem 1). That algorithm only works for the edge version, because it will use the property of a 2-approximation solution of Edge Multiterminal Cut and this property does not hold in the vertex version. In Section 4, we show that Vertex Three-terminal Cut can be solved in $O(2.059^kT(n,m))$ (See Theorem 2) and Vertex Multiterminal Cut can be solved in $O(l^kT(n,m))$ (See Theorem 3). In Section 5, an alternative algorithm for Vertex Multiterminal Cut with running time $O(\frac{(k!)^2}{2^k}T(n,m))$ is given (See Theorem 4). Furthermore, in Section 6, we show that our results will lead to an $O(p^{2l}2^kT(n,m))$ algorithm for Edge Multicut, where $p = \min(\sqrt{2l}, k)$ and $O((2l)^{l+k/2}T(n,m))$ or $O(\frac{(k!)^2l^l}{2^{k-l}}T(n,m))$ algorithm for Vertex Multicut. For most cases, our results improve the previously known result of $O(2^{kl}4^{k^3}n^{O(1)})$ [16] on Multicuts.

2 Definitions and Structural Properties

It is easy to transform an Edge Multiterminal Cut problem to a Vertex Multiterminal Cut problem in polynomial time by using some standard techniques. But no polynomial reduction in the opposite direction is found, which implies that the edge version might be easier than the vertex version. In this paper our algorithms for the edge version are also faster than those for the vertex version. There are certain relations between those two versions as they have similar definitions and properties. Our algorithms for the vertex version are still applicable to the edge version and so on. Considering their similarities, we will not discuss the two versions separately. In the following sections, if no explicit statement is given, the defined terms are applicable for both the edge and vertex version. For the vertex version cut problems, some references [1], [12] also discuss the case that terminals can be selected into the solution set. They call it *unrestricted vertex version* when terminals can be moved into the solution set, and *restricted vertex version* when terminals are not allowed to be deleted. An unrestricted vertex terminal cut instance can be reduced to a restricted vertex version instance simply by adding, for each terminal t_i, a new terminal t'_i adjacent only to t_i. And with some modifications our algorithms for the restricted vertex version are still applicable for the unrestricted vertex version. So we do not discuss the unrestricted vertex version problem individually. In this paper, vertex version always means restricted vertex version. The proofs of the main claims in this section can be found in the full version paper.

2.1 Basic Definitions

In this paper, k always denotes the upper bound of the solution size and l denotes the number of terminals (or the number of terminal pairs in Multicut problems). $T = \{t_1, t_2, \cdots, t_l\}$ is the terminal set, $T_{-i} = T - \{t_i\}$ is the set of terminals except t_i, S is the *deletion set* or *solution set*, the edge or vertex set to be deleted in our algorithms. We will require $|S| \leq k$. $E(G)$ and $V(G)$ denote the edge set and vertex set of graph G respectively. We call an edge (non-terminal vertex) subset an *edge (vertex) multiterminal cut*, if deleting it separates the l terminals. Among all the multiterminal cuts, those with the minimum size are called *minimum multiterminal cuts*. A *minimal multiterminal cut* is a multiterminal cut in case that any of its proper subsets is not a multiterminal cut. If with no explicit statement, multiterminal cuts in this paper always mean minimal multiterminal cuts. And we assume that the initial graph is connected.

We call a subset C_i of edges (vertices) an *isolating cut* for terminal t_i, if deleting C_i separates t_i from all other terminals T_{-i}. A *minimum isolating cut* for t_i is an isolating cut for t_i with the minimum size. Usually the minimum isolating cut is not unique and can be found by using the methods for the Minimum (s, t) Cut problem.

After deleting a minimal multiterminal cut S, the graph is separated into several components, each of which contains at most one terminal (For the edge version, there are right l components). Let T_i be the component containing terminal t_i and $C_i \subseteq S$ be the edge (vertex) set incident on (adjacent to) the vertices in T_i. Then C_i is an isolating cut for t_i. For a minimal edge multiterminal cut S, each edge in S will appear in two isolating cuts C_i's, since each edge connects two components. We have that $\sum_{i=1}^{l} |C_i| = 2|S|$. But minimal vertex multiterminal cuts have no such a good property. A vertex in a vertex multiterminal cut may appear in more than 2 such isolating cuts. This difference is the main point that makes the faster algorithms in Section 3 suitable only for the edge version but not for the vertex version.

In our algorithms, we often branch at an edge (vertex) by either including or excluding it in the solution set. When an edge (vertex) is not allowed to be selected into the solution set, we can implement it by *shrinking* the edge (*dissolving* the vertex). Shrinking an edge means that we identify the two endpoints of it keeping all edges, including possible parallel edges, incident on the two endpoints and delete the self-loops. To dissolve a vertex we will delete this vertex and all the edges incident on it, and add an edge between any two of its neighbors if there is not an edge between them. See Figure 1 for an illustration of these two operations. Sometimes we will *contract* a subgraph into a single vertices, i.e., we replace this subgraph with a single vertex, keeping all the edges incident on it (also keeping the parallel edges).

2.2 Farthest Minimum Isolating Cuts

A notion, called *farthest minimum isolating cut*, is frequently used in our algorithms. Based on this notion, we present some properties of multiterminal cuts,

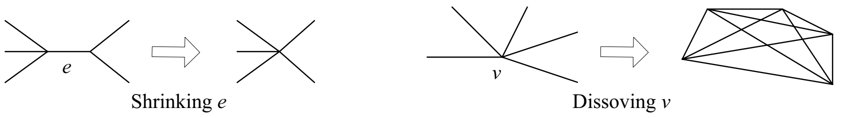


Fig. 1. Shrinking and dissolving

which provide a deeper understanding about the structure of cut problems and faster algorithms for finding small multiterminal cuts. A minimum isolating cut C_i for terminal t_i splits the graph into two partitions. One that contains t_i is called the *residual* of C_i and denoted by R_i. The other one is the remaining part containing T_{-i}, and is called the *outside* of C_i. The *farthest minimum isolating cut* for terminal t_i is the minimum isolating cut that makes the residual R_i of the maximum cardinality over all the minimum isolating cuts. Here we use 'the minimum isolating cut' as opposed to 'a minimum isolating cut', since the farthest minimum isolating cut for a given terminal is unique. The uniqueness is discussed in many flow monographs and used in some references [13], [5], [11]. It can be easily derived from the Max flow/Min cut theorem [13]. How fast we can find the farthest minimum isolating cut is directly related to the running time of our algorithms since finding the farthest minimum isolating cut is a subroutine in our algorithms. The farthest minimum isolating cut for t_i can be found in linear time, if a maximum flow from t_i to T_{-i} is given. Many algorithms for the Minimum (s,t) Cut problem just find the farthest minimum isolating cut for s or t. (Note that given a maximum flow, in the residual graph some vertices, including s, can not reach t. Let X be the set of all the vertices can reach t in the residual graph, then edges between $V-X$ and X is the farthest minimum isolating cut for s). There are many fast algorithms computing max flow/min cut. Reader can refer to [10] as a survey. In this paper, we will simply use Dinic's $O(\min(n^{2/3}, m^{1/2})m)$ algorithm [6] finding max flows in unweighted graphs. Thus, $T(n,m) = O(\min(n^{2/3}, m^{1/2})m)$. And we use C_i to denote the farthest minimum isolating cut for t_i, when no ambiguity will be aroused.

Lemma 1. *The farthest minimum isolating cut for t_i will not change in the graph when e shrinks (dissolves, for the vertex version) , where e is an edge (a vertex) not in the farthest minimum isolating cut for t_i.*

Lemma 2. *Let S be a minimum edge (respectively, vertex) multiterminal cut of graph G, C_i be a minimum isolating cut for terminal t_i, and R_i be the residual of C_i, $i \in \{1, 2, \cdots, l\}$. Suppose $|S \cap E(R_i)| = p$ ($|S \cap V(R_i)| = p$). Then G has a minimum multiterminal cut S' such that $S' \cap E(R_i) = \emptyset$ ($S' \cap V(R_i) = \emptyset$), and $|S' \cap C_i| \geq p$.*

Corollary 1. *Let C_i be the farthest minimum isolating cut for terminal t_i in G, and G' be the graph after merging R_i into a new terminal t'_i. Then any minimum multiterminal cut in G' is a corresponding minimum multiterminal cut in G.*

Lemma 2 and Corollary 1 imply that we can find a minimum multiterminal cut in part of the graph, the outside of a minimum isolating cut and the cut itself, ignoring the residual. Note that Lemma 2 can also be inferred from the proof of Goldschmidt and Hochbaum's main theorem in [11]. And Dahlhaus *et al.* [5] also proved results similar to Lemma 2 and Corollary 1 when they discuss reducing the instance size. However their arguments only work for edge version. Our proof in the full version paper is suitable for both the edge and the vertex versions.

Then we give another important definition of the *ith layer farthest isolating cut*. For the edge version, if the farthest minimum edge isolating cut C_i is not incident on any other terminals except t_i, i.e., no endpoint of an edge in C_i is a terminal except t_i, we can contract the residual R_i together with cut C_i into a new terminal $t_i^{(2)}$ without merging two terminals together. Let $G^{(2)}$ be the new graph after contracting R_i and C_i. Now we find the farthest minimum isolating cut $C_i^{(2)}$ for terminal $t_i^{(2)}$ in $G^{(2)}$. We call the corresponding set of $C_i^{(2)}$ in the original graph G *the second layer farthest isolating cut* (or just *the second layer*) for terminal t_i in G. Similarly, we can calculate the third layer $C_i^{(3)}$, the forth layer $C_i^{(4)}$ and so on, if they exist. The ith layer farthest isolating cut for the vertex version can be defined in the same way. To avoid merging two terminals together or having no vertex cut anymore we say that the jth layer exists if the $(j-1)$th layer $C_i^{(j-1)}$ is not incident on (adjacent to) any other terminals except t_i. And the first layer $C_i^{(1)}$ is just regarded as C_i.

Lemma 3. *Let $C_i^{(j)}$ and $C_i^{(j+1)}$ be the jth and $(j+1)$th layer farthest isolating cuts for terminal t_i in graph G respectively. Then $|C_i^{(j+1)}| \geq |C_i^{(j)}| + 1$ and $|C_i^{(j)}| \geq j$.*

3 Algorithms for Edge Multiterminal Cut

In this section we present an algorithm for Edge Multiterminal Cut with running time $O(2^{\frac{l-2}{l-1}k} lT(n,m))$. We will describe the algorithm in terms of the edge version. Initially let the solution set S be an emptyset. The main steps of our algorithm are listed as follows:

Step 1: *Reduce the graph by iteratively choosing a terminal t_i, calculating the farthest minimum isolating cut C_i for it, and contracting the residual R_i into a new terminal t'_i until all the l terminals are chosen. (Remind that we will keep parallel edges when contract R_i. And for convenience, we just use t_i to denote the new terminal t'_i in the new graph)*
Step 2: *Let $B = \bigcup_{i=1}^{l} C_i$. If $|B| \leq (k - |S|)\frac{l}{l-1}$, return a solution $S \longleftarrow S + (B - C_{i'})$ directly and stop the algorithm, where $C_{i'}$ satisfying $|C_{i'}| = \max_{i=1}^{l} |C_i|$. Else goto the next step.*
Step 3: *Delete all the edges that directly connect two terminals and move them into the solution set S (Note that all of those edges are in B).*
Step 4: *Branch at an edge e in B by including it in the solution set S or excluding it from the solution set: in the former branch, we delete it in the current graph; in the latter branch, we just shrink it.*

Step 5: *Loop Step 1-4 until one of the following three stop conditions is satisfied: (1) All the l terminals are separated away, and at that time output S as a solution; (2) $|S| > k$; (3) $2|S| + \sum_{i=1}^{l} |C_i| > 2k$.*

Lemma 4. *The above algorithm finds an edge multiterminal cut of size $\leq k$ if it exists.*

The first two stop conditions in **Step 5** are intuitive. The last one seems a bit complicated. We will explain it in the following analysis of correctness of the algorithm.

According to Corollary 1, we can reduce the graph in **Step 1**. When $|B| \leq (k - |S|)\frac{l}{l-1}$ and $|C_{i'}| = \max_{i=1}^{l} |C_i|$, $|B - C_{i'}| \leq (k - |S|)\frac{l}{l-1}\frac{l-1}{l} = k - |S|$, $S + (B - C_{i'})$ is solution with size at most k. In this case, we will find a solution in **Step 2** directly. To separate all the terminals, the edges that directly connect two terminals must be deleted. We can put such edges into S directly in **Step 3**. Now we consider set B. There are two situations for each edge in B: in the solution or not. If the edge is in the solution then we can add it into S directly and delete it in the graph and reduce the parameter k by 1; if it is not, shrinking it will not cause problems according to Lemma 1. So we will cover all cases. Stop condition (1) and (2) in **Step 5** are easy to understand. We prove that when $2|S| + \sum_{i=1}^{l} |C_i| > 2k$, there is not a valid multiterminal cut (a multiterminal cut with size at most k) containing current solution set S. Suppose $S' \supseteq S$ is a valid multiterminal cut containing the current solution set S. Obviously, $S'' = S' - S$ is a multiterminal cut for the current graph. Let $S'_i \subseteq S''$ be the minimal isolating cut for terminal t_i contained in S''. $|C_i| \leq |S'_i|$, and then

$$\sum_{i=1}^{l} |C_i| \leq \sum_{i=1}^{l} |S'_i| = 2|S''| = 2|S'| - 2|S| \leq 2k - 2|S|. \tag{1}$$

So $2|S| + \sum_{i=1}^{l} |C_i| \leq 2k$. When $2|S| + \sum_{i=1}^{l} |C_i| > 2k$, we can stop this subbranch.

Next, we prove that this algorithm always stops. Let $p' = 2|S| + \sum_{i=1}^{l} |C_i|$. We show that p' will increase in each iteration. Since the upper bound of p' is finite (stop condition 3), the algorithm always stops. Only **Step 3** and **Step 4** will possibly affect p'. We show that p' will not change in **Step 3** and increase by at least 1 in each subbranch of **Step 4**. In **Step 3**, when an edge is selected into S, $\sum_{i=1}^{l} |C_i|$ decreases by 2 and $|S|$ increases by 1. p' will not change. In **Step 4**, when we include an edge $e \in C_i$ in S, $|S|$ increases by 1. Now we prove that in this subbranch $\sum_{i=1}^{l} |C_i|$ decreases by 1. After deleting e, $C'_i = C_i - \{e\}$ is the farthest minimum isolating cut for terminal t_i. And for any j ($j \neq i$), the size of the farthest minimum isolating cut C'_j will not decrease. Note that $C''_j = C'_j \bigcup \{e\}$ is still an isolating cut for terminal t_j. Assume to the contrary that $|C'_j| \leq |C_j| - 1$, then $|C''_j| \leq |C_j|$. But $C''_j \neq C_j$, in contradiction with the uniqueness of the farthest minimum isolating cut. So $\sum_{i=1}^{l} |C_i|$ decreases by 1. Since $\sum_{i=1}^{l} |C_i|$ decreases by 1 and $|S|$ increases by 1, totally p' increases by 1. When e shrinks, it is easy to see that the farthest minimum isolating cut for t_i

will increase its size by at least 1 and other farthest minimum isolating cuts will not change according to Lemma 1. Totally $\sum_{i=1}^{l} |C_i|$ will increase by at least 1. Thus, in each subbranch of **Step 4**, p' will increase by at least 1.

As for the time complexity of the algorithm, we build a recurrence relation related to a parameter $p = 2k - p'$. In each branch, no matter whether the edge is included in S or not, p' increases by 1 and then p decreases by 1. In each iteration we need to find at most l farthest minimum isolating cuts to get B. So we have

$$B(p) \le B(p-1) + B(p-1) + O(lT(n,m)), \tag{2}$$

where $B(p)$ is the running time of our algorithm when the parameter is p, and $T(n,m)$ is the running time of finding a farthest minimum isolating cut.

It is easy to verify that

$$B(p) = O(2^p lT(n,m)) \tag{3}$$

satisfies (2).

We only need to confirm an upper bound for p, and then we can get the time complexity for this algorithm. Recall $p = 2k - \sum_{i=1}^{l} |C_i| - 2|S|$. In the beginning, S is an empty set. We look at $B = \bigcup_{i=1}^{l} C_i$ the union of all the farthest minimum isolating cuts. For any $i \in \{1, 2, \cdots, l\}$, $B - C_i$ is a multiterminal cut. Suppose $|C_{i'}| = \max_{i=1}^{l} |C_i|$. Then $|C_{i'}| \ge \frac{1}{l}\sum_{i=1}^{l} |C_i| \ge \frac{1}{l}|B|$. If $|B - C_{i'}| \le k$, then we find a solution directly in **Step 2**. Else $\sum_{i \ne i'} |C_i| \ge |B - C_{i'}| > k$, then $\sum_{i=1}^{l} |C_i| > k + |C_{i'}| \ge k + \frac{1}{l}\sum_{i=1}^{l} |C_i|$, $\sum_{i=1}^{l} |C_i| > \frac{l}{l-1}k$. So

$$p = 2k - \sum\nolimits_{i=1}^{l} |C_i| - 2|S| < \tfrac{l-2}{l-1}k. \tag{4}$$

Combining (3) and (4), we get that $B(p) = O(2^{\frac{l-2}{l-1}k} lT(n,m))$.

Theorem 1. *Edge Multiterminal Cut can be solved in* $O(2^{\frac{l-2}{l-1}k} lT(n,m))$ *time.*

Corollary 2. *Edge Three-terminal Cut can be solved in* $O(1.415^k T(n,m))$ *time.*

4 Algorithms for Vertex Multiterminal Cut

In this section we show that when l is a constant and $k = O(\log n)$, Vertex Multiterminal Cut is also polynomial-time solvable by presenting an $O(l^k T(n,m))$ algorithm for it. The algorithm is based on the algorithm for Edge Multiterminal Cut presented in Section 3. We prove that the idea in Section 3 can still be used for solving Vertex Multiterminal Cut, but the running time may cost more. In the next two sections, we will describe the algorithms in terms of the vertex version, although they are applicable to both Vertex Multiterminal Cut and Edge Multiterminal Cut.

We only need to modify the stop conditions of the algorithm in Section 3 as the following **Step** $5'$ and describe the algorithm in terms of the vertex version.

Step $5'$**:** *Loop Step 1-4 until one of the following three stop conditions is satisfied: (1) All the l terminals are separated away, and at that time output S as a solution; (2) $|S| > k$; (3) $l|S| + \sum_{i=1}^{l} |C_i| > lk$.*

The third stop condition is changed. The reason lies in that for vertex version problems a vertex in a multiterminal cut may appear in more than 2 isolating cut C_i's, as opposed to exact 2.

The proof of the algorithm is also based on the analysis in Section 3. We only need to show that when $l|S| + \sum_{i=1}^{l} |C_i| > lk$, there is not a valid multiterminal cut containing S and the algorithm always stops. Since each vertex in a multiterminal cut may appear in up to l isolating cuts C_i's, we modify (1) to $\sum_{i=1}^{l} |C_i| \leq \sum_{i=1}^{l} |S_i'| \leq l|S''| = l|S'| - l|S| \leq lk - l|S|$. Then in the same way we can prove it. We also can prove that this algorithm always stops by showing that $p = lk - l|S| - \sum_{i=1}^{l} |C_i|$ will decrease in each iteration.

Only **Step 3** and **Step 4** will possibly affect p. In **Step 3**, when an edge is selected into S, $\sum_{i=1}^{l} |C_i|$ decreases by at most l and $|S|$ increases by 1. p will not increase. In **Step 4**, when we include a vertex $v \in C_i$ in S, $|S|$ increases by 1. By the same way discussed in Section 3 we can prove that in this subbranch $\sum_{i=1}^{l} |C_i|$ decreases by 1. Since $\sum_{i=1}^{l} |C_i|$ decreases by 1 and $|S|$ increases by 1, totally p decreases by $l-1$. When v is dissolved, $|S|$ does not change and $\sum_{i=1}^{l} |C_i|$ increases by at least 1. In this subbranch p will decrease by at least 1.

We get the following recurrence relation

$$B(p) \leq B(p-1) + B(p-(l-1)), \tag{5}$$

where $B(p)$ is the number of search paths in the search tree of our algorithm with parameter p.

In the beginning, S is an empty set and $\sum_{i=1}^{l} |C_i| > \frac{l}{l-1}k$.

$$p = lk - \Sigma_{i=1}^{l}|C_i| - l|S| < (l-1)k - \tfrac{1}{l-1}k. \tag{6}$$

When $l = 3$, it is easy to verify that $B(p) = O(2.059^k)$ satisfies (5) and (6).

Theorem 2. *Vertex Three-terminal Cut can be solved in $O(2.059^k T(n,m))$ time.*

For general l, we will relax more to get a simplified running time bound.

$$\begin{aligned} B(p) &\leq B(p-1) + B(p-(l-1)) \leq B(p-2) + 2B(p-(l-1)) \\ &\leq \cdots \leq B(p-(l-1)) + (l-1)B(p-(l-1)) = lB(p-(l-1)). \end{aligned}$$

$$B(p) < l^{\left\lceil \frac{p}{l-1} \right\rceil} < l^{k - \left\lfloor \frac{k}{(l-1)^2} \right\rfloor} \leq l^k.$$

Theorem 3. *Vertex Multiterminal Cut can be solved in $O(l^k T(n,m))$ time.*

5 A Simple Algorithm For Multiterminal Cut

In this section, we present a simple algorithm for Vertex Multiterminal Cut with running time $O(\frac{(k!)^2}{2^k} T(n,m))$. The exponential part of the running time is only related to parameter k. When k is sufficiently less than l, this algorithm may be faster than the algorithm in Section 4.

Our algorithm involves the following two iterative steps until a solution is found or the size of the current solution set is greater than k: first, find a vertex candidate set B in the current graph such that at least one vertex in B is in a solution, if a solution dose exist; second, we branch at each vertex in B by including it in the current solution set and deleting it from the graph. Our algorithm guarantees that the size of B is bounded by $\frac{k(k-1)}{2}$. So we get the running time $O(\frac{k!(k-1)!}{2^k}kT(n,m))$, where $kT(n,m)$ is the time required for finding B.

t_i is an arbitrarily selected terminal. Let the candidate set B be $\sum_{j=1}^{b} C_i^{(j)}$, where $C_i^{(j)}$ is the jth layer of t_i and b is the smallest number satisfying that: the $(b+1)$th layer $C_i^{(b+1)}$ does not exist or $|C_i^{(b+1)}| > k$. It is easy to see that B contains at least one vertex in a solution if it exists by the definition of the layers in Section 2.2. Let $R_i^{(j)}$ be the residual of $C_i^{(j)}$ $(j = 1, \cdots, b)$ and S be a minimum multiterminal cut. Note that if $(V(R_i^{(j)}) \cup C_i^{(j)}) \cap S = \emptyset$, we can merge $R_i^{(j)}$ and $C_i^{(j)}$ into a new terminal and then consider $C_i^{(j+1)}$. Else we know that $C_i^{(j)}$ contains at least one vertex in a minimum multiterminal cut by Lemma 2. Now we only need to confirm an upper bound for $|B|$. According to Lemma 3, $|C_i^{(1)}| < |C_i^{(2)}| < \cdots < |C_i^{(b)}| < k$ (We also ignore the trivial case $|C_i^{(b)}| = k$). Then $|B| = \sum_{j=1}^{b} |C_i^{(j)}| \le \frac{k(k-1)}{2}$.

To find the candidate set B we need at most $b \le k$ farthest minimum isolating cut computations. So we get the recurrence relation

$$B(k) \le \tfrac{k(k-1)}{2} B(k-1) + O(kT(n,m)), \tag{7}$$

where $B(k)$ is the running time of our algorithm when the parameter is k.

$B(k) = O(\frac{(k!)^2}{2^k}T(n,m))$ satisfies (7).

Theorem 4. *Vertex Multiterminal Cut can be solved in* $O(\frac{(k!)^2}{2^k}T(n,m))$ *time.*

Remark. Our algorithms for Vertex Multiterminal Cut are still applicable to Multiterminal Cut in hypergraphs. In a Hypergraph Multiterminal Cut problem, we are asked to delete at most k hyperedges to separate l given terminals. It is trivial to extend lemmas in Section 2 and algorithms in Section 4 and Section 5 for Vertex Multiterminal Cut to the corresponding results for Hypergraph Multiterminal Cut. Theorem 3 and Theorem 4 still hold for Hypergraph Multiterminal Cut. The only difference is that in the running time $T(n,m)$ should be the time bound for finding minimum (s,t) cuts in hypergraphs.

6 Improvements on Multicuts

The Multicut problem is another important graph cut problem and extensively studied in the literature [4], [1], [12]. In this problem, l pairs of terminals, instead of l terminals, are given, and we are required to separate each of the l pairs by deleting at most k edges or vertices. Multiterminal Cut can be regarded as a special case of Multicut. For each pair of terminals in a Multiterminal Cut problem, we set it as a terminal pair and then get an equivalent Multicut problem.

We present a simple idea to solve Multicut by solving Multiterminal Cuts as a subroutine. Given a minimum multicut, the graph is separated into several connected components after the minimum multicut is removed. Initially let each component be a *compatible set*. We iteratively merge two compatible sets into one compatible set when there is no terminal pair in the union of the two compatible sets until no more compatible set can be merged. There are l terminal pairs, so at last there are at most $\sqrt{2l}$ compatible sets left (Note that a complete graph with l edges has at most $\sqrt{2l}$ vertices). We consider such a Multiterminal Cut problem I': the graph is the same graph as the Multicut problem I, and the union of all the terminals in each compatible set is a terminal for the Multiterminal Cut problem I'. A multiterminal cut of I' obviously splits all the l terminal pairs of I and the minimum multicut of I is still a multiterminal cut of G'. Thus, a minimum multiterminal cut of I' will be a minimum multicut of I. There are at most $2l$ different terminals in the Multicut problem and each one will be in one of the $\sqrt{2l}$ compatible sets. We try all the $\sqrt{2l}^{2l} = (2l)^l$ possibilities. Then we can solve a Multicut problem by solving at most $(2l)^l$ Multiterminal Cut problems. We also note that in the edge version cut problems, the initial graph is connected and by deleting one edge we can get at most one component more. So the number of compatible sets for edge cut problems is not greater than $k+1$. For Edge Multicut, the number of compatible sets is bounded by $\min(\sqrt{2l}, k+1)$.

Theorem 5. *Edge Multicut can be solved in $O(p^{2l}2^kT(n,m))$ time, where $p = \min(\sqrt{2l}, k+1)$.*

Theorem 6. *Vertex Multicut can be solved in $O(\min(\frac{(k!)^2}{2^k}, \sqrt{2l}^k)(2l)^lT(n,m))$ time.*

The best previous result on Multicut was $O(2^{kl}4^{k^3}n^{O(1)})$ [16]. Our result on Edge Multicut has greatly improved it. And in most cases, such as when $2^k > l$, our algorithms for Vertex Multicut are also faster than the best previous algorithm.

7 Remarks

In this paper, based on the notion farthest minimum isolating cut, we have presented several simple and improved algorithms for Multiterminal Cuts. This notion is helpful for us to characterize the structure of optimal cut problems and design fast algorithms. In addition, we have proved that Edge Multiterminal Cut can be solved in polynomial time when $k = O(\log n)$ and a comparatively weaker result for Vertex Multiterminal Cut.

The initial version of this paper was presented at the Institute for Theoretical Computer Science of Tsinghua University in 2006. Recently, Chen et al. [3] developed an $O(4^kkn^3)$ algorithm for Vertex Multiterminal Cut by using different techniques, indicating Vertex Multiterminal Cut also can be solved in polynomial time when $k = O(\log n)$. They extended our result on Edge Multiterminal Cut to Vertex Multiterminal Cut. But our algorithms for Edge Multiterminal Cut and Vertex {3,4}-terminal Cuts are still the fastest algorithms.

References

1. Calinescu, G., Fernandes, C., Reed, B.: Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. J. Algorithms 48(2), 333–359 (2003)
2. Calinescu, G., Karloff, H.J., Rabani, Y.: An improved approximation algorithm for multiway cut. Journal of Computer and Systems Sciences 60(3), 564–574 (2000); a preliminary version appeared in STOC 1998 (1998)
3. Chen, J., Liu, Y., Lu, S.: An Improved Parameterized Algorithm for the Minimum Node Multiway Cut Problem. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, Springer, Heidelberg (2007)
4. Costa, M.-C., Létocart, L., Roupin, F.: Minimal multicut and maximal integer multiflow: A survey. European Journal of Operational Research 162(1), 55–69 (2005)
5. Dahlhaus, E., Johnson, D., Papadimitriou, C., Seymour, P., Yannakakis, M.: The complexity of multiterminal cuts. SIAM J. Comput. 23(4), 864–894 (1994)
6. Dinic, E.: Algorithm for solution of a problem of maximum flow in networks with power estimation. Soviet Math. Dokl. 11, 1277–1280 (1970)
7. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
8. Feige, U., Mahdian, M.: Finding small balanced separators. In: Proceedings of the 38th Annual ACM symposium on Theory of Computing (STOC 2006) (2006)
9. Garg, N., Vazirani, V.V., Yannakakis, M.: Multiway cuts in node weighted graphs. J. Algorithms 50(1), 49–61 (2004)
10. Goldberg, A.V., Rao, S.: Beyond the flow decomposition barrier. J. ACM 45(5), 783–797 (1998)
11. Goldschmidt, O., Hochbaum, D.: A polynomial algorithm for the k-cut problem for fixed k. Mathematics of Operations Research 19(1), 24–37 (1994)
12. Guo, J., Hüffner, F., Kenar, E., Niedermeier, R., Uhlmann, J.: Complexity and exact algorithms for multicut. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831. Springer, Heidelberg (2006)
13. Ford, J.R., Fullkerson, D.R.: Flows in networks. Princeton University Press, Princeton (1962)
14. Karger, D.R., Klein, P.N., Stein, C., Thorup, M., Young, N.E.: Rounding algorithms for a geometric embedding relaxation of minimum multiway cut. In: Proceedings of the 31th Annual ACM Symposium on Theory of Computing (STOC 1999) (1999)
15. Marx, D.: The closest substring problem with small distances. In: Proceedings of the 46th Annual Symposium on Foundations of Computer Science (FOCS 2005) (2005)
16. Marx, D.: Parameterized graph separation problems. Theoret. Comput. Sci. 351(3), 394–406 (2006)
17. Naor, J., Zosin, L.: A 2-approximation algorithm for the directed multiway cut problem. SIAM J. Comput. 31(2), 477–482 (2001)
18. Yeh, W.-C.: A simple algorithm for the planar multiway cut problem. J. Algorithms 39(1), 68–77 (2001)

Two Sources Are Better Than One for Increasing the Kolmogorov Complexity of Infinite Sequences

Marius Zimand[⋆]

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

Abstract. The randomness rate of an infinite binary sequence is characterized by the sequence of ratios between the Kolmogorov complexity and the length of the initial segments of the sequence. It is known that there is no uniform effective procedure that transforms one input sequence into another sequence with higher randomness rate. By contrast, we display such a uniform effective procedure having as input two independent sequences with positive but arbitrarily small constant randomness rate. Moreover the transformation is a truth-table reduction and the output has randomness rate arbitrarily close to 1.

Keywords: Kolmogorov complexity, Hausdorff dimension.

1 Introduction

It is a basic fact that no function can increase the amount of randomness (i.e., entropy) of a finite structure. Formally, if X is a distribution on a finite set A, then for any function f mapping A into A, the (Shannon) entropy of $f(X)$ cannot be larger than the entropy of X. As it is usually the case, the above fact has an analogue in algorithmic information theory: for any finite binary string x and any computable function f, $K(f(x)) \leq K(x) + O(1)$, where $K(x)$ is the Kolmogorov complexity of x and the constant depends only on the underlying universal machine. The above inequality has an immediate one-line proof, but the analoguous statement when we move to infinite sequences is not known to hold. For any $\sigma \in [0, 1]$, we say that an infinite binary sequence x has randomness rate σ, if $K(x(1 : n)) \geq \sigma n$ for all sufficiently large n, where $x(1 : n)$ denotes the initial segment of x of length n.[1] The question becomes: if x has randomness rate $0 < \sigma < 1$, is there an effective transformation f such that $f(x)$ has randomness rate greater than that of x? Unlike the case of finite strings, infinite sequences

[⋆] The author is supported by NSF grant CCF 0634830. Part of this work was done while visiting University of Auckland, New Zealand.

[1] The randomness rate of x is very close to the notion of constructive Hausdorff dimension of x [Lut03, May02, Rya84, Sta05]; however since this paper is about handling randomness and not about measure-theoretical issues we prefer the randomness terminology.

with positive randomness rate possess an infinite amount of randomness (even though it is sparsely distributed) and thus it cannot be ruled out that there may be a way to concentrate it and obtain a sequence with higher randomness rate.

