Andrei A. Bulatov
Arseny M. Shur (Eds.)

LNCS 7913

Computer Science – Theory and Applications

8th International Computer Science Symposium in Russia, CSR 2013
Ekaterinburg, Russia, June 2013
Proceedings

Springer

Lecture Notes in Computer Science 7913

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

Andrei A. Bulatov Arseny M. Shur (Eds.)

Computer Science – Theory and Applications

8th International Computer Science Symposium in Russia, CSR 2013
Ekaterinburg, Russia, June 25-29, 2013
Proceedings

Springer

Volume Editors

Andrei A. Bulatov
Simon Fraser University
School of Computing Science
8888 University Drive, Burnaby, BC V5A 1S6, Canada
E-mail: abulatov@sfu.ca

Arseny M. Shur
Ural Federal University
Institute of Mathematics and Computer Science
51 Lenina Ave., 620083 Ekaterinburg, Russia
E-mail: arseny.shur@usu.ru

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38535-3 e-ISBN 978-3-642-38536-0
DOI 10.1007/978-3-642-38536-0
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013938303

CR Subject Classification (1998): F.2, F.3, F.1, G.2, F.4, G.3

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

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

Printed on acid-free paper

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

Preface

The 8th International Computer Science Symposium in Russia (CSR 2013) was held during June 25–29, 2013, in Ekaterinburg, Russia, hosted by the Ural Federal University. It was the eighth event in the series of regular international meetings following CSR 2006 in St. Petersburg, CSR 2007 in Ekaterinburg, CSR 2008 in Moscow, CSR 2009 in Novosibirsk, CSR 2010 in Kazan, CSR 2001 in St. Petersburg, and CSR 2012 in Nizhny Novgorod.

The opening lecture was given by Mario Szegedy and seven other invited plenary lectures were given by Thomas Colcombet, Gilles Dowek, Alexandr Kostochka, Nicole Schweikardt, Jeffrey Shallit, Paul Spirakis, and Ryan Williams.

This volume contains the accepted papers and abstracts of the invited talks. The scope of the proposed topics for the symposium is quite broad and covers a wide range of areas in theoretical computer science and its applications. We received 52 papers in total, and out of these the Program Committee selected 29 papers for presentation at the symposium and for publication in the proceedings.

As usual, Yandex provided the Best Paper Awards; the recipients of these awards were selected by the Program Committee:

- Best Paper Award: Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein
  "Information Lower Bounds via Self-Reducibility"
- Best Student Paper Award: Luke Friedman and Yixin Xu
  "Exponential Lower Bounds for Refuting Random Formulas Using Ordered Binary Decision Diagrams"

The reviewing process was organized using the EasyChair conference system created by Andrei Voronkov. We would like to acknowledge that this system helped greatly to improve the efficiency of the committee work.

The following satellite events were co-located with CSR 2013:

- The Second Workshop on Current Trends in Cryptology (CTCrypt 2013)
- The 4th Workshop on Program Semantics, Specification and Verification: Theory and Applications (PSSV 2013)
- The 6th School for Students and Young Researchers "Computer Science Ekaterinburg Days" (CSEdays 2013)

We are grateful to our sponsors:

- Russian Foundation for Basic Research
- SKB Kontur
- Eastwind
- Yandex

We greatly appreciate the assistance of the many departments of the Ural Federal University. Our special thanks to Dmitry Bugrov, the first vice-rector, and to Magaz Asanov, the director of the Institute of Mathematics and Computer Science. We also thank the local organizers: Alexei Borbunov, Alexandr Galperin, Irina Polyakova, Elena Pribavkina, Ekaterina Shcherbakova, Mikhail Volkov, and Alexei Zverev.

June 2013

Andrei Bulatov
Arseny Shur

Organization

CSR 2013 was organized by the Ural Federal University in Ekaterinburg.

Program Committee Chair

Andrei Bulatov	Simon Fraser University, Canada

Program Committee

Max Alekseyev	University of South Carolina, USA
Andris Ambainis	University of Latvia, Latvia
Maxim Babenko	Higher School of Economics, Russia
Patrick Baillot	ENS Lyon, France
Glencora Borradaile	Oregon State University, USA
Yijia Chen	Shanghai Jiao Tong University, China
Victor Dalmau	Universitat Pompeu Fabra, Spain
Yevgeniy Dodis	Courant Institute of Mathematical Sciences, New York University, USA
Manfred Droste	University of Leipzig, Germany
Anna Frid	Sobolev Institute of Mathematics, Novosibirsk, Russia
Hamed Hatami	McGill University, Canada
Tero Harju	University of Turku, Finland
Michal Koucky	Mathematical Institute of Czech Academy of Sciences, Czech Republic
Stephan Kreutzer	Technical University of Berlin, Germany
Alexander Kulikov	Steklov Institute of Mathematics at St. Petersburg, Russia
Konstantin Makarychev	Microsoft Research, Redmond, USA
Simone Martini	University of Bologna, Italy
Jaroslav Nesetril	Charles University, Czech Republic
Jean-Eric Pin	Université Paris 7, France
Harald Raecke	University of Munich, Germany
Alexander Razborov	University of Chicago, USA, and Steklov Mathematical Institute, Russia
Mikhail Volkov	Ural Federal University, Russia

Symposium Chair

Arseny Shur	Ural Federal University, Russia

Organizing Committee

Alexandr Galperin	Ural Federal University, Russia
Elena Pribavkina	Ural Federal University, Russia
Arseny Shur (Chair)	Ural Federal University, Russia
Mikhail Volkov	Ural Federal University, Russia
Alexei Zverev	Ural Federal University, Russia

Steering Committee

Anna Frid	Sobolev Institute of Mathematics, Russia
Edward A. Hirsch	Steklov Institute of Mathematics at St. Petersburg, Russian Academy of Sciences, Russia
Juhani Karhumäki	University of Turku, Finland
Ernst W. Mayr	Technical University of Munich, Germany
Alexander Razborov	University of Chicago, USA, and Steklov Mathematical Institute, Russia
Mikhail Volkov	Ural Federal University, Russia

External Reviewers

Artem Babenko	Anton Korobeynikov	Michel Rigo
Ivan Bliznets	Gregory Kucherov	Michael Roizner
Gabor Braun	Peter Leupold	Rahul Santhanam
Christian Choffrut	Ingmar Meinecke	Maximilian Schlund
Diego Figueira	Theresa Migler	Sylvain Schmitz
Alexander Golovnev	Ivan Mihajlin	Alexander Shen
Yuri Gurevich	Sergey Nikolenko	George Voutsadakis
Pooya Hatami	Vsevolod Oparin	John Watrous
Shuai Jiang	Ivan Pouzyrevsky	Thomas Weidner
Stanislav Kikot	Narad Rampersad	
Ignat Kolesnichenko	Christian Retore	

Sponsoring Institutions

Russian Foundation for Basic Research
SKB Kontur
Eastwind
Yandex

Table of Contents

Invited Lecture 2

Session 3: Logic, Proof Complexity

Invited Lecture 3

Session 4: Complexity 1

Invited Lecture 4

Session 5: Words and Languages

Invited Lecture 5

Session 6: Algorithms 2

Invited Lecture 6

Session 7: Complexity 2

The Lovász Local Lemma – A Survey

Mario Szegedy*

Rutgers University, Piscataway NJ, USA

Abstract. The Local Lemma of Lovász has affected multiple sciences. We survey its impact on mathematics, computer science and statistical physics.