This is a natural question, first raised by Reimann [Rei04], which has received significant attention recently (it is Question 10.1 in the list of open questions of Miller and Nies [MN06]). So far, there exist several partial results, mostly negative, obtained by restricting the type of transformation. Reimann and Terwijn [Rei04, Th 3.10] have shown that for every constant $c < 1$, there exists a sequence x such that if f is a many-one reduction, then the randomness rate of $f(x)$ cannot be larger than c. This result has been improved by Nies and Reimann [NR06] to wtt-reductions. More precisely, they showed that for all rational $c \in (0,1)$, there exists a sequence x with randomness rate c such that for all wtt-reductions f, $f(x)$ has randomness rate $\leq c$. Bienvenu, Doty, and Stephan [BDS07] have obtained an impossibility result for the general case of Turing reductions, which, however, is valid only for *uniform* reductions. Building on the result of Nies and Reimann, they show that for every Turing reduction f and all constants c_1 and c_2, with $0 < c_1 < c_2 < 1$, there exists x with randomness rate $\geq c_1$ such that $f(x)$, if it exists, has randomness rate $< c_2$. In other words, loosely speaking, no effective uniform transformation is able to raise the randomness rate from c_1 to c_2. Thus the question "Is there any effective transformation that on input $\sigma \in (0,1]$, $\epsilon > 0$, and x, a sequence with randomness rate σ, produces a string y with randomness rate $\sigma + \epsilon$?" has a negative answer. On the positive side, Doty [Dot07] has shown that for every constant c there exists a uniform effective transformation f able to transform any x with randomness rate $c \in (0,1]$ into a sequence $f(x)$ that, for infinitely many n, has the initial segments of length n with Kolmogorov complexity $\geq (1-\epsilon)n$ (see Doty's paper for the exact statement). However, since Doty's transformation f is a wtt-reduction, it follows from Nies and Reimann's result that $f(x)$ also has infinitely many initial segments with no increase in the Kolmogorov complexity.

In the case of finite strings, as we have observed earlier, there is no effective transformation that increases the absolute amount of Kolmogorov complexity. However, some positive results do exist. Buhrman, Fortnow, Newman, and Vereshchagin [BFNV05] show that, for any non-random string of length n, one can flip $O(\sqrt{n})$ of its bits and obtain a string with higher Kolmogorov complexity. Fortnow, Hitchcock, Pavan, Vinodchandran, and Wang [FHP$^+$06] show that for any $0 < \alpha < \beta < 1$, there is a polynomial-time procedure that on input x with $K(x) > \alpha|x|$, using a constant number of advice bits (which depend on x), builds a string y with $K(y) \geq \beta|y|$ and y is shorter than x by only a multiplicative constant.

Our main result concerns infinite sequences and is a positive one. Recall that Bienvenu, Doty and Stephan have shown that there is no uniform effective way to increase the randomness rate when the input consists of *one* sequence with positive randomness rate. We show that if instead the input consists of *two* such sequences that are independent, then such a uniform effective transformation exists.

Theorem 1. *(Main Result) There exists an effective transformation* $f : Q \times \{0,1\}^\infty \times \{0,1\}^\infty \to \{0,1\}^\infty$ *with the following property: If the input is* $\tau \in (0,1]$ *and two independent sequences* x *and* y *with randomness rate* τ*, then* $f(\tau, x, y)$ *has randomness rate* $1 - \delta$*, for all* $\delta > 0$*. Moreover, the effective transformation is a truth-table reduction.*

Effective transformations are essentially Turing reductions that are uniform in the parameter τ; see Section 2. Two sequences are independent if they do not contain much common information; see Section 3.

One key element of the proof is inspired from Fortnow et al.'s [FHP$^+$06], who showed that a randomness extractor can be used to construct a procedure that increases the Kolmogorov complexity of finite strings. Their procedure for increasing the Kolmogorov complexity runs in polynomial time, but uses a small amount of advice. To obtain the polynomial-time efficiency, they had to use the multi-source extractor of Barak, Impagliazzo, and Wigderson [BIW04], which requires a number of sources that is dependent on the initial min-entropy of the sources and on the desired quality of the output. In our case, we are not concerned about the efficiency of the transformation (this of course simplifies our task), but, on the other hand, we want it completely effective (with no advice), we want it to work with just two sources, and we want it to handle infinite sequences. In place of an extractor, we provide a procedure with similar functionality, using the probabilistic method which is next derandomized in the trivial way by brute force searching. Since we handle infinite sequences, we have to iterate the procedure infinitely many times on finite blocks of the two sources and this necessitates solving some technical issues related to the independence of the blocks.

2 Preliminaries

We work over the binary alphabet $\{0,1\}$. A string is an element of $\{0,1\}^*$ and a sequence is an element of $\{0,1\}^\infty$. If x is a string, $|x|$ denotes its length. If x is a string or a sequence and $n, n_1, n_2 \in \mathbf{N}$, $x(n)$ denotes the n-th bit of x and $x(n_1 : n_2)$ is the substring $x(n_1)x(n_1+1)\ldots x(n_2)$. The cardinality of a finite set A is denoted $\|A\|$. Let M be a standard Turing machine. For any string x, define the *(plain) Kolmogorov complexity* of x with respect to M, as

$$K_M(x) = \min\{|p| \mid M(p) = x\}.$$

There is a universal Turing machine U such that for every machine M there is a constant c such that for all x,

$$K_U(x) \leq K_M(x) + c. \tag{1}$$

We fix such a universal machine U and dropping the subscript, we let $K(x)$ denote the Kolmogorov complexity of x with respect to U. For the concept of conditional Komogorov complexity, the underlying machine is a Turing machine that in addition to the read/work tape which in the initial state contains the input p, has a second tape containing initially a string y, which is called the conditioning information. Given such a machine M, we define the Kolmogorov complexity of x conditioned by y with respect to M as

$$K_M(x \mid y) = \min\{|p| \mid M(p, y) = x\}.$$

Similarly to the above, there exist universal machines of this type and they satisfy the relation similar to Equation 1, but for conditional complexity. We fix such a universal machine U, and dropping the subscript U, we let $K(x \mid y)$ denote the Kolmogorov complexity of x conditioned by y with respect to U.

We briefly use the concept of *prefix-free complexity*, which is defined similarly to plain Kolmogorov complexity, the difference being that in the case of prefix-free complexity the domain of the underlying machines is required to be a prefix-free set.

Let $\sigma \in [0, 1]$. A sequence x has randomness rate σ if $K(x(1:n)) \geq \sigma \cdot n$, for almost every n (i.e., the set of n's violating the inequality is finite).

An effective transformation f is represented by a two-oracle Turing machine M_f. The machine M_f has access to two oracles x and y, which are binary sequences. When M_f makes the query "n-th bit of first oracle?" ("n-th bit of second oracle?"), the machine obtains $x(n)$ (respectively, $y(n)$). On input $(\tau, 1^n)$, where τ is a rational (given in some canonical representation), M_f outputs one bit. We say that $f(\tau, x, y) = z \in \{0, 1\}^\infty$, if for all n, M_f on input $(\tau, 1^n)$ and working with oracles x and y halts and outputs $z(n)$. (Effective transformations are more commonly called Turing reductions. If τ would be embedded in the machine M_f, instead of being an input, we would say that z is Turing-reducible to (x, y). Our approach emphasizes the fact that we want a family of Turing reductions that is uniform in the parameter τ.) In case the machine M_f halts on all inputs and with all oracles, we say that f is a truth-table reduction.

3 Independence

We need to require that the two inputs x and y that appear in the main result are really distinct, or in the algorithmic-information theoretical terminology, *independent*.

Definition 1. *Two infinite binary sequences x, y are independent if for all natural numbers n and m,*

$$K(x(1:n)y(1:m)) \geq K(x(1:n)) + K(y(1:m)) - O(\log(n) + \log(m)).$$

The definition says that, modulo additive logarithmic terms, there is no shorter way to describe the concatenation of any two initial segments of x and y than having the information that describes the initial segments.

It can be shown that the fact that x and y are independent is equivalent to saying that for every natural numbers n and m,

$$K(x(1:n) \mid y(1:m)) \geq K(x(1:n)) - O(\log(n) + \log(m)). \tag{2}$$

and

$$K(y(1:m) \mid x(1:n)) \geq K(y(1:m)) - O(\log(n) + \log(m)). \tag{3}$$

Thus, if two sequences x and y are independent, no initial segment of one of the sequence can help in getting a shorter description of any initial segment of the other sequence, modulo additive logarithmical terms. The notion of independence for infinite sequences is extensively studied in [CZ08], where the notion from Definition 1 is called *finitary-independence* to distinguish it from a stronger type of independence.

In our main result, the input consists of two sequences x and y that are independent and that have Kolmogorov rate σ for some positive constant $\sigma < 1$. We sketch an argument showing that such sequences exist. In our sketch we take $\sigma = 1/2$.

We start with an arbitrary random (in the Martin-Löf sense) sequence x. Next using the machinery of Martin-Löf tests relativized with x we infer the existence of a sequence y that is random relative to x. From the theory of Martin-Löf tests, we deduce that there exists a constant c such that for all m, $H(y(1:m) \mid x) \geq m - c$, where $H(\cdot)$ is the prefix-free version of complexity. Since $H(y(1:m)) \leq m + O(\log m)$, for all m, we conclude that $H(y(1:m) \mid x) \geq H(y(1:m)) - O(\log m)$, for all m. Therefore, $H(y(1:m)) \mid x(1:n)) \geq H(y(1:m) \mid x) - O(\log n) \geq H(y(1:m)) - O(\log n + \log m)$, for all n and m. Since the prefix-free complexity $H(\cdot)$ and the plain complexity $K(\cdot)$ are within $O(\log m)$ of each other, it follows that $K(y(1:m)) \mid x(1:n)) \geq K(y(1:m)) - O(\log n + \log m))$, for all n and m. This implies $K(x(1:m)y(1:n)) \geq K(x(1:n)) + K(y(1:m)) - O(\log(n) + \log(m))$, for all n, m. Next we construct x' and y' by inserting in x and respectively y, the bit 0 in all even positions, i.e., $x' = x_1 0 x_2 0 \ldots$ (where x_i is the i-th bit of x) and $y' = y_1 0 y_2 0 \ldots$. Clearly, $K(x(1:n))$ and $K(x_1 0 \ldots x_n 0)$ are within a constant of each other, and the same holds for y and y'. It follows that x' and y' are independent and have randomness rate 1/2.

4 Proof of Main Result

4.1 Proof Overview

We present in a simplified setting the main ideas of the construction. Suppose we have two independent strings x and y of length n such that $K(x) = \sigma n$ and $K(y) = \sigma n$, for some $\sigma > 0$. We want to construct a string z of length m such that $K(z) > (1-\epsilon)m$. The key idea (borrowed from the theory of randomness extractors) is to use a function $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ such that every large enough rectangle of $\{0,1\}^n \times \{0,1\}^n$ maps about the same number of pairs into all elements of $\{0,1\}^m$. We say that such a function is *regular* (the formal Definition 2 has some parameters which quantify the degree of regularity). To illustrate the idea, suppose for a moment that we have a function $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ that, for all subsets $B \subseteq \{0,1\}^n$ with $\|B\| \approx 2^{\sigma n}$, has the property that any $a \in \{0,1\}^m$ has the same number of preimages in $B \times B$, which is of course $\|B \times B\|/2^m$. Then for any $A \subseteq \{0,1\}^m$, $E^{-1}(A) \cap (B \times B)$ has size $\frac{\|B \times B\|}{2^m} \cdot \|A\|$. Let us take $z = E(x, y)$ and let us suppose that $K(z) < (1-\epsilon)m$. Note that the set $B = \{u \in \{0,1\}^n \mid K(u) = \sigma n\}$ has size $\approx 2^{\sigma n}$, the set $A = \{v \in \{0,1\}^m \mid K(v) < (1-\epsilon)m\}$ has size $< 2^{(1-\epsilon)m}$ and that x and y are in $E^{-1}(A) \cap (B \times B)$. By the above observation the set $E^{-1}(A) \cap (B \times B)$ has size $\leq \frac{2^{\sigma n} \cdot 2^{\sigma n}}{2^{\epsilon m}}$. Since $E^{-1}(A) \cap (B \times B)$ can be enumerated effectively, any pair of strings in $E^{-1}(A) \cap B \times B$ can be described by its rank in a fixed enumeration of $E^{-1}(A) \cap B \times B$. In particular (x, y) is such a pair and therefore $K(xy) \leq 2\sigma n - \epsilon m$. On the other hand, since x and y are independent, $K(xy) \approx K(x) + K(y) = 2\sigma n$. The contradiction we have reached shows that in fact $K(z) \geq (1-\epsilon)m$.

A function E having the strong regularity requirement stated above may not exist. Fortunately, using the probabilistic method, it can be shown (see Section 4.3) that, for all $m \leq n^{0.99\sigma}$, there exist a function $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ such that all strings $a \in \{0,1\}^m$ have at most $2(\|B \times B\|/2^m)$ preimages in any $B \times B$ as above (instead of $(\|B \times B\|/2^m)$ preimages in the ideal, but not realizable, setting we used above). Once we know that it exists, such a function E can be found effectively by exhaustive search. Then the argument above, with some minor modifications, goes through. In fact, when we apply this idea, we only know that $K(x) \geq \sigma n$ and $K(y) \geq \sigma n$ and therefore we need the function E to satisfy a stronger variant of regularity. However, the main idea remains the same.

Thus there is an effective way to produce a string z with Kolmogorov complexity $(1-\epsilon)m$ from two independent strings x and y of length n and with Kolmogorov complexity σn. Recall that, in fact, the input consists of two independent *infinite* sequences x and y with randomness rate $\tau > 0$. To take advantage of the procedure sketched above which works for finite strings, we split x and y into finite strings $x_1, x_2, \ldots, x_n, \ldots$, and respectively $y_1, y_2, \ldots, y_n, \ldots$, such that the blocks x_i and y_i, of length n_i, have still enough Kolmogorov complexity, say $(\tau/2)n_i$, conditioned by the previous blocks $x_1, \ldots, x_{i-1}$ and $y_1, \ldots, y_{i-1}$. The splitting of x and y into blocks and the properties of the blocks are presented in Section 4.2. Then using a regular function $E_i : \{0,1\}^{n_i} \times \{0,1\}^{n_i} \to \{0,1\}^{m_i}$, we build $z_i = E_i(x_i, y_i)$. By modifying slightly the argument described above, it can be shown that $K(z_i \mid x_1, \ldots, x_{i-1}, y_1, \ldots, y_{i-1}) > (1-\epsilon)m_i$, i.e., z_i has high Kolmogorov complexity even conditioned by the previous blocks $x_1, \ldots, x_{i-1}$ and $y_1, \ldots, y_{i-1}$. It follows that $K(z_i \mid z_1, \ldots, z_{i-1})$ is also close to m_i. We finally take $z = z_1 z_2 \ldots$, and using the above property of each z_i, we infer that for every n, the prefix of z of length n has randomness rate $> (1-\epsilon)n$. In other words, z has randomness rate $(1-\epsilon)$, as desired.

4.2 Splitting the Two Inputs

The two input sequences x and y from Theorem 1 are broken into finite blocks $x_1, x_2, \ldots, x_i, \ldots$ and respectively $y_1, y_2, \ldots, y_i, \ldots$. The division is done in such a manner that x_i (respectively, y_i) has high Komogorov complexity rate conditioned by the previous blocks $x_1, \ldots, x_{i-1}$ (respectively by the blocks $y_1, \ldots, y_{i-1}$). The following lemma shows how this division is done.

Lemma 1. *(Splitting lemma) Let $x \in \{0,1\}^\infty$ with randomness rate τ, for some constant $\tau > 0$. Let $0 < \sigma < \tau$. For any n_0 sufficiently large, there is $n_1 > n_0$ such that*

$$K(x(n_0+1 : n_1) \mid x(1 : n_0)) > \sigma(n_1 - n_0).$$

Furthermore, there is an effective procedure that on input n_0, τ and σ calculates n_1.

Proof. Let σ' be such that $0 < \sigma' < \tau - \sigma$. Take $n_1 = \lceil \frac{1-\sigma}{\sigma'} \rceil n_0$. Suppose $K(x(n_0+1 : n_1) \mid x(1 : n_0)) \leq \sigma(n_1 - n_0)$. Then $x(1 : n_1)$ can be reconstructed from: $x(1 : n_0)$, the description of $x(n_0+1 : n_1)$ given $x(1 : n_0)$, n_0, $\log n_0$ bits for delimiting these pieces of information, extra constant number of bits describing the procedure. So

$$\begin{aligned} K(x(1 : n_1)) &\leq n_0 + \sigma(n_1 - n_0) + 2\log n_0 + O(1) \\ &= \sigma n_1 + (1-\sigma)n_0 + 2\log n_0 + O(1) \\ &\leq \sigma n_1 + \sigma' n_1 + 2\log n_0 + O(1) \\ &< \tau n_1 \text{ (if } n_0 \text{ suffic. large)}\ , \end{aligned} \tag{4}$$

which is a contradiction if n_1 is sufficiently large. □

Now we define the points where we split x and y, the two sources. Take a, the point from where the Splitting Lemma holds. For the rest of this section we consider $b = \lceil \frac{1-\sigma}{\sigma'} \rceil$. The following sequence represents the cutting points that will define the blocks. It is defined recursively, as follows: $t_0 = 0$, $t_1 = a$, $t_i = b(t_1 + \ldots + t_{i-1})$. It can be seen that $t_i = ab(1+b)^{i-2}$, for $i \geq 2$. Finally, we define the blocks: for each $i \geq 1$, $x_i := x(t_{i-1}+1 : t_i)$ and $y_i = y(t_{i-1}+1 : t_i)$, and $n_i := |x_i| = |y_i| = ab^2(1+b)^{i-3}$ (the last equality holds for $i \geq 3$). We also denote by $\bar{x}_i$ the concatenation of the blocks $x_1, \ldots, x_i$ and by $\bar{y}_i$ the concatenation of the blocks $y_1, \ldots, y_i$.

Lemma 2

1. *$K(x_i \mid \bar{x}_{i-1}) > \sigma n_i$, for all $i \geq 2$ (and the analogue relation holds for the y_i's).*
2. *$\log |x_i| = \Theta(i)$ and $\log |\bar{x}_i| = \Theta(i)$, for all i (and the analogue relation holds for the y_i's).*

Proof. The first point follows from the Splitting Lemma 1, and the second point follows immediately from the definition of n_i (which is the length of x_i) and of t_i (which is the length of $\bar{x}_i$). □

The following facts state some basic algorithmic-information theoretical properties of the blocks $x_1, x_2, \ldots.$ and $y_1, y_2, \ldots.$ The proof is available in the full version of this paper; it is based on the Symmetric Information Theorem $|K(vu) - (K(u) + K(v \mid u))| \leq O(\log K(u) + \log K(v))$ (for example, see Alexander Shen's lecture notes [She00]).

Lemma 3. *For all i and j, (a) $\left|K(x_i \mid \bar{x}_{i-1}\bar{y}_j) - K(x_i \mid \bar{x}_{i-1})\right| < O(i + j)$. (b) $\left|K(y_i \mid \bar{x}_j\bar{y}_{i-1}) - K(y_i \mid \bar{y}_{i-1})\right| < O(i + j)$. (c) $K(x_iy_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) \geq K(x_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) + K(y_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) - O(i)$.*

4.3 Regular Functions

The construction of z from x and y proceeds block-wise: we take as inputs the blocks x_i and y_i and, from them, we build z_i, the i-th block of z. The input strings x_i and y_i, both of length n_i, have Kolmogorov complexity σn_i, for some positive constant σ, and the goal is to produce z_i, of length m_i (which will be specified later), with Kolmogorov complexity (even conditioned by $z_1 z_2 \dots z_{i-1}$) at least $(1-\epsilon)m_i$, for positive ϵ arbitrarily small. This resembles the functionality of randomness extractors and, indeed, the following definition captures a property similar to that of extractors that is sufficient for our purposes.

Definition 2. *A function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ is (σ, c)-regular, if for any $k_1, k_2 \geq \sigma n$, any two subsets $B_1 \subseteq \{0,1\}^n$ and $B_2 \subseteq \{0,1\}^n$ with $\|B_1\| = 2^{k_1}$ and $\|B_2\| = 2^{k_2}$ have the following property: for any $a \in \{0,1\}^m$,*

$$\|f^{-1}(a) \cap (B_1 \times B_2)\| \leq \frac{c}{2^m}\|B_1 \times B_2\|.$$

Remarks: Let $[N]$ be the set $\{1, \dots, N\}$. We identify in the standard way $\{0,1\}^n$ with $[N]$, where $N = 2^n$. We can view $[N] \times [N]$ as a table with N rows and N columns and a function $f : [N] \times [N] \rightarrow [M]$ as an assignment of a color chosen from $[M]$ to each cell of the table. The function f is (σ, c)-regular if in any rectangle of size $[K] \times [K]$, with $k \geq \sigma n$, no color appears more than a fraction of c/M times. (The notion of regularity is interesting for small values of c because it says that in all rectangles, unless they are small, all the colors appear approximately the same number of times; note that if $c = 1$, then all the colors appear the same number of times.)

We show using the probabilistic method that for any $\sigma > 0$, $(\sigma, 2)$-regular functions exist. Since the regularity property for a function f (given via its truth table) can be effectively tested, we can effectively construct $(\sigma, 2)$- regular functions by exhaustive search

We take $f : [N] \times [N] \rightarrow [M]$, a random function. First we show that with positive probability such a function satisfies the definition of regularity for sets A and B having size 2^k, where k is *exactly* $\lceil \sigma n \rceil$. Let's temporarily call this property the *weak regularity* property. We will show that in fact weak regularity implies the regularity property as defined above (i.e., the regularity should hold for all sets B_1 and B_2 of size 2^{k_1} and respectively 2^{k_2}, for k_1 and k_2 *greater or equal* $\lceil \sigma n \rceil$).

Lemma 4. *For every $\sigma > 0$ and for every N that is sufficiently large, if $M \leq N^{0.99\sigma}$, then it holds with probability > 0 that f satisfies the $(\sigma, 2)$- weak regularity property as defined above.*

Proof. Fix $B_1 \subseteq [N]$ with $\|B_1\| = N^\sigma$ (to keep the notation simple, we ignore truncation issues). Fix $B_2 \subseteq [N]$ with $\|B_2\| = N^\sigma$. Let $j_1 \in B_1 \times B_2$ and $j_2 \in [M]$ be fixed values. As discussed above, we view $[N] \times [N]$ as a table with N rows and N columns. Then $B_1 \times B_2$ is a rectangle in the table, j_1 is a cell in the rectangle, and j_2 is a color out of M possible colors. Clearly, $\mathrm{Prob}(f(j_1) = j_2) = 1/M$. By Chernoff bounds,

$$\mathrm{Prob}\Bigg(\left(\frac{\text{no. of } j_2\text{-colored cells in } B_1 \times B_2}{N^\sigma \cdot N^\sigma} - \frac{1}{M}\right) > \frac{1}{M}\Bigg) < e^{-(1/M)\cdot N^\sigma \cdot N^\sigma \cdot (1/3)}.$$

By the union bound

$$\mathrm{Prob}(\text{ the above holds for some } j_2 \text{ in } [M]\ ) < Me^{-(1/M)\cdot N^\sigma \cdot N^\sigma \cdot (1/3)}. \qquad (5)$$

The number of rectangles $B_1 \times B_2$ is

$$\tbinom{N}{N^\sigma} \cdot \tbinom{N}{N^\sigma} \leq \left(\left(\tfrac{eN}{N^\sigma}\right)^{N^\sigma}\right)^2 = e^{2N^\sigma} \cdot e^{2N^\sigma \cdot (1-\sigma)\ln N}. \qquad (6)$$

Note that if there is no rectangle $B_1 \times B_2$ and j_2 as above, then f satisfies the weaker $(\sigma, 2)$-regularity property. Therefore we need that the product of the right hand sides in equations (5) and (6) is < 1. This is equivalent to

$$(1/M) \cdot N^{2\sigma} \cdot 1/3 - \ln(M) > 2N^\sigma + 2N^\sigma \cdot (1-\sigma)\ln N,$$

which holds true for $M \leq N^{0.99\sigma}$, and N sufficiently large. $\square$

As promised, we show next that weak regularity implies regularity.

Lemma 5. *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ such that for every $B_1 \subseteq \{0,1\}^n$ with $\|B_1\| = 2^k$, for every $B_2 \subseteq \{0,1\}^n$ with $\|B_2\| = 2^k$, and for every $a \in \{0,1\}^m$ it holds that $\|f^{-1}(a) \cap (B_1 \times B_2)\| \leq p$.*

Then for every $k_1 \geq k$ and every $k_2 \geq k$, for every $B_1' \subseteq \{0,1\}^n$ with $\|B_1'\| = 2^{k_1}$, for every $B_2' \subseteq \{0,1\}^n$ with $\|B_2'\| = 2^{k_2}$, and for every $a \in \{0,1\}^m$ it holds that $\|f^{-1}(a) \cap (B_1' \times B_2')\| \leq p$.

Proof. We partition B_1' and B_2' into subsets of size 2^k. So, $B_1' = A_1 \cup A_2 \cup \ldots \cup A_s$, with $\|A_i\| = 2^k$, $i = 1, \ldots, s$ and $B_2 = C_1 \cup C_2 \cup \ldots \cup C_t$, with $\|C_j\| = 2^k$, $j = 1, \ldots, t$. Then,

$$\begin{aligned}\|f^{-1}(a) \cap (B_1' \times B_2')\| &= \textstyle\sum_{i=1}^s \sum_{j=1}^t \|f^{-1}(a) \cap (A_i \times C_j)\| \\ &\leq \textstyle\sum_{i=1}^s \sum_{j=1}^t p \cdot \|A_i \times C_j\| \\ &= p \cdot \|B_1' \times B_2'\|.\end{aligned}$$

4.4 Increasing the Randomness Rate

We proceed to the proof of our main result, Theorem 1. We give a “global” description of the effective mapping $f : Q \times \{0,1\}^\infty \times \{0,1\}^\infty \to \{0,1\}^\infty$. It will be clear how to obtain the n-th bit of the output in finitely many steps, as it is formally required.

Construction

Input: $\tau \in Q \cap (0,1]$, $x, y \in \{0,1\}^\infty$ (the sequences x and y are oracles to which the procedure has access).
Step 1: Split x into $x_1, x_2, \ldots, x_i, \ldots$ and split y into $y_1, y_2, \ldots, y_i, \ldots$, as described in Section 4.2 taking $\sigma = \tau/2$ and $\sigma' = \tau/4$.
For each i, let $|x_i| = |y_i| = n_i$ (as described in Section 4.2).
By Lemma 2, $K(x_i \mid \bar{x}_{i-1}) > \sigma n_i$ and $K^x(y_i \mid \bar{y}_{i-1}) > \sigma n_i$.
Step 2: As discussed in Section 4.3, for each i, construct by exhaustive search $E_i : \{0,1\}^{n_i} \times \{0,1\}^{n_i} \to \{0,1\}^{m_i}$ a $(\sigma/2, 2)$-regular function, where $m_i = i^2$.
We recall that this means that for all $k_1, k_2 \geq (\sigma/2)n_i$, for all $B_1 \subseteq \{0,1\}^{n_i}$ with $\|A\| \geq 2^{k_1}$, for all $B_2 \subseteq \{0,1\}^{n_i}$ with $\|B_2\| \geq 2^{k_2}$, and for all $a \in \{0,1\}^{m_i}$,

$$\|E_i^{-1}(a) \cap B_1 \times B_2\| \leq \frac{2}{2^{m_i}} \|B_1 \times B_2\|.$$

We take $z_i = E_i(x_i, y_i)$.
Finally $z = z_1 z_2 \ldots z_i \ldots$.

It is obvious that the above procedure is a truth-table reduction (i.e., it halts on all inputs).

In what follows we will assume that the two input sequences x and y have randomness rate τ and our goal is to show that the output z has randomness rate $(1 - \delta)$ for any $\delta > 0$.

Lemma 6. *For any $\epsilon > 0$, for all i sufficiently large, $K(z_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) \geq (1 - \epsilon) \cdot m_i$.*

Proof. Suppose $K(z_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) < (1 - \epsilon) \cdot m_i$.

Let $A = \{z \in \{0,1\}^{m_i} \mid K(z \mid \bar{x}_{i-1}\bar{y}_{i-1}) < (1 - \epsilon) \cdot m_i\}$. We have $\|A\| < 2^{(1-\epsilon)m_i}$.

Let t_1, t_2, B_1, B_2 be defined as follows: $t_1 = K(x_i \mid \bar{x}_{i-1}\bar{y}_{i-1})$, $t_2 = K(y_i \mid \bar{x}_{i-1}\bar{y}_{i-1})$, $B_1 = \{x \in \{0,1\}^{n_i} \mid K(x \mid \bar{x}_{i-1}\bar{y}_{i-1}) \leq t_1\}$, and $B_2 = \{y \in \{0,1\}^{n_i} \mid K(y \mid \bar{x}_{i-1}\bar{y}_{i-1}) \leq t_2\}$.

Since $K(x_i \mid \bar{x}_{i-1}) > \sigma n_i$, and taking into account Lemma 3, it follows that $t_1 > \sigma n_i - O(i) > (\sigma/2)n_i$, for all i sufficiently large. By the same argument as above, $t_2 > (\sigma/2)n_i$. We have $\|B_1\| \leq 2^{t_1+1}$. Take B_1' such that $\|B_1'\| = 2^{t_1+1}$ and $B_1 \subseteq B_1'$. We also have $\|B_2\| \leq 2^{t_2+1}$. Take B_2' such that $\|B_2'\| = 2^{t_2+1}$ and $B_2 \subseteq B_2'$. The bounds on t_1 and t_2 imply that B_1' and B_2' are large enough for E_i to satisfy the regularity property on them. In other words, for any $a \in \{0,1\}^{m_i}$, $\|E_i^{-1}(a) \cap B_1' \times B_2'\| \leq \frac{2}{2^{m_i}} \|B_1' \times B_2'\|$. So,

$$\begin{aligned}
\|E_i^{-1}(A) \cap B_1 \times B_2\| &\leq \|E_i^{-1}(A) \cap B_1' \times B_2'\| \\
&= \textstyle\sum_{a \in A} \|E_i^{-1}(a) \cap B_1' \times B_2'\| \\
&\leq 2^{(1-\epsilon)m_i} \tfrac{2}{2^{m_i}} \|B_1' \times B_2'\| \\
&\leq 2^{t_1 + t_2 - \epsilon m_i + 3}.
\end{aligned}$$

There is an algorithm that, given $(x_1, x_2, \ldots, x_{i-1})$, $(y_1, y_2, \ldots, y_{i-1})$, $(1-\epsilon)m_i$, t_1 and t_2, enters an infinite loop during which it enumerates the elements of the set $E_i^{-1}(A) \cap B_1 \times B_2$. Therefore, the Kolmogorov complexity of any element of $E_i^{-1}(A) \cap B_1 \times B_2$ is bounded by its rank in some fixed enumeration of this set, the binary encoding of the input (including the information needed to separate the different components), plus a constant number of bits describing the enumeration procedure.

Formally, for every $(u, v) \in E_i^{-1}(A) \cap B_1 \times B_2$,

$$K(uv|\bar{x}_{i-1}\bar{y}_{i-1}) \le t_1 + t_2 - \epsilon m_i + 2(\log(1-\epsilon)m_i + \log t_1 + \log t_2) + O(1) = t_1 + t_2 - \Omega(i^2).$$

We took into account that $m_i = i^2$, $\log t_1 = O(i)$, and $\log t_2 = O(i)$. In particular,

$$K(x_i y_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) \le t_1 + t_2 - \Omega(i^2).$$

On the other hand, by Lemma 3,

$$\begin{aligned} K(x_i y_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) &\ge K(x_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) + K(y_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) - O(i) \\ &= t_1 + t_2 - O(i). \end{aligned}$$

The last two inequations are in conflict, and thus we have reached a contradiction. □

The following lemma concludes the proof of the main result.

Lemma 7. *For any $\delta > 0$, the sequence z obtained by concatenating in order $z_1, z_2, \ldots$, has randomness rate at least $1 - \delta$.*

Proof. Take $\epsilon = \delta/4$. By Lemma 6, $K(z_i \mid \bar{x}_{i-1}\bar{y}_{i-1}) \ge (1-\epsilon) \cdot m_i$, for all i sufficiently large. This implies $K(z_i \mid z_1 \ldots z_{i-1}) > (1-\epsilon)m_i - O(1) > (1-2\epsilon)m_i$ (because each z_j can be effectively computed from x_j and y_j). By induction, it can be shown that $K(z_1 \ldots z_i) \ge (1-3\epsilon)(m_1 + \ldots + m_i)$. For the inductive step, we have

$$\begin{aligned} K(z_1 z_2 \ldots z_i) &\ge K(z_1 \ldots z_{i-1}) + K(z_i \mid z_1 \ldots z_{i-1}) - O(\log(m_1 + \ldots + m_{i-1}) + \log(m_i)) \\ &\ge (1-3\epsilon)(m_1 + \ldots + m_{i-1}) + (1-2\epsilon)m_i - O(\log(m_1 + \ldots + m_i)) \\ &> (1-3\epsilon)(m_1 + \ldots + m_i). \end{aligned}$$

Now consider some z' which is between $z_1 \ldots z_{i-1}$ and $z_1 \ldots z_i$, i.e., for some strings u and v, $z' = z_1 \ldots z_{i-1}u$ and $z_1 \ldots z_i = z'v$. Suppose $K(z') < (1-4\epsilon)|z'|$. Then $z_1 \ldots z_{i-1}$ can be reconstructed from:

(a) the descriptor of z', which takes $(1-4\epsilon)|z'| \le (1-4\epsilon)(m_1 + \ldots + m_i)$ bits,
(b) $O(1)$ bits for describing the reconstruction procedure.

This implies that

$$\begin{aligned} K(z_1 \ldots z_{i-1}) &\le (1-4\epsilon)(m_1 + \ldots + m_i) + O(1) \\ &= (1-4\epsilon)(m_1 + \ldots + m_{i-1}) + (1-4\epsilon)m_i + O(1) \\ &= \left(1 - 4\epsilon + (1-4\epsilon)\frac{m_i}{m_1 + \ldots + m_{i-1}}\right) \cdot (m_1 + \ldots + m_{i-1}) + O(1) \\ &< (1-3\epsilon)(m_1 + \ldots + m_{i-1}). \end{aligned}$$

(The last inequality holds if $\frac{m_i}{m_1 + \ldots + m_{i-1}}$ goes to 0, which is true for $m_i = i^2$.) This is a contradiction.

Thus we have proved that for every n sufficiently large, $K(z(1:n)) > (1-\delta)n$. □

The main result can be stated in terms of constructive Hausdorff dimension, a notion introduced in measure theory. The constructive Hausdorff dimension of a sequence $x \in \{0,1\}^\infty$ turns out to be equal to $\liminf \frac{K(x(1:n))}{n}$(see [May02, Rya84, Sta05]).

Corollary 1. *For any $\tau > 0$, there is a truth-table reduction f such that if $x \in \{0,1\}^\infty$ and $y \in \{0,1\}^\infty$ are independent and have constructive Hausdorff dimension at least τ, then $f(x, y)$ has Hausdorff dimension 1. Moreover, f is uniform in the parameter τ.*

We next observe that Theorem 1 can be strengthened by relaxing the requirement regarding the independence of the two input sequences. For a function $g : \mathbb{N} \to \mathbb{R}^+$, we say that two sequences $x \in \{0,1\}^\infty$ and $y \in \{0,1\}^\infty$ have dependency g, if for all natural numbers n and m,

$$K(x(1:n)) + K(y(1:m)) - K(x(1:n)y(1:m)) \leq O(g(n) + g(m)).$$

In Theorem 1, the assumption is that the two input sequences have dependency $g(n) = \log n$. Using essentially the same proof as the one that demonstrated Theorem 1, one can obtain the following result.

Theorem 2. *For any $\tau > 0$, there exist $0 < \alpha < 1$ and a truth-table reduction $f : \{0,1\}^\infty \times \{0,1\}^\infty \to \{0,1\}^\infty$ such that if $x \in \{0,1\}^\infty$ and $y \in \{0,1\}^\infty$ have dependency n^α and randomness rate τ, then $f(x, y)$ has randomness rate $1 - \delta$, for any positive δ. Moreover, f is uniform in the parameter τ.*

In Theorem 1 it is required that the initial segments of x and y have Kolmogorov complexity at least $\tau \cdot n$, for a positive constant τ. We do not know if it is possible to obtain a similar result for sequences with lower Kolmogorov complexity. However, using the same proof technique, it can be shown that if x and y have their initial segments with Kolmogorov complexity only $\Omega(\log n)$, then one can produce an infinite sequence z that has very high Kolmogorov complexity for infinitely many of its prefixes.

Theorem 3. *For any $\delta > 0$, there exist a constant C and a truth-table reduction $f : \{0,1\}^\infty \times \{0,1\}^\infty \to \{0,1\}^\infty$ with the following property:*

If the input sequences x and y are independent and satisfy $K(x(1:n)) > C \cdot \log n$ and $K(y(1:n)) > C \cdot \log n$, for every n, then the output $z = f(x, y)$ satisfies $K(z(1:n)) > (1 - \delta) \cdot n$, for infinitely many n. Furthermore, there is an infinite computable set S, such that $K(z(1:n)) > (1 - \delta) \cdot n$, for every $n \in S$.

Acknowledgments

The author thanks Ted Slaman for bringing to his attention the problem of extracting Kolmogorov complexity from infinite sequences, during the Dagstuhl seminar on Kolmogorov complexity in January 2006. The author is grateful to Cristian Calude for insightful discussions.

References

[BDS07] Bienvenu, L., Doty, D., Stephan, F.: Constructive dimension and weak truth-table degrees. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, Springer, Heidelberg (to appear, 2007); Available as Technical Report arXiv:cs/0701089 ar arxiv.org

[BFNV05] Buhrman, H., Fortnow, L., Newman, I., Vereshchagin, N.: Increasing Kolmogorov complexity. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 412–421. Springer, Heidelberg (2005)

[BIW04] Barak, B., Impagliazzo, R., Wigderson, A.: Extracting randomness using few independent sources. In: Proceedings of the 36th ACM Symposium on Theory of Computing, pp. 384–393 (2004)

[CZ08] Calude, C., Zimand, M.: Algorithmically independent sequences, CORR Technical report arxiv:0802-0487 (2008)

[Dot07] Doty, D.: Dimension extractors and optimal decompression. Technical Report arXiv:cs/0606078, Computing Research Repository, arXiv.org, Theory of Computing Systems (to appear, May 2007)

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

[Lut03] Lutz, J.: The dimensions of individual strings and sequences. Information and Control 187, 49–79 (2003)

[May02] Mayordomo, E.: A Kolmogorov complexity characterization of constructive Hausdorff dimension. Information Processing Letters 84, 1–3 (2002)

[MN06] Miller, J., Nies, A.: Randomness and computability. Open questions. Bull. Symb. Logic 12(3), 390–410 (2006)

[NR06] Nies, A., Reimann, J.: A lower cone in the wtt degrees of non-integral effective dimension. In: Proceedings of IMS workshop on Computational Prospects of Infinity, Singapore (to appear, 2006)

[Rei04] Reimann, J.: Computability and fractal dimension. Technical report, Universität Heidelberg, Ph.D. thesis (2004)

[Rya84] Ryabko, B.: Coding of combinatorial sources and Hausdorff dimension. Doklady Akademii Nauk SSR 277, 1066–1070 (1984)

[She00] Shen, A.: Algorithmic information theory and Kolmogorov complexity. Technical Report 2000-034, Uppsala Universitet (December 2000)

[Sta05] Staiger, L.: Constructive dimension equals Kolmogorov complexity. Information Processing Letters 93, 149–153 (2005); Preliminary version: Research Report CDMTCS-210, Univ. of Auckland (January 2003)

Multilayer Neuro-fuzzy Network for Short Term Electric Load Forecasting

Yevgeniy Bodyanskiy, Sergiy Popov, and Taras Rybalchenko

Control Systems Research Laboratory
Kharkiv National University of Radio Electronics
bodya@kture.kharkov.ua, serge.popov@gmx.net

Abstract. The problem of short term electric load forecasting is considered in the case when a part of input variables is given in a nonnumeric form. Novel neuro-fuzzy network architecture and learning algorithms are proposed, which enable high-rate processing of information given in different measurements scales (quantitative, ordinal, and nominal). Types and parameters of the employed membership functions may be determined by the amount of available explicit prior knowledge. Experimental comparison to a traditional neural network confirms superiority of the proposed approach.

1 Introduction

Short term electric load forecasting (STLF) is an integral part of management and planning in power supply companies. Currently, two approaches to STLF problems are widely employed: traditional forecasting techniques (regression, correlation, spectral analysis, Box-Jenkins method, exponential smoothing, adaptive predictors, etc.) and advanced techniques based on artificial intelligence and data mining methods. Traditional forecasting techniques have advantages of simplicity of use and wide range of available software implementations. However, because electric load often depends on influencing factors in a complicated nonlinear manner, it is not always possible to achieve acceptable forecasting accuracy.

A good alternative to traditional approaches is to apply computational intelligence methods, first of all, artificial neural networks and fuzzy inference systems. Their efficiency is based on universal approximation capabilities and ability to learn during the forecasting process. These methods have proven their efficiency in solution of a wide range of problems related to forecasting in power systems [1-12]. The use of computational intelligence techniques may be complicated when part of the information is given not in a quantitative form, but in the ordinal or nominal scale. Traditional neural or neuro-fuzzy networks are poorly suited for processing of the information given in a form like “bad, normal, good weather”, “weak or strong wind”, “cloudy, fog, clear”, etc.

This paper addresses the problem of synthesis of a forecasting neuro-fuzzy network, which is able to process information given in different scales, and its learning algorithm with high convergence rate and ability to process information in real time.

2 Architecture of Forecasting Neuro-fuzzy Network

The proposed neuro-fuzzy network has a four-layer feedforward architecture depicted in fig. 1.

From the receptive (zero) layer information is fed to the first hidden layer of delays and input signals fuzzification. Here, the forecasted signal's history is formed along with membership functions of factors, given in different measurement scales. From this layer's output, information in numeric form is fed to the second and third hidden layers formed by elementary Rosenblatt-type neurons. The output layer is formed by a single neuron with nonlinear activation function, which produces the forecast.

The following information is given to the input of the first hidden layer:

- quantitative variables:
  - current value of the forecasted signal $y(k)$ (here $k = 0, 1, 2,..., N$ is a discrete time, N – data set length);
  - air temperature;
- ordinal variables:
  - relative humidity in the form «low – medium – high»;
  - wind speed in the form «calm – weak – strong – storm»;
  - cloudiness in the form «clear – light – heavy»;
  - hour index: *0, 1, 2, ..., 23*;
  - day of week in the form «Monday – Tuesday – ... – Sunday»;
- nominal variables:
  - type of day in the form «weekday – weekend – holiday – regional holiday»;
  - type of weather in the form «fair – fog – rain – snow».

At first, variables are scaled to the interval [0, 1] by the following transformation:

$$\tilde{x}_l = \frac{x_l - x_{l\min}}{x_{l\max} - x_{l\min}},$$

$$x_l = x_{l\max}\tilde{x}_l - x_{l\min}\left(\tilde{x}_l - 1\right),$$

where x_l – is the value of the l-th input variable in the original scale: MWh, °C; $\tilde{x}_l$ – scaled value of the l-th input variable; $x_{l\min}, x_{l\max}$ – minimum and maximum values of the l-th input variable in original scale.

Then, in the first hidden layer using delay elements z^{-1}, the forecasted signal's history is formed: $y(k-1)$, $y(k-2)$, $y(k-24)$, $y(k-48)$, $y(k-168)$, $y(k-336)$, which is fed to the second hidden layer in the form $o_1^{[1]}(k)$, $o_2^{[1]}(k)$, $o_3^{[1]}(k)$, $o_4^{[1]}(k)$, $o_5^{[1]}(k)$, $o_6^{[1]}(k)$. Different delays may be used for other forecasting horizons.

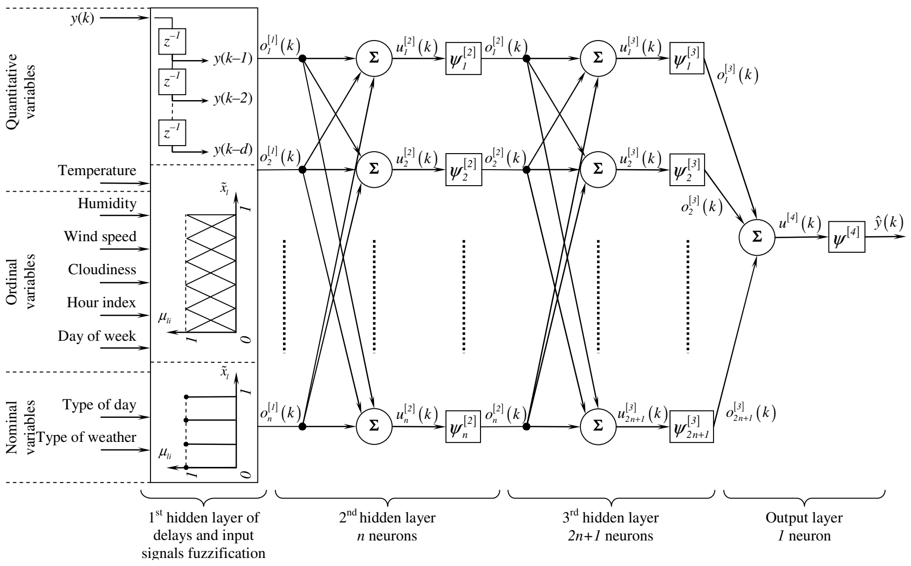


Fig. 1. Forecasting neuro-fuzzy network architecture

In the same layer, air temperature, hour index, relative humidity, wind speed, cloudiness and type of day are fuzzified using triangular membership functions uniformly distributed in the interval [0, 1]:

$$\mu_{l1} = \frac{c_{l2} - \tilde{x}_l}{c_{l2}}, \tilde{x}_l \in [0, c_{l2}],$$

$$\mu_{li} = \begin{cases} \dfrac{\tilde{x}_l - c_{l,i-1}}{c_{li} - c_{l,i-1}}, \tilde{x}_l \in \left[c_{l,i-1}, c_{li}\right], \\ \dfrac{c_{l,i+1} - \tilde{x}_l}{c_{l,i+1} - c_{li}}, \tilde{x}_l \in \left[c_{li}, c_{l,i+1}\right], \\ i = 2, \ldots, p_l - 1, \end{cases}$$

$$\mu_{l,p_l} = \frac{\tilde{x}_l - c_{l,p_l-1}}{1 - c_{l,p_l-1}}, \tilde{x}_l \in \left[c_{l,p_l-1}, 1\right],$$

where c_{li} – position of the center of the i-th membership function of the l-th variable, p_l – number of membership functions of the l-th variable. Triangular membership functions are employed because of their computational simplicity and easy satisfaction of condition $\sum_{i=1}^{p_l} \mu_{li}\left(\tilde{x}_l\right) = 1, \forall \tilde{x}_l$.

Fig. 2 shows fuzzification of air temperature with seven triangular membership functions, temperatures lying outside of the interval [–30°C, 30°C] are projected onto

[0, 1]. The choice of the shape and centers of membership functions may be subject to specific weather conditions at the location, for which the STLF problem is being solved. This will require, for instance, explicit knowledge of temperature sensitivity zones that may be hard to obtain. In this paper we only demonstrate the possibility of such fuzzification, therefore the simplest (uniform) option is shown.

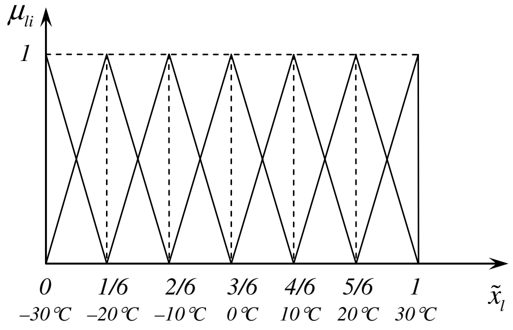


Fig. 2. Air temperature fuzzification

The inputs to the second hidden layer are obtained by fuzzification of variables in the first hidden layer as shown in fig. 3.

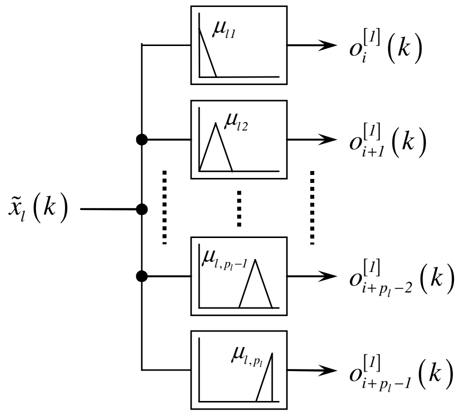


Fig. 3. Fuzzification of inputs to the second hidden layer

Fuzzification of nominal variables is performed in the same manner except for the use of singleton membership functions as shown in fig. 4. Basically, singleton membership functions provide a transparent means of conversion of a single nominal variable into a set of independent numeric flags.

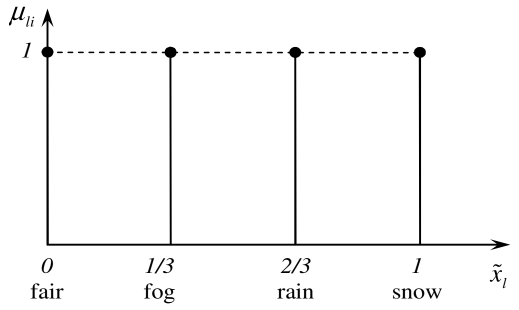


Fig. 4. Fuzzification of the type of weather

Thus, fuzzification with different types of membership functions provides a consistent method for conversion of nonnumeric variables in a quantitative form. The choice of a particular set of membership functions for a given variable depends on availability of explicit knowledge about the variable's physical interpretation and relations between its states. When the states of the variable are completely independent (or the dependency is not known, like in the case of nominal variables in our study), singleton membership functions are employed. When some explicit knowledge is available (e.g., states ranking for ordinal variables), this knowledge can be coded by an appropriate set of overlapping membership functions: triangular, bell-shaped, etc. Centers and shape parameters of these functions can be determined by experts or can be extracted from available data using intelligent data mining techniques.

At the output of the first hidden layer signals $o_1^{[1]}, o_2^{[1]}, \ldots, o_n^{[1]}$ are formed, which are fed to the second hidden layer in the form of $(n+1)\times 1$-vector $x^{[2]} = \left(1, o_1^{[1]}, o_2^{[1]}, \ldots, o_n^{[1]}\right)^T$, where the unity component represents bias signal.

The second hidden layer of the proposed neuro-fuzzy network consists of n identical neurons with nonlinear sigmoidal activation functions $\psi_j^{[2]}, j = 1, 2, \ldots, n$ and contains $n(n+1)$ tuned synaptic weights $w_{ji}^{[2]}$. The output signal of the j-th neuron of the second hidden layer is

$$o_j^{[2]} = \psi_j^{[2]}\left(u_j^{[2]}\right) = \psi_j^{[2]}\left(\sum_{i=0}^{n} w_{ji}^{[2]} x_i^{[2]}\right)$$

(here $w_{j0}^{[2]} \equiv \theta_j^{[2]}$ – bias of the j-th neuron), and the output of the layer –

$$o^{[2]} = \Psi^{[2]}\left(W^{[2]} x^{[2]}\right), \tag{1}$$

where $o^{[2]} - (n\times 1)$-vector signal, which is fed to the third hidden layer in the form $x^{[3]} = \left(1, o^{[2]T}\right)^T$, $\Psi^{[2]} = \mathrm{diag}\left\{\psi_j^{[2]}\right\} - (n\times n)$-matrix activation function, $W^{[2]} - n\times(n+1)$-matrix of tuned synaptic weights.

The third hidden layer consists of *2n+1* neurons and forms the following signals

$$o_j^{[3]} = \psi_j^{[3]}\left(u_j^{[3]}\right) = \psi_j^{[3]}\left(\sum_{i=0}^{n} w_{ji}^{[3]} x_i^{[3]}\right),$$

$$o^{[3]} = \Psi^{[3]}\left(W^{[3]} x^{[3]}\right), \tag{2}$$

where $\Psi^{[3]} = \mathrm{diag}\left\{\psi_j^{[3]}\right\} - \left((2n+1)\times(2n+1)\right)$-matrix activation function, $W^{[3]} - \left((2n+1)\times(n+1)\right)$-matrix of tuned synaptic weights, $o^{[3]} - \left((2n+1)\times 1\right)$-vector signal, which is fed to the output layer in the form $x^{[4]} = \left(1, o^{[3]T}\right)^T$.

The output layer consists of a single neuron that forms scalar forecasting signal

$$\hat{y} = \psi^{[4]}\left(u^{[4]}\right) = \psi^{[4]}\left(\sum_{i=0}^{2n+1} w_i^{[4]} x_i^{[4]}\right) = \psi^{[4]}\left(w^{[4]T} x^{[4]}\right), \tag{3}$$

where $w^{[4]} - \left((2n+2)\times 1\right)$-vector of tuned synaptic weights.

Combining relations (1)-(3), one obtains the transfer function of the whole network

$$\hat{y} = \psi^{[4]}\left(w^{[4]T} \Psi^{[3]}\left(W^{[3]} \Psi^{[2]}\left(W^{[2]} x^{[2]}\right)\right)\right).$$

3 Network Training

The network training is performed according to standard local quadratic criterion

$$E(k) = \frac{1}{2} e^2(k) = \frac{1}{2}\left(y(k) - \hat{y}(k)\right)^2$$

using sigmoidal activation function

$$\psi^{[S]}\left(u^{[S]}(k)\right) = \frac{1}{1 - e^{-\gamma u^{[S]}(k)}},$$

where parameter $\gamma > 0$, setting steepness of the activation function, can also be tuned, $S = 2, 3, 4$ – layer index.

Omitting for brevity intermediate results, we obtain the following relations for updating network weights:

– traditional backpropagation procedure:

$$w_j^{[S]}(k+1) = w_j^{[S]}(k) + \eta^{[S]}(k)\delta_j^{[S]}(k) x^{[S]}(k); \tag{4}$$

– applying one-step modification of Levenberg-Marquardt algorithm [13], we obtain:

$$w_j^{[S]}(k+1) = w_j^{[S]}(k) + \frac{\delta_j^{[S]}(k)x^{[S]}(k)}{\left\|x^{[S]}(k)\right\|^2}, \tag{5}$$

which structurally coincides with optimal Kaczmarz algorithm [14];

– to improve learning stability at the nearly flat ends of sigmoidal activation functions, regularized Chan-Fallside algorithm [15] can be used:

$$w_j^{[S]}(k+1) = w_j^{[S]}(k) + \eta^{[S]}\delta_j^{[S]}(k)x^{[S]}(k) + \rho \vartriangle w_j^{[S]}(k-1), \tag{6}$$

Where $\eta^{[S]} = \text{const} > 0$, $\delta_j^{[S]}(k) = \frac{\partial \psi_j^{[S]}\left(u_j^{[S]}(k)\right)}{\partial u_j^{[S]}(k)} \sum_{i=1}^{h} \delta_i^{[S+1]}(k) w_{ij}^{[S+1]}(k)$,

$\delta^{[4]}(k) = e(k)\frac{\partial \psi^{[4]}\left(u^{[4]}(k)\right)}{\partial u^{[4]}(k)} = \frac{\partial E(k)}{\partial u^{[4]}(k)}$, $1 > \rho \geq 0$ – regularization parameter, $\vartriangle w^{[S]}(k-1) = w^{[S]}(k) - w^{[S]}(k-1)$, h – number of neurons.

Choice of a particular modification of the learning algorithm will depend to a large extent on statistical properties of the data set and therefore is left to a specific application.

4 Experimental Results

To validate theoretical findings, we conducted hourly forecasting for *24* hours ahead of the time series describing electric load of the Burshtyn energy island (Western Ukraine) using the proposed neuro-fuzzy network (NFN) and traditional multilayer perceptron (MLP) network with a similar architecture (the same number of hidden layers and neurons in each layer). For the MLP network, ordinal variables are converted to the numeric form by incrementally assigning an integer (*1*, *2*, *3*, …) for each subsequent rank; nominal variables are presented as a set of *1*/*0* flags for each possible state. Having the data set of one year, we used *10* months for training (January-October, 2006) and *2* months for testing (November-December, 2006). MAPE (Mean Absolute Percentage Error) is used as the forecasting quality measure.

$$MAPE = \frac{1}{N+1}\sum_{k=0}^{N}\frac{\left|\hat{y}(k) - y(k)\right|}{y(k)} \bullet 100\% .$$

Forecasting results are given in table 1 (best results are given in bold) and fig. 5, 6 (only one week of data is visualized for the sake of readability).

Table 1.

	Training set	Test set
Multilayer perceptron	*3.11%*	*3.25%*
Proposed neuro-fuzzy network	***2.24%***	***2.31%***

Analysis of graphs and forecasting errors shows that the proposed approach provides better forecasting accuracy comparing to traditional technique, producing lower mean and peak errors.

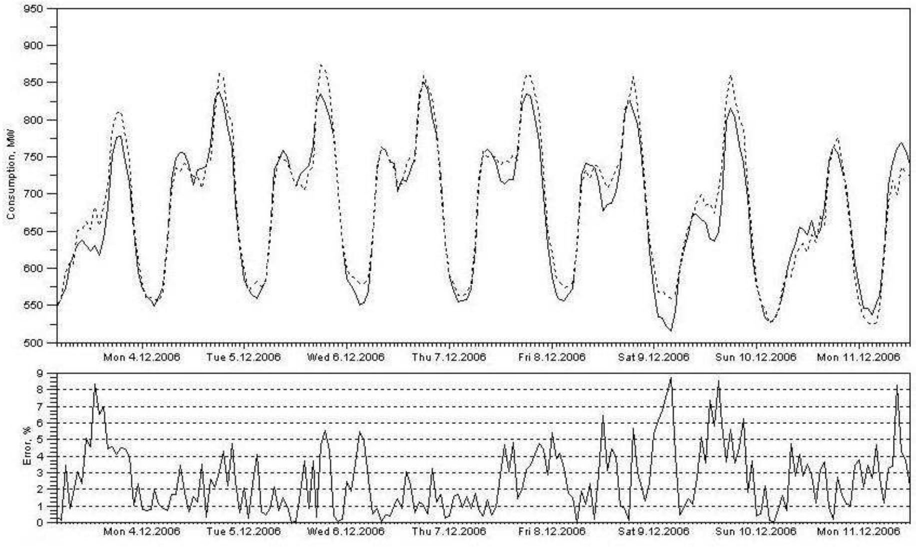


Fig. 5. Forecasting results using MLP (solid line – original series, dotted line – forecast)

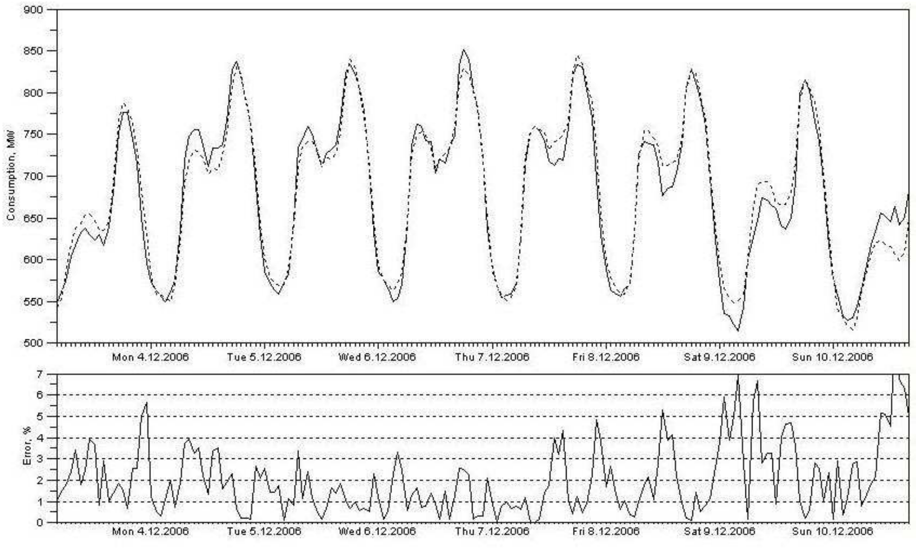


Fig. 6. Forecasting results using NFN (solid line – original series, dotted line – forecast)

5 Conclusions

The presented neuro-fuzzy network for short term electric load forecasting allows processing of information given in different scales (quantitative, ordinal, and nominal) by means of fuzzification of input variables. Types of the applied membership functions, their count and parameters may be chosen by expert judgment, based on statistical properties of the data set, or experimentally. Available prior knowledge about characteristics of input variables may be incorporated in this process. In this paper we used equally spaced triangular and singleton activation functions for the sake of simplicity, just demonstrating the basic idea.

The proposed approach can be further modified to handle multiple interconnected time series, describing for example, electric load for several regions joined by common power system, or electric load, thermal power, water and gas consumption in one region, etc. To achieve this, it is necessary to expand input and output signal vectors, modifying the proposed network architecture and learning algorithms accordingly, which is a topic for our further research.

References

1. Mori, H., Hidenori, K.: Optimal fuzzy inference for short-term load forecasting. IEEE Transactions on Power Systems 11(1), 390–396 (1996)
2. Vermaak, J., Botha, E.C.: Recurrent neural networks for short-term load forecasting. IEEE Transactions on Power Systems 13(1), 126–132 (1998)
3. Mueller, H., Petrisch, G.: Energy and load forecasting by fuzzy-neural networks. In: Proc. EUFTT 1998, September 7-10, pp. 1925–1929 (1998)
4. Shumilova, G.P., Gotman, N.E., Starceva, T.B.: Short term electric load forecasting using artificial neural networks. Elektrichestvo 10, 6–12 (1999) (in Russian)
5. Khan, M.R.: Short term load forecasting for large distribution systems using artificial neural networks and fuzzy logic. – PhD Thesis, UVEE, FEI, VUT Brno, Czech Republic (2001)
6. Khan, M.R., Zak, L., Ondrusek, C.: Fuzzy logic based short-term electric load forecasting. In: 4th International Scientific Conference Elektro-2001 – Faculty of Electrical Engineering, University of Zilina, Slovak Republic, pp. 19–24 (2001)
7. Tzafestas, S., Tzafestas, E.: Computational intelligence techniques for short-term electric load forecasting. Journal of Intelligent and Robotic Systems 31, 7–68 (2001)
8. Tkachenko, R.O., Pavlyuk, O.M.: Forecasting electric power consumption in Lviv region using artificial neural networks. Visnyk NU «Lvivska politechnika». Kompyuterna inzheneriya ta informaciyni technologii 450, 76–80 (2002) (in Ukrainian)
9. Ling, S.H., Leung, F.H.F., Lam, H.K., Tam, P.K.S.: Short-term electric load forecasting based on a neural fuzzy network. IEEE Transactions on Industrial Electronics 50(6), 1305–1316 (2003)
10. Fan, S., Chen, L.: Short-term load forecasting based on an adaptive hybrid method. IEEE Transactions on Power Systems 21(1), 392–401 (2006)
11. Bardachev Yu.N., Grinavcev, O.V., Litvinenko, V.I., Fefelov, A.A.: Synthesis and performance analysis of fuzzy neural networks using immune algorithms in the problems of electric load forecasting. Modelyuvannya ta keruvannya stanom ekologo-ekonomichnyh system regionu 3, 47–68 (2006)

12. Bodyanskiy Ye.V., Popov, S.V., Rybalchenko, T.V.: Adaptive short term electric load forecasting using artificial neural network. In: «Avtomatizaciya: problemy, idei, resheniya»: Proc. Int. Conf., Sebastopol, September 10-15, pp. 17–18 (2007) (in Russian)
13. Bodyanskiy Ye.V., Mikhalyov, O.I., Pliss, I.P.: Adaptive fault detection in controlled objects using artificial neural networks. – Dnipropetrovsk: Systemni technologii, p. 140 (in Ukrainian) (2000)
14. Kaczmarz, S.: Approximate solution of systems of linear equations. Int. J. Control 57(6), 1269–1271 (1993)
15. Cichocki, A., Unbehauen, R.: Neural Networks for Optimization and Signal Processing, Stuttgart, Teubner, p. 526 (1993)

Invariant Generation for P-Solvable Loops with Assignments

Laura Kovács[⋆]

EPFL, Switzerland
laura.kovacs@epfl.ch

Abstract. We discuss interesting properties of a general technique for inferring polynomial invariants for a subfamily of imperative loops, called the P-solvable loops, with assignments only. The approach combines algorithmic combinatorics, polynomial algebra and computational logic, and it is implemented in a new software package called `Aligator`. We present a collection of examples illustrating the power of the framework.

1 Introduction

Techniques for automatically checking and finding loop invariants and intermediate assertions have been studied and developed since the early works of [5,9,3]. Following the "trend" of developing powerful algorithms for automatically inferring polynomial invariants, in our work we apply advanced methods from symbolic summation [24,16,10] together with polynomial algebra algorithms [2,23].

Based on the shape of the loop body and on the structure of assignment statements, in [12] we defined a certain family of loops with assignments, sequencing and conditionals, called *P-solvable*, for which all test conditions are ignored and the value of each program variable can be expressed as a polynomial of the initial values of variables, the loop counter, and some new variables where there are algebraic dependencies among the new variables. For such loops we derived a systematic method for generating *polynomial loop invariants* of the form $p_1 = 0 \wedge \cdots \wedge p_r = 0$, where $p_1, \ldots, p_r$ are polynomials over the program variables. In the sequel we will call a *polynomial equality* any equality of the form $p = 0$, where p is a polynomial, thus an invariant is polynomial if it is a conjunction of polynomial equalities.

Our approach is implemented in a new software package `Aligator` [12], on top of the computer algebra system *Mathematica*. `Aligator` stands for **a**utomated **l**oop **i**nvariant **g**eneration by **a**lgebraic **t**echniques **o**ver the **r**ationals, and supports algebraic reasoning about loops. We successfully tried it on many programs working on numbers.

Moreover, our approach to polynomial invariant generation is shown to be complete for some special cases. By completeness we mean that it generates a polynomial invariant $p_1 = 0 \wedge \cdots \wedge p_r = 0$ such that any other polynomial invariant is a logical

[⋆] Research was done while the author was at the Research Institute for Symbolic Computation (RISC), Linz, Austria, The work was partially supported by BMBWK (Austrian Ministry of Education, Science, and Culture), BMWA (Austrian Ministry of Economy and Work) and MEC (Romanian Ministry of Education and Research) in the frame of the e-Austria Timişoara project, and by FWF (Austrian National Science Foundation) - SFB project F1302.

consequence of $p_1 = 0 \wedge \cdots \wedge p_r = 0$. Using polynomial ideal theory, $q = 0$ is a logical consequence of $p_1 = 0 \wedge \cdots \wedge p_r = 0$ if and only if q is in the ideal generated by $\{p_1, \ldots, p_r\}$. Since any ideal has a finite basis, the completeness of our approach equivalently means the generation of a basis $\{p_1, \ldots, p_r\}$ for the ideal of all polynomials q such that $q = 0$ is an invariant. Our method thus first derives such a polynomial basis $\{p_1, \ldots, p_r\}$, from which the polynomial loop invariant is constructed by taking the conjunction of the polynomial equalities $p_i = 0$. If the loop does not have (non-trivial) polynomial invariants, the method returns `True` as loop invariant corresponding to the $\{\}$ polynomial basis. For a detailed presentation of the soundness and completeness aspects of our approach we refer to [12,13].