1 Introduction

Lovász in the early seventies invented his celebrated local lemma (LLL) [11] to prove the *existence* of combinatorial objects that satisfy a prescribed sparse set of constraints. His beautiful proof was inherently non-constructive. Nearly two decades later J. Beck [5] presented a *constructive proof*, which was however seen as technical. In 2008 Robin A. Moser [23], and in 2009 Moser and Gábor Tardos [24] turned the LLL research around by giving a constructive proof with a simple RESAMPLE process (algorithm) at its heart. Although the process (or similar ones) had been under the radar screen of other researchers [25][29], finding its connection to the LLL was a milestone. We survey past and recent research related to the LLL, raise some questions, and take a look at the plethora of subjects it has impacted from mathematics through computer science to statistical physics.

2 Versions of the Lovász Local Lemma

The *variable version* (or setting) of the Lovász Local Lemma is most elegantly defined by Moser and Tardos [23][24]. Let $\{X_1, \ldots, X_m\}$ be mutually independent random variables. In the space they define we have n events, viewed as *constraints*, and we denote by A_i $(1 \leq i \leq n)$ the event that the i^{th} constraint does not hold. We denote by $\text{vbl}(A_i)$ the set of variables on which A_i depends. In particular, if $\text{vbl}(A_i) \cap \text{vbl}(A_i) = \emptyset$ then A_i and A_j are independent. Let

p_i:= $\text{Prob}(A_i)$, the probability of the event we try to avoid;
G:= the *dependency graph*: $V(G) = [n]$, $E(G) = \{(i,j) \mid \text{vbl}(A_i) \cap \text{vbl}(A_i) \neq \emptyset\}$.

The LLL gives conditions in terms of G and $\overline{p} = (p_1, \ldots, p_n)$ for the existence of an assignment to $X_1, \ldots, X_m$ which satisfies all constraints (i.e. avoids all A_is). In the history of the lemma many different conditions have been considered [11][32]. The simplest: $p_i \leq \frac{1}{e(d+1)}$ $(1 \leq i \leq n)$, where d is the maximum degree of G. Let $N(i)$ be the set of neighbors of a node i in G, together with i itself. The most cited condition is

* The author was supported by NSF grant CCF-0832787.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 1–11, 2013.

$\angle(G,\overline{p})$**:** There are $0 < x_i < 1$ such that $p_i \leq x_i \prod_{j\in N(i)\setminus\{i\}}(1-x_j)$.

Abstract Setting. While almost all applications in mathematics and computer science use the variable version of LLL, Lovász's original formulation was more abstract and more general: Let $\{A_1,\ldots,A_m\}$ be a family of events in any probability space with $\mathrm{Prob}(A_i) = p_i$, and let G be a *dependency graph* on [n], i.e. it must hold that for each i the event A_i is independent from the σ-algebra generated by the events $\{A_j \mid j \in [n]\setminus N(i)\}$ (recall that $N(i)$ is the set of neighbors of i plus i itself). Throughout the paper we assume that G is undirected, but in a section we will also discuss the case of directed dependency graphs.

Remark 1. Interesting examples show, that the sparsest graph G satisfying the above is not necessarily unique.

As in the variable version, under some Condition$(G,\overline{p})$ we want to arrive at the conclusion

$$\mathrm{Prob}(\bigcap_{i=1}^{n}\overline{A_i}) > 0.$$

Clearly, for every Condition$(G,\overline{p})$ under which the abstract holds, the variable version does too. The standard Lovász Local Lemma states that in the abstract setting:

Theorem 1 (Standard LLL). *If* $\angle(G,\overline{p})$ *holds then* $\mathrm{Prob}(\bigcap_{i=1}^{n}\overline{A_i}) > 0$.

The strongest possible Condition for the abstract setting was given by Shearer [31]. To state it, we need the notion of the *independence polynomial*. It is a multivariate polynomial, $\mathrm{Z}(G,\overline{z}) = \sum_{I\in\mathrm{Ind}(G)}\prod_{i\in I} z_i$. Here $\mathrm{Ind}(G)$ is the set of all independent sets of G, including the empty set. Shearer's condition is then:

$\angle^{+}(G,\overline{p})$**:** For every subgraph G' of G it holds, that $\mathrm{Z}(G',-\overline{p}|_{V(G')}) > 0$.

Shearer has also proved, that for every G and $\overline{p}$ that do not meet his condition there is an instance, that violates the Lovász Local Lemma's conclusion. Kashyap Kolipaka and the author have shown, that if we restrict ourselves to the variable version, Shearer's bound is not sharp for $G = C_4$ [19].

Efficient LLL. While the LLL is an existence statement, in the case of the variable version it makes sense to ask how efficiently a solution can be found. Efficiency in this case means constructing an evaluation of the variables in random polynomial time (i.e. in time random poly(m,n)) that satisfies all of the n constraints. The first efficient algorithm was due to Beck [5], which was followed by several improvements [1][22][9][33]. In all of these the conditions on G and $\overline{p}$ under which $\mathrm{Prob}(\bigcap_{i=1}^{n}\overline{A_i}) > 0$ was shown were more binding than in the non-efficient version. Furthermore, all algorithms and their proofs contained a lot of technicalities, and in particular, they used the original LLL as a component. The turnaround came when R. Moser [23] and a year later, in a more general and streamlined way, R. Moser and G. Tardos [24] invented their breakthrough Resample algorithm:

Algorithm 1. RESAMPLE

- Assign random values to $X_1, \ldots, X_m$;
- While $\exists\ 1 \leq i \leq n$ such that A_i is not violated by the current assignment, do:
  - Choose the smallest such i and **resample** all variables $X_j \in \mathrm{vbl}(A_i)$;
- Return the current assignment;

Theorem 2 (Moser-Tardos). *If $\angle(G, \overline{p})$ holds for a system of constraints in the variable setting, then* RESAMPLE *finds a solution in expected time $\sum_{i=1}^{n} \frac{x_i}{1-x_i}$. In particular, in the variable setting $\angle(G, \overline{p})$ implies that the solution exists.*

In a further development K. Kolipaka and the author have proven, that condition $\angle$ above can be replaced with the stronger $\angle^+$ [19].

The RESAMPLE **algorithm**. One might consider it a historic accident that RESAMPLE was analyzed after the LLL was invented. Had it happened in the other way around, the natural question would be: Under what conditions is the success of the RESAMPLE algorithm guaranteed for a system of constraints? It is not a priory clear that we want to answer to this question solely in therms of G and $\overline{p}$, but if we do, first it would be good to know how far we may be able to go:

Problem: What is the ultimate Condition(G,$\overline{p}$) for the variable version of LLL?

We know from [19] that the answer is not $\angle^+$. Let us denote the ultimate condition for the variable version by $\angle^?(G, \overline{p})$. Since a solution can be found only if one exists, the most generous condition, that depends only on G and $\overline{p}$, under which RESAMPLE may find a solution is $\angle^?(G, \overline{p})$.

Problem: Does RESAMPLE *succeed in polynomial time under the condition $\angle^?$?*

Observe, that if we drop the restriction "polynomial time" from the Problem, the answer becomes obviously "yes," since if there is a solution at all, RESAMPLE will eventually find it. It is a small miracle that the efficiency of RESAMPLE is proven (in a very natural way) exactly up-to $\angle^+$. Can we understand better what is behind this? The issue with RESAMPLE is further complicated by the fact that we have swept under the rug the method of selecting a violated constraint in the resample step. Indeed, the Moser-Tardos theorem holds for any (even adversarial) choice of a violated constraint. Experiments however indicate that the selection process matters.

Problem: Try to find the best method of selecting a violated constraint in the resample step of RESAMPLE.

The Lopsided LLL. We can compute $\mathrm{Prob}(\bigcap_{i=1}^{n} \overline{A_i})$ from the inclusion-exclusion formula as long as we know the probability of the other $2^n - 1$ atomic events created by the A_is. This is barely helpful because of the large number of atoms and the alternating signs in the formula. The LLL is valuable exactly because we can deduce the positiveness of the above probability from simple information on G and $\overline{p}$. It may occur however, that simple information on the A_is other than G and $\overline{p}$

can lead us to the conclusion we want to get. There is at least one well known condition, that relies on other than G and $\overline{p}$ alone. This is the *lopsided* LLL [12]. Here we also have a graph G, but now we can choose it arbitrarily. The price for this is that we have to replace every p_i with

$$q_i = \max_{S:S\subseteq[n]\setminus N(i)} \mathrm{Prob}(\, A_i \mid \bigcap_{j\in S} \overline{A_j}\,).$$

If we replace $\overline{p}$ with $\overline{q}$, and the dependency graph with the new G then we can turn conditions $\angle$ and $\angle^+$ into new conditions, $\angle_\ell$ and $\angle_\ell^+$ such that the LLL still holds for these. Again a little miracle happens: $\angle_\ell$ and $\angle_\ell^+$ also guarantee the efficiency of Resample.

Despite of the generality of the lopsided LLL, it is not easy to apply it since the max takes exponentially many arguments. In some cases, however, we can find this maximum value easily. Let us restrict ourselves to the variable version. Moser and Tardos define A_i and A_j to be *lopsidependent*, if there exist two evaluations, α and β, of the variable sequence that differ in values only on the variables in $vbl(A_i)\cap vbl(A_j)$ such that α violates A_i and β violates A_j but either α does not violate A_j or β does not violate A_i. If instead of the usual dependency graph we connect i and j when A_i and A_j are lopsidependent as above, we get a graph that has the same or less edges than the original dependency graph, and $\overline{q} = \overline{p}$.

Problem: find other simple local conditions for the constraints (in the variable framework) that advantageously translate to some abstract lopsided condition.

The Clique LLL. Another example, when a condition becomes very natural in the variable setting is the *Clique LLL* of K. Kolipaka, Y. Xu and the author [20]. It is also a constraint that involves only G and $\overline{p}$, and is similar to $\angle$ in the sense that its application requires appropriately set real numbers that range in $[0,1]$, but now we have more than n numbers. For simplicity here we only describe the condition for the variable version of LLL:

$\angle^{\angle}$ (in the variable setting): Let $j \rightsquigarrow i$ denote $X_j \in \mathrm{vbl}(A_i)$. For every $1 \le i \le n$ and $1 \le j \le m$ such that $j \rightsquigarrow i$ there is a number $x_{i,j} \in [0,1]$ such that

- $\forall\ 1 \le j \le m:\ \sum_{i:\ j\rightsquigarrow i} x_{i,j} < 1$,
- $\forall\ 1 \le i \le n, \forall\ 1 \le j \le m:\ \ p_i \le x_{i,j} \prod_{j'\neq j:\ j'\rightsquigarrow i}(1 - \sum_{i'\neq i:\ j'\rightsquigarrow i'} x_{i',j'})$.

Other Intermediate Versions: Condition $\angle^{\angle}$ gives strictly better bounds, than $\angle$, but worse bounds, than $\angle^+$. Its great advantage, however, that it is poly-concise, unlike $\angle^+$. In [20] a hierarchy of decreasingly concise bounds are presented between $\angle$ and $\angle^+$, which includes $\angle^{\angle}$. A bound due to R. Bissacot, R. Fernandez, A. Procacci and B. Scoppola [7] is also intermediate between $\angle$ and $\angle^+$ To compare their condition with $\angle$ we restate the latter as

$\angle(G,\overline{p})$ **(restated):** There are $0 \le \mu_i$ such that $p_i \le \mu_i / \prod_{j\in N(i)}(1+\mu_j)$.

Notice that $\prod_{j\in N(i)}(1+\mu_j) = \sum_{H\subseteq N(i)} \prod_{j\in H} \mu_j$ (the empty product is 1 by definition). We do not increase the sum if we let it run only over the independent sets of $N(i)$, thus getting a more generous condition:

$\angle^{\odot}(G,\overline{p})$**:** There are $0 \le \mu_i$ such that $p_i \le \mu_i / \sum_{H\subseteq N(i);\, H\in \mathrm{Ind}(N(i))} \prod_{j\in H} \mu_j$.

It turns out that LLL holds under $\angle^{\odot}(G,\overline{p})$ as well.

Lefthanded Local Lemma. An interesting variant is the lefthanded local lemma, due to W. Pegden [26], [27]. The lemma does not hold for all dependency graphs, but for the ones it does, it gives an equal or stronger bound than the Standard LLL. For chordal graphs the lemma is optimal, meaning, it is equivalent to Shearer's [27], but the expression involved resembles far more to $\angle$ than to $\angle^{+}$, and in particular it does not use sums that run over subsets of events, and it is easily verifiable. In other words, Pegden shows that $\angle^{+}$ greatly simplifies when G is chordal.

The Issue of Undirectedness. For Shearer's bound ($\angle^{+}$) the graph G has to be undirected. On the other hand, even for the simplest LLL as usually stated

Theorem 3. *Let $A_1, A_2, ..., A_n$ be events such that each A_i is is mutually independent of all but d of the others. If $p\cdot e\cdot(d+1)\le 1$ then* $\mathrm{Prob}(\cap_{i=1}^{n}\overline{A_i}) > 0$.

The underlying graph is directed. The following example may be enlightening. Let H be a graph, and we randomly color its vertices with k colors. Let A_i be the event that the i^{th} edge, $e_i = (u,v)$, is not well colored, i.e. its two endpoints u and v get the same color. Let S_u and S_v be the set of edges that are incident to u and v, respectively. In the Moser-Tardos framework, discussed in the beginning, the neighbors of event A_i are $\{A_j \mid j \in S_u \cup S_v\}$. But we can do better: notice, that A_i is mutually independent from the system of events that contain all but the events in $\{A_j \mid j \in S_u\}$. The same holds when we replace u with v. This shows two phenomena at once. First, the set of events to be excepted, so that A_i is independent from the rest, may not be unique, and second, that if directed dependency graphs exist, they may be sparser than any undirected dependency graph for the same system. We will briefly return to this example in Section 3.