The generated polynomial loop invariant, together with user-asserted non-polynomial invariants, can be used further in the verification of (partial) correctness of programs.

In this paper, we focus our attention on *P-solvable loops with only assignment statements*. Several properties of such loops will be presented, followed by motivating examples. The case of affine loops is separately discussed. The examples from the paper highlight the power and limitations of our method. More examples and experimental results can be found in [12].

The current paper extends our conference paper [13] in two aspects.

– We give a detailed comparison with related work on generating polynomial invariants for loops with assignments only (see Sections 2 and 4).
– Most importantly, in Section 4 we present interesting properties of P-solvable loops with assignments only. Firstly, we extend the framework by relaxing the P-solvable constraints of an imperative loop, and show the applicability of our approach to such cases as well. Next, we present the usage of the method in deriving automatically combinatorial identities for algorithms implementing special numbers, and finally we focus our attention on a special case of P-solvable loops, called the affine loops.

All theoretical results as well as comparison with related work are illustrated by examples. The purpose of this paper is to demonstrate the applicability and the usefulness of several algebraic and combinatorial techniques for automatic generation of polynomial invariants, and, in more general terms, to emphasize the power of combining techniques from computational logic with techniques from computer algebra for program verification.

The rest of the paper is structured as follows. Section 2 presents related work on polynomial invariant generation for loops with assignments only. Section 3 contains a short overview of our approach to invariant generation, and briefly introduces the reader to the algebraic considerations used throughout the paper. In Section 4 we discuss properties of P-solvable loops with assignments only, and demonstrate the results by a number of examples. Section 5 concludes the work.

2 Related Work

In [9], M. Karr proposed a general technique for finding affine relationships among program variables. However, Karr's work used quite complicated operations (transformations on invertible/non-invertible assignments, affine union of spaces) and had a limitation on arithmetical operations among the program variables. For these reasons, extension of his work has recently become a challenging research topic.

One line of work uses a *generic polynomial relation* of an a priori *fixed degree* [14,15,21,8,19]. The coefficients of this polynomial are replaced by variables, and constraints over the values of the coefficients are derived. The solution space of this constraint system characterizes the coefficients of all polynomial invariants up to the fixed degree. However, in case when the program has polynomial invariants of different degrees, these approaches have to be applied separately for the different degrees. This is not the case of our algorithm. Our restriction is not on the degree of sought polynomial relations, but on the type of assignments (recurrence equations) present in the loop body. The shape of assignments restricts our approach to the class of P-solvable loops, and thus we cannot handle loops with arbitrary polynomial assignments, nor tests in loop condition. However, for P-solvable loops with assignments only our method returns the generator set of all polynomial relations (of different degrees) among the program variables by the application of our method only once.

A different line of research imposes structural constraints on the assignment statements of the loop. Based on the theory of Gröbner basis, in [20] a fixpoint procedure for invariant generation is presented for so–called *simple* loops having *solvable mappings with positive rational eigenvalues*. This fixpoint is the ideal of polynomial invariants. The restriction of assignment mappings being solvable with positive rational eigenvalues ensures that the program variables can be polynomially expressed in terms of the loop counter and some auxiliary *rational* variables. Hence, the concept of solvable mapping is similar to the definition of P-solvable loop. However, contrarily to [20], in our approach we compute closed form solutions of program variables for a wider class of recurrence equations (assignment statements). The restriction on the closed form solution for P-solvable loops brings our approach also to the case of having closed forms as polynomials in the loop counters and additional new variables, but, unlike [20], the new variables can be arbitrary *algebraic* numbers, and not just rationals.

3 Overview

We begin with a brief overview of our approach to *invariant generation for P-solvable loops with assignments only*. We present the algebraic notions used later in the paper, and also fix some relevant notation. Further, a short description of our invariant generation algorithm will be given. More details can be found in [12,13].

We assume that $\mathbb{K}$ is a field of characteristic zero (e.g. $\mathbb{Q}$, $\mathbb{R}$, etc.), and by $\bar{\mathbb{K}}$ we denote its algebraic closure. Throughout this paper, $X = \{x_1, \dots, x_m\}$ $(m > 1)$ denotes the set of loop variables, $\mathbb{K}[X]$ is the ring of polynomials in the variables X with coefficients from $\mathbb{K}$, and $n \in \mathbb{N}$ stands for the loop counter.

P-solvable Loops. In our approach for generating polynomial invariants, test conditions in the loops are ignored. When we ignore conditions of loops, we will deal with non-deterministic programs. Using regular-expression like notation, in [12] we introduced the syntax and semantics of the class of non-deterministic programs that we consider. We called this class *basic non-deterministic* programs. Essentially, when we omit the condition b from a conditional statement $\texttt{If}[b\ \texttt{Then}\ S_1\ \texttt{Else}\ S_2]$, where S_1 and S_2

are sequences of assignments, we will write it as `If`$[\ldots$ `Then` S_1 `Else` $S_2]$ and mean the basic non-deterministic program $S_1|S_2$. Similarly, we omit the condition b from a loop `While`$[b, S]$, where S is a sequence of assignments, and write it in the form `While`$[\ldots, S]$ to mean the basic non-deterministic program S^*.

In our work, we developed a systematic method for generating (all) polynomial invariants for loops of such a non-deterministic syntax and semantics. Namely, we identified a class of loops with assignments, sequencing and conditionals, called the P-solvable loops, for which tests are ignored, and the value of each loop variable is expressed as a polynomial in the initial values of variables (those when the loop is entered), the loop counter, and some new variables, where there are polynomial relations among the new variables. A precise definition of P-solvable loops can be found in [12].

Polynomial Ideals and Invariants. A non-empty subset $I \subseteq \mathbb{K}[X]$ is an *ideal* of $\mathbb{K}[X]$ if $p_1 + p_2 \in I$ for all $p_1,\ p_2 \in I$ and $pq \in \mathbb{K}[X]$ for all $p \in I$ and $q \in \mathbb{K}[X]$. As observed in [18], the set of polynomials p such that $p = 0$ is a polynomial invariant forms a polynomial ideal, called *polynomial invariant ideal*.

By Hilbert's basis theorem [1], any ideal, and in particular thus the polynomial invariant ideal, has a finite basis. Using the Buchberger Algorithm [2], a special ideal basis called Gröbner basis $\{p_1, \ldots, p_r\}$ ($p_i \in \mathbb{K}[X]$) of the polynomial invariant ideal can be effectively computed. Hence, the conjunction of the polynomial equations corresponding to the polynomials from the computed basis (i.e. $p_i(X) = 0$) characterizes completely the polynomial invariants of the loop. Namely, any other polynomial invariant can be derived as a logical consequence of $p_1 = 0 \wedge \cdots \wedge p_r = 0$.

In the process of deriving a basis for the polynomial invariant ideal, we rely on efficient methods from algorithmic combinatorics, as presented below.

Sequences and Recurrences. From the assignments statements of a P-solvable loop, recurrence equations of the variables are built and solved, using the loop counter n as the recurrence index.

In what follows, $f : \mathbb{N} \to \mathbb{K}$ defines a *(univariate) sequence* of values $f(n)$ from $\mathbb{K}$ ($n \in \mathbb{N}$). A *recurrence equation* for the sequence f is a rational function defining the values of $f(n + r)$ in terms of the previous values $f(n), f(n-1), \ldots, f(n+r-1)$, where $r \in \mathbb{N}$ is called the *order* of the recurrence. A solution of the recurrence equation $f(n)$, that is a *closed-form solution*, expresses the value of $f(n)$ as a function of the summation variable n and some given initial values, e.g. $f(0), \ldots, f(r-1)$. A detailed presentation of sequences and recurrences can be found in [4,7]. In our research, we only consider special classes of recurrence equations, as follows.

A *C-finite recurrence* $f(n)$ is of the form $f(n+r) = a_0(n)f(n) + a_1(n)f(n+1) + \ldots + a_{r-1}(n)f(n+r-1)$, where the constants $a_0, \ldots, a_{r-1} \in \mathbb{K}$ do not depend on n. The closed form of a C-finite recurrence can always be computed [24,4], and it is a linear combination of polynomials in n and algebraic exponential sequences $\theta^n \in \bar{\mathbb{K}}$, with $\theta \in \bar{\mathbb{K}}$, where there exist polynomial relations among the exponential sequences. By adding any linear combination of polynomials and exponential sequences in n to the rhs of a C-finite recurrence, we obtain an *inhomogeneous linear recurrence* $f(n+r) = a_0(n)f(n) + a_1(n)f(n+1) + \ldots + a_{r-1}(n)f(n+r-1) + \sum_i q_i(n)\theta_i^n$ *with constant coefficients*, where $q_i\bar{\mathbb{K}}[n], \theta_i \in \bar{\mathbb{K}}$. Such recurrences can be transformed into C-finite ones, and thus their closed forms can be always computed. For solving such recurrences, we rely on our *Mathematica* implementation integrated into `Aligator`. For example, the closed form of $f(n+1) = 4f(n) + 2$ is $f(n) = 4^n f(0) - \frac{2}{3}(4^n - 1)$, where $f(0)$ is the initial value of $f(n)$.

A *Gosper-summable* recurrence $f(n)$ is of the form $f(n+1) = f(n) + h(n)$, where the sequence $h(n)$ can be a product of rational function-terms, exponentials, factorials and binomials in the summation variable n (all these factors can be raised to an integer power). The closed form solution of a Gosper-summable recurrence can be exactly computed using the decision algorithm given by [6]. In our research, we use a *Mathematica* implementation of the Gosper-algorithm given by the RISC Combinatorics group [16]. For example, the closed form of $f(n+1) = f(n) + n^5$ is $f(n) = f(0) - \frac{1}{12}n^2 + \frac{5}{12}n^4 - \frac{1}{2}n^5 + \frac{1}{6}n^6$, where $f(0)$ is the initial value of $f(n)$.

In our work, we only consider P-solvable loops whose assignment statements describe Gosper-summable or C-finite recurrences. Thus, the closed forms of loop variables can be computed as presented above.

Algebraic Dependencies. As mentioned already, the closed form solutions of the variables of a P-solvable loop are polynomial expressions in the summation variable n and algebraic exponential sequences in the summation variable n. We thus need to relate this sequences in a polynomial manner, such that the exponential sequences can be eliminated from the closed forms of the loop variables, and polynomial invariants can be subsequently derived. In other words, we need to compute the *algebraic dependencies* among the exponential sequences $\theta_1^n, \ldots, \theta_s^n \in \bar{\mathbb{K}}$ of the algebraic numbers $\theta_1, \ldots, \theta_s \in \bar{\mathbb{K}}$ present in the closed forms.

An *algebraic dependency* (or *algebraic relation*) of these sequences over $\bar{\mathbb{K}}$ is a polynomial $p \in \bar{\mathbb{K}}[y_1, \ldots, y_s]$ in s distinct variables $y_1, \ldots, y_s$, such that $p(\theta_1^n, \ldots, \theta_s^n) = 0, \forall n \in \mathbb{N}$. Computing algebraic dependencies reduces thus to compute a Gröbner basis of the ideal of all algebraic dependencies. We integrated in our framework a *Mathematica* implementation for deriving such a basis [10]. For example, $\theta_1^n \theta_2^n - 1 = 0$ generates the ideal of algebraic dependencies among the exponential sequences of $\theta_1 = 4$ and $\theta_2 = \frac{1}{4}$, whereas there is no algebraic dependency among the exponential sequences of $\theta_1 = 4$ and $\theta_2 = 3$.

Invariant Generation for P-solvable Loops with Assignments Only. We have now all necessary ingredients to synthesize our invariant generation algorithm for P-solvable loops with assignments only. This is achieved in Algorithm 3.1. Algorithm 3.1 starts first with extracting and solving the recurrence equations of the P-solvable loop's variables X. Next, the ideal A of algebraic dependencies among the exponential sequences from the derived closed forms is computed. Finally, from the polynomial closed form system and A, the loop counter and the exponential sequences are eliminated, using Gröbner basis computation. The polynomial invariant ideal G is thus derived.

Algorithm 3.1. P-solvable Loops with Assignments Only
Input: P-solvable loop with only assignment statements S
Output: The polynomial invariant ideal $G \trianglelefteq \mathbb{K}[X]$
Assumption: The recurrence equations of X are of order at least 1, $n \in \mathbb{N}$, X_0 are the initial values of X

```
1  Extract and solve the recurrence equations of the loop variables. We obtain:
```

$$\begin{cases} x_1[n] = q_1(n, \theta_1^n, \ldots, \theta_s^n) \\ \vdots \\ x_m[n] = q_m(n, \theta_1^n, \ldots, \theta_s^n) \end{cases}, \text{ where } \begin{array}{l} \theta_j \in \bar{\mathbb{K}},\ q_i \in \bar{\mathbb{K}}[n, \theta_1^n, \ldots, \theta_s^n], \\ q_i \text{ are parameterized by } X_0, \\ j = 1, \ldots, s,\ i = 1, \ldots, m \end{array}$$

```
2  Compute the ideal $A$ of algebraic dependencies among $n, \theta_1^n, \ldots, \theta_s^n$.
3  Denote $z_0 = n, z_1 = \theta_1^n, \ldots, z_s = \theta_s^n$.
4  Consider $I = \langle x_1 - q_1, \ldots, x_m - q_m) \rangle + A \trianglelefteq \bar{\mathbb{K}}[z_0, z_1, \ldots, z_s, x_1, \ldots, x_m]$
5  return $G = I \cap \mathbb{K}[x_1, \ldots, x_m]$.
```

Steps 1 and 2 of Algorithm 3.1 are essential. If the recurrence equations determined by the assignments of the loop cannot be solved using the techniques presented on page 353, or the algebraic dependencies among exponential sequences of the closed form system cannot be computed, Algorithm 3.1 does not succeed in generating polynomial invariants.

EXAMPLE 3.1. Consider the program fragment taken from [17], implementing an algorithm for computing the sum of the first n integers at power 5.

$$x := 0;\ y := 0;\ \texttt{While}[y \neq k,\ y := y + 1;\ x := x + y^5].$$

By applying Algorithm 3.1 and using `Aligator`, the polynomial invariants of the above loop are obtained as follows.
Step 1:

$$\begin{cases} y[n+1] = y[n] + 1 \\ x[n+1] = x[n] + y[n]^5 \end{cases} \Longrightarrow \begin{cases} y[n] \underset{Gosper}{=} y[0] + n \\ x[n] \underset{Gosper}{=} x[0] - \frac{1}{12}n^2 + \frac{5}{12}n^4 - \frac{1}{2}n^5 + \frac{1}{6}n^6 \end{cases}$$

where $y[0],\ x[0]$ denote the initial values of $y,\ x$ before the loop.

Steps 2, 3: $z_0 = n, A = \langle z_0 - n \rangle$.

Steps 4,5: The Gröbner basis computation with $z_0 \succ x \succ y$ yields:

$G = \langle 12\,x + y^2 - 5\,y^4 + 6\,y^5 - 2\,y^6 - 12\,x[0] - y[0]^2 + 5\,y[0]^4 - 6\,y[0]^5 + 2\,y[0]^6 \rangle$, that is

$$G = \langle 12x + y^2 - 5y^4 + 6y^5 - 2y^6 \rangle,$$

by initial value substitutions.

We thus derived the polynomial loop invariant

$$12x + y^2 - 5y^4 + 6y^5 - 2y^6 = 0,$$

from which any other polynomial invariant can be inferred.

Note that the closed form solution of loop variables is the strongest invariant that can be obtained, other properties can be derived from it. However, the strongest invariant contains non-polynomial expressions and the quantifier over the loop counter. Therefore, although all other invariants do logically follow from the strongest one, they cannot be derived from it in a constructive sense, for example, by a theorem prover. To obtain a tractable class of invariants, that is polynomial ones, from the strongest invariant we apply techniques from algorithmic combinatorics and polynomial algebra, as presented above. In fact, additional polynomial dependencies among loop variables stem from the desire of obtaining the strongest polynomial invariant.

4 P-Solvable Loop Properties and Examples

In this section interesting properties of P-solvable loops will be presented, justified by motivating examples. Algorithm 3.1 can be applied only for P-solvable loops. (i) The solvability of the recurrence equations of loop variables yielding a polynomial closed form system in n, and (ii) the existence of polynomial relations among the exponential sequences in n are crucial requirements of our approach to invariant generation. Without fulfilling these conditions, Algorithm 3.1 fails in generating polynomial invariants.

In what follows, we relax condition (ii) and prove that such loops do not have polynomial invariants. In other words, we prove that the "failure" of our method is actually a "correct" behavior due to the non-existence of valid polynomial relations among the loop variables.

THEOREM 4.1. Given an imperative loop with assignment statements only.
If the closed form system of the recursively changed loop variables $\{x_1, \dots, x_m\}$ is as in step 1 of Algorithm 3.1, with the property that there are no algebraic dependencies among the exponential sequences θ_i^n, then the loop has no polynomial invariant.

Proof. Consider a loop with closed form system as in step 1 of Algorithm 3.1, with the property that there are no algebraic dependencies among $\theta_1^n, \dots, \theta_s^n$. Using notation from Algorithm 3.1, we thus have $A = \emptyset$.

Let's assume there is a polynomial relation $g \in \mathbb{K}[X]$ among $x_1, \dots, x_m$ such that $g = 0$ is a polynomial invariant, i.e. it is valid at any loop iteration n. Using the closed form solutions of X, we derive that $g \in A$ whenever evaluated at $(x_1, \dots, x_m) = (q_1, \dots, q_m)$, contradicting the assumption that $A = \emptyset$. ■

EXAMPLE 4.2. Consider the loop `While`$[\dots, x := 4x; y = 3y]$. The recurrence equations of the loop are C-finite, thus solvable, and their closed form solutions involve the exponential sequence 3^n and 4^n. As mentioned on page 353, there are no algebraic dependencies among these sequences. Hence, by Theorem 4.1, the loop has no polynomial invariant.

In the process of automatically inferring polynomial invariants for P-solvable loops, techniques from symbolic summations are involved. Hence, Algorithm 3.1 can be applied on a rich class of practical imperative loops with assignments, implementing non-trivial algorithms working on numbers. From these assignments, the values of variables are expressed in terms of their previously computed values, and the loop is thus modeled by means of recurrence equations.

Moreover, using recurrence equations of order greater or equal to 2, we are able to handle imperative loops where auxiliary variables are used in order to refer to values of variables computed at finitely many loop iterations before (the number of previous loop iterations is given by the order of the recurrence). Such a loop behavior is presented below.

PROPOSITION 4.3. Given the P-solvable loop:

$$\texttt{While}[\ldots,\ t := r;\ r := b * r + a * q;\ q := t;\ x := a * x],$$

where t, r, q, x are loop variables and $a, b \in \mathbb{K}$ are constants.
Then there is always a polynomial invariant $p \in \mathbb{K}[r, q, x]$ among r, q and x.

Proof. Denoting by $n \geq 0$ the loop counter, the given imperative loop can be expressed in terms of C-finite recurrence equations as follows.

$$\begin{cases} r[n+2] &= b * r[n+1] + a * r[n] \\ q[n+2] &= r[n+1] \\ x[n+2] &= a * x[n+1]. \end{cases} \tag{1}$$

The recurrence of r is of order 2. Therefore, in its closed form computation two initial values of r are needed (that are the initial values of r and q before entering the loop). Thus, the one-to-one correspondence between the loop counter and the recurrence index does not hold anymore, but, denoting by j the recurrence index, we have $j = n + 1$, where $j \geq 1$ and $n \geq 0$.

Solving (1) reduces to solving C-finite recurrences, and thus can always be solved. The derived closed forms are linear combinations of polynomials in j and algebraic exponential sequences, whose algebraic dependencies are obtained using the method discussed on page 353. Hence, (1) is P-solvable. Applying Algorithm 3.1, the polynomial invariant among the loop variables r, q, x is:

$$-a^2\, x^2\, q[0]^4 - 2\, a\, b\, x^2\, q[0]^3\, r[0] + 2\, a\, x^2\, q[0]^2\, r[0]^2 - b^2\, x^2\, q[0]^2\, r[0]^2 +$$
$$2\, b\, x^2\, q[0]\, r[0]^3 - x^2\, r[0]^4 + a^2\, q^4\, x[0]^2 + 2\, a\, b\, q^3\, r\, x[0]^2 - 2\, a\, q^2\, r^2\, x[0]^2 +$$
$$b^2\, q^2\, r^2\, x[0]^2 - 2\, b\, q\, r^3\, x[0]^2 + r^4\, x[0]^2 = 0. \qquad \blacksquare$$

Note that computing the above invariant requires computation of algebraic dependencies among arbitrary algebraic sequences, and thus not just rationals. Hence, [20] would fail in generating polynomial relations as invariants. Moreover, [14,21] would need to guess first the degree of the polynomial, i.e. in this case 4, and then, fixing to 4 the degree of the generic polynomial invariant, a large number of constraints on the unknown coefficients of the polynomial would be generated. Proposition 4.3 highlights thus one of the advantages of combining algorithmic combinatorics and polynomial algebra for inferring invariant properties.

By substituting concrete values for the symbolic constants a and b, (1) generates a rich class of interesting examples. For example, taking $a = b = 1$ (and simplifying the loop body), we derive polynomial invariants for a loop implementing the computation of Fibonacci numbers, as presented below.

EXAMPLE 4.4. Fibonacci Numbers [7,12].
Given the program fragment:

$$r := 1;\ q := 0;\ \texttt{While}[\ldots,\ t := r;\ r := r + q;\ q := t].$$

By applying Algorithm 3.1 and using `Aligator`, we obtain the polynomial invariant

$$q^4+2q^3\,r-q^2\,r^2-2q\,r^3+r^4-q[0]^4-2q[0]^3\,r[0]+q[0]^2\,r[0]^2+2q[0]\,r[0]^3-r[0]^4=0,$$

that is $-1+q^4+2\,q^3r-q^2r^2-2\,q\,r^3+r^4=0$, by initial values substitutions.

Similarly to Proposition 4.3, using recurrences of order $1,2,3,\ldots$, our work thus offers an algorithmic approach for inferring polynomial invariants of programs computing other special numbers, e.g. the Tribonacci numbers.

EXAMPLE 4.5. Tribonacci numbers [22,12].
Given the program fragment:

$r:=1;\ a:=1;\ b:=0;$ `While`$[\ldots,\ s:=t;\ t:=r;\ r:=r+a+b;\ a:=t;\ b:=s\,].$

By applying Algorithm 3.1 and using `Aligator`, we obtain the polynomial invariant

$$2a^3+2a^2b+2ab^2+b^3-2abr+b^2r-2ar^2-br^2+r^3-2a[0]^3-2a[0]^2b[0]-$$
$$2a[0]b[0]^2-b[0]^3+2a[0]b[0]r[0]-b[0]^2r[0]+2a[0]r[0]^2+b[0]r[0]^2-r[0]^3=0,$$

that is $-1+2a^3+2a^2b+2ab^2+b^3-2abr+b^2r-2ar^2-br^2+r^3=0$, by initial values substitutions.

We finally focus our attention on a special class of loops, called *the affine loops*, defined as below.

DEFINITION 4.6. An imperative loop with only affine assignments is an *affine loop*. Considering $n\geq 0$ as the loop counter, the variables X of an affine loop satisfy the matrix equation

$$X[n+1]=A*X[n]+B, \tag{2}$$

with the matrix $A\in\mathbb{K}^{m\times m}$ and the (column) vector $B\in\mathbb{K}^m$.

In what follows, whenever we refer to an affine loop, we have in my mind an affine loop with *ignored test condition*.

Conform Section 3, affine assignments of variables define C-finite recurrences. Hence, any closed form solution of a loop variable defined by an affine relation among the loop variables is a linear combination of polynomials and algebraically related exponentials in the summation variable n. For such cases the P-solvable loop properties are thus fulfilled. We have the following theorem.

THEOREM 4.7. Affine loops are P-solvable. The polynomial invariant ideal for an affine loop is algorithmically computable by Algorithm 3.1.

Unlike [14,21], in the process of inferring automatically a basis for the polynomial invariant ideal of an affine loop, our method does not need to fix a priori the degree of sought polynomials. Moreover, due to powerful techniques from algorithmic combinatorics (C-finite solving and computing algebraic dependencies of exponential sequences), unlike [20] where the affine assignments have to be with positive rational eigenvalues, we do not impose any restriction on the shape of affine assignments. Based on Theorem 4.7, many interesting properties of affine loops can be thus automatically derived by symbolic summation algorithms.

EXAMPLE 4.8. Consider the affine loop given below, implementing an algorithm for computing the cubic root r for a given integer number a [11,20].

$$x := a;\ r := 1;\ s := 13/4;$$
$$\texttt{While}[x - s > 0,\ x := x - s;\ s := s + 6 * r + 3;\ r := r + 1].$$

By applying Algorithm 3.1 and using `Aligator`, we obtain the polynomial invariant

$$-3r^2 + s + 3r[0]^2 - s[0] = 0\ \wedge$$
$$r - 3r^2 + 2r^3 + 2x - r[0] + 3r[0]^2 - 6rr[0]^2 + 4r[0]^3 + 2rs[0] - 2r[0]s[0] - 2x[0] = 0,$$

yielding

$$-1 - 12r^2 + 4s = 0 \wedge -1 - 4a + 3r - 6r^2 + 4r^3 + 4x = 0,$$

by initial values substitutions.

5 Conclusions

In this paper, we present interesting properties of P-solvable loops with assignments only. These properties involve algebraic reasoning for automatically inferring polynomial invariants. Our approach is implemented in a new software package `Aligator`, in top of the computer algebra system *Mathematica*.

Our experiments show that many non-trivial algorithms working on numbers can be implemented using P-solvable loops. A collection of examples successfully worked out using the framework is presented in [12], some of them are given in this paper. For all these examples a basis of the polynomial invariant ideal has been automatically derived. The generated polynomial invariants, together with additional non-polynomial properties asserted by the user, can be subsequently used for verifying properties of programs.

Acknowledgements. The author wishes to thank Tudor Jebelean, Andrei Voronkov, Deepak Kapur and Manuel Kauers for their help and comments.

References

1. Adams, W.W., Loustaunau, P.: An Introduction to Gröbner Bases. Graduate Studies in Math. 3. AMS (1994)
2. Buchberger, B.: An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. Journal of Symbolic Computation 41(3-4), 475–511 (2006); Phd thesis 1965, University of Innsbruck, Austria (1965)
3. Cousot, P., Cousot, R.: Automatic Synthesis of Optimal Invariant Assertions: Mathematical Foundations. In: Proc. of the 1977 Symposium on Artificial Intelligence and Programming Languages, pp. 1–12 (1977)
4. Everest, G., van der Poorten, A., Shparlinski, I., Ward, T.: Recurrence Sequences. Mathematical Surveys and Monographs, vol. 104. American Mathematical Society, Providence, RI (2003)

5. German, S.M., Wegbreit, B.: A Synthesizer of Inductive Assertions. IEEE Transactions on Software Engineering 1, 68–75 (1975)
6. Gosper, R.W.: Decision Procedures for Indefinite Hypergeometric Summation. Journal of Symbolic Computation 75, 40–42 (1978)
7. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics, 2nd edn. Addison-Wesley Publishing Company, Reading (1989)
8. Kapur, D.: A Quantifier Elimination based Heuristic for Automatically Generating Inductive Assertions for Programs. Journal of Systems Science and Complexity 19(3), 307–330 (2006)
9. Karr, M.: Affine Relationships Among Variables of Programs. Acta Informatica 6, 133–151 (1976)
10. Kauers, M., Zimmermann, B.: Computing the Algebraic Relations of C-finite Sequences and Multisequences. Technical Report 2006-24, SFB F013 (2006)
11. Knuth, D.E.: The Art of Computer Programming. In: Seminumerical Algorithms, 3rd edn., vol. 2, Addison-Wesley, Reading (1998)
12. Kovács, L.: Automated Invariant Generation by Algebraic Techniques for Imperative Program Verification in Theorema. PhD thesis, RISC, Johannes Kepler University Linz (2007)
13. Kovacs, L.: Reasoning Algebraically About P-Solvable Loops. In: Proc. of TACAS, Budapest, Hungary. LNCS, vol. 4963, pp. 249–264 (to appear, 2008)
14. Müller-Olm, M., Seidl, H.: Computing Polynomial Program Invariants. Indormation Processing Letters 91(5), 233–244 (2004)
15. Müller-Olm, M., Seidl, H.: Precise Interprocedural Analysis through Linear Algebra. In: Proc. of the 31st POPL, pp. 330–341 (2004)
16. Paule, P., Schorn, M.: A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities. Journal of Symbolic Computation 20(5-6), 673–698 (1995)
17. Petter, M.: Berechnung von polynomiellen Invarianten. Master's thesis, Technical University Münich, Germany (2004)
18. Rodriguez-Carbonell, E., Kapur, D.: Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In: Proc. of ISSAC 2004 (2004)
19. Rodriguez-Carbonell, E., Kapur, D.: Automatic Generation of Polynomial Invariants of Bounded Degree using Abstract Interpretation. Science of Computer Programming 64(1) (2007)
20. Rodriguez-Carbonell, E., Kapur, D.: Generating All Polynomial Invariants in Simple Loops. J. of Symbolic Computation 42(4), 443–476 (2007)
21. Sankaranaryanan, S., Sipma, H.B., Manna, Z.: Non-Linear Loop Invariant Generation using Gröbner Bases. In: Proc. of POPL 2004 (2004)
22. Stanley, R.P.: Enumerative Combinatorics. Cambridge Studies in Advanced Mathematics, vol. 1. Cambridge University Press, Cambridge (1997)
23. Winkler, F.: Polynomial Algorithms in Computer Algebra. Springer, Heidelberg (1996)
24. Zeilberger, D.: A Holonomic System Approach to Special Functions. Journal of Computational and Applied Mathematics 32, 321–368 (1990)

Using Coloured Petri Nets to Model and Verify Telecommunications Systems

Valery Nepomniaschy, Dmitry Beloglazov, Tatiana Churina, and Mikhail Mashukov

A.P.Ershov Institute of Informatics Systems
Siberian Division of Russian Academy of Sciences
6, Lavrentiev ave., Novosibirsk 630090, Russia
vnep@iis.nsk.su

Abstract. A method for translation from SDL into Coloured Petri Nets (CPN) is presented. A tool STSV (SDL Telecommunications Systems Verifier) including a translator from SDL into CPN, a verifier of the net models and using CPN Tools [16] for simulation of CPN is described. For verification, the tool STSV uses a model-checking method. As case studies, we apply the tool STSV to RE-protocol [3], ATMR-protocol [8] and to detection of features interaction in telephone networks.

1 Introduction

The analysis, validation and verification of telecommunications systems are a challenge for computer science. In spite of considerable progress in theoretical research, obtained results find a limited use in modern practice. The formal description technique SDL accepted as an international standard [17] is widely used to represent telecommunications systems. Therefore, development of methods and tools for analysis and verification of SDL specified telecommunications systems is an important problem. It should be noted that high expressive power of SDL increases difficulties in verification of telecommunications systems.

A natural approach to overcome the problem is to use the models like finite state machines, Petri nets or their generalizations. Coloured Petri Nets (CPN) [9] should be distinguished among them because they have significant expressive power, a wide application, and simulation and analysis tools available [12, 16].

However, constructing the models of telecommunications systems manually is unreliable. Therefore, the problem of automatic construction of the net models arises for SDL specifications. A method for translation from SDL into high level Petri nets called hierarchical timed typed nets has been presented in [14]. Translation from SDL into so-called SDL time nets which extend conventional Petri nets by time intervals and guards for transitions has been described in [4]. Translation from SDL into high level Petri nets called M-nets has been described in [5]. A method for translation from an SDL dialect called TNSDL and from the standard SDL into high level Petri nets similar to Predicate / Transition nets has been described in [7] and [1], respectively. The problem of translation from SDL into CPN was mentioned in [9] (vol.3) as an open problem.

Different tools have been implemented for the analysis, simulation and verification of these net models. Such tools as SITE [4], PEP [6], Emma [7], Maria [1], Design/CPN [12], CPN Tools [16] and SPV [14] should be mentioned. A model-checking method is used by the tools SPV, PEP, Emma and Maria for verification of the net models.

The purpose of the paper is to describe a method for translation from SDL into CPN, a tool STSV (SDL Telecommunications Systems Verifier) and its application to modeling and verification of telecommunications systems. The paper consists of 5 sections. The translation method is described in Section 2. The tool STSV is presented in Section 3. Application of the tool STSV to RE-protocol [3] and ATMR protocol [8] as well as to detection of features interaction in telephone networks [10] is outlined in Section 4. Results and perspectives of our approach are discussed in Section 5.

This work is partly supported by Russian Foundation for Basic Research under grant 07-07-00173.

2 Translation of SDL Specifications into Coloured Petri Nets

A nonhierarchical CPN consists of three parts: a net structure, declarations and a net marking. A hierarchical coloured net is a composition of a number of nonhierarchical nets called pages. Pages can contain a special kind of transitions called modules. Modules are connected with places on a page in the same way as transitions. A module is a subnet placed on a separate page. The behavior of a hierarchical net is equivalent to the behavior of a nonhierarchical net in which each module is replaced by the corresponding subnet. Connections within nonhierarchical nets are those that obtained by substitution of transitions in hierarchical nets: each prototype is glued with all its copy-places.

Translation of a SDL specification to a CPN is made by stepwise refinement. Details of our translation method can be found in [2]. At the first step, we build a CPN which represents the general structure of the SDL specification. This net is placed at the first page and contains one module for each block.

At the second step, each module is replaced by a subnet which represents a block substructure and it is placed on a separate page. The subnet can also contain other modules. At the next steps, processes and transitions of the SDL system are translated into modules or CPN transitions (N-transitions).

Modeling is based on the fact that each unit has its fixed place in a hierarchical structure of units defining the system at different levels. The process definition describes an arbitrary set of the process instances. SDL provides a static communication interface for SDL processes. A system is modeled by a net, and the process instances are modeled by tokens.

Different instances of one process will be modeled by one subnet constructed according to the description of this process but by different tokens in the places of the net. Tokens in all places, except otherwise specified, will be marked by PId value (process identifier) of the process instances. All tokens relating to a process instance are marked by the same PId value.

Declarations of the user data types and signals are translated to the sets of colours. The net declarations should contain the sets of colours which represent data types:

integer, boolean and real. SDL uses the sorts which are generated using patterns and sorts defined before. For example, such a sort is an array represented in the net by a list. To represent a queue of signals, a set of colours list is also used. The type record is represented in the net by colour sets formed with the constructor product or record.

Generation of top-level nets. One module corresponds to a block definition in the net. Channels which connected blocks to the system boundary are represented by places connected by arcs with the modules. A one-direction channel becomes one place, and a bi-direction channel becomes two places, where one of them is the entry place, another is the exit place of the block.

Net transition guards are not defined at this step, because modules corresponding to the blocks will be replaced by subnets during further construction.

The service place *NewPId* is created for the PId value generation. This place contains one token. The initial marking is one token with the value $n+1$, where n is the number of instances created at the system initiation. The place *NewPId* is the input and output place for each transition which models the block containing the CREATE construction.

During translation of each construction OUTPUT, potential receiver processes are defined. A service place *InstPIds* is created which contains one token for each process. Initially each token carries the value which consists of the process PId and a list of identifiers of the process instances.

As a result of translation of OUTPUT, at the top page the places are created which model the queues of signals coming into processes inside the blocks.

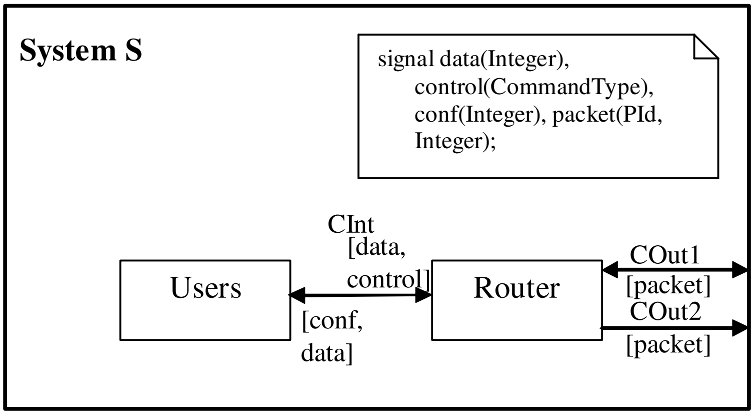

System S
signal data(Integer), control(CommandType), conf(Integer), packet(PId, Integer);
CInt
[data, control]
Users
[conf, data]
Router
COut1
[packet]
COut2
[packet]

Fig. 1. The system S

A net for the system *S* given on Fig.1 is shown on Fig.2. The net declarations are omitted to avoid overloading of the figure. For clarity, the names of the blocks and channels are used in the paper as the transition and place names. The system *S* contains three channel definitions and two block definitions. The net contains the modules *User* and *Router* and seven places. For example, the places *COut1_1* and *COut1_2* model the bi-direction channel COut1. The places *MQueue* and *UQueue* are created after translation of the OUTPUT construction, the places *Mqueue* and *UQueue* model the ports of the process which are inside the Users and Router blocks, correspondingly.

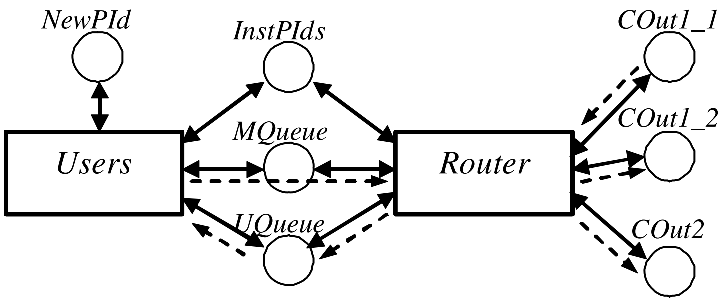


Fig. 2. The net for the system S

Only one token corresponds to a queue, so any place representing the queue is input and output for the net transition which operates the queue. Signal transmitting directions are shown on Fig. 2 by dotted lines which do not belong to the net.

Further, in the net construction, the net declarations are supplemented with variables included in arc expressions and guards of the transitions and, sometimes, with new sets of colours provided that the block and process definitions contain their own definitions of sorts, signals and lists of signals.

Block and Process modeling. A subnet for a block is created similar to the top-level net. The subnet contains one module for each new block, one or two places (depending on whether the channel is one- or bi-directional) for each new channel.

On the lower hierarchical level, a block definition contains at least one process definition. Each block definition which contains the processes is translated into CPN similar to the block substructure. One module is set according to each process definition.

At the next step, each module is replaced by a subnet which represents the inner structure of the process definition and is placed on a separate page. The subnet contains one module for each SDL-transition of the process. All data types defined in the process, as well as a set of states, signal definitions and signal list definitions, are transformed to sets of colours. The subnet contains one place for each variable (including arrays).

The set of colours assigned to the place modeling the variable is a record, where the second field is a set of colours obtaned after mapping the descriptions of the variable. The value of the first field of the token at the place-variable is the PID value of the process instance.

The subnet contains the service places *Queue* and *State*. The place *State* is the input and output place of each module modeling the SDL-transition. The value of the second field is defined by the states of this instance. The service place *Queu*e models the input ports of all instances of the same process and contains the tokens of the type list. This place is an input and output place for each transition corresponding to the SDL-transition, except for transitions of the construction of a continuous signal.

At this step the service transitions *Link* are created. They permit us to connect the places corresponding to the input channels with the places which model ports of the processes. The service transition *Delete* is created to model removing of a signal from the process port in the case when the process instance being in a certain state does not interpret the signal which comes first in the signal queue.

In all process instances, three values may be used: *sender*, *parent*, *offspring*. So, a subnet modeling the process definition has service places with the same names. At the service place, tokens are also marked by the PId value.

In the net corresponding to the process whose instance is created at the initialization of the system, as many tokens appear at each service place as the number of instances created.

Initially all places in the net modeling the process whose instances are dynamically created have no tokens. Thus, only that "net pattern" will be created, where tokens appear after firing of the transition, modeling the CREATE statement in the net corresponding to the process parent.

Let us consider the generation of offspring processes. The place *Create_id,* where *id* is the process name, is created. This place is an input for the module representing the process *offspring* and an output for the module representing the process *parent.*

A service transition *Create* is build in the net representing a process whose instances are dynamically created. Each service place and the places modeling the parameters, timers and variables are output places for it. At the transition *Create* firing, each output place gets one token. All tokens are marked by the PId value of the creating instance.

The procedure definition is translated into the net in the same way as a process definition.

Transition modeling. Each module corresponding to an SDL-transition is replaced by the net allocated on the page associated with this module. This step is the last step of the translation, if the SDL-transition can be modeled by one N-transition. In this case the guard and the expressions on the surrounding arcs are defined. These SDL-transitions are called *simple* transitions, and others are called *complex*.

The guards are defined by signals and states indicated after the keywords INPUT and STATE, respectively, in the SDL-transition. Each input arc has an arc expression. The first variable in the arc expression is the same as in expressions of other input arcs of this transition, except for the arcs connected to the places. Since all entrances of a variable into the guard and arc expressions are replaced with the same value, tokens belonging to one process instance are bound at firing the transitions. The statement NEXTSTATE and the statements of the SDL-transition define the arc expressions.

The copies of all places which are connected with the module are presented at this page. The input/output arcs repeat connection of the prototype places with the module.

A global state of a process instance is defined by its state, by a queue of signals in its port and by the values of all its variables. The global state is modeled by the net marking. Firing of a net transition that models a simple SDL-transition is possible on some marking if and only if this SDL-transition can be executed in the corresponding global state. The definition of coloured nets ensures that the execution of a simple SDL-transition and firing of the corresponding net transition result in an equivalent change of the global state of the process instance and the net marking, respectively.

Let us consider the SDL transition of the process PSender (Fig. 3).

```
process  PSender(1,1);
      ...
state prior_send;
input switch_norm;
nextstate norm_send;
      ...
endprocess Sender;
```

Fig. 3. The process PSender

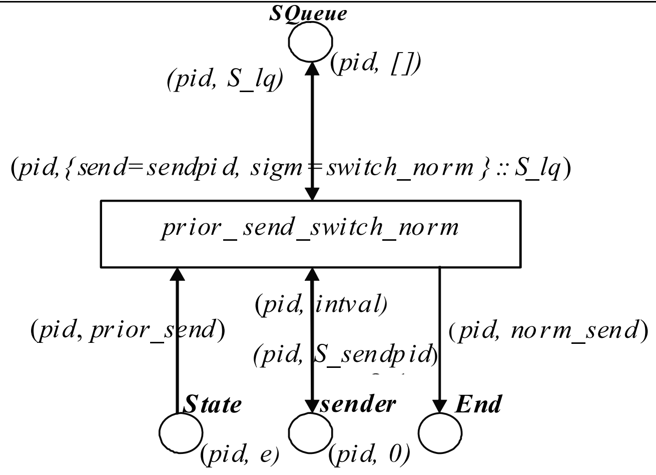


Fig. 4. The net modeling the SDL transition

The process has one instance. Let us suppose that this instance is modeled by tokens marked by the PId value *pid*. The SDL-transition in the net (see Fig. 4) is presented by one N-transition.

The complex SDL-transition is divided into fragments. Each fragment is translated separately and is represented by a subnet. The subnets are connected consequently by means of the connective places *Connect*. Two service transitions *begin* and *end* restrict the chain of subnets representing the body of a complex SDL transition. Then we can speak about the first and last transitions of the net modeling the SDL-transition.

The execution of a complex SDL-transition is modeled by consequent firing of all net transitions from *begin* to *end*. The place *State* is the input place of the transition *begin* and the output place of the transition *end*. Thus, the atomicity of execution of the SDL-transition is guaranteed.

The action CREATE is represented in the net by transitions *Generate*, *GenerateNul* and *Create*. The transitions *Generate* и *GenerateNul* belong to the subnet corresponding to the parent process. Firing of the transition *Create* models the creation of the process instance. The transition *GenerateNul* fires when the maximum allowable number of instances of the process are generated.

Destruction of the process instance is performed by removing the tokens which model it.

Modeling of the construction SET. Let us consider an SDL-transition which contains the construction SET(*N*, *t*). The modeling net is shown on Fig. 5. For mapping the timer *t* in the net, the place *T* has been created. If instances of the process are created at the initiation time of the SDL system, this place contains tokens with the value (*pid*,-1), where *pid* is a PId value of the process instance. The token (*pid*,-1) means that the timer of this process is inactive. The token (*pid*, 0) means that a signal from the timer is in the queue of this process instance. A token of any other colour means that the timer is set, but its value is greater than the current time in the model.

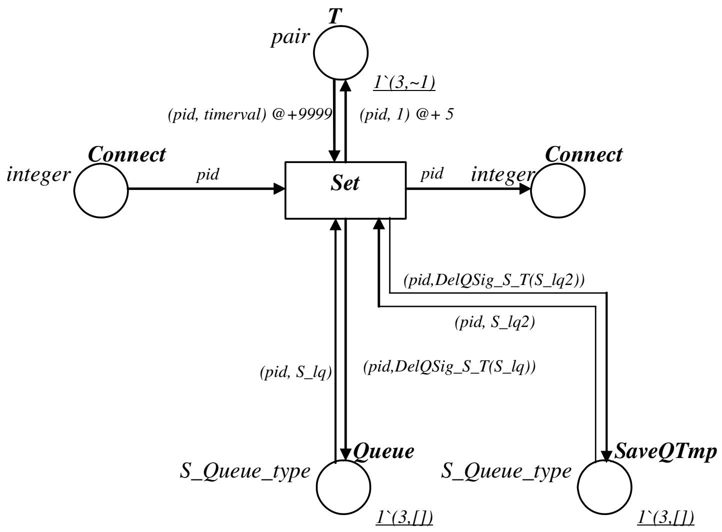


Fig. 5. A net for the statement SET

The N-transition *Send* models the firing of the timer in SDL. This transition can fire if the value of the token is *(pid, 1)* and the time stamp of the token is equal to the current model time.

Consumption of a signal from a timer by a simple SDL-transition is modeled by firing of a corresponding N-transition. This transition removes the first record corresponding to a signal of the timer from the process queue. After that, the timer of this process becomes inactive. Consumption of a signal from the timer by a complex SDL-transition is modeled by firing of the transition *begin* in the subnet corresponding to this SDL-transition.

A new setting of the timer should cancel the previous setting. Two variants are possible in this case, depending on whether a signal from the timer was sent to the queue of the process instance at the previous setting, i.e., whether the current time of the system reached the value set by the previous timer setting. If a signal from the timer has not been sent to the queue yet, then a new setting of the timer should change the previous timer value, otherwise a signal from the previous timer setting should be deleted from the queue of signals of this process instance. For this purpose a special function is created.

Modeling of the statement RESET is made in a similar way.

Optimization and net size bound. As a result of the translation, the number of pages, as well as the sets of objects which are placed on each page, are defined. Then the net optimization is made for each page. Using some patterns, the constructions which can be substituted by more simple ones are defined. Coordinates are assigned to the page objects. The page objects are placed into several columns so that the overwhelming majority of the arcs connect the objects of the neighbor columns.

In order not to take into account modules and copy-places, we consider a bound of an equivalent nonhierarchical net. The net size linearly depends on the number of statements, variables, SDL transitions, channels and processes. However, the net bound becomes quadratic with respect to *n* for each OUTPUT, where *n* is the number of the processes which can receive a signal from this OUTPUT.

3 Software Environment

STSV is a complex solution for modeling and verification of telecommunications systems specified in SDL or using Colored Petri Nets (CPN). This solution consists of the following modules:

SDL -> CPN translator. Given a SDL specification, this translator builds a corresponding CPN model.

Simulator. For simulation of CPNs we use CPN Tools.

Reachability graphs builder. CPN Tools builds reachability graphs in its internal format. We use our reachability graphs builder to transform it to our format and to build full description of each vertex, which represents the state of the modeled system.

Predicates builder. Given a reachability graph with full description and a file with description of properties, this module identifies "predicates" – sets of graph vertices where each property holds.

Model checker. It takes 3 files as an input: a model (reachability graph), a mu-formula and a predicate description. The predicate description is required because usually mu-formulas depend on predicates. Details of the model checker can be found in [11].

4 Case Studies

Ring Protocols. The first two systems we verified were ring network protocols: RE-protocol and ATMR-protocol. Their SDL specifications were translated to CPN, then the reachability graphs were built and model checking was performed.

The first case study was held for RE-protocol. In this protocol each station sends a frame fulfilled with data to its downstream neighbor. The frame has two special service bits – R and E – which are used to control the correctness of network functioning. This control is performed by a special kind of a station in the ring – a monitor. According to values of R and E bits the monitor decides which action to perform – reinitialize the ring or do nothing.

The RE-protocol was studied for cases of reliable and unreliable medium with up to 3 stations and a monitor. It was checked to be satisfying the following properties:

1. *Presence of deadlocks.* This property can be identified at the stage of reachability graph construction or with mu-formula $\neg$<to>*true,* where *true* corresponds to all states in the model. RE-protocol in cases studied has no deadlocks.
2. *Safety.* This property can be described with mu-formula $\mu X.(\langle to \rangle(received \vee X))$, and it holds in systems where it is possible to receive all sent messages. The predicate *received* corresponds to states in the model where a message is received. This property holds for RE-protocol in all cases studied.

3. *Extended safety.* Described with formula $sent \rightarrow \mu X.(received \vee [to]\ X)$, this property means "all sent messages are received". The predicate *sent* corresponds to states in the model where a message is sent. This property holds only for cases with reliable medium.
4. *Repeating messages.* We found that if the medium is unreliable, the message sent by one station to another may eventually come more than one time to its recipient. This property can be described with the following mu-formula: $received \wedge \langle to \rangle(\mu X.(received \vee (\neg sent \wedge \langle to \rangle X)))$. Our verification experiment confirmed that in case of unreliable medium the repetition of messages appears. This is not happening for models with reliable medium.

The second protocol to verify was ATMR-protocol. It is also a ring protocol and it's similar to RE-protocol in its basics, however there's no special station to control the correctness of network functioning. The ATMR-protocol is a high-speed protocol and it has no unreliable medium handler, it is supposed that high-level protocols should take care of re-sending messages. That's why we studied ATMR-protocol for cases with reliable medium only. The example ring-networks had 3 stations. We checked the same properties as for the RE-protocol. ATMR protocol appeared to have no deadlocks, satisfy safety and extended safety and have no repeating messages. And that was quite expected, since the medium was reliable.

Such experiments were fulfilled with these protocols using the tool SPV and HTT-nets [14]. As for model size and performance, we achieved better results with CPN comparing to HTT-nets. In most cases, nets and reachability graphs were smaller for CPNs comparing to corresponding HTT-nets.

Feature Interaction Problem in Telephone Networks. The next case study was the feature interaction problem (FIP) in telephone networks. Our approach to modeling was building CPN directly using the CPN Tools.

The modeling of telephone networks always begins with modeling of Basic Call State Model (BCSM). It is a set of basic rules which work in every telephone network and allow subscribers to call each other. For example: "if an idle subscriber off-hooks, he will receive the dial tone".

We built the CPN for BCSM using the following basic principles:

1) For each state of a subscriber we created a place with integer color. So, each integer token represents a subscriber. For example, marking {1'1, 1'2} in IDLE place means that subscribers 1 and 2 are idle.
2) For each possible subscriber action in each place we create appropriate transitions. For example, we create an unconditional transition from IDLE to DIALTONE. This is for the situation when idle subscriber off-hooks and receives the dial tone.

We built this CPN in all-in-one style, when the subscriber logic and the telephone station logic are united.

After we had built the CPN for the Basic Call State Model, we added several features to it:

- Call Waiting (CW). Suppose the subscriber A has this feature enabled. If he is in a call with the subscriber B, and the subscriber C calls him, he will be able to see this incoming call and answer it.

- Call Forwarding when Busy (CFB). Suppose the subscriber A has this feature enabled, and the forwarding phone number is D. If he is in a call with the subscriber B, and the subscriber C calls him, the incoming call will be forwarded to D.
- Denied Termination (DT). All incoming calls for subscribers of this feature will be denied – a caller will receive the busy tone.
- Direct Connect (DC). If the subscriber A has this feature enabled and it points to the phone number B, then each time A off-hooks, he'll be instantly connected to B.
- Emergency Call (EMG). This feature is used for emergency calls in organizations like police. Suppose the subscriber A is police with EMG enabled. When B is talking to A and drops the call (on-hooks), the call won't actually be dropped. Instead, it will be put on hold until B off-hooks again.

We added these features by appending new conditions to transitions, new transitions and places.

We built several models with different combinations of these features. The properties we studied were:

1. *Presence of deadlocks.* This is the same property as we checked for the ring protocols.
2. *Presence of loops.* The presence of loops which don't go through initial state. Since none of the models had deadlocks, we could identify the presence of such loops by checking the formula $\mu X.(\langle to\rangle(\mathit{begin} \vee X))$, where *begin* is a predicate which is true in initial state.
3. *Non-determinism.* The presence of two conflicting transition in the CPN model, which correspond to two features. The formulae for this property are feature-dependent. In general, it can be easily created from the following statement: "the non-determinism occurs when in some state both features A and B can trigger, and in the next state feature A is triggered, while B becomes disabled".
4. *Conditions violation.* In the set of features we studied only DT has a condition: "No one can call the subscriber of DT feature". This property doesn't require model checking to be discovered. It is easily described in terms of CPN and is identified with the predicate builder.

The results of our studies are given in the following table:

Features	Dead-locks	Loops	Non-determinism	Conditions violation
CW + CFB	false	false	**true**	false
CW + DT	false	false	**true**	**true**
CFB + DT	false	false	**true**	**true**
CFB + CFB	false	**true**	false	false
DC + DT	false	false	false	**true**
EMG + EMG	false	**true**	false	false
CW+CFB+DT	false	false	**true**	**true**
CW+DC+DT	false	false	**true**	**true**
All Features	false	**true**	**true**	**true**

5 Conclusion

To the best of our knowledge, our translation method is the first one that gives a solution the open problem of translation of SDL specifications into CPN.

Our approach has the following advantages:

- The translation method covers all basic SDL constructs including ones which are difficult for translation such as dynamic ones, procedures, constructs for mapping signal routes onto channels.
- An efficiency of the translation method is confirmed by linear bounds of the size of resulted nets in many cases.
- The tool STSV based on the translation method allows us to perform simulation, analysis and verification of SDL specified telecommunications systems using both STSV means and the power of CPN Tools.
- The tool STSV extends CPN Tools by our model checker [11].

Experiments with RE-protocol and ATMR-protocol have been successfully performed. The ineffectiveness of RE-protocol has been proven using the model checking method. Also a modified effective version of the protocol has been verified.

Another application of our approach was features interaction detection in telephone networks. We used CPN Tools to build CPN models for telephone networks with features and STSV tool to detect features interaction in these models. In difference to the paper [13], where CPN models are used, we applied the automated model checking method to features interaction detection.

It is supposed to apply the tool STSV to verification of different communication protocols and telecommunication systems with additional features.

References

1. Aalto, A., Husberg, N., Varpaaniemi, K.: Automatic formal model generation and analysis of SDL. In: Reed, R., Reed, J. (eds.) SDL 2003. LNCS, vol. 2708, pp. 285–299. Springer, Heidelberg (2003)
2. Mashukov, M., Churina, T.: Modeling SDL-specifications via coloured Petri nets. Institute of Informatics Systems, Russian Academy of Sciences, Novosibirsk 144, 1–70 (preprint, 2007) (in Russian)
3. Cohen, R., Segall, A.: An efficient reliable ring protocol. IEEE Transactions on Communications 39(11), 1616–1624 (1991)
4. Fisher, J., Dimitrov, E.: Verification of SDL 1992 specifications using extended Petri nets. In: Proc. IFIP 15th Intern. Symp. on Protocol Specification, Testing and Verification, Warsaw, Poland, pp. 455–458 (1995)
5. Fleischhack, H., Grahlmann, B.: A compositional Petri net semantics for SDL. In: Desel, J., Silva, M. (eds.) ICATPN 1998. LNCS, vol. 1420, pp. 144–164. Springer, Heidelberg (1998)
6. Grahlmann, B.: Combining Finite Automata. In: Steffen, B. (ed.) ETAPS 1998 and TACAS 1998. LNCS, vol. 1384, pp. 102–117. Springer, Heidelberg (1998)
7. Husberg, N., Manner, T.: Emma: Developing an Industrial Reachability Analyser for SDL. In: Wing, J.M., Woodcock, J.C.P., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 642–661. Springer, Heidelberg (1999)

8. Imai, K., Ito, T., Kasahara, H., Morita, N.: ATMR: Asynchronous transfer mode ring protocol. Computer Networks and ISDN Systems 26, 785–798 (1994)
9. Jensen, K.: Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 1,2,3. Springer, Heidelberg (1997)
10. Keck, D.O., Kuehn, P.J.: The feature and service interaction problem in telecommunications systems: a survey. IEEE Trans. on Software Eng. 24(10), 779–796 (1998)
11. Kozura, V.E., Nepomniaschy, V.A., Novikov, R.M.: Verification of distributed systems modeled by high-level Petri nets. In: Proc. Intern. Conf. on Parallel Computing in Electrical Engineering, Warsaw, Poland, pp. 61–66. IEEE Comp. Society, Los Alamitos (2002)
12. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured Petri nets. Internat. J. Software Tools for Technology Transfer 2(2), 98–132 (1998)
13. Nakamura, M.: Design and evaluation of efficient algorithms for feature interaction detection in telecommunication services. PhD dissertation, Osaka University (January 1999)
14. Nepomniaschy, V.A., Alekseev, G.I., Argirov, V.S., Beloglazov, D.M., Bystrov, A.V., Chetvertakov, E.A., Churina, T.G., Mylnikov, S.P., Novikov, R.M.: Application of modified coloured Petri nets to modeling and verification of SDL specified communication protocols. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 303–314. Springer, Heidelberg (2007)
15. Peng, H., Tahar, S., Khendek, F.: SPIN vs. VIS: A case study on the formal verification of the ATMR protocol. In: Proc. 3rd Intern. Conf. on Formal Engineering Methods, pp. 79–87. IEEE Comp. Society, Los Alamitos (2000)
16. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN Tools for editing, simulating and analysing coloured Petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 450–462. Springer, Heidelberg (2003)
17. Specification and Description Language (SDL). Recommendation, Z.100, CCITT (1992)

Additive Preconditioning for Matrix Computations*

Victor Y. Pan[1,3], Dmitriy Ivolgin[2], Brian Murphy[1], Rhys Eric Rosholt[1], Yuqing Tang[2], and Xiaodong Yan[2]

[1] Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
`firstname.lastname@lehman.cuny.edu`
[2] Ph.D. Program in Computer Science
The City University of New York
New York, NY 10036 USA
`firstnameinitiallastname@gc.cuny.edu`
[3] `http://comet.lehman.cuny.edu/vpan/`

Abstract. Our weakly random additive preconditioners facilitate the solution of linear systems of equations and other fundamental matrix computations. Compared to the popular SVD-based multiplicative preconditioners, these preconditioners are generated more readily and for a much wider class of input matrices. Furthermore they better preserve matrix structure and sparseness and have a wider range of applications, in particular to linear systems with rectangular coefficient matrices. We study the generation of such preconditioners and their impact on conditioning of the input matrix. Our analysis and experiments show the power of our approach even where we use very weak randomization and choose sparse and/or structured preconditioners.

Keywords: Matrix computations, Additive preconditioning, Weak randomization.

1 Introduction

1.1 Background: Multiplicative Preconditioning

Originally, preconditioning of a linear systems of equations $A\mathbf{y} = \mathbf{b}$ meant the transition to an equivalent but better conditioned linear systems $MA\mathbf{y} = M\mathbf{b}$, $AN\mathbf{x} = \mathbf{b}$, or more generally $MAN\mathbf{x} = M\mathbf{b}$ for $\mathbf{y} = N\mathbf{x}$ and readily computable nonsingular matrices M and/or N, called preconditioners (see [1]–[3] and the bibliography therein). Such systems can be solved faster and/or more accurately. Generally, however, Gaussian elimination is less costly than computing desired multiplicative preconditioners M and N, and so preconditioning only florishes for large but special classes of matrices A.

* Supported by PSC CUNY Awards 68291–0037 and 69330–0038.

1.2 Additive Preprocessing

As an alternative or complementary tool, we propose *weakly random additive preprocessing* $A \leftarrow C = A + P$, i.e., we add a matrix P (a preconditioner, having a smaller rank and/or structured) to the input matrix A to obtain its *additive modification* C with a smaller condition number. Hereafter we use the abbreviations "A-" for "additive", "*APPs*" for A-preprocessors, "AC" for "A-complements", and "APCs" for "A-preconditioners". ACs (resp. APCs) are the APPs P such that the input matrix A is rank deficient (resp. ill conditioned), whereas the matrix $C = A + P$ is not.

For ill conditioned matrices A with at most r small singular values, we arrive at well conditioned matrices C quite regularly provided P is a random matrix of a rank of at least r and the ratio $||A||/||P||$ is neither large nor small. The concepts "large", "small", "well" and "ill" are commonly quantified in the context of the computational tasks and computer environment. In our presentation we assume the customary IEEE model of numerical computing.

The cited power of random APPs actually holds for very weakly random APPs P, e.g., APPs whose randomization is restricted by specified patterns of structure and sparseness. Random multiplicative preconditioning does not work because random matrices tend to be well conditioned and because $\operatorname{cond} A \le \prod_i \operatorname{cond} F_i$ if $A = \prod_i F_i$.

To sum up, our APCs are generated more readily and for a much larger class of matrices than multiplicative preconditioners. Furthermore they better preserve matrix structure and sparseness and have a wider range of applications. In particular they remain effective for rectangular and rank deficient matrices A.

In this paper we generate ACs and APCs and study their impact on conditioning. Further material, including the results of our extensive tests and acknowledgements, can be found in [4]. The paper [5], the references therein, and a number of Tech. Reports by the first author with coauthors cover effective applications of such APCs to the solution of singular and nonsingular linear systems of equations, eigen-solving, and the computation of determinants. See Tech. Reports in the Ph.D. (Doctoral) Program in Computer Science of the Graduate Center of the City University of New York in 2005–2008 at `http://www.cs.gc.cuny.edu/tr/files/TR-200xxxx.pdf`

1.3 Organization of Our Paper

We organize our paper as follows. We begin with the definitions in the next section. We generate random AC and APCs in Section 3 and sparse and structured APCs in Section 5. We study conditioning of A-modifications of an input matrix A theoretically in Section 4 and experimentally in [4]. In Section 6 we sketch a further extension of our approach with application to the solution of linear systems of equations. Our numerical tests have been designed by the first author and performed by his coauthors. Otherwise this work with all typos and other errors is due to the first author.

2 Basic Definitions

Most of our basic definitions reproduce or slightly modify the customary definitions in [6], [7] for matrix computations, in particular, for Hermitian, unitary (orthogonal), singular, full-rank and rank deficient matrices, the $k \times k$ identity matrices I_k, $k \times l$ matrices $0_{k,l}$ filled with zeros, the transpose A^T and the Hermitian transpose A^H of an $m \times n$ matrix A, its rank $\rho = \operatorname{rank} A$, singular values $\sigma_j(A)$, $j = 1, \ldots, \rho$, in nonincreasing order, 2-norm $||A|| = \sigma_1(A)$, and the condition number $\operatorname{cond} A$, its *Moore-Penrose generalized inverse* A^+ (also called *pseudo inverse* and equal to the inverse A^{-1} for nonsingular matrices A), left nullity $\operatorname{lnul} A = m - \rho$, right nullity $\operatorname{rnul} A = n - \rho$, and nullity $\operatorname{nul} A = l - \rho$ for $l = \min\{m, n\}$. A matrix A is normalized if $||A|| = 1$. We write $n \gg d$ where the ratio n/d is large. We say that $r = \operatorname{nnul} A$ is the numerical nullity and $l - r = \operatorname{nrank} A$ is the numerical rank of the matrix A if the ratio $\sigma_1(A)/\sigma_{l-h}(A)$ is not large, whereas $\sigma_1(A) \gg \sigma_{l-h+1}(A)$, that is if the matrix has exactly r singular values that are small relative to $||A|| = \sigma_1(A)$. $(B_1, \ldots, B_k)$ and $\operatorname{diag}(B_i)_{i=1}^k$ denote the $1 \times k$ block matrix with the blocks $B_1, \ldots, B_k$ and $k \times k$ block diagonal matrix with the diagonal blocks $B_1, \ldots, B_k$, respectively. We write $Q(M)$ for the Q-factor of the size $m \times n$ in the thin QR factorization of an $m \times n$ matrix M of the full rank where the R-factor has positive diagonal entries. We represent an $m \times n$ APPs P of a rank r by a pair of *generators* U of size $m \times r$ and V of size $n \times r$ such that $P = UV^H$. $\mathbb{C}$ is the field of complex numbers.

3 Generation of ACs and APCs

3.1 The Basic Theorem for ACs

Suppose $A, C \in \mathbb{C}^{n \times n}$, $U, V \in \mathbb{C}^{n \times r}$, and U and V have full rank r. Then

$$\{\operatorname{rank} C = n\} \Longrightarrow \{r \geq \operatorname{nul} A\},$$

$$\{r \geq \operatorname{nul} A \text{ for random } U \text{ and } V\} \Longrightarrow \{\operatorname{rank} C = n \text{ (likely)}\}.$$

Let us formalize these simple relationships.

Random sampling of elements from a finite set Δ is their selection from the set Δ at random, independently of each other, under the uniform probability distribution on Δ. A matrix is *random* if its entries are randomly sampled (from a fixed finite set Δ). A $k \times l$ *random unitary* matrix is the $k \times l$ Q-factor $Q(M)$ in the QR factorization of random $k \times l$ matrix M of full rank. (QR factorization reveals if a matrix has full rank, and if not, we can generate a new matrix M.)