Problem: Investigate if there is an analogue of Shearer's bound for directed dependency graphs. We do not anticipate to have a simple analogue. Another question is if we can somehow exploit, when there are more than one dependency graphs.

Btw, graph G for the Standard LLL as stated in the original [11] is directed. We do not have to restate anything, just replace "neighbors" with "out-neighbors" in the definition and we get the directed version, which then implies Theorem 3. This is all in the abstract setting. In the Moser-Tardos variable setting the dependency is defined via the non-emptyness of $\mathrm{vbl}(A_i) \cap \mathrm{vbl}(A_i)$ (thus G is undirected) which is stronger than actual dependency. Our example above shows that actual dependency can be a strictly weaker requirement.

Problem: If we replace the Moser-Tardos notion of dependency (in the variable framework) with actual dependency (or with a notion in-between) can we say anything about the running time of Resample*?*

The Quantum LLL. It has become customary to extend probabilistic statements to the quantum setting, and LLL could not avoid this fate either, due to Andris Ambainis, Julia Kempe and Or Sattath [4]. Actually, there is nothing quantum about the quantum LLL, except its consequence to k-QSAT (a quantum problem). It talks about vector spaces and dimensions. For a subspace A of vector space X we define $R(A) = \dim(A)/\dim(X)$. We say that a subspace A is mutually independent from a set $\mathcal{S}$ of subspaces if for any $\{B_1, \ldots, B_k\} \subseteq \mathcal{S}$ of this set it holds that $\dim(A \cap \bigcap_{i=1}^k B_k)/\dim(\bigcap_{i=1}^k B_k) = R(A)$.

Theorem 4 (Quantum LLL). *Let $A_1, A_2, ..., A_n$ be subspaces, where $R(A_i) \geq 1 - p$ and such that each subspace is mutually R-independent of all but d of the others. If $p \cdot e \cdot (d+1) \leq 1$ then $\dim(\cap_{i=1}^n A_i) > 0$.*

3 Applications in Computer Science and Mathematics

The most typical application of LLL is to show that sparse constraint systems (with the sparsity parameter depending on the constraint types) always have solutions. We start our discussion with a well known statement of this type that does *not* rely on LLL:

Proposition 1. *If an undirected graph has maximal degree $k-1$ it is k colorable.*

If instead of the usual greedy coloring argument we try to prove this proposition from the Lovász Local Lemma, we get a weaker statement. Let A_i be the event that under random coloring with k colors the i^{th} edge is illegally colored. Then $\text{Prob}(A_i) = \frac{1}{k}$. We then use Theorem 3 and the structure of the *directed* dependency graph for the above problem, we have discussed in the paragraph after Theorem 3. We get that to have a good coloring the following is sufficient:

$$p = \frac{1}{k} \leq \frac{1}{e(d+1)} \quad \longrightarrow \quad d \leq \frac{k}{e} - 1.$$

The resulting bound on d, although not the best possible, is in the right order of magnitude: $d = \theta(k)$. In addition, due to [24], if $d \leq \frac{k+e}{2e}$, then Resample (i.e. randomly recoloring bad edges until we have none) runs in polynomial time.

For any k determine the largest possible d such that k coloring graphs with maximum degree d, with the Resample *algorithm, takes polynomial time.*

The power of LLL reveals itself in the hyper-graph two-coloring problem: Color the vertices of a k-hyper graphs such that no hyper-edge becomes monochromatic. The constraints thus correspond to the hyper-edges. Under a random coloring each hyper edge is well-colored with probability $1 - 2^{-k+1}$. We say that two hyper-edges are neighbors if they properly intersect (the edge-graph of the hyper-graph). If now the hyper-graph is sparse, i.e. d is suitably small, then there is always a good coloring. How small d is small enough? The LLL gives:

$$p = \frac{1}{2^{-k+1}} \leq \frac{1}{e(d+1)}$$

and so $d = \theta(2^k)$. This turns out to be the right order of magnitude. No argument (or algorithm) other than LLL (and RESAMPLE) seems to be of help in achieving a similarly strong conclusion.

The kSAT Problem. Every constraint C_i of a *strict* (or *exact*) kSAT instance is a disjunction $C_i = \ell_{1,i} \wedge \ldots \wedge \ell_{k,i}$ of k literals (variables or their negations), such that the underlying variables are all different. The latter condition is necessary to ensure that the probability of the event A_i that the i^{th} constraint does not hold under a random assignment is exactly 2^{-k}. A kSAT instance is typically written as $\Phi = \bigwedge_{i=1}^{n} C_i$, i.e. as a Bool formula we want to satisfy (the alternative would be just to list the constraints). In computer science kSAT is the "archetypical NP-hard problem" [15], which is under constant investigation. The density of a kSAT instance is measured by not only one, but at least by two different parameters:

1. Let $f(\Phi) = \max_{i=1}^{m} f_i$, where f_i is the number of occurrences of x_i (negated or not) in Φ. For a positive integer k let $f(k)$ denote the largest integer such that for every kSAT formula Φ it holds that if $f(\Phi) \leq f(k)$ then Φ is satisfiable. The notion $f(k)$ was introduced in [21].
2. Let $l(\Phi)$ denote the minimal integer such that all clauses of Φ intersect at most $l(\Phi)$ other clauses of Φ. For a positive integer k let $l(k)$ denote the largest integer such that for every kSAT formula Φ it holds that if $l(\Phi) \leq l(k)$ then Φ is satisfiable. The notion $l(k)$ was introduced in [14].

Theorem 3 immediately implies that $l(k) \geq \left\lfloor \frac{2^k}{e} \right\rfloor - 1$. Since for every kSAT formula Φ the inequality $(k-1)f(\Phi) \geq l(\phi)$ trivially holds, we immediately get that $f(\Phi) \geq \left\lfloor \frac{l(\phi)}{k} \right\rfloor + 1 \geq \left\lfloor \frac{2^k}{e \cdot k} \right\rfloor$. It would be a good guess that the lower bounds we get for $f(k)$ and $l(k)$ above are tight within a factor of $1 + o(1)$. This is indeed the case for $l(k)$, but for $f(k)$ it turns out that one can improve on the lower bound by a factor of two. The version of LLL needed for the tighter bound is the lopsided version [6], [15]. In [15] it is shown:

$$\left\lfloor \frac{2^{k+1}}{ek} \right\rfloor - 1 \leq f(k) \leq \left(1 + \frac{O(1)}{\sqrt{k}}\right) \frac{2^{k+1}}{ek} \tag{1}$$

$$\left\lfloor \frac{2^k}{e} \right\rfloor - 1 \leq l(k) \leq \left(1 + \frac{O(1)}{\sqrt{k}}\right) \frac{2^k}{e} \tag{2}$$

The upper bounds are of course constructions of sparse kSAT instances that are not satisfiable. A sequence of constructions [28], [18], [13] have lead to the construction in [15]. (A trivial construction that consists of all possible k-clauses on k variables already implies $l(k) < 2^k - 1$. To get the right order of magnitude for $f(k)$ is much trickier.)

Problem: Determine $f(k)$ and $l(k)$ more accurately.

The exact value of $f(k)$ has the following significance: Kratochvíl, Savický, and Tuza [21] have shown that the satisfiability problem for kSAT instances Φ

with $f(\Phi) = f(k) + 1$ is already NP complete. Thus at $f(k)$ a sharp complexity "phase transition" occurs (for deterministic instances) from trivial to hard.