Lemma 1. *[8] (cf. also [9], [10]). For a finite set Δ of cardinality $|\Delta|$ in a ring* $\mathbf{R}$*, let a polynomial in m variables have total degree d, let it not vanish identically on the set Δ^m, and let the values of its variables be randomly sampled from the set Δ. Then the polynomial vanishes with a probability of at most $d/|\Delta|$.*

Theorem 1. *For a finite set Δ of cardinality $|\Delta|$ in a ring* $\mathbf{R}$ *and four matrices $A \in \mathbf{R}^{n \times n}$ of a rank ρ, U and V in $\Delta^{r \times n}$, and $C = A + UV^T$, we have*

a) $\operatorname{rank} C \leq r + \rho$,
b) $\operatorname{rank} C = n$ *with a probability of at least* $1 - \frac{2r}{|\Delta|}$ *if* $r + \rho \geq n$ *and either the entries of both matrices* U *and* V *have been randomly sampled from the set* Δ *or* $U = V$ *and the entries of the matrix* U *have been randomly sampled from this set,*
c) $\operatorname{rank} C = n$ *with a probability of at least* $1 - \frac{r}{|\Delta|}$ *if* $r + \rho \geq n$*, the matrix* U *(respectively* V*) has full rank* r*, and the entries of the matrix* V *(respectively* U*) have been randomly sampled from the set* Δ*.*

Proof. Part a) is verified immediately. Now let $r+\rho \geq n$. Then clearly, $\operatorname{rank} C = n$ if $U = V$ and if the entries of the matrix U are indeterminates. Since $\det C$ is a polynomial of a total degree of at most $2(n-\rho) \leq 2r$ in these entries, part b) follows from Lemma 1. Part c) is proved similarly to part b).

3.2 Generation of Randomized ACs and APCs

In virtue of Theorem 1 a random APP UV^H of a rank r is likely to be an AC if $r \geq \operatorname{nul} A$, whereas an APP of a rank r is never an AC otherwise. Randomized linear or binary search for the value $\operatorname{nul} A$ can rely on these properties.

Likewise, assuming $M \in \mathbb{C}$ and $\operatorname{nnul} M = r$, we can generate APCs based on the following extension of our sketch of Theorem 1:

$$\{\operatorname{nrank} C = n\} \Longrightarrow \{r \geq \operatorname{nnul} A\},$$

$$\{r \geq \operatorname{nnul} A \ \text{ and random unitary } U \text{ and } V\} \Longrightarrow \{\operatorname{nrank} C = n \ \ (\text{likely})\}.$$

Seeking $\operatorname{nnul} A$, however, we should choose well conditioned APPs which are properly scaled, so that the ratio $||UV^H)||/||A||$ would be neither large nor small, and surely in such a search we should test the candidate A-modifications for being well conditioned rather than having full rank. The algorithm only requires a random number generator and crude estimates for the condition numbers of the candidate matrices $P = UV^H$ and C and for the ratio $||UV^H||/||A||$ (cf. [6, Sections 2.3.2, 2.3.3, 3.5.4, and 12.5] and [7, Sections I.5.3 and I.5.4] on the 2-norm and condition estimators).

In the unlikely case where our randomization works poorly, we can apply some effective refinement techniques in [11], [12] to APPs. Alternatively we can just reapply A-preconditioning with a new (weakly) random APP. Both ways have very good chances for success, according to the test results in [4].

4 APPs and Conditioning

In this section we estimate the ratio $(\operatorname{cond} A)/\operatorname{cond} C$ from above but first recall the following sharp lower bound from [11]).

Theorem 2. *For any* $n \times n$ *nonnegative definite matrix* $A \geq 0$*, we have*

$$\min_{P \geq 0, \operatorname{rank} P \leq k} \operatorname{cond}(A+P) = \tfrac{\sigma_1(A)}{\sigma_{n-k}(A)}.$$

4.1 Randomized Upper Estimates (The Objective and the Two Main Steps)

Our analysis and extensive tests show that the value $\operatorname{cond} C$ is likely to be roughly of the order of $\sigma_1(A)/\sigma_{n-r}(A)$ provided A is an $n \times n$ matrix and an APP UV^H of a rank r is well conditioned, (weakly) random and scaled so that the ratio $||UV^H||/||A||$ is neither large, nor small. We first show this property for a singular well conditioned matrix A with a nullity r. Then in Sections 4.4 and 4.5 we extend our study to nonsingular ill conditioned matrices A with numerical nullity $r = \operatorname{nnul} A$.

4.2 ACs and Conditioning: The Basic Estimates

We first factorize the A-modification C.

Theorem 3. *Let $A = \Sigma = \operatorname{diag}(\Sigma_A, 0_r)$ be an $n \times n$ diagonal matrix of a rank $\rho = n - r$ where $\Sigma_A = \operatorname{diag}(\sigma_j)_{j=1}^{\rho}$ is the diagonal matrix of the (positive) singular values of the matrix A. Let U and V be $n \times r$ matrices such that the $n \times n$ matrix $C = A + UV^H$ is nonsingular. Write*

$$U = \begin{pmatrix} U_\rho \\ U_r \end{pmatrix}, \quad V = \begin{pmatrix} V_\rho \\ V_r \end{pmatrix}, \quad R_U = \begin{pmatrix} I_\rho & U_\rho \\ 0 & U_r \end{pmatrix}, \quad R_V = \begin{pmatrix} I_\rho & V_\rho \\ 0 & V_r \end{pmatrix}$$

where U_r and V_r are $r \times r$ block submatrices. Then

a) $C = R_U \operatorname{diag}(\Sigma_A, I_r) R_V^H$ and
b) the matrices R_U, R_V, U_r, and V_r are nonsingular.

Proof. Observe that

$$C = \Sigma + UV^H, \quad R_U \Sigma R_V^H = \Sigma, \quad U = R_U \begin{pmatrix} 0 \\ I_r \end{pmatrix}, \quad \text{and} \quad V = R_V \begin{pmatrix} 0 \\ I_r \end{pmatrix}.$$

Deduce that $\tilde{C} = R_U \Sigma R_V^H + R_U \operatorname{diag}(0, I_r) R_V^H = R_U \operatorname{diag}(\Sigma_A, I_r) R_V^H$ and arrive at part a). Part b) follows because the matrix C is nonsingular.

Corollary 1. *Under the assumptions of Theorem 3 we have*

$$\frac{||\operatorname{diag}(\Sigma_A, I_r)||}{||R_U^{-1}||\,||R_V^{-1}||} \;\le\; ||C|| \;\le\; ||\operatorname{diag}(\Sigma_A, I_r)||\,||R_U||\,||R_V||,$$

$$\frac{||\operatorname{diag}(\Sigma_A^{-1}, I_r)||}{||R_U||\,||R_V||} \;\le\; ||C^{-1}|| \;\le\; ||\operatorname{diag}(\Sigma_A^{-1}, I_r)||\,||R_U^{-1}||\,||R_V^{-1}||,$$

so that

$$\frac{\operatorname{cond}\operatorname{diag}(\Sigma_A, I_r)}{(\operatorname{cond} R_U)\operatorname{cond} R_V} \;\le\; \operatorname{cond} C \;\le\; (\operatorname{cond} R_U)(\operatorname{cond} R_V)\operatorname{cond}\operatorname{diag}(\Sigma_A, I_r).$$

Proof. The corollary follows from Theorem 3 because $\operatorname{cond} M = ||M||\,||M^+||$ and $||M^H|| = ||M||$ for any matrix M.

4.3 ACs and Conditioning: Refined Estimates

Lemma 2. *For any pair of matrices X and Y of compatible sizes we have*

$$\max\{||X||, ||Y||\} \ \le\ ||(X,Y)|| = ||(X,Y)^H|| \ \le\ \sqrt{||X||^2 + ||Y||^2}.$$

Proof. Let $||(X,Y)\begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix}|| = ||(X,Y)||$ for two vectors $\mathbf{u}$ and $\mathbf{v}$ such that

$$||\begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix}||^2 = ||\mathbf{u}||^2 + ||\mathbf{v}||^2 = 1.$$

Recall that $(X,Y)\begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix} = X\mathbf{u} + Y\mathbf{v}$ and deduce that

$$||(X,Y)|| = ||X\mathbf{u} + Y\mathbf{v}|| = ||(X,Y)\begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix}||.$$

Apply Cauchy–Schwartz bound and obtain that

$$||(X,Y)||^2 \ \le\ (||X||^2 + ||Y||^2)(||\mathbf{u}||^2 + ||\mathbf{v}||^2) \ =\ ||X||^2 + ||Y||^2,$$

which is the claimed upper bound. Now let $||X\mathbf{w}|| = ||X||$ where $||\mathbf{w}|| = 1$. Then $||X|| = ||(X,Y)\begin{pmatrix}\mathbf{w}\\ \mathbf{0}\end{pmatrix}|| \le ||(X,Y)||$. Similarly we obtain that $||Y|| \le ||(X,Y)||$. Finally recall that $||L|| = ||L^H||$ for any matrix L.

Theorem 4. *Suppose the matrices U and V have full rank. Then we have*

$$\begin{array}{rcl}
\max\{1, ||U||^2\} \le & ||R_U||^2 & \le 1 + ||U||^2,\\
\max\{1, ||V||^2\} \le & ||R_V||^2 & \le 1 + ||V||^2,\\
1 \le & ||R_U^{-1}||^2 & \le 1 + (1 + ||U||^2)||U_r^{-1}||^2,\\
1 \le ||R_V^{-H}||^2 = & ||R_V^{-1}||^2 & \le 1 + (1 + ||V||^2)||V_r^{-1}||^2.
\end{array}$$

Proof. The bounds on the norms $||R_U||$ and $||R_V||$ follow from Lemma 2 because $R_U = (I_{n,\rho}, U)$ and $R_V = (I_{n,\rho}, V)$ where $I_{n,\rho} = \begin{pmatrix} I_\rho \\ 0\end{pmatrix}$. The lower bounds on the norms $||R_U^{-1}||$ and $||R_V^{-1}||$ are obvious. To bound the norm $||R_U^{-1}||$, first observe that $R_U^{-1} = \operatorname{diag}(I_\rho, 0) + \begin{pmatrix} -U_\rho \\ I_r\end{pmatrix} U_r^{-1}$. Now apply Lemma 2 at first to this matrix and then to the matrix $\begin{pmatrix} -U_\rho \\ I_r\end{pmatrix}$ and obtain that

$$||R_U^{-1}||^2 \ \le\ 1 + ||\begin{pmatrix} -U_\rho \\ I_r\end{pmatrix} U_r^{-1}||^2 \ \le\ 1 + ||\begin{pmatrix} -U_\rho \\ I_r\end{pmatrix}||^2 ||U_r^{-1}||^2$$

and

$$||\begin{pmatrix} -U_\rho \\ I_r\end{pmatrix}||^2 \ \le\ 1 + ||U||^2.$$

By combining the latter bounds obtain the desired estimate for the norm $||R_U^{-1}||$. The norm $||R_V^{-1}||$ is estimated similarly.

The next two theorems are immediately verified,

Theorem 5. *Under the assumptions of Theorem 3, suppose that*

$$\sigma_{n-r} \leq 1 \leq \sigma_1. \tag{4.1}$$

Then $||\operatorname{diag}(\Sigma_A, I_r)|| = ||A||$ *and* $||(\operatorname{diag}(\Sigma_A, I_r))^{-1}|| = \sigma_{n-r}$.

Theorem 6. *Let us write* $\theta = ||UV^H||/||A||$. *Then we have*

$$|1-\theta| \leq ||C||/||A|| \leq (1+\theta).$$

Corollary 2. *Write*

$$\theta = ||UV^H||/||A||, \quad q = ||R_U||\ ||R_V|| \ \text{ and } \ p = ||R_U^{-1}||\ ||R_V^{-1}||,$$

so that

$$\max\{1, ||U||, ||V||, ||U||\ ||V||\} \ \leq \ q \ \leq \ \sqrt{(1+||U||^2)(1+||V||^2)},$$

$$1 \ \leq \ p^2 \ \leq \ (1+(1+||U||^2)||U_r^{-1}||^2)(1+(1+||V||^2)||V_r^{-1}||^2).$$

Then under the bounds (4.1) and the assumptions of Theorems 3 and 6 we have

a) $\max\{|1-\theta|,\ 1/p\} \ \leq \ ||C||/||A|| \ \leq \ \min\{1+\theta,\ q\}$,
b) $1/q \ \leq \ \sigma_{n-r}||C^{-1}|| = ||C^{-1}||/||A^+|| \ \leq \ p$,
c) $\max\{|1-\theta|,\ 1/p\}/q \ \leq \ (\operatorname{cond} C)/\operatorname{cond} A \ \leq \ \min\{1+\theta,\ q\}p$.

Proof. Parts a) and b) follow from Corollary 1 and Theorems 4-6. Part c) follows from parts a) and b).

The corollary shows that the transition $A \to C$ tends to yield the full rank property but changes the norms and condition numbers of the matrices only within the factors p and q. Clearly we can nicely bound the parameter q by properly scaling the matrices U and V. We estimate the bound p in Section 4.5.

Now suppose we represent an ill conditioned matrix of full rank as the sum $A + E$ where E is a small norm matrix and A is a well conditioned and rank deficient matrix. Then perturbation by the matrix E little affects our analysis, and so our next results extend it to the A-modification $C = A + E + UV^H$ for a random APP UV^H.

4.4 The Impact of A-Modification on Full Rank Matrices

Hereafter $M \geq 0$ means that M is a nonnegative definite Hermitian matrix. We extend the bounds of Corollary 2 in the cases where $||E||$ is small or $C \geq 0$ and $E \geq 0$.

Theorem 7. *Let the matrices* C *and* $\tilde{C} = C+E$ *be nonsingular. Write* $\delta = ||E||$ *and* $\delta_C = \delta||C^{-1}||$. *Then we have*

a) $||\tilde{C}|| \leq \delta + ||C||$,
b) *if* $\delta_C < 1$, *then* $||\tilde{C}^{-1}|| \leq ||C^{-1}||/(1-\delta_C)$,
so that $\operatorname{cond}\tilde{C} \leq (\operatorname{cond} C + \delta_C)/(1-\delta_C)$,
c) *if* $C \geq 0$ *and* $E \geq 0$, *then* $||\tilde{C}^{-1}|| \leq ||C^{-1}||$,
so that $\operatorname{cond}\tilde{C} \leq (1+\delta/||C||)\operatorname{cond} C$.

Proof. Parts a) and c) follow immediately. Part b) follows because

$$||\tilde{C}^{-1}|| = 1/\sigma_n(\tilde{C}) \leq 1/(\sigma_n(C)-\delta) = 1/(1/||C^{-1}||-\delta).$$

4.5 Further Comments

1. Bounds (4.1) in Corollary 2 are no loss of generality because the ratios $||C||/||A||$, $||C^{-1}||/||A^+||$, and $(\operatorname{cond} C)/\operatorname{cond} A$ do not change, whereas $\sigma_j \to s\sigma_j$ if the matrices A, UV^H, and $C = A + UV^H$ are scaled by the same scalar s.
2. An APP UV^H cannot be an APC if the ratio $\theta = ||UV^H||/||A||$ is small because $\sigma_n(C) \leq \sigma_n(A) + ||UV^H||$. Furthermore, an APP UV^H cannot be an APC if the ratio θ is large and if $\operatorname{rank}(UV^H) < \operatorname{rank} C$. Corollary 2 provides us, however, with reasonable bounds on the ratio $(\operatorname{cond} C)/\operatorname{cond} A$ as long as the norms $||U||$, $||V||$, $||U_r^{-1}||$, and $||V_r^{-1}||$ are reasonable. We can assume that $||U|| = ||V|| = 1$ and then obtain that $1 \leq q \leq 2$ and $1 \leq p^2 \leq (1+2||U_r^{-1}||)(1+2||V_r^{-1}||)$ in Corollary 2.
3. The value p in Corollary 2 is expected to be reasonably small if the matrices U_r and V_r are well conditioned. Practically we just need to randomize these matrices because there is huge empirical evidence that random matrices tend to be well conditioned. This observation also has some formal support in [13] and the references therein. Clearly the matrices U_r and V_r are random if so are the matrices SU and TV. The latter matrices are random where the generators U and V are random or just independent of the matrices S and T, respectively. Realistically this is a very light restriction, which explains why we regularly yield APCs $P = UV^H$ for generators U and V endowed with various patterns of structure and sparseness (see Section 5 and the Appendix).
4. For our random APPs the value $\operatorname{cond} C$ is expected to be reasonably bounded, but if it is not, we can readily detect this at a low computational cost and then resample the random matrices U and V.
5. In virtue of Theorem 1, random APPs of appropriate rank are ACs with a high probability for a rank deficient matrix A. Corollary 2 shows that they are likely to preserve the order of the condition number $\operatorname{cond} A$ in the transition to the full rank matrix C. We can turn an ill conditioned matrix A into a well conditioned matrix of a smaller rank by zeroing the smallest singular values. For a very large class of ill conditioned matrices, this transformation and its reverse are just small-norm perturbations, which must keep the matrices well conditioned in virtue of Theorem 7b. In virtue of Theorem 7c, the same property holds for a Hermitian and nonnegative definite input

matrix A, even where the above perturbation has a large norm. (This class is highly important [6], [7] and quite universal because a nonsingular linear system $A\mathbf{y} = \mathbf{b}$ is equivalent to the Hermitian and positive definite systems $A^H A\mathbf{y} = A^H\mathbf{b}$ and $AA^H\mathbf{x} = \mathbf{b}$, $\mathbf{y} = A^H\mathbf{x}$.)

6. The bounds in Theorem 7b rely on the worst case assumption that the perturbation by the matrix E is directed strictly towards decreasing the value $\sigma_n(C) = 1/||C^{-1}||$, that is destroying the effect of random A-preconditioning. Such a behavior, however, is completely opposite to the known effect of random perturbation (see [13] and the references therein), which means that the bounds in Theorem 7b are overly pessimistic. Our tests have confirmed this conclusion.
7. According to our extensive tests (see the Appendix), the estimated impact of A-preconditioning on the singular values is quite regular so that random APPs can be used for detecting large jumps in the spectra of the singular values and for computing numerical rank and numerical nullity. This application can be reinforced with the techniques in the next section.

5 Structured and Sparse APPs

All APPs of small ranks are structured, but next we supply various examples of sparse and/or structured APPs of any rank. In our extensive tests, these APPs were typically APCs for all classes of tested input matrices. Hereafter we call these APPs *pseudo random*. We hope to welcome more such examples from the readers.

Example 1. **Circulant APPs.** $UV^H = F^{-1}D_rF$ for the $n \times n$ unitary matrix

$$F = \frac{1}{\sqrt{n}}(\exp\frac{2\pi ij\sqrt{-1}}{n})_{i,j=0}^{n-1}$$

of the discrete Fourier transform at the n-th roots of unity and for the $n \times n$ diagonal matrix $D_r = \mathrm{diag}(d_i)_{i=0}^{n-1}$ that has exactly r nonzero entries fixed or sampled at random in r fixed sets $\mathbb{S}_1, \ldots, \mathbb{S}_r$ and placed at r fixed or random positions on the diagonal. Such an APP UV^H is a circulant matrix of the rank r that has the first column $F^{-1}\mathbf{d}$ for $\mathbf{d} = (d_i)_{i=0}^{n-1}$ (cf., e.g., [14, Theorem 2.6.4]). It is sufficient to perform $O(n\max\{r, \log n\})$ ops to multiply it by a vector. The bound decreases to $O(n\log r)$ where the r nonzeros occupy r successive positions on the diagonal. If $\mathbb{S}_1, \ldots, \mathbb{S}_r$ are real sets, then the APP is Hermitian. If the sets $\mathbb{S}_1, \ldots, \mathbb{S}_r$ lie in the annulus $\{x : d_- \le |x| \le d_+\}$, then $\mathrm{cond}(UV^H) = \mathrm{cond}\, D_r \le d_+/d_-$.

Example 2. **f-circulant APPs** [14, Section 2.6]**.** In the previous example replace the matrix F with the matrix FD_- where $D_- = \mathrm{diag}(g^i)_{i=0}^{n-1}$ and g is a primitive n-th root of a nonzero scalar f. In this case the APP is f-circulant. (It is circulant for $f = 1$ and skew-circulant for $f = -1$.) As in the previous example, one can readily bound the condition number of the APP and the arithmetic cost of its multiplication by a vector.

Example 3. **Toeplitz-like APPs I.** Define an $n \times r$ well conditioned Toeplitz matrix U of full rank. Either fix such a matrix or define it by varying u random parameters for a nonnegative integer $u < n+r$ until you yield well conditioning. Output FAILURE if this does not work. Define a matrix V a) either similarly or b) set $V = U$ (to produce a Hermitian APP). The APP UV^H has a rank of at most r and a displacement rank of at most four and can be multiplied by a vector in $O(n \log r)$ ops (cf. [14]).

Example 4. **Structured or sparse APPs I.** Define a matrix $U = PW$, P for a fixed or random $n \times n$ permutation matrix P (in the simplest case $P = I_n$) and a fixed or random $n \times r$ block W of the $n \times n$ matrix of the discrete Fourier, sine or cosine transform [14, Section 3.11], or of another well conditioned matrix with a fixed structure such as the sparseness structure [15], the displacement structure of Toeplitz, Hankel, Vandermonde, or Cauchy types (cf. [14] and the bibliography therein), or the semi- and quasi-separable (rank) structure (cf. the bibliography in [16]). One can apply random diagonal scaling to sparse and semi- and quasi-separable matrices. Example 3 is the special case where $P = I_n$ and W is a Toeplitz matrix. Define a matrix V a) either similarly or b) set $V = U$ (to produce a Hermitian APP). Define an APP UV^H. The complexity of its multiplication by a vector can be linear or nearly linear, depending on its structure.

Example 5. **Toeplitz-like APPs II.** Define an $n \times r$ Toeplitz matrix

$$U = (T_1, 0_{r,n_1}, \dots, T_k, 0_{r,n_k})^T.$$

Here T_i are $r \times r$ Toeplitz matrices, $0_{r,n_i}$ are $r \times n_i$ matrices filled with zeros for $i = 1, \dots, k$, and k, $n_1, \dots, n_k$ are positive integers (fixed or random) such that $kr + n_1 + \cdots + n_k = n$. Fix or choose at random the Toeplitz matrices T_i such that the resulting matrix U is well conditioned. T_i can denote general Toeplitz matrices or special, e.g., circulant, f-circulant, triangular Toeplitz or banded Toeplitz matrices. Define a matrix V a) either similarly or b) set $V = U$ (to produce a Hermitian APP). For general Toeplitz matrices $T_1, \dots, T_k$ and the shift operators associated with the Toeplitz structure, the APP UV^H has a displacement rank of at most $2k \le 2\lfloor n/r \rfloor$ and can be multiplied by a vector in $O(kr \log r)$ flops. For banded Toeplitz matrices T_i with a constant bandwidth we only need $O(kr)$ flops to multiply the APP by a vector. For $T_i = c_i I_r$ the matrix U has orthogonal columns, and we make it unitary by choosing the scalars c_1, $\dots, c_k$ such that $c_1^2 + \cdots + c_k^2 = 1$.

Example 6. **Structured or sparse APPs II.** Define a well conditioned matrix

$$U = P(T_1, 0_{r,n_1}, \dots, T_k, 0_{r,n_k})^T$$

for an $n \times n$ permutation matrix P and integers k, $n_1, \dots, n_k$ chosen as in Example 5 but for all i let T_i be $r \times r$ fixed or random structured matrices, e.g., the matrices of the discrete Fourier, sign or cosine transforms, matrices with a fixed displacement structure, semi- and quasi-separable (rank structured) matrices, or sparse matrices with fixed patterns of sparseness (see the bibliography listed in Example 4). Define a matrix V a) either similarly or b) set $V = U$ (to produce a Hermitian APP). Define an APP UV^H. Example 5 is the special case where $P = I_n$ and T_i are Toeplitz matrices.

Finally, we can generate APCs by appending pairs of (block) rows and (block) columns that preserve any structure of an input matrix, e.g., the structure of a block Hankel matrix with Hankel blocks.

6 Discussion

Define $\operatorname{range} M$, the linear space spanned by the columns of a matrix M, and its null space $N(M) = \{\mathbf{z} : M\mathbf{z} = \mathbf{0}\}$. A matrix B is a null matrix basis for M if $\operatorname{range} B = N(M)$.

Now let us sketch a modification of A-preconditioning for computing a null matrix basis for a matrix A (see some details in [17]). The solution of a linear system $A\mathbf{y} = \mathbf{b}$ is a special case because $(-\mathbf{b}, A)\begin{pmatrix} 1 \\ \mathbf{y} \end{pmatrix} = \mathbf{0}$.

Assume a (possibly ill conditioned) matrix $A \in \mathbb{C}^{n\times n}$, extend our study in Section 4, and deduce that for a sufficiently large integer $k < n$ and a (weakly) random matrix $B \in \mathbb{C}^{n\times k}$, the matrix $\tilde{A} = (B, A)$ is likely to be well conditioned. Generate such a matrix B for a possibly smaller integer k and apply any of the two recipes below to compute an integer $s = \operatorname{nul} \tilde{A} < n$ and a null matrix basis $Z \in \mathbb{C}^{(n+k)\times s}$ for the matrix $\tilde{A}$.

Next write $Z = \begin{pmatrix} Z_0 \\ Z_1 \end{pmatrix}$ where $Z_0 \in \mathbb{C}^{s\times k}$, and compute a null matrix basis $X \in \mathbb{C}^{s\times r}$ for the matrix Z_0. Finally obtain a null matrix basis $Z_1 X$ for the matrix A. To compute a null matrix basis X we can reapply the same algorithm, noting that $s < n$.

Here are two recipes for computing a null matrix basis Z by solving nonsingular well conditioned linear systems of equations.

1. Generate a random unitary matrix $W \in \mathbb{C}^{(n+k)\times(n+k)}$ (we can use its approximation), compute the matrix $C = \tilde{A}W = (B, A)W$, observe that $\operatorname{cond} C = \operatorname{cond} \tilde{A}$, write $C = (C_0, C_1)$ where $C_0 \in \mathbb{C}^{n\times n}$, and extend our analysis in Sections 3 and 4 to deduce that the matrix C_0 is expected to be nonsingular and to have the condition number of the order of $\operatorname{cond} C$. Finally compute a null matrix basis $Z = \begin{pmatrix} -C_0^{-1}C_1 \\ I_k \end{pmatrix}$.
2. Generate a (weakly) random $k \times (n+k)$ matrix V, define the matrix $C = \begin{pmatrix} V \\ \tilde{A} \end{pmatrix}$, and extend our analysis in Sections 3 and 4 to deduce that the matrix C_0 is expected to be nonsingular and to have the condition number of the order of $\operatorname{cond} \tilde{A}$. Finally compute a null matrix basis $Z = C^{-1}\begin{pmatrix} I_k \\ 0 \end{pmatrix}$.

Clearly, the same techniques can be applied where $A \in \mathbb{C}^{m\times n}$ and $m < n$.

The study in [17] and extensive tests support this approach. Wherever A is an ill conditioned matrix, the matrix Z must be computed with a higher precision. To yield such a high precision output, the paper [17] employs extended iterative refinement in solving linear systems in the two recipes above and, in the first of them, computes the product $C = (B, A)W$ with no errors.

References

1. Greenbaum, A.: Iterative Methods for Solving Linear Systems. SIAM, Philadelphia (1997)
2. Benzi, M.: Preconditioning Techniques for Large Linear Systems: a Survey. J. of Computational Physics 182, 418–477 (2002)
3. Chen, K.: Matrix Preconditioning Techniques and Applications. Cambridge University Press, Cambridge (2005)
4. Pan, V.Y., Ivolgin, D., Murphy, B., Rosholt, R.E., Tang, Y., Yan, X.: Additive Preconditioning for Matrix Computations, Technical Report TR 2007003, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York (2007), `http://www.cs.gc.cuny.edu/tr/files/TR-2007003.pdf`
5. Pan, V.Y., Ivolgin, D., Murphy, B., Rosholt, R.E., Tang, Y., Yan, X.: Additive Preconditioning and Aggregation in Matrix Computations. Computers and Math. with Applications 55(8), 1870–1886 (2008)
6. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore, Maryland (1996)
7. Stewart, G.W.: Matrix Algorithms, Vol I: Basic Decompositions; Vol II: Eigensystems. SIAM, Philadelphia (1998)
8. Demillo, R.A., Lipton, R.J.: A Probabilistic Remark on Algebraic Program Testing. Information Processing Letters 7(4), 193–195 (1978)
9. Zippel, R.E.: Probabilistic Algorithms for Sparse Polynomials. In: EUROSAM 1979 and ISSAC 1979. LNCS, vol. 72, pp. 216–226. Springer, Berlin (1979)
10. Schwartz, J.T.: Fast Probabilistic Algorithms for Verification of Polynomial Identities. Journal of ACM 27(4), 701–717 (1980)
11. Wang, X.: Affect of Small Rank Modification on the Condition Number of a Matrix. Computer and Math (with Applications) 54, 819–825 (2007)
12. Pan, V.Y.: Null Aggregation and Extensions, Technical Report TR2007009, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York (2007), `http://www.cs.gc.cuny.edu/tr/files/TR-2007009.pdf`
13. Tao, T., Van Wu,: The Condition Number of a Randomly Perturbed Matrix. In: Proc. Annual Symposium on Theory of Computing (STOC 2007). ACM Press, New York (2007)
14. Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser, Boston, and Springer, New York (2001)
15. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van Der Vorst, H.A.: Numerical Linear Algebra for High-Performance Computers. SIAM, Philadelphia (1998)
16. Vandebril, R., Van Barel, M., Golub, G., Mastronardi, N.: A Bibliography on Semiseparable Matrices. Calcolo 42(3–4), 249–270 (2005)
17. Pan, V.Y.: Matrix Computations with Randomized Expansion and Aggregation (preprint, 2008)

Network as a Computer: Ranking Paths to Find Flows

Dusko Pavlovic[⋆]

Oxford University and Kestrel Institute
dusko@kestrel.edu, dusko@comlab.ox.ac.uk

Abstract. We explore a simple mathematical model of network computation, based on Markov chains. Similar models apply to a broad range of computational phenomena, arising in networks of computers, as well as in genetic, and neural nets, in social networks, and so on. The main problem of interaction with such spontaneously evolving computational systems is that the data are not uniformly structured. An interesting approach is to try to extract the semantical content of the data from their distribution among the nodes. A concept is then identified by finding the community of nodes that share it. The task of data structuring is thus reduced to the task of finding the network communities, as groups of nodes that together perform some non-local data processing. Towards this goal, we extend the ranking methods from nodes to paths. This allows us to extract some information about the likely flow biases from the available static information about the network.

1 Introduction

Initially, Web search was developed as an instance of *information retrieval*, optimized for a particularly large distributed database. With the advent of online advertising, Web search got enhanced by a broad range of *information supply* techniques where the search results are expanded by additional data, extrapolated from user's interests, and from search engine's stock of information. From the simple idea to match and coordinate the push and the pull of information on the Web as a new computational platform [18] sprang up a new generation of web businesses and social networks. Similar patterns of information processing are found in many other evolutionary systems, from gene regulation, protein interaction and neural nets, through the various networks of computers and devices, to the complex social and market structures [15].

This paper explores some simple mathematical consequences of the observation that the Web, and similar networks, are much more than mere information repositories: besides storing, and retrieving, and supplying information, they also generate, and process information. We pursue the idea that the Web can be modeled as a computer, rather than a database; or more precisely, as a vast multi-party computation [6], akin to a market place, where masses of selfish

[⋆] Supported by ONR and EPSRC.

agents jointly evaluate and generate public information, driven by their private utilities. While this view raises interesting new problems across the whole gamut of Computer Science, the most effective solutions, so far, of the problem of *semantical* interactions with the Web computations were obtained by rediscovering and adopting the *ranking* methods, deeply rooted in the sociometric tradition [11,10], and adapting them for use on very large indices, leading to the whole new paradigm of *search* [19,12,13]. Implicitly, the idea of the Web as a computer is tacitly present already in this paradigm, in the sense that the search rankings are extracted from the link structure, and other intrinsic information, generated on the Web itself, rather than stored in it.

Outline of the paper. In Sect. 2 we introduce the basic network model, and describe a first attempt to extract information about the flows through a network from the available static data about it. In Sects. 3 and 4, we describe the structure which allows us to lift the notion of rank, described in Sect. 5, to path networks in Sect. 6. Ranking paths allows us to extract a random variable, called attraction bias, which allows measuring the *mutual information* of the distributions of the inputs and the outputs of the network computation, which can be viewed as an indicator of non-local information processing that takes place in the given network. In the final section, we describe how the obtained data can be used to detect semantical structures in a network. The experimental work necessary to test the practical effectiveness of the approach is left for future work.

2 Networks

Basic model. We view a network as an edge-labelled directed graph $A = \left(R \xleftarrow{\gamma} E \underset{\varrho}{\overset{\delta}{\rightrightarrows}} N\right)$, where N and E are, respectively, the finite sets of *nodes*, and *links*, or *edges*, whereas R is an ordered field of *values* (in some applications an ordered ring of functions). A link $i \overset{e}{\to} j$ is thus an element $e \in E$ with $\delta(e) = i$ and $\varrho(e) = j$. The value $\gamma(e)$ is the cost (when positive), or payoff (when negative) of the traffic over e. These data induce the *adjacency matrix* $E = (E_{ij})_{N\times N}$ and the *capacity matrix* $A = (A_{ij})_{N\times N}$, with the entries $E_{ij} = \{e \in E \mid i \overset{e}{\to} j\}$ and $A_{ij} = \sum_{e\in E_{ij}} A_e$, where $A_e = 2^{-\gamma(e)}$ is the capacity of the link e.