Non-regular Dependency Graphs. Thus far we could apply Theorem 3 because the probabilities of the bad events were the same. The problems we shall consider in the sequel are such that the constraints greatly vary in size and the associated probabilities are different. In these cases we need the Standard LLL or one of its improvements.

Acyclic Edge Coloring. Given an undirected graph G, an acyclic edge coloring is a proper edge coloring such that no cycle is 2-edge-colored. The acyclic edge chromatic number of G is the minimum number of colors in an acyclic edge coloring of G and is denoted by $a'(G)$. Let $a'(d)$ denote the maximum value $a'(G)$ over all graphs with max-degree at most d. Alon, McDiarmid and Reed in [3] consider acyclic edge coloring among other things, and use the LLL to prove that $a'(d) < 64d$. The bound is currently $a'(d) \leq 8.6d$ [20].

Non-repetitive Vertex Coloring. Given an undirected graph G, a non- repetitive vertex coloring is a proper coloring of the vertices of G such that there is no simple path with an even number of vertices such that the sequence of colors in the first half is the same as the sequence of colors in the second half. The minimum number of colors in such a coloring for G, called the Thue number of G, is denoted by $\pi(G)$. Let $\pi(d) = \max\{\pi(G) \mid \text{max-degree of } G \leq d\}$. Alon, Grytczuk, Hauszczak and Riordan [2] proved that $\pi(d)$ is in $O(d^2)$ and in $\Omega(d^2/\log d)$. Their proof of the upper bound uses the LLL. In a recent development, Dujmovic, Joret, Kozik and Wood [10] have shown that $\pi(d) \leq (1 + o(1))d^2$, employing an entropy-compression argument similar to the one in Moser and Tardos [23], [24].

Games. W. Pegden in [26] applies LLL to show that in certain games a particular player has a winning strategy. A representative example of the types of games he considers: Player 1 and Player 2 take turns to create an infinite binary string character by character. Player 2 wins if in the string a contiguous block of $k > 100$ characters is never repeated in $h(k)$ or less steps, where $h(k)$ is carefully given. The existence of a winning strategy for Player 2 is via the LLL.

Super-Polynomial Number of Events. In [17] a setting is described, where the number of events, n, is possibly super-polynomial in the number of variables, m. We run RESAMPLE as before, but assume access to an oracle that provides an A_i that holds, if one exists. It turns out that "when a LLL application provides a small amount of slack, the number of resamplings of the Moser-Tardos algorithm is nearly linear in the number of underlying independent variables (not events!)." The authors "obtain the first constant-factor approximation algorithm for the Santa Claus problem by making a LLL-based proof of Feige constructive" [17]. Further improvements over some of their results are made in [19].

De-randomization. While it was a decisive step to make the LLL efficient in a very elegant way, the most efficient algorithm is randomized. The first de-randomization of the Moser-Tardos algorithm was given by Moser [23] and

Moser and Tardos [24] themselves, but their running time bound, $n^{\theta(k^2)}$ for kSAT, grows exponentially with k, although remains polynomial in the number of clauses, n. An $O(n^{2+1/\epsilon})$ was presented in [8], which eliminates this dependence on k, but questions about de-randomization still remain.

4 The Lovász Local Lemma and Statistical Mechanics

Physicists have known for some time the connection between LLL and the Repulsive Lattice Gas model [35][16][34]. The connection is rather formal, and only through expressions that are "accidentally" identical. The independence set polynomial,

$$\mathrm{Z}(G, \overline{z}) = \sum_{I \in \mathrm{Ind}(G)} \prod_{i \in I} z_i$$

of a graph G happens to coincide with the *partition function* of the system, where we have a "container" G of gas, with the rules that:

1. Each node of G is either occupied by a (single) particle or is empty;
2. Two particles cannot occupy two neighboring nodes (hard-core repulsive property);
3. The particles are indistinguishable;
4. They can be created or destroyed, so the particle number is not fixed.

Clearly, the possible states of such a physical system are in one-one correspondence with the elements of $\mathrm{Ind}(G)$. The probability that the system is in any one of the states is a mathematical expression, with $\mathrm{Z}(G, -\overline{z})$ in its denominator. Here vector $\overline{z}$ is a physical parameter that depends on the system (together with the graph G). In statistical mechanics it is important to determine the range of parameter $\overline{z}$ for which $\mathrm{Z}(G, -\overline{z}) \neq 0$, since this is the domain, where the above probabilities make sense. Alexander D. Scott and Alan D. Sokal [30] in their ninety pages article describe many formulas associated with $\mathrm{Z}(G, -\overline{z})$. Their "Fundamental theorem" rewrites condition $\angle^+$ in many equivalent ways, and in particular it proves that:

Theorem: $\mathrm{Z}(G, \overline{z}) \neq 0$ for every $|\overline{z}| \leq \overline{p}$ (coordinate-wise) if and only if $\angle^+(G, \overline{p})$ holds.

This, of course, doubles the motivation to study sufficient conditions for $\angle^+(G, \overline{p})$ such as $\angle^{\angle}$ and $\angle^{\odot}$. One of the goals of Scott and Sokal was to connect the efforts of physicists and computer scientists in finding easy sufficient conditions for $\angle^+$.

Problem: Find a more essential connection between statistical mechanics and LLL, and in particular a physical interpretation of the Resample *algorithm.*

Acknowledgements. Naturally, in this short survey we could not do justice to all LLL related research, for instance we have not mentioned the very interesting case of infinite number of events or the applications of LLL to Ramsey theory. The original article of Erdős and Lovász is cited 547 times according to Google Scholar. Kashyap Kolipaka has multiply contributed to this survey.

References

1. Alon, N.: A parallel algorithmic version of the Local Lemma. In: FOCS, pp. 586–593 (1991)
2. Alon, N., Grytczuk, J., Haluszczak, M., Riordan, O.: Nonrepetitive colorings of graphs. Random Struct. Algorithms 21(3-4), 336–346 (2002)
3. Alon, N., McDiarmid, C., Reed, B.A.: Acyclic coloring of graphs. Random Struct. Algorithms 2(3), 277–288 (1991)
4. Ambainis, A., Kempe, J., Sattath, O.: A quantum lovász local lemma. In: STOC, pp. 151–160 (2010)
5. Beck, J.: An algorithmic approach to the Lovász Local Lemma. i. Random Struct. Algorithms 2(4), 343–366 (1991)
6. Berman, P., Karpinski, M., Scott, A.D.: Approximation hardness and satisfiability of bounded occurrence instances of sat. Electronic Colloquium on Computational Complexity (ECCC) 10(022) (2003)
7. Bissacot, R., Fernandez, R., Procacci, A., Scoppola, B.: Combinatorics Probability and Computing 20(5), 709–719 (2011)
8. Chandrasekaran, K., Goyal, N., Haeupler, B.: Deterministic algorithms for the lovász local lemma. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, pp. 992–1004. Society for Industrial and Applied Mathematics, Philadelphia (2010)
9. Czumaj, A., Scheideler, C.: Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász Local Lemma. In: SODA, pp. 30–39 (2000)
10. Dujmovic, V., Joret, G., Wood, D.R.: Nonrepetitive colouring via entropy compression. CoRR abs/1112.5524 (2011)
11. Erdös, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In: Hajnal, A., Rado, R., Sós, V.T. (eds.) Infinite and Finite Sets (to Paul Erdös on his 60th birthday), pp. 609–627 (1975)
12. Erdös, P., Spencer, J.: Lopsided Lovász Local Lemma and latin transversals. Discrete Applied Mathematics 30(2-3), 151–154 (1991)
13. Gebauer, H.: Disproof of the neighborhood conjecture with implications to sat. Combinatorica 32(5), 573–587 (2012)
14. Gebauer, H., Moser, R.A., Scheder, D., Welzl, E.: The lovász local lemma and satisfiability. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms. LNCS, vol. 5760, pp. 30–54. Springer, Heidelberg (2009)
15. Gebauer, H., Szabó, T., Tardos, G.: The local lemma is tight for sat. In: SODA, pp. 664–674 (2011)
16. Guttmann, A.J.: Comment: Comment on the exact location of partition function zeros, a new method for statistical mechanics. Journal of Physics A Mathematical General 20, 511–512 (1987)
17. Haeupler, B., Saha, B., Srinivasan, A.: New constructive aspects of the lovász local lemma. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, FOCS 2010, pp. 397–406. IEEE Computer Society, Washington, DC (2010), http://dx.doi.org/10.1109/FOCS.2010.45
18. Hoory, S., Szeider, S.: A note on unsatisfiable k-cnf formulas with few occurrences per variable. SIAM J. Discrete Math. 20(2), 523–528 (2006)
19. Kolipaka, K.B.R., Szegedy, M.: Moser and Tardos meet Lovász. In: STOC, pp. 235–244 (2011)
20. Kolipaka, K., Szegedy, M., Xu, Y.: A sharper local lemma with improved applications. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX/RANDOM 2012. LNCS, vol. 7408, pp. 603–614. Springer, Heidelberg (2012)

21. Kratochvíl, J., Savický, P., Tuza, Z.: One more occurrence of variables makes satisfiability jump from trivial to np-complete. SIAM J. Comput. 22(1), 203–210 (1993)
22. Molloy, M., Reed, B.A.: Further algorithmic aspects of the Local Lemma. In: STOC, pp. 524–529 (1998)
23. Moser, R.A.: A constructive proof of the Lovász Local Lemma. In: STOC, pp. 343–350 (2009)
24. Moser, R.A., Tardos, G.: A constructive proof of the general Lovász Local Lemma. J. ACM 57(2) (2010)
25. Papadimitriou, C.H.: On selecting a satisfying truth assignment (extended abstract). In: FOCS, pp. 163–169 (1991)
26. Pegden, W.: Highly nonrepetitive sequences: Winning strategies from the local lemma. Random Struct. Algorithms 38(1-2), 140–161 (2011)
27. Pegden, W.: The lefthanded local lemma characterizes chordal dependency graphs. Random Struct. Algorithms 41(4), 546–556 (2012)
28. Savický, P., Sgall, J.: Dnf tautologies with a limited number of occurrences of every variable. Theor. Comput. Sci. 238(1-2), 495–498 (2000)
29. Schöning, U.: A probabilistic algorithm for k-sat and constraint satisfaction problems. In: FOCS, pp. 410–414 (1999)
30. Scott, A.D., Sokal, A.D.: On dependency graphs and the lattice gas. Combinatorics, Probability & Computing 15(1-2), 253–279 (2006)
31. Shearer, J.B.: On a problem of Spencer. Combinatorica 5(3), 241–245 (1985)
32. Spencer, J.: Asymptotic lower bounds for Ramsey functions. Discrete Mathematics 20(0), 69–76 (1977)
33. Srinivasan, A.: Improved algorithmic versions of the Lovász Local Lemma. In: SODA, pp. 611–620 (2008)
34. Todo, S.: Transfer-matrix study of negative-fugacity singularity of hard-core lattice gas. International Journal of Modern Physics C 10, 517–529 (1999)
35. Wood, D.W.: The exact location of partition function zeros, a new method for statistical mechanics. Journal of Physics A: Mathematical and General 18(15), L917 (1985)

An Improved Knapsack Solver for Column Generation*

Klaus Jansen and Stefan Kraft

Department of Computer Science, Theory of Parallelism,
Christian-Albrechts-University to Kiel, 24098 Kiel, Germany
{kj,stkr}@informatik.uni-kiel.de

Abstract. The Knapsack Problem (KP) and its variants are well-known NP-hard problems. Their study is also driven by approximation algorithms for optimization problems like Bin Packing: these algorithms must often solve KP instances as subproblems. In this paper, we introduce the Knapsack Problem with Inversely Proportional Profits (KPIP), a generalization of KP: in it, one of several knapsack sizes has to be chosen. At the same time, the item profits are inversely proportional to the chosen knapsack size so that it is non-trivial to take the right knapsack. We adapt Lawler's approximation scheme for KP to faster solve KPIP. Thus, we are able to improve the running time of an approximation scheme for Variable-Sized Bin Packing that solves KPIP as a subproblem.

Keywords: Knapsack Problem, Unbounded Knapsack Problem, Bounded Knapsack Problem, Variable-Sized Bin Packing, Column Generation.

An instance I of the Knapsack Problem (KP) consists of a list of items $a_1, \ldots, a_n$, $n \in \mathbb{N}$, where every item has a profit $p(a_j) = p_j \in \mathbb{N}$ and a size $s(a_j) = s_j \in \mathbb{N}$. Moreover, a knapsack size $c \in \mathbb{N}$ is given. In the 0-1 Knapsack Problem (0-1 KP), a subset $V \subset \{a_1, \ldots, a_n\}$ has to be chosen such that the total profit of V is maximized and the total size of the items in V is at most c. Mathematically, the problem can be defined by $\max\{\sum_{j=1}^{n} p_j x_j | \sum_{j=1}^{n} s_j x_j \leq c; x_j \in \{0,1\} \forall j\}$. In the bounded variant (BKP), up to $d_j \in \mathbb{N}$ copies of each item a_j may be taken (i.e. $x_j \in \{0, \ldots, d_j\}$), and in the unbounded variant (UKP), an arbitrary number of copies of every item is allowed (i.e. $x_j \in \mathbb{N}$).

In the 0-1 Knapsack Problem with Inversely Proportional Profits (0-1 KPIP), the items have sizes $s_j \in (0,1]$ and basic profits $p_j \in (0,1]$. Moreover, M knapsack sizes $0 < c_1 < \ldots < c_M = 1$ are given. If an item a_j is packed into the knapsack of size c_l for $l \in \{1, \ldots, M\}$, its profit counts as $\frac{p_j}{c_l}$. A smaller knapsack size $c_{l_1} < c_{l_2}$ may therefore have a solution with a larger profit than an optimal solution for knapsack c_{l_2}. The goal is to determine the knapsack size c_l and the corresponding item set V such that the total profit is maximized. Mathematically, the problem is defined by $\max_{l \in \{1,\ldots,M\}} \max\{\sum_{j=1}^{n} \frac{p_j}{c_l} x_j | \sum_{j=1}^{n} s_j x_j \leq$

* Research supported by DFG project JA612/14-1, "Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme".