Remark. The term “capacity” is used here as in network flow theory.[1] The cost or the payoff of a link may represent its value in a pay-per-click model of a fragment of the Web; or it may denote the proximity of the web pages within the same site, or within a group of interconnected sites. In a protein network, the energy cost or payoff may be derived from the chemical affinities between the nodes. While this parameter can be abstracted away, simply by taking $\gamma(e) = 0$ for all

[1] The information theoretic homonym has a different, albeit related meaning, which motivates the choice of $\gamma(e) = -\log_2 A_e$.

links e, its role will become clear in Sects. 3 and 6, where it allows discounting and eliminating some paths.

Basic dynamics. The simplest model of network dynamics is based on the assumption that the traffic flows are distributed proportionally to the link capacities. Randomly sampling the Web traffic, we shall thus find a surfer on a link e with the probability $\alpha_e = \frac{A_e}{A_\bullet}$, where $A_\bullet = \sum_{f \in E} A_f$.

In order to find the communities in a network, we need to detect the traffic biases between its nodes. We assume that the traffic between the nodes within the same community will be higher than the capacity of the links between them would lead us to expect; and that the traffic between the different communities will be lower than expected. To measure such traffic biases, we normalize the capacity matrix A to get the *capacity distribution* $\alpha = (\alpha_{ij})_{N \times N}$ as $\alpha_{ij} = \frac{A_{ij}}{A_{\bullet\bullet}}$, where $A_{\bullet\bullet} = \sum_{k,\ell \in N} A_{k\ell}$. The entry α_{ij} is thus the probability that a random sample of traffic on A, following the simple dynamics proportional to capacity, will be found on a link from i to j. On the other hand, the marginals of the probability distribution α,

$$\alpha_{i\bullet} = \sum_{j \in N} \alpha_{ij} \qquad\qquad \alpha_{\bullet j} = \sum_{i \in N} \alpha_{ij}$$

correspond, respectively, to the probabilities that a random sample of traffic will have i as its source, and j as its destination. Let us call $\alpha_{i\bullet}$ the *out-rank* of i, and $\alpha_{\bullet j}$ the *in-rank* of j, because they can be viewed as the simplest, albeit degenerate cases of the notion of rank.

If the in-rank and the out-rank are statistically independent, then (by the definition of independence) the probability that a random traffic sample goes from i to j will be $\alpha_{i\bullet}\alpha_{\bullet j}$. Their dependency is thus measured by the *traffic bias* matrix $\upsilon = (\upsilon_{ij})_{N \times N}$ with the entries $\upsilon_{ij} = \alpha_{ij} - \alpha_{i\bullet}\alpha_{\bullet j}$ falling in the interval $[-1, 1]$. The higher the bias, the more unexpected traffic there is. For a set of nodes $U \subseteq N$ the values

$$\mathsf{Coh}(U) = \sum_{i,j \in U} \upsilon_{ij} \qquad\qquad \mathsf{Adh}(U) = \sum_{i \in U, j \notin U} \upsilon_{ij} + \upsilon_{ji}$$

can thus be construed as the *cohesion* and the *adhesion* forces: the total traffic bias within the group, and with its exterior, respectively. A network community U can thus be recognized as a set of nodes with a high cohesion and a low adhesion [16]. The idea that semantically related nodes can be captured as members of the same network communities, derived from the graph structure, is a natural extension of the ranking approach, which has been formalized in [9,17].

The only problem with applying that idea to the above model is that our initial assumption — that the traffic distribution on A is proportional to its link capacities — is not very realistic. It abstracts away all traffic dynamics. On the other hand, the static network model, as given above, does not provide any data about the actual traffic. We explore the ways to solve this problem, and extract increasingly more realistic views of traffic dynamics from a static network model.

3 Adding Paths

A path $i \xrightarrow{a} j$ in a network A is a sequence of links $i \xrightarrow{a_1} k_1 \xrightarrow{a_2} k_2 \to \cdots \xrightarrow{a_n} j$. In many cases of interest, traffic dynamics on a network depends on the path selections, rather than just on single links.

One idea is to add the paths to the structure of a network, and to annotate how the links compose into paths, and how the paths compose into longer paths. This amounts to generating the free category [14] over the network graph. Unfortunately, adding all paths to a network usually destroys some essential information, just like the transitive closure of a relation does. E.g., in a social network, a friend of a friend is often not even an acquaintance. Taking the transitive closure of the friendship relation obliterates that fact. Moreover, the popular "small world" phenomenon suggests that almost *every two people* can be related through no more than six friends of friends of friends... So already adding all paths of length six to a social network, with a symmetric friendship relation, is likely to generate a complete graph. In fact, the average probability that two of node's neighbors in an undirected graph are also linked with each other is an important factor, called *clustering coefficient* [21]. On the other hand, in some networks, e.g. of protein interactions, a link $i \to k$ which shortcuts the links $i \to j \to k$ often denotes a direct *feed-forward* connection, rather than a composition of the two links, and leads to essentially different dynamics.

So only "short" paths must be added to a network: composition must be penalized.

Definition 1. *For a given network* $A = \left(R \xleftarrow{\gamma} E \overset{\delta}{\underset{\varrho}{\rightrightarrows}} N\right)$, *a cutoff value* $v \in R$, *and a composition penalty* $d \in R$, *we define the* v-completion *to be the network* $A^{*v} = \left(R \xleftarrow{\gamma} E^{*v} \overset{\delta}{\underset{\varrho}{\rightrightarrows}} N\right)$, *where*

$E^{*v} = \{a \in E^* \mid \gamma(a) \leq v\}$ *and*

$$\gamma\left(i_0 \xrightarrow{a_1} i_1 \xrightarrow{a_2} i_2 \to \cdots \xrightarrow{a_n} i_n\right) = (n-1)d + \gamma(a_1) + \cdots + \gamma(a_n)$$

and E^* *is the set of all nonempty paths in* A.

Remarks. E^* can be obtained as the matrix of sets $E^* = \sum_{n=0}^{\infty} E^n$ where each E^n is a power of the adjacency matrix E. If the entry E_{ij} is viewed as the set of links $\{i \xrightarrow{e} j\}$, then the entry $E^2_{ij} = \sum_{k=1}^{N} E_{ik} \cdot E_{kj}$ corresponds to the set of 2-hop paths $\{i \xrightarrow{e_1} k \xrightarrow{e_2} j\}$ through the various nodes k; the matrix E^3 similarly corresponds to the matrix of 3-hop paths, and so on.

The v-closed network A^{*v} is closed under the composition of low cost paths, but not if the cost is greater than v. It is not hard to see that the v-completion is an idempotent operation, i.e. $A^{*v*v} = A^{*v}$, but it may fail to be a proper closure operation, because a link e in A, with $\gamma(e) > v$, may lead to $A \not\subseteq A^{*v}$.

In the rest of the paper, we assume that the networks are v- complete for some v, i.e. $A = A^{*v}$. This means that the relevant pathways are already represented as links, with the composition penalty absorbed in the cost.

4 Network Dynamics

In order to derive network dynamics from a static network model, one first specifies the way in which the behavior of a computational agent, processing data on the network, is influenced by the network structure, and then usually derives a Markov chain that drives the traffic. The network features that influence its dynamics can then be incrementally refined, yielding more and more information.

4.1 Forward and Backward

Random walks on networks are often represented in terms of the behavior of surfers on the Web, following the hyperlinks.[2] The simplest surfer behavior chooses an out-link uniformly at random at each node. A visitor of a node i will thus proceed to a node j with probability $A^{\triangleright}_{ij} = \frac{A_{ij}}{A_{i\bullet}}$, where $A_{i\bullet} = \sum_{k=1}^{N} A_{ik}$ is the out-degree of i. The row-stochastic matrix $A^{\triangleright} = (A^{\triangleright}_{ij})_{N\times N}$ represents *forward dynamics* of a network A. The entries $A^{\triangleright}_{ij}$ are called the *pull* coefficients of i by j.

Dually, *backward dynamics* of a network A is represented by a column-stochastic matrix $A^{\triangleleft} = (A^{\triangleleft}_{ij})_{N\times N}$, where the entry $A^{\triangleleft}_{ij} = \frac{A_{ij}}{A_{\bullet j}}$, with $A_{\bullet j} = \sum_{k=1}^{N} A_{kj}$ denoting the in-degree of j, describes the probability that a surfer who is on the node j came there from the node i. The entries $A^{\triangleleft}_{ij}$ are called the *push* coefficients.

Remark. Note that the capacity matrix can be normalized to get $A^{\triangleright}$ and $A^{\triangleleft}$ as above only if no rows, resp. columns, consist of 0s alone. This means that every node of the network must have at least one out-link, resp. at least one in-link. Networks that do not satisfy this requirement need to be modified, in one way or another, in order to enable analysis. Adding a high-cost link between every two nodes is clearly the minimal perturbation (with maximal entropy) that achieves this. Alternatively, the problem can also be resolved by adjoining a fresh node, and the high-cost links in and out of it [2]. Either way, the quantitative effect of such modifications can be made arbitrarily small by increasing the cost of the added links.

4.2 Forward-Out and Backward-In Dynamics

The next example can be interpreted in two ways, either to show how forward and backward dynamics can be refined to take into account various navigation capabilities, or how to abstract away irrelevant cycles. Suppose that a surfer searches for the hubs on the network: he prefers to follow the hyperlinks that lead to the nodes with a higher out-degree. This preference may be realized by annotating the hyperlinks according to the out-rank of their target nodes. Alternatively, the surfer may explore the hyperlinks ahead, and select those

[2] The surfers deserve their name by following the "waves", i.e. obeying the same dynamics.

with the highest out-degree; but we want to ignore the exploration part, and simply assume that he proceeds according to the out-rank of the nodes ahead. The probability that this surfer will move from i to j is thus

$$A^{\blacktriangleright}_{ij} \quad = \quad A^{\triangleright}_{ij} \cdot \alpha_{j\bullet} \quad = \quad \frac{A_{ij}A_{j\bullet}}{A_{i\bullet}A_{\bullet\bullet}}$$

We call this the *forward-out* dynamics. In the dual, *backward-in* dynamics, the surfers are more likely to arrive to j from i if this is a frequently visited node, i.e. if its in-rank is higher

$$A^{\blacktriangleleft}_{ij} \quad = \quad \alpha_{\bullet i} \cdot A^{\triangleleft}_{ij} \quad = \quad \frac{A_{\bullet i}A_{ij}}{A_{\bullet\bullet}A_{\bullet j}}$$

These dynamics will be the particularly convenient to demonstrate an example of bias analysis in Sect. 6, because they clearly display clearly how the simple traffic bias v from Sect. 2 can be refined by the various dynamics factors.

4.3 Teleportation and Preference

The main point of formulating network dynamics, especially in the Markov chain form, is to be able to compute the node ranking as its invariant distribution. However, since the network graphs are usually *not* strongly connected, the Markov chains, derived from their structure, are often reducible to classes of nodes with no way out.

The simplest remedy is the idea of *teleportation*, going back to [19]. A general interpretation is that, whichever dynamics a surfer might follow, at each node he tosses a biased coin, and with a probability $d \in (0, 1)$ follows that dynamics. Otherwise, with a probability $1 - d$, he "teleports" to a randomly chosen node, ignoring all hyperlinks and other structure. Following, say, forward dynamics, the probability that he will go from i to j is thus $A^{P}_{ij} \; = \; dA^{\triangleright}_{ij} + \frac{1-d}{N}$. This is roughly the PageRank dynamics, from which the Google search engine had started [19][3]. The induced dynamics is thus $A^{P} \; = \; dA^{\triangleright} + (1 - d)P$, where $P = (P_{ij})_{N\times N}$ has all entries $P_{ij} = \frac{1}{N}$. In the networks without a cost function, this is interpreted as adding a link between every two nodes. The influence of such links can be controlled using the cost functions. In any case, the resulting Markov chains become irreducible, and their stationary distributions do not get captured in any closed components. Furthermore, the model can be *personalized* by capturing surfer's preferences in terms of the biases in P: the entries P_{ij} can be interpreted as i's *trust* in j [8]. The extensions of the *backward* dynamics by teleportation yields to different interpretations, which the reader may wish to consider on her own.

[3] The original version allowed $A^{\triangleright}_{ij}$ to be 0, if $A_{i\bullet}$ is 0, i.e. if i is a "sink-hole", and the teleportation factor was added to save dynamics from such sinkholes. Other modifications were introduced later.

5 Ranking

Intuitively, the rank of a node is the probability that randomly sampled traffic will be found to visit that node. In search, this is taken as a generic relevance measure. The technical implication is that the rank can be obtained as a stationary distribution of the Markov chain capturing dynamics. Each notion of dynamics thus induces a corresponding notion of rank. Since a Markov chain can be viewed as a linear, and hence continuous transformation of the simplex of distributions, which is closed and compact, already Brouwer's fixed point theorem guarantees that the rank always exists. Finding a meaningful, useful notion of rank is another matter.

First of all, as already mentioned, networks often decompose into loosely connected subnets. In the long run, all traffic is likely to get captured in some such subnet. This results in multiple stationary distributions, each concentrated in a closed subnet, zero otherwise. Dynamics derived directly from the network graph therefore result in uninformative ranking data. In order to assure that the relevant Markov chains are irreducible and aperiodic, and thus induce unique and nondegenerate stationary distributions, network dynamics usually need to be perturbed, using a damping and stabilizing factor such as teleportation. Another sort of problems arise when the unique stationary distribution is not an attractor, or when the rate of convergence is unfeasibly slow [7,3].

While very important in concrete applications, these problems, and their solutions, have less impact on the conceptual analyses pursued in this paper. *We shall henceforth assume that all processes have been adjusted to induce unique and effectively computable ranking.*[4]

5.1 Promotion and Reputation

We now explain the intuition behind the simplest notions of rank.

In social terms, the push coefficient $A^{\lhd}_{ij} = \frac{A_{ij}}{A_{\bullet j}}$ can be interpreted as measuring how much i supports (or advocates) j. The concept of *promotion* can then be formalized as a probability distribution $r^{\lhd}$, such that $r^{\lhd}_i = \sum_{k=1}^{N} A^{\lhd}_{ik} r^{\lhd}_k$. In words, the promotion rank (or *push rank*) $r^{\lhd}_i$ of a node i is the sum of the promotion ranks $r^{\lhd}_k$ of its children nodes, each allocated to i according to the push coefficient $A^{\lhd}_{ik}$, measuring i's support for k.

Dually, the pull coefficient $A^{\rhd}_{ij}$ can be interpreted as measuring how much i trusts j. The concept of *reputation* can then be formalized as a probability distribution $r^{\rhd}$, such that $r^{\rhd}_j = \sum_{k=1}^{N} r^{\rhd}_k A^{\rhd}_{kj}$. This reputation rank (or *pull rank*) $r^{\rhd}_i$ of a node i is thus the sum of the reputation ranks $r^{\rhd}_k$ of its parent nodes, each allocated according to the pull coefficient $A^{\rhd}_{kj}$, of k's trust in j.

Gathering the promotion values in a column vector $r^{\lhd}$ and the reputation values in a row vector $r^{\rhd}$, we can rewrite the definitions of $r^{\lhd}$ and $r^{\rhd}$ in the matrix form

$$r^{\lhd} = A^{\lhd} r^{\lhd} \qquad\qquad r^{\rhd} = r^{\rhd} A^{\rhd}$$

[4] This implies that all notions of dynamics that we consider have a tacit damping factor. We do not display it only because it needlessly complicates formulas.

The refined notions of promotion $r^{\blacktriangleleft}$ and reputation $r^{\blacktriangleright}$ are defined and interpreted along the same lines, as the stationary distributions of the processes $A^{\blacktriangleleft}$ and $A^{\blacktriangleright}$ respectively.

5.2 Expected Flow

While dynamics of reputation has been studied for a long time [11,10], and with increased attention recently, since it become a crucial tool of Web search [19,13], the dual dynamics of promotion does not seem to have attracted much attention. We need both notions to define the expected traffic flow.

The expected flow from j to k, under the assumption that they are independent, is caused only by a "traffic pressure", resulting from the pull to j and the push from k. Following this idea, we define

$$r^{\bowtie}_{jk} = r^{\blacktriangleright}_j r^{\blacktriangleleft}_k \tag{1}$$

The expected flow $r^{\bowtie}$ is thus a probability distribution over $N \times N$, which can be represented as the matrix $r^{\bowtie} = r^{\blacktriangleleft} \cdot r^{\blacktriangleright}$, obtained by multiplying the column vector $r^{\blacktriangleleft}$ and the row vector $r^{\blacktriangleright}$. Since $r^{\blacktriangleleft}$ and $r^{\blacktriangleright}$ are the principal eigenvectors of $A^{\blacktriangleleft}$ and $A^{\blacktriangleright}$, $r^{\bowtie}$ is the unique distribution satisfying $r^{\bowtie} = A^{\blacktriangleleft} \cdot r^{\bowtie} \cdot A^{\blacktriangleright}$, i.e. $r^{\bowtie}_{jk} = \sum_{i=1}^{N} \sum_{\ell=1}^{N} A^{\blacktriangleright}_{ij} r^{\bowtie}_{i\ell} A^{\blacktriangleleft}_{k\ell}$. Intuitively, this means that the flow pressure from i to ℓ propagates to cause a flow pressure from j to k proportionally to the force of the traffic from i to j and to the force of traffic flows from k to ℓ — *provided* that j and k are independent. In order to measure their dependency, we attempt to capture how the *actual flows* from i to ℓ (rather than mere flow pressure) may get diverted, say by the high costs and the low capacities, to cause actual flows from j to k.

6 Path Networks

Definition 2. *Given a v-closed network* $A = \left(R \xleftarrow{\gamma} E \overset{\delta}{\underset{\varrho}{\rightrightarrows}} N\right)$, *we define the* path network

$\widehat{A} = \left(R \xleftarrow{\gamma} \widehat{E} \overset{\delta}{\underset{\varrho}{\rightrightarrows}} \widehat{N}\right)$, *where* $\widehat{N} = E$, *and* $\widehat{E} = \sum_{a,b\in E} \widehat{E}_{ab}$, *with*

$$\widehat{E}_{ab} = \left\{f = \langle f_0, f_1\rangle \in E_{ij} \times E_{k\ell} \mid \gamma(f_0) + \gamma(b) + \gamma(f_1) - \gamma(a) \le v - 2d\right\} \tag{2}$$
$$\gamma(f) = 2d + \gamma(f_0) + \gamma(b) + \gamma(f_1) - \gamma(a) \tag{3}$$

$$\begin{tikzcd}
i \arrow[dr, "f_0"] \arrow[ddd, "a"'] & \\
& j \arrow[d, "b"] \\
& k \arrow[dl, "f_1"] \\
\ell &
\end{tikzcd}$$

Dynamics of path selection. Recalling that $\widehat{A}_{ab} = \sum_{f \in \widehat{E}_{ab}} 2^{-\gamma(f)}$, we define the forward and the backward dynamics, and the pull rank and the push rank just like before:

$$\widehat{A}^{\triangleright}_{ab} = \frac{\widehat{A}_{ab}}{\widehat{A}_{a\bullet}} \qquad\qquad \widehat{A}^{\triangleleft}_{ab} = \frac{\widehat{A}_{ab}}{\widehat{A}_{\bullet b}}$$

$$\widehat{r}_b = \sum_{a \in \widehat{N}} \widehat{r}_a \widehat{A}^{\triangleright}_{ab} \qquad\qquad \widehat{r}^{\triangleleft}_a = \sum_{b \in \widehat{N}} \widehat{A}^{\triangleleft}_{ab} \widehat{r}^{\triangleleft}_b$$

Intuitively, $\widehat{A}^{\triangleright}_{ab}$ is now the probability that traffic through a is diverted to b (rather than to some other path); while $\widehat{A}^{\triangleleft}_{ab}$ is the probability that traffic through b is diverted from a (and not from some other path). The pull rank $\widehat{r}_b$, i.e. the probability that b will be traversed, can thus be understood as its *attraction*; whereas $\widehat{r}^{\triangleleft}_a$ is the probability that a will be avoided.

Using the pull rank of the paths, we can now define the *node attraction* between j and k to be the total attraction of all paths between them:

$$\widehat{r}_{jk} = \sum_{j \xrightarrow[b]{} k} \widehat{r}_b \tag{4}$$

The idea is that this notion of attraction the nodes will allow us to refine the estimate of the traffic bias υ as described in Sect. 2. In particular, consider *attraction bias*

$$\Upsilon_{jk} = \widehat{r}_{jk} - r^{\bowtie}_{jk} \tag{5}$$

To motivate this, note that expanding the formula for $r^{\bowtie}_{jk}$ in Sect. 5.2 shows that $r^{\bowtie}$ is the stationary distribution of the Markov chain $A^{\bowtie} = \left(A^{\bowtie}_{(ij)(k\ell)}\right)_{N^2 \times N^2}$, where

$$A^{\bowtie}_{(ij)(k\ell)} = \frac{A_{ij} A_{j\bullet} A_{\bullet k} A_{k\ell}}{A_{i\bullet} A^2_{\bullet\bullet} A_{\bullet \ell}} \qquad \text{and} \qquad r^{\bowtie}_{jk} = \sum_{i,\ell \in N} A^{\bowtie}_{(ij)(k\ell)} r^{\bowtie}_{i\ell}$$

On the other hand, the node attraction $\widehat{r}$ turns out to be a stationary distribution of a process that refines $A^{\bowtie}$.

Definition 3. *Given a network A, its* attraction dynamics *is a Markov chain $\widehat{A} = \left(\widehat{A}_{(ij)(k\ell)}\right)_{N^2 \times N^2}$, with the entries*

$$\widehat{A}_{(ij)(k\ell)} = \frac{A_{ij} A_{jk} A_{k\ell}}{A_{i\bullet} A_{\bullet\bullet} A_{\bullet\ell}} \tag{6}$$

where $A_{i\bullet} A_{\bullet\bullet} A_{\bullet\ell} = \sum_{m,n \in N} A_{im} A_{mn} A_{n\ell}$.

Proposition 1. *Suppose that a given network A is v-complete for a sufficiently large v. Then the node attraction $\widehat{r}$, defined in (4), is the stationary distribution of its attraction dynamics (6). In other words, for every j, k holds*

$$\widehat{r}_{jk} = \sum_{i,\ell \in N} \widehat{A}_{(ij)(k\ell)} \widehat{r}_{i\ell} \tag{7}$$

The proof can be found in [20]. It is based on the following lemma.

Lemma 1. *For a network A, which is v- complete for a sufficiently large cutoff value v, the following equations hold for $i \xrightarrow{a} \ell$ and $j \xrightarrow{b} k$*

$$\widehat{A}_{ab} = \frac{A_b}{4^d A_a} A_{ij} A_{k\ell} \tag{8}$$

$$\sum_{j \underset{c}{\to} k} \widehat{A}_{ac} = \frac{1}{4^d A_a} A_{ij} A_{jk} A_{k\ell} \tag{9}$$

$$\widehat{A}_{a\bullet} = \frac{1}{4^d A_a} A_{i\bullet} A_{\bullet\bullet} A_{\bullet\ell} \tag{10}$$

On the other hand, proposition 1 implies the following corollary, which establishes that formula (5) can be used to measure the attraction bias, as intended.

Corollary 1. *The directed reputation and promotion ranks are the marginals of the node attraction*

$$\sum_{k \in N} \widehat{r}_{jk} = r_j^{\blacktriangleright} \tag{11}$$

$$\sum_{j \in N} \widehat{r}_{jk} = r_k^{\blacktriangleleft} \tag{12}$$

All proofs are in the Appendix of [20].

Interpretation. To understand the meaning of attraction bias, consider a v-complete network A, with the forward-out and backward-in dynamics. The pull rank $r_j^{\blacktriangleright}$ tells how likely it is that a randomly sampled traffic path arrives to j; whereas the push rank $r_k^{\blacktriangleleft}$ tells how likely it is that a randomly sampled traffic path departs from k.

On the other hand, the attraction dynamics in the induced path network $\widehat{A}$ gives the node attraction $\widehat{r}_{jk}$, which tells how likely it is that a randomly sampled traffic path traverses a path from j to k. In summary, we have

$$r_j^{\blacktriangleright} = \mathsf{Prob}\big(\bullet \xrightarrow{\xi} j \mid \xi \in A^{\blacktriangleright}\big)$$

$$r_k^{\blacktriangleleft} = \mathsf{Prob}\big(k \xrightarrow{\xi} \bullet \mid \xi \in A^{\blacktriangleleft}\big)$$

$$\widehat{r}_{jk} = \mathsf{Prob}\big(j \xrightarrow{\xi} k \mid \xi \in \widehat{A}\big)$$

Although the notation suggests that $r^{\blacktriangleright}$, $r^{\blacktriangleleft}$, and $\widehat{r}$ are sampled from different processes, corollary 1 establishes that $\widehat{r}$ is in fact the joint distribution of $r^{\blacktriangleright}$ and $r^{\blacktriangleleft}$.

Nevertheless, a diligent reader will surely notice a twist of j and k in the last three equations, and wonder why is the probability that traffic goes from j to k related with the probabilities that it arrives *to* j, and that it departs *from* k? — The answer to this question makes the forward-*out* and the backward-*in* dynamics into a more interesting example than its many dynamical cousins. Briefly, if the surfers are more likely to flow with $\bullet\!\rightarrow\! j$ if the capacity of the links out of j is higher, and if they are more likely to flow with $k\!\rightarrow\!\bullet$ if the capacity of the links into k is higher, then the surfers are most likely to follow both these flows, i.e. into j and out of k — if there is a high capacity of the links $j \to k$.

Mutual information of the inputs and the outputs. The fact that $\widehat{r}$ is the joint distribution of the processes expressed by $r^{\blacktriangleright}$ and $r^{\blacktriangleleft}$ allows us to extract their *mutual information* [4]

$$I(r^{\blacktriangleright}\ ;\ r^{\blacktriangleleft}) \quad = \quad D(\widehat{r}\ ||\ r^{\bowtie}) \quad = \quad \sum_{j=1}^{N}\sum_{k=1}^{N} \widehat{r}_{jk}\ \log \frac{\widehat{r}_{jk}}{r^{\blacktriangleright}_{j} r^{\blacktriangleleft}_{k}}$$

Its expression in terms of relative entropy $D(\widehat{r}\ ||\ r^{\bowtie})$ [*ibidem*] shows that it measures how much we lose, in the efficiency of encoding of $\widehat{r}$ if we assume that $r^{\blacktriangleright}$ and $r^{\blacktriangleleft}$ are mutually independent. Intuitively, the mutual information $I(r^{\blacktriangleright}\ ;\ r^{\blacktriangleleft})$ can thus be taken as a measure of the *locality* of information processing in A. If this is an entirely local process, then every path must begin and end at the same node, and the random walks δ and ϱ, selecting the sources and the destinations of the paths, must coincide. But if $\delta = \varrho$, then the push rank and the pull rank must obey the same distribution $r^{\blacktriangleleft} = r^{\blacktriangleright} = r$, and their mutual information is $I(r^{\blacktriangleright}\ ;\ r^{\blacktriangleleft}) = H(r)$, their entropy. In the other extreme case, the random walks δ and ϱ are independent, and their joint distribution is just the product of their distributions $\widehat{r}_{jk} = r^{\blacktriangleright}_{j} r^{\blacktriangleleft}_{k}$. Their mutual information is then $I(r^{\blacktriangleright}\ ;\ r^{\blacktriangleleft}) = 0$.

7 Conclusions and Future Work

When the Web is viewed as a global data store, the problem of its semantics is the problem of determining a uniform meaning for the data published by its various participants. The search engines are dealing with this problem on the level of the human-Web interaction (e.g., distinguishing the meanings of the word “jaguar”, sometimes denoting a car, sometimes an animal [12], or deciding whether “Paris Hilton”, in a given context, refers to a person or to a hotel, etc.), whereas the Semantic Web project [1] deals with the computer-Web interactions. When the Web is viewed as a computer, the problem of its semantics is not just a matter of assigning some meanings to some data stored in it, but also to its data processing operations. For programming languages, this is what we usually call operational semantics. However, unlike a programming language, the Web, and other spontaneously evolving networks, do not have a formally defined set of data structures and operations: data are transformed by many random walks, running concurrently. Operational semantics of network computation requires a toolkit for incremental analysis of such processes. In this paper, we described a path ranking method, which is may become a useful piece of that toolkit. Now we sketch a way to test it experimentally. Using the notion of attraction bias, we lift the graph theoretic notion of (maximal) *clique* into rank analysis, while retaining network dynamics as a graph structure over such generalized cliques. We call these generalized cliques *concepts* and the links between them *associations.*

Communities and concepts. Taking the notion of attraction bias back to the idea of communities as sets of nodes with high cohesion, from which we started in the Introduction, we now reformulate the notion of cohesion in a different norm (ℓ_∞ instead of ℓ_1), and define cohesion of a set of nodes $U \subseteq N$ to be their minimal symmetric attraction bias

$$\Upsilon(U) = \bigwedge_{i,j\in U} (\Upsilon_{ij} \vee \Upsilon_{ji})$$

For each $\varepsilon \in [0,1]$, we define an ε- *community* to be a set of nodes $U \subseteq N$ such that $\Upsilon(U) \geq \varepsilon$. Denoting by $\wp_\varepsilon N$ the set of ε-communities, note that $\varepsilon_1 \leq \varepsilon_2$ implies that $\wp_{\varepsilon_1} N \supseteq \wp_{\varepsilon_2} N$. The partial ordering of $U, V \in \wp_\varepsilon N$ is given by $U \sqsubseteq V \iff U \subseteq V \wedge \Upsilon(U) \leq \Upsilon(V)$ This gives a *directed complete partial order (dcpo).* It is not a lattice because some communities cannot be extended by new nodes without decreasing their cohesion; so there are pairs of communities that cannot be joined, and do not have an upper bound. However, *directed* sets of communities (i.e., where each pair has an upper bound) do have least upper bounds, which are just their set theoretic unions. Directed complete partial orders are often used in denotational semantics of programming languages [5]. According to that interpretation, communities can be thought of as pieces of *partial information*, their $\sqsubseteq$- ordering as the increase of information, and the existence of an upper bound of two communities as the *consistency* of the informations that they carry.

The maximal elements of $\wp_\varepsilon N$, i.e. the communities that cannot be extended by new nodes without losing cohesion, can be construed as *ε-concepts.* A set $U \in \wp_\varepsilon N$ is thus an ε-concept if $\Upsilon(\{i,j\}) \geq \varepsilon$ holds for all $i, j \in U$, but for every $k \in N \setminus U$ there is a $j \in U$ such that $\Upsilon(\{k,j\}) < \varepsilon$.

The community and concept structure of a network A can be analyzed by studying the sequence of hypergraphs A_ε, where the ε-concepts, or the ε-communities approximating them, are viewed as hyperedges. The sequence $(A_\varepsilon)_{\varepsilon\in[0,1]}$ decreases as the cohesion parameter ε increases, and the highly cohesive communities and concepts can be feasibly analyzed.

A level further, concepts and communities can be viewed as the nodes of a network. The most interesting definition of the links between them, intuitively thought of as associations, is based on a variant of a path network, complementing definition 2. A sketch of this definition is in the next, final subsection.

Associations. Let $\mathcal{N}^\varepsilon$ denote the set of ε-concepts in a network A. The *concept network* $\mathcal{A}^\varepsilon$, induced by a network A, has the ε-concepts as its nodes. Its edges are called *concept associations*. The set of associations between $U, V \in \mathcal{N}^\varepsilon$ is

$$\mathcal{E}^\varepsilon_{UV} = \sum_{U \underset{a}{\rightarrow} U\cap V} \; \sum_{U\cap V \underset{b}{\rightarrow} V} \tilde{E}_{ab}$$

where $U \overset{\xi}{\rightarrow} V$ abbreviates $\delta(\xi) \in U \;\wedge\; \varrho(\xi) \in V$, and

$$\tilde{E}_{ab} = \left\{ f = \langle f_0, f_1 \rangle \in E_{ij} \times E_{k\ell} \mid \gamma(f_0) + \gamma(b) \leq v - d \text{ and } \gamma(a) + \gamma(f_1) \leq v - d \right\}$$

An association $f \in \mathcal{A}_{UV}$ is thus a quadruple $f = \langle a, b, f_0, f_1 \rangle$

$$\begin{tikzcd} i \arrow[r, "f_0"] \arrow[d, "a"'] & j \arrow[d, "b"] \\ k \arrow[r, "f_1"'] & \ell \end{tikzcd}$$

such that $i, j, k \in U$ and $j, k, \ell \in V$. Its cost is $\gamma(f) = \gamma(f_0) + \gamma(b) - \gamma(a) - \gamma(f_1)$. The cost of an association from U to V is lower if the traffic from $i \in U$ to $\ell \in V$ gets less costly when it crosses to V earlier.

While the general network analysis tools apply to concept networks, the various notions of dynamics acquire new meanings on this level. At this point, understanding which of the possible interpretations may lead to useful tools for extracting and analyzing the relevant concepts, processed in a network, seems to call for experimentation with real data.