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 12–23, 2013.

$c_l; x_j \in \{0,1\} \forall j\}$. The bounded variant (BKPIP) and the unbounded variant (UKPIP) are defined similar to BKP and UKP.

1 Introduction

1.1 Known Results

The 0-1 knapsack problem and its variants are well-known NP-hard problems [2]. They can be optimally solved in pseudo-polynomial time by dynamic programming [1,9]. Furthermore, approximation schemes $(A_\varepsilon)_{\varepsilon>0}$ are known: for every $\varepsilon > 0$, there is an algorithm A_ε that finds a solution with profit $A_\varepsilon(I) \geq (1-\varepsilon)OPT(I)$, where $OPT(I)$ denotes the optimal value. Such schemes with running time polynomial in n and $\frac{1}{\varepsilon}$ are called fully polynomial time approximation schemes (FPTAS). Several FPTAS are known for 0-1 KP [5,10,11][9, pp. 166–183], BKP [13, p. 296],[9] and UKP [5,10,9].

Many algorithms of optimization problems like Bin Packing rely on fast KP solvers for column generation: the algorithms have to solve linear programs, but enumerating all columns of the linear programs would take too much time. Instead, the algorithms solve KP instances whose solutions correspond to the columns needed. Examples can be found in [3,4,8,13].

In fact, the study of KPIP is motivated by column generation for Variable-Sized Bin Packing (VBP). In this generalization of the Bin Packing Problem, several bin sizes $C = \{c_1, \ldots, c_M\}$, with $0 < c_1 < \ldots < c_M = 1$, are given together with a set I of n items of size in $(0,1]$. An arbitrary number of copies of every bin size may be taken to pack the items, and the goal is to find the smallest total volume $OPT(I,C)$ of bins needed. Note that VBP becomes the normal Bin Packing Problem if there is only one bin size $c_1 = c_M = 1$. We [6] have recently presented an improved asymptotic fully polynomial time approximation scheme (AFPTAS), i.e. there is an algorithm for every $\varepsilon > 0$ that finds a packing of value of at most $(1+\varepsilon)OPT(I,C)+C(\frac{1}{\varepsilon})$, and its running time is polynomial in n and $\frac{1}{\varepsilon}$. This new AFPTAS has a smaller constant $C(\frac{1}{\varepsilon})$ and a better running time of $O\left(\frac{1}{\varepsilon^7}\log^2\frac{1}{\varepsilon} + (M+n)\log\frac{1}{\varepsilon}\right)$ than previously known algorithms [12,15]. The algorithm has to approximately solve instances of UKPIP for column generation. Hence, a faster UKPIP solver would directly improve the running time of the algorithm.

There is a simple way to solve an instance of 0-1 KPIP, BKPIP or UKPIP with n items and M knapsack sizes: take an FPTAS for the corresponding variant of the knapsack problem that can also handle fractional profits and sizes. For each knapsack c_l, solve the KP instance with item profits $\frac{p_j}{c_l}$. Return the solution and knapsack size of highest profit. Lawler's algorithm [10] for UKP yields a running time of $O(M \cdot (n + \frac{1}{\varepsilon^3}))$ and a space bound of $O(M + n + \frac{1}{\varepsilon^2})$ for UKPIP.

1.2 Our Result

By adapting Lawler's algorithm, we get an improved running time.

Theorem 1. *Let $\varepsilon > 0$. Let $c_{\min}$ be the smallest knapsack size of a UKPIP instance with M knapsacks and n item sizes. There is an FPTAS that solves this problem in time*

$$O\left(n \log M + \min\left\{\left\lfloor \log \frac{1}{c_{\min}} \right\rfloor + 1, M\right\} n + \min\left\{\left\lfloor \log \frac{1}{c_{\min}} \right\rfloor + 1, M\right\} \frac{1}{\varepsilon^3} + \frac{M}{\varepsilon^2}\right)$$

and space $O(M + n + \frac{1}{\varepsilon^2})$.

This directly improves the column generation subroutine of the AFPTAS for VBP [6]. Thus, the running time of the AFPTAS decreases by a factor of $\frac{1}{\varepsilon}$ (as shown in Section 5). Another small modification removes the factor $\log \frac{1}{\varepsilon}$ of M.

Theorem 2. *Let $\varepsilon > 0$. There is an AFPTAS for Variable-Sized Bin Packing that finds a solution for an instance with n items and M bin sizes in time $O\left(\frac{1}{\varepsilon^6} \log^2 \frac{1}{\varepsilon} + M + n \log \frac{1}{\varepsilon}\right)$.*

1.3 Techniques

As mentioned above, we adapt the classical knapsack algorithms by Lawler [10] for 0-1 KP as well as BKP and UKP to obtain FPTAS for 0-1 KPIP, BKPIP and UKPIP. The algorithms avoid redundancy and therefore have a faster running time.

For a normal 0-1 KP instance, Lawler's algorithm first computes an approximate solution with profit $P_0 \geq \frac{1}{2} OPT(I)$. Based on P_0, a threshold T is defined and the items are partitioned into large ones with profit $p_j > T$ and small ones with profit $p_j \leq T$. Since an optimal solution cannot contain too many items of large profit, it is sufficient to take a subset of them. The profits of these large items are then scaled, and the well-known dynamic programming by profits is used to compute several candidates. Small items are greedily added to each candidate to obtain several solutions, and the solution of highest profit is returned. For the unbounded and the bounded case, copies of large items are created to transform the respective problem instances into normal 0-1 KP instances.

Solving KPIP by applying Lawler's algorithm M times, i.e. once for each knapsack size, results in using dynamic programming M times, which is time-consuming. For large M, our algorithm avoids redundancy by first partitioning the knapsacks into intervals $C_b := (c_{\min}^{(b+1)w}, c_{\min}^{bw}]$ for $b \in \{0, \ldots, \lfloor \frac{1}{w} \rfloor\}$ and a suitable $w \leq 1$. The basic item profits p_j are scaled in a way suitable for all $c_l \in C_b$. Then, the dynamic programming is applied to these scaled profits to determine the candidates for all $c_l \in C_b$ at once: thus, dynamic programming is only executed once for the $c_l \in C_b$, which yields an improved running time. As above, the small items are then greedily added to the candidates. Finally, the solutions for $c_l \in C_b$ are scaled by $\frac{1}{c_l}$ to get the solutions for profits $\frac{p_j}{c_l}$. When all C_b have been considered, the solution of highest profit for all c_l, $l \in \{1, \ldots, M\}$, is returned. However, $\frac{c_{\min}^{bw}}{c_{\min}^{(b+1)w}} = \frac{1}{c_{\min}^{w}}$ is part of the running time for each execution of the dynamic programming. Hence, w is chosen in such a way that $O(\frac{1}{c_{\min}^{w}} \cdot \frac{1}{w})$, i.e. the running time of one dynamic programming computation times the number of computations, is minimized.