References

1. Berners-Lee, T.: Semantic Web road map (October 1998)
2. Bianchini, M., Gori, M., Scarselli, F.: Inside PageRank. ACM Trans. Inter. Tech. 5(1), 92–128 (2005)
3. Boldi, P., Santini, M., Vigna, S.: PageRank as a function of the damping factor. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, pp. 557–566. ACM Press, New York, NY, USA (2005)
4. Cover, T.M., Thomas, J.A.: Elements of information theory. Wiley-Interscience, New York, NY, USA (1991)
5. Gierz, G., Hoffmann, K.H., Keimel, K., Lawson, J., Mislove, M., Scott, D.: Continuous Lattices and Domains. Encyclopedia of Mathematics and its Applications, vol. 93. Cambridge University Press, Cambridge (2003)
6. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, New York, NY, USA (2004)
7. Golub, G.H., Greif, C.: An Arnoldi-type algorithm for computing PageRank. BIT Numerical Mathematics 43(1), 1–18 (2003)
8. Gyöngyi, Z., Garcia-Molina, H., Pedersen, J.: Combating Web spam with TrustRank. In: VLDB, pp. 576–587 (2004)

9. Haveliwala, T.H.: Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. IEEE Trans. Knowl. Data Eng. 15(4), 784–796 (2003)
10. Hubbell, C.H.: An input-output approach to clique identification. Sociometry 28, 377–399 (1965)
11. Katz, L.: A new status index derived from sociometric analysis. Psychometrika 18, 39–43 (1953)
12. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. Journal of the ACM 46(5), 604–632 (1999)
13. Langville, A.N., Meyer, C.D.: Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, Princeton, NJ, USA (2006)
14. Mac Lane, S.: Categories for the Working Mathematician. Number 5 in Graduate Texts in Mathematics. Springer, Heidelberg (1971)
15. Newman, M., Barabasi, A.-L., Watts, D.J.: The Structure and Dynamics of Networks. Princeton Studies in Complexity. Princeton University Press, Princeton, NJ, USA (2006)
16. Newman, M.E.J.: Modularity and community structure in networks. PNAS 103(23), 8577–8582 (2006)
17. Ollivier, Y., Senellart, P.: Finding related pages using Green measures: An illustration with Wikipedia. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence, Menlo Park, California, July 2007, pp. 1427–1433. AAAI Press (2007)
18. O'Reilly, T.: What is Web 2 (September 2005)
19. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project (1998)
20. Pavlovic, D.: Network as a computer: ranking paths to find flows (February 2008), `http://aps.arxiv.org/abs/0802.1306` (Preliminary version of this paper, with proofs)
21. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393(6684), 440–442 (1998)

A Unified Categorical Approach for Attributed Graph Rewriting

Maxime Rebout, Louis Féraud, and Sergei Soloviev

IRIT, Université Paul Sabatier
118 Route de Narbonne, F-31062 TOULOUSE CEDEX 9
{rebout,feraud,soloviev}@irit.fr

Abstract. Attributed graphs are often used in software engineering. Mainly algorithms concerning programs and models transformations are based on rewriting techniques. We suggest a unified categorical approach for the description and the verification of such algorithms and programs. This contribution which is a generalization of the double pushout approach can be seen as a mix between pushout and pullback. This will facilitate the computations on attributes within a unified framework. It should be particularly helpful for model to model transformation in the domain of "Model Driven Architecture".

1 Introduction

When dealing with graphs, there are generally two points of view. The first one concentrates mostly on a static analysis of graph properties such as the shortest path or the maximal flow, cycles, connected components; such an approach is nowadays mainly found in mathematics. The second one is more on the dynamic side. Here the graphs are used to describe the transitions from one state to another when an "action" is performed. Since the 1970s the description of such processes can be done in a formal way through the use of "graph grammars" which are a generalization of the textual Chomsky's grammars applied to graphs. The concept of graph grammars has been successfully used in many scientific domains. Many topics in computer science can be addressed using graph grammars. Let us quote some: program optimisation and/or transformation [3], image processing [2,18], software engineering. Since nearly a decade, the so called Model Driven Architecture (abbreviated MDA) has increased graph notation and language popularity [13]. Subsequently, the theory of graph grammars has known a considerable revival of activity [4]. In fact, modeling and meta modeling in software engineering is mainly based on attributed graph rewriting. Among the formalisms with the most solid theoretical foundations, Ehrig's double pushout approach has already a long tradition [8] and various applications [9]. These seminal works are very interesting because they allow not only to implement structural transformations but also to compute attributes that decorate the structures. In fact, software engineering, in contrast *e.g.* to image processing where structural transformations are primordial, needs to attach semantic information to the nodes or edges of graphs: attributes for models or semantic networks, tokens for Petri nets, *etc.* For Ehrig and his colleagues, the background of their rewriting systems mainly relies on graph categories and pushout operations. These characteristics are very convenient to define structural transformations but not sufficient to implement computations with attributes. Because of certain difficulties with the direct approach, Ehrig suggests to use another formalism (Σ-algebras) [12]. This combination leads to a hybrid theoretical construction which can be cumbersome theoretically and practically. Our contribution aims at a homogeneous theoretical framework for graph rewriting unifying in a common formalism structural transformations and computations with attributes.

This paper is organized as follows: in section 2, the current solutions are discussed, in section 3, the basic definitions of our contribution are presented and in section 4 and 5 the method to construct pushouts and graph rewritings in our framework is given.

2 The Current Solutions

For textual grammars, rewriting is based on manipulations of linear strings of characters such as substitution or concatenation. For graphs grammars, operations on strings are no more sufficient. Many generalizations are possible: *e.g.*, the connexion relations in the "node replacement" approach or the external nodes in the "hyperedge replacement" approach [20]. The operations based on the (double) pushout are really interesting because they offer an algebraic framework that rests on category theory to perform the rewriting. The first uses of the pushout to rewrite graphs, by Hartmut Ehrig, can be found in [8,11]. There, a rewriting rule p is given by three graphs K, L and R and two morphisms $l : K \to L$ and $r : K \to R$. The elements of L with no pre-image by l will be deleted by the application of the rule and the elements of R with no pre-image by r will be created. The graph K is the "glue" graph linking the graphs L and R. The application of such rules on a graph G is decomposed in three steps: finding a morphism between L and G, construction of the pushout-complement and construction of the pushout. To have a unique solution, the morphism between L and G is often restricted to injective morphisms. A useful property of this theory is that all rewriting rules are reversible.

In the last decades, this theory has evolved in order to generalize the graphs which can be rewritten. One of the most advanced solutions is given by the double pushout approach (abbreviated as DPO approach) in the "adhesive high level replacement categories" (see [12]) in which the graphs can be attributed and typed. The adhesive HLR categories (based on the adhesive categories introduced by Lack and Sobocinski in [17]) are a concept which permits to get certain interesting properties for graph rewriting systems, for example, the local Church-Rosser and parallelism properties, confluence or critical pair analysis, *etc.* Some refinements of this approach have been studied: *e.g.*, the work of Kastenberg [16] can be cited. We now give some definitions and propositions of the HLR approach.

Definition 1. *An E-graph* $G = (V_1, V_2, E_1, E_2, E_3, (source_i, target_i)_{i \in [\![1,3]\!]})$ *is given by five sets:* V_1 *and* V_2 *respectively called graph nodes set and data nodes set,* E_1*,* E_2*, and* E_3 *respectively called edges set, node attributes set and edge attributes set and six functions giving for each set of edges the source and the target of the edges.*

An E-graph morphism $f : G_1 \to G_2$ *is given by five functions* $(f_{V_1}, f_{V_2}, f_{E_1}, f_{E_2}, f_{E_3})$ *with* $f_{V_i} : G_{1,V_i} \to G_{2,V_i}$ *for* $i \in [\![1,2]\!]$ *and* $f_{E_j} : G_{1,E_j} \to G_{2,E_j}$ *for* $j \in [\![1,3]\!]$ *such that all these functions commute with all the functions source and target of* G_1 *and* G_2*.*

Definition 2. *Let us give a data signature* $DSIG = (S_D, OP_D)$ *with the attribute value sorts* $S'_D \subset S_D$*. An attributed graph* $AG = (G, D)$ *is given by an E-graph* G *and a DSIG-algebra* D *such that* $\bigcup_{s \in S'_D} D_s = G_{V_2}$*.*

The class of all attributed graphs with the attributed graph morphisms constitutes an adhesive HLR category. Consequently, all the properties already given are present in the framework. The definition of the rules are very similar to the definition with simple graphs: some conditions on the morphisms l and r are added in order to ensure that rewriting is possible.

Definition 3. *A transformation rule* $p : L \leftarrow K \rightarrow R$ *is given by three attributed graphs (with variables)* K*,* L*, and* R *and two morphisms* $l : K \to L$ *and* $r : K \to R$ *which have to be injective on the graph structure and isomorphic over the graph attributes.*

In order to describe computations on the attributes, we have to use terms that contain variables; for example, in the graph R, an attribute $x + y$ can be found if in the graph K the variables x and y are present.

The main drawback of this approach is the heterogeneous structures used to define a graph (sets and algebraic signatures). The way of dealing with attributes leads to a huge graph: a node is created for each value of a variable. Changing the value of an attribute attached to a node consists in canceling the edge connected to the old value to the node and creating a new edge whose target is the new value. If this solution is theoretically acceptable, it cannot be easily implemented. For this reason, in the AGG environment [21] the computations of attributes are directly executed in an external programming language.

3 The "Double Pushout Pullback" Approach

The main idea of our study was to find a uniform framework to describe the graphs and the morphisms. Keeping to the same conceptual scheme as in the above constructions, the goal was to put the attribute computation to work in a more uniform way. We would like to remain within the same theory for implementing computations.

The Double Pushout Pullback solution (abbreviated as DPoPb) uses inductive types to code the attributed graphs: finite types to describe the structure of the graphs and general inductive types to define the data types. This choice has been mainly made for two reasons. The first one was to take advantage of the power of type theory in order to be able to prove some logical properties on the graph transformations (*e.g.* giving pre-conditions and post-conditions for a transformation for each rule of the system). The second reason concerns the future implementation of this theory: some proof assistants like Coq [7] are very efficient to work with the inductive types; moreover, in Coq, libraries for categorical computations (such as pushout or pullback) are available.

One of the earlier problems met was working with the attributes: for example, how to copy the value of an attribute during the transformation? This transformation has to stand during the second pushout because it is a creation process. So we have an attribute attached to a node of the graph K and we want to copy it to the graph R. With a classical morphism from K to R, this attribute can have only one image in the graph R and it is then impossible to share the information it is carrying between the two copies.

The central idea of the DPoPb approach is to use the power of the pullback for computing attribute values. *E.g.*, in the theory of graph transformations based on pullback (see for example the introduction of [15]), copies of arbitrary graphs are easily described using only one rule, while in the pushout approach, one rule is needed for each graph. The use of the pullback is justified by the possibility of reversing the arrows between the attributes (see definitions below). The whole construction can be seen as a pushout in a suitable category, but also, for a more intuitive point of view, as a mix between a pushout and a pullback.

We will now proceed in giving the formal definitions for the category of attributed graph used for the DPoPb approach.

Structure. The following definition uses "Coq-style" notation [7]:

Definition 4. *The graph structure G consists of an inductive finite type for the nodes* $S^G = \mathtt{Ind}()(\mathtt{S^G} : \mathtt{Set} := \mathtt{s_1} : \mathtt{S^G}, \ldots, \mathtt{s_n} : \mathtt{S^G})$, *with a function* A^G *for the edges. This function is typed by* $A^G : (x : S^G)(y : S^G) FiniteTypes$.

The number of constructors s_i in the type S^G gives the number of the nodes in the graph. Similarly, for each pair of nodes (s_1, s_2), the number of constructors in $A^G(s_1, s_2)$ stands for the number of edges starting from s_1 and arriving on s_2.

Attributes. For clarity, only attributes on the nodes will be considered below. The extension of definitions to data on edges can be given in a similar way.

To attach attributes to the structure, we will proceed in two steps. The first step consists in giving to each node the types of the attributes we want to bind to the node. Then, the values of the attributes are defined.

During the first step, we want to retain the possibility to put on a node several attributes of the same type. For this reason, we use a relation rather than a partial function as described in [14]. It must also be possible to distinguish these attributes. To that purpose, we use the notion of conform copies of inductive types (see [5]). An inductive type T' will be called a conform copy of the inductive type T if T' differs from T only in the names of the constructors.

Definition 5. *A labelling relation is a relation between the nodes of a graph and the copies of the inductive types (we have to define before how many copies of a given inductive type are needed for the graph).*

This relation allows to provide each node with the types of the possible data we want to attach to this node.

For a graph G, the labelling relation will be denoted by $\mathcal{R}^G$ and for a node s in S^G, $\mathcal{R}^G_s$ will denote the set of data type in relation with the node s.

Now, we define a function that takes a node from a graph and a data type in relation with this node and gives the attribute. An attribute does not have to be in the inductive type: it can also be undefined, it will then be called a joker and be denoted by ❀. This possibility will be useful for the graph transformations: the joker will play the role of a variable.

Definition 6. *For the structure of a graph G and a labelling relation $\mathcal{R}^G$, the attribution function is a map of type:*

$$AS^G : \left(\prod s : S^G \left(\prod DT : \mathcal{R}^G_s \, (DT \cup ❀)\right)\right).$$

Definition 7. *An attributed graph will be given by a structure (nodes S^G and edges A^G), a labelling relation $\mathcal{R}^G$, and an attribution function AS^G.*

This definition shows a main difference between the DPoPb approach and the HLR theory. In [10], the attributes are encapsulated in some special nodes of the graph. So in order to attach information to a node s of the structure, it is necessary to put an edge between s and the node which carries the right value for the attribute. As a consequence, if a sort has an infinity numbers of values, the resulting graph will theoretically speaking be infinite. With the above definitions, we only attach the required attributes to nodes.

Let G and H be two attributed graphs. Now, the notion of morphism between G and H is defined.

Structure morphisms. On the structure, the morphism will just be defined by classical functions: $f_S : S^G \rightarrow S^H$ for the nodes and f_A of type $(x : S^G_n)(y : S^G_n)(A^G(x, y) \longrightarrow A^H(f_S(x), f_S(y))$ for the edges. In order to avoid losing information during morphism application, a new compatibility condition is added. For each node, its image by f_S must contain at least the same data types; *i.e.*, for each node s in the graph G and for each data type DT in relation with s then DT is also in relation with $f_S(s)$. In the following, the pair (f_S, f_A) is denoted by "structural morphism".

Data type morphisms. We now suppose that we have a structural morphism $f = (f_S, f_A)$ from the graph G to the graph H. The intuitive idea in the definition of morphisms on the attributes is the following: unlike the arrows between the nodes and the edges, the arrows between the data types are starting from the graph H and finishing on the graph G. The first step of the construction is to detect which attributes of H are related to those of G (definition 9 and definition 10). With this construction, the morphisms between data types can be constituted by arrows with multiple sources (but only one target). Such "arrows" are called trees and can describe complex relations: for example, if we want to add two natural numbers, we will get a tree with two sources (the numbers to add) and the target will be the result. In the second step, for each tree, a computation function will be added in order to describe the computation (definition 11).

Definition 8. *Let A and B be two sets. A multi-valued relation between A and B is a relation which allows $a \in A$ and $b \in B$ to be in relation several times. Formally, a multi-valued relation can be seen as a multiset on the Cartesian product $A \times B$.*

The first requirement about this relation is that a data type DT attached to the node s in the graph G has to be in relation with the same data type attached to the node $f_S(s)$ in the graph H (this data type exists in H due to the condition on the morphism in section 3). With this condition, the information carried by the attributes is not lost during the application of the morphism.

Definition 9. *Let $\mathcal{R}^f$ be a multi-valued relation between the dependent types $\left(\prod s : S^H \ \mathcal{R}_s^H\right)$ and $\left(\prod s : S^G \ \mathcal{R}_s^G\right)$. This relation will be compatible with the structural morphism f if the following condition holds: for all node s in S^G, for all data type DT in relation with s,*

$$(s, DT) \ \mathcal{R}^f \ (f_S(s), DT)\,.$$

This condition defines the "necessary elements" of a compatible relation. The (multi-)set of the data types in the graph H in relation with (s, DT) is denoted $\mathcal{R}^f_{(s,DT)}$.

The following definition explains how other attributes of the graph G can influence attributes of the graph G.

Definition 10. *Let $\mathcal{R}^f$ be a relation compatible with f. A partitioning associated to f and $\mathcal{R}^f$ is given by a partition of $\mathcal{R}^f_{(s,DT)}$ for each node $s \in S^G$ and each data type DT in relation with s.*

Each element of one of these partitions will be called a tree. The compatible relation with f and the partitioning will then be called the forest of the morphism.

Attribute morphisms. As we now have defined how data types are related to the others, it is possible to show the way how this relation works. For this purpose, for each tree of the morphism, an application is defined.

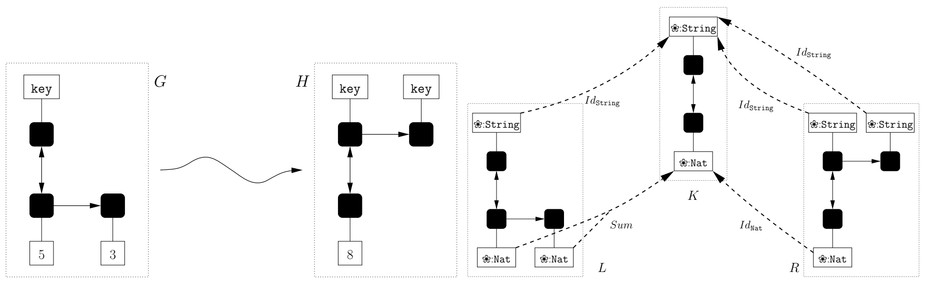
Definition 11. *A computation function is a function defined for each tree of a morphism. Its domain is the Cartesian product of the data types at the leaves of the tree (corresponding to the graph H) and its codomain is the data type at the root (corresponding to the graph G). Is is coherent with the graph G and H if the image of the attributes at the leaves of the tree is equal to the attributes on* *the root. In case of jokers, the condition is weaker: a computation function can send any attribute on a joker, but a joker is necessarily sent on a joker.*

Definition 12. *A morphism between two attributed graphs is given by a structural morphism, a forest, and for each tree of the forest, a computation function.*

Theorem 1. *The class of attributed graphs forms a category, called* **AttGraph**, *with the morphisms as arrows.*

The importance of reversing arrows for computation functions. To understand the reasons to change the orientation of arrows for computation functions, let us study a simple example. Figure 1(a) shows the transformation we want to apply to a source graph: the two attributes of the graph G have to be added and the result has to be stored in an attribute of H (the node carrying the second attribute is deleted during the transformation); parallely, a new node is created on H with a copy of the string in the source graph G.

Since the context graph of such transformation contains only two nodes and two data types (Nat and $String$), addition of two natural numbers has to take place during the first pushout. We have then to define a function between the Nat data type of the context graph and the two Nat data types of the source graph G summing the two numbers. It is then more natural to start from the graph G. Then the computation function will follow the intuitive definition of the transformation. Moreover, it is impossible to find an inverse function to the summing function. Looking for having the arrows for compution functions in the same direction as the ones for the structure is then no use. The other part of the transformation is similar: since there is only one $String$ data type in the context graph, the copy of the word `key` has to be done in the second pushout. With a classical function, the value of an attribute can only be sent to one place; but by changing the orientation of the arrows for the functions, the attribute in graph R can easily "go and pick up" any value of attributes in the graph K; then, several attributes can then share the same value.



(a) A graph rewrite (b) The computation functions of the rule

Fig. 1. Reversing the arrows

4 The Pushout

Let A, B, and C be three attributed graphs and $b : A \to B$ and $c : A \to C$ two morphisms.

Definition 13. *The pushout of* b *and* c *is given by an attributed graph* D *with two morphisms* $b' : C \to D$ *and* $c' : B \to D$ *such that the square* $ABCD$ *is commutative (i.e.* $c' \circ b = b' \circ c$*) and the following universal property is fulfilled: for all attributed graph* D' *and morphisms* $\tilde{b} : C \to D'$ *and* $\tilde{c} : B \to D'$ *with* $\tilde{c} \circ b = \tilde{b} \circ c$*, there is a unique morphism* $d : D \to D'$ *such that* $\tilde{b} = d \circ b'$ *and* $\tilde{c} = d \circ c'$*.*

Intuitively speaking, this construction allows to glue the graphs B and C along a pattern (the graph A).

This leads to another question: if the morphisms b and c' are given, is it possible to find a graph C and morphisms c and b' in order to complete the square? The solution to this problem is called the pushout-complement of b and c'.

In the category of simple graphs, the objects are built with sets (one set for the nodes and another one for the edges). Due to this fact, the pushout of two morphisms always exists, since we know how to construct arbitrary pushouts in the category of sets.

In the category **AttGraph**, due to the trees and the computation functions, it is not possible to use the same simple method to find the pushout of two morphisms. Moreover, it is very easy to find examples where the construction is impossible. For example, contradictions may arise in the placement of the computation functions: on the same tree, we would have to attach two different values of the associated computation function. In order to avoid these drawbacks, we will impose some conditions on the morphisms. These conditions will not be very restrictive for graph rewriting: most of them could be seen as natural for a transformation rule.

The first class of morphisms we will define is used to describe graph transformations. In a first time, two technical conditions are imposed for this class to ensure the existence of the pushout in the most general case. In section 5, it will be explained how to get around of this restrictions.

Definition 14. *The class* $\mathcal{M}$ *of morphisms of attributed graphs is composed by the morphisms of which the structural part is injective and the computation functions are bijective.*

The second class will be useful in order to extract exactly a graph into a larger one: the morphism has to be injective on the structure (the condition a of the definition 15) and the attributes should be equal in the two graphs (except on the jokers) (the conditions b and c).

Definition 15. *The class* $\mathcal{N}$ *of morphisms of attributed graphs is composed by the morphisms verifying the following properties:*

a. the structural part is injective;
b. the relation between data types is restricted to the necessary elements;
c. the computation functions are the identity functions.

From the intuitive point of view, since the arrows for computation functions are reversed, the construction of the pushout in the category **AttGraph** can be seen as a pullback construction for these computation functions. Hence, we call our approach the "Double Pushout Pullback" approach, even if we do not compute effectively any pullback during a graph transformation.

Theorem 2. *If the morphism b is in the class $\mathcal{M}$ and the morphism c is in the class $\mathcal{N}$, then the pushout of b and c exists up to the verification of the compatibility of the attributes.*

Theorem 3. *Similarly, the construction of the pushout-complement of a morphism $b \in \mathcal{M}$ and a morphism $c' \in \mathcal{N}$ exists up to the verification of the compatibility of the attributes.*

In these two theorems, the construction of the new attributed graph can be blocked by differences of concrete values of some attributes. This problem is not just a theoretical complication but reflects real modeling situations: if we want to merge two attributes with different values, it is normal to stop on this difficulty because we cannot choose a natural value for the new attribute.

There are several ways to address this problem. The conservative one is to say that the transformation is impossible. We are then certain to get no incoherent result. The second solution consists in "factorizing" the data types corresponding to problematic attributes in order to identify the incompatible values (the constrains resolution induced by this identification may lead in some cases to a data type with only one element). With this approach, even if some information is lost during the pushout, the construction can be completed and in some cases, it could be interesting to always get a result, even if it is partial.

5 Graph Rewriting

Before explaining how graphs are rewritten, we describe how to generalize the above construction of the pushout by getting around of the restrictions imposed in the definition of the class $\mathcal{M}$. In the explicit constructions of the pushout and the pushout-complement, the following observations can be made: the condition of injectivity on the structural part of the morphism b is useless in the construction of the pushout and the condition on the computation functions of b is useless in the construction of pushout-complement. These observations are very important for the description of transformation rules. The first step in the application of a transformation is a computation of a pushout-complement, so in the left-hand part of the rule, the morphism does not need to have isomorphic computation functions. This allows to do complex computations with the attributes. The second step of the transformation is the construction of a pushout, so the right-hand part of the rule could be non-injective and then the merging of nodes or edges will be possible.

Definition 16. *The class $\mathcal{M}'$ of morphisms of attributed graphs is composed by the morphisms which the structural part is injective.*

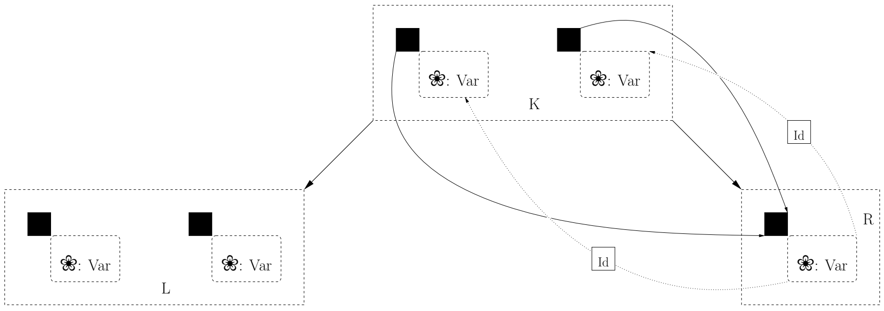
Definition 17. *The class $\mathcal{M}''$ of morphisms of attributed graphs is composed by the morphisms with isomorphic computation functions.*

Definition 18. *A transformation rule in the* DPoPb *approach is given by the attributed graphs K, L and R and two morphisms $l : K \to L$ belonging to the class $\mathcal{M}'$ and $r : K \to R$ belonging to the class $\mathcal{M}''$.*

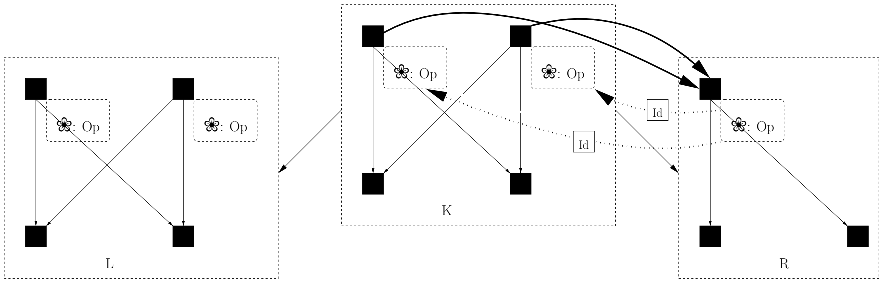
To apply such a rule to a graph G, we have to find a match of the graph L in the graph G, *i.e.* a morphism $m : L \to G$ in the class $\mathcal{N}$. Then we compute the pushout-complement of l and m to find a new graph D with a morphism $m' : K \to D$. This construction is useful to delete elements of the structure and make complex computations on the attributes. Finally we compute the pushout of m' and r to find the result. During this step, structural elements can be created and attributes be copied.

6 Example

We give here an example corresponding to the transformation of syntax trees for arithmetic expressions into direct acyclic graphs. This transformation, detailed



(a) Merging variables



(b) Merging operations

Fig. 2. The two rules to transform a syntax tree

in [1], consists in recognizing identical subtrees in the original tree in order to reduce its size. The rewriting consists in a first step to merge the nodes with the same variable and in a second step to merge nodes and edges representing the same sub-expression. The two rewriting rules are shown in figure 2. For clarity, only the non trivial computation functions are represented on the schemes. For these two rules, the left hand side is the identity, thus we can simplify the construction of the transformation by computing only one pushout. Moreover, in order to ensure the correct application of these rules, some negative application conditions need to be added (see [19] for more details).

7 Conclusion

We outline the difficulties of the heterogeneous character of attributed graph transformations and the computations which are concerned. We suggest a combined approach by pushouts and pullbacks which can express many kinds of computations. The use of inductive types opens a way for efficient use of very established proof assistant systems like Coq. It is to be noticed that this formal framework offers all the interesting properties required to work efficiently with a graph rewriting system (the local Church-Rosser and parallelism properties, confluence, or critical pair analysis). Moreover, like in the DPO approach in the adhesive HLR categories, transformations of typed graph are possible. A work dedicated to implementation will be a natural continuation of this study.

Acknowledgments. We thank Reiko Heckel for helpful discussions and comments. This work was supported by the European project TYPES and the pluridisciplinary action PEPS from French CNRS.

References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques and Tools. Addison-Wesley, Reading (1988)
2. Aizawa, K., Nakamura, A.: Path-controlled graph grammars for multi-resolution image processing and analysis. In: Ehrig, H., Schneider, H.-J. (eds.) Dagstuhl Seminar 1993. LNCS, vol. 776, pp. 1–18. Springer, Heidelberg (1994)
3. Assmann, U.: How to uniformly specify program analysis and transformation with graph rewrite systems. In: Gyimóthy, T. (ed.) CC 1996. LNCS, vol. 1060. Springer, Heidelberg (1996)
4. Baresi, L., Heckel, R.: Tutorial introduction to graph transformation: A software engineering perspective. In: Corradini et al. [6], pp. 402–429
5. Chemouil, D.: Types inductifs, isomorphismes et récriture extensionnelle. PhD thesis, Université Paul Sabatier (2004)
6. Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.): ICGT 2002. LNCS, vol. 2505. Springer, Heidelberg (2002)
7. The Coq development team. The Coq proof assistant reference manual: Version 8.1. Technical report, LogiCal Project (2006)

8. Ehrig, H.: Introduction to the algebraic theory of graph grammars (A survey). In: Ng, E.W., Ehrig, H., Rozenberg, G. (eds.) Graph Grammars 1978. LNCS, vol. 73, pp. 1–69. Springer, Heidelberg (1979)
9. Ehrig, H., Ehrig, K.: Overview of formal concepts for model transformations based on typed attributed graph transformation. Electr. Notes Theor. Comput. Sci. 152, 3–22 (2006)
10. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
11. Ehrig, H., Pfender, M., Schneider, H.J.: Graph-grammars: An algebraic approach. In: FOCS, pp. 167–180. IEEE, Los Alamitos (1973)
12. Ehrig, H., Prange, U., Taentzer, G.: Fundamental theory for typed attributed graph transformation. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 161–177. Springer, Heidelberg (2004)
13. Grunske, L., Geiger, L., Zündorf, A., Van Eetvelde, N., Van Gorp, P., Varró, D.: Using Graph Transformation for Practical Model Driven Software Engineering. In: Model Driven Software Engineering, pp. 91–118. Springer, Heidelberg (2005)
14. Habel, A., Plump, D.: Relabelling in graph transformation. In: Corradini et al. [6], pp. 135–147
15. Kahl, W.: A relational-algebraic approach to graph structure transformation. PhD thesis, Universität der Bundeswehr München (2001)
16. Kastenberg, H.: Towards attributed graphs in Groove: Work in progress. Electr. Notes Theor. Comput. Sci. 154(2), 47–54 (2006)
17. Lack, S., Sobocinski, P.: Adhesive categories. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 273–288. Springer, Heidelberg (2004)
18. Lladós, J., Sánchez, G.: Symbol recognition using graphs. In: ICIP (2), pp. 49–52 (2003)
19. Rebout, M.: Algebraic transformations for attributed graphs. Technical report, IRIT, Toulouse (2007)
20. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1. World Scientific, Singapore (1997)
21. Taentzer, G.: AGG: A tool environment for algebraic graph transformation. In: Münch, M., Nagl, M. (eds.) AGTIVE 1999. LNCS, vol. 1779, pp. 481–488. Springer, Heidelberg (2000)

Author Index

Allender, Eric 3
Artemov, Sergei 30
Arvind, Vikraman 40
Avron, Arnon 52

Babenko, Maxim A. 64
Bedon, Nicolas 76
Beloglazov, Dmitry 360
Bès, Alexis 76
Bläser, Markus 86
Bodyanskiy, Yevgeniy 339
Boros, Endre 99

Carton, Olivier 76
Caucal, Didier 110
Cherukhin, Dmitriy Yu. 122
Churina, Tatiana 360

Dal Lago, Ugo 134
Das, Bireswar 40
Dell, Holger 86
Deviatov, Rostislav 146

Elbassioni, Khaled 99

Féraud, Louis 398
Fiala, Jiří 158
Fitting, Melvin 168
Flarup, Uffe 180
Formenti, Enrico 194

Ganguly, Sumit 204
Gurvich, Vladimir 99

Hassen, Stéphane 110
Hofmann, Martin 134

Ivolgin, Dmitriy 372

Kari, Jarkko 194
Khasidashvili, Zurab 11
Köbler, Johannes 40, 216
Kovács, Laura 349
Kuznets, Roman 228

Limaye, Nutan 240
Lyaudet, Laurent 180

Mahajan, Meena 240
Makino, Kazuhisa 99
Makowsky, Johann A. 86
Mashukov, Mikhail 360
Meyer, Antoine 240
Murphy, Brian 372

Nepomniaschy, Valery 360
Nogina, Elena 30

Osin, Denis 252
Oudalov, Vladimir 99

Pan, Victor Y. 372
Paulusma, Daniël 158
Pavlovic, Dusko 384
Podolskii, Vladimir V. 261
Popov, Sergiy 339
Pudlák, Pavel 13

Rebout, Maxime 398
Rispal, Chloé 76
Rosholt, Rhys Eric 372
Rybalchenko, Taras 339

Savateev, Yury 273
Shpilrain, Vladimir 252, 283
Shur, Arseny M. 289
Soloviev, Sergei 398
Starikovskaya, Tatiana A. 64

Taati, Siamak 194
Tang, Yuqing 372

V'yugin, Vladimir V. 302
Verbitsky, Oleg 216

Wigderson, Avi 1
Wong, Limsoon 28

Xiao, Mingyu 314

Yan, Xiaodong 372

Zamansky, Anna 52
Zimand, Marius 326
Zwick, Uri 29