1.4 Notation and Remarks

$OPT_{c_l} = \max\{\sum_{j=1}^{n} \frac{p_j}{c_l} x_j | \sum_{j=1}^{n} s_j x_j \leq c_l; x_j \in D_j\}$ refers to the optimal solution for the current variant of KPIP when only knapsack size c_l is considered (with $D_j = \{0,1\}$ for 0-1 KPIP, $D_j = \{0,\ldots,d_j\}$ for BKPIP and $D_j = \mathbb{N}$ for UKPIP). $OPT = \max_{l\in\{1,\ldots,M\}} OPT_{c_l}$ denotes the global optimum.

We will also consider the case where the items only have their unscaled basic profit p_j, independent of the knapsack in which they are packed. $\overline{OPT}_{c_l} = \max\{\sum_{j=1}^{n} p_j x_j | \sum_{j=1}^{n} s_j x_j \leq c_l; x_j \in D_j\}$ denotes the optimum for knapsack size c_l (with D_j defined as above) and $\overline{OPT} = \max_{l\in\{1,\ldots,M\}} \overline{OPT}_{c_l}$ the global optimum. (Note that $\overline{OPT} = \overline{OPT}_{c_M}$.) This notation will also be used for the normal Knapsack Problem.

As stated above, we denote the basic profit of an item a by $p(a)$ or $p(a_j) = p_j$ if $a = a_j$, and we do the same for the size $s(a)$ and $s(a_j) = s_j$. We also define the total basic profit $p(V) := \sum_{a\in V} p(a)$ and the total size $s(V) := \sum_{a\in V} s(a)$ of an item set V.

We assume throughout the paper that basic arithmetic operations as well as computing the logarithm can be performed in $O(1)$. Missing proofs can be found in the technical report [7].

2 Observations

Two simple observations help us to handle that the profits depend on the knapsack size.

Lemma 3. *For every variant of KPIP, it is sufficient to calculate a $(1-\varepsilon)$ approximate solution with profits $\frac{p_j}{c_l}$ for every knapsack c_l and take as final solution the maximum over all knapsack sizes.*

Lemma 4. *Let c_l be a knapsack size. A $(1-\varepsilon)$ approximate solution $(x_1,\ldots,x_n)$ of one variant of KP for knapsack size c_l with the basic profits p_j is also a $(1-\varepsilon)$ solution for the corresponding variant of KPIP with profits $\frac{p_j}{c_l}$, and vice versa.*

The following algorithm therefore finds an approximate solution for any variant of KPIP:

Algorithm $MaxSolution(a_1,\ldots,a_n; c_1,\ldots,c_M)$
(1) **for** all c_l, $l \in \{1,\ldots,M\}$ **do**
Find a solution $(x_1^{(c_l)},\ldots,x_n^{(c_l)})$ with $\sum_{j=1}^{n} p_j x_j^{(c_l)} \geq (1-\varepsilon)\,\overline{OPT}_{c_l}$ **od**
(2) **for** all c_l, $l \in \{1,\ldots,M\}$ **do** Compute $P_{c_l} := \frac{1}{c_l} \sum_{j=1}^{n} p_j x_j^{(c_l)}$ **od**
(3) **return** $\max_{l\in\{1,\ldots,n\}} P_{c_l}$.

In fact, Lemma 4 shows that $P_{c_l} \geq (1-\varepsilon)OPT_{c_l}$. Taking the maximum over all P_{c_l} then yields a solution with objective value of at least $(1-\varepsilon)OPT$ according to Lemma 3.

3 Adapting Lawler's Algorithm

We have seen that Algorithm *MaxSolution* calculates the desired approximate solution. In this section, we will see how to adapt Lawler's algorithm for 0-1 KP [10] to 0-1 KPIP such that we obtain the approximate solutions in Step (1) of the algorithm, and how we avoid redundancy when calculating them. BKPIP and UKPIP will then be considered in Section 4.

First, we will introduce a basic technique for the Knapsack Problem.

3.1 Dynamic Programming

Dynamic Programming by Profits [1,9] is a well-known technique to optimally solve a 0-1 KP instance in pseudo-polynomial time. Let $F_j(i) \in \mathbb{R}_{\geq 0}$ be the minimum size necessary to obtain profit $i \in \mathbb{N}$ with the items $a_1, \ldots, a_j$, i.e. $F_1(0) = 0$ and $F_1(p_1) = s_1$. If a profit i cannot be obtained with the first j items, $F_j(i) = \infty$ is set, i.e. $F_1(i) = \infty$ for $i \neq 0, p_1$. The $F_j(i)$ are computed by $F_j(i) = \min\{F_{j-1}(i), F_{j-1}(i - p_j) + s_j\}$. Let $c_1 < \ldots < c_M$ be several knapsack sizes. Obviously, $\overline{OPT}_{c_l} = \max\{i \mid F_n(i) \leq c_l\}$ holds (see also [3]), and the corresponding items can be found by backtracking in the table of the $F_j(i)$. Note that it is possible that $i \leq i'$, but $F_n(i) \geq F_n(i')$: a larger profit i' is achieved with a smaller size $F_n(i')$. Hence, $F_n(i)$ is *dominated* by $F_n(i')$.

Lemma 5. *Dominated table entries $F_n(i)$ can be discarded in $O(\overline{PB})$, where $\overline{PB}$ is an upper bound for the maximum profit* $\max_{l \in \{1,\ldots,M\}} \overline{OPT}_{c_l} = \overline{OPT}_{c_M}$.

All $\overline{OPT}_{c_l}$ can be found by a single scan of the $F_n(i)$ in time $O(M + \overline{PB})$.

Theorem 6. *The $F_j(i)$ can be determined in time and space $O(\overline{PB} \cdot n)$. Then, the optimal values $\overline{OPT}_{c_l}$ for $l \in \{1, \ldots, M\}$ can be found in time $O\left(\overline{PB} + M\right)$ and space $O(M)$. Dynamic programming also works for non-integral item sizes $s_j \in \mathbb{R}_{>0}$ and knapsack sizes $c_l \in \mathbb{R}_{>0}$.*

3.2 Bounds for the Optimum: A Simple $\frac{1}{2}$ Algorithm

We present a simple algorithm that provides a bound for $\overline{OPT}_{c_l}$. Let $S_1 := \{a_i \mid s(a_i) \leq c_1\}$ and $S_l := \{a_i \mid c_{l-1} < s(a_i) \leq c_l\}$ for $l \in \{2, \ldots, M\}$. (Note that we still have $c_1 < c_2 < \ldots < c_M$.)

Obviously, $\bar{S}_l := \bigcup_{l'=1}^{l} S_{l'}$ is the set of the items that have to be considered for a solution of knapsack size c_l. Based on an idea by Lawler [10], it is not difficult to construct the S_l: create M stacks, one for each S_l. Each item a is then added to the right stack by binary search.

Lemma 7. *The S_l can be created in time $O(n \log M + M)$ and space $O(n + M)$.*

The idea of the approximation for any c_l is simple: start with the item of largest efficiency $\frac{p(a)}{s(a)}$ in $\bar{S}_l$ and add items in $\bar{S}_l$ to the knapsack in non-increasing order of their efficiency until no more items can be added. Return the maximum of

this value and of $p_{l,\max} := \max\{p_j | s_j \leq c_l\}$. As seen in [10], this value $\bar{P}_{c_l}$ satisfies $\frac{1}{2}\overline{OPT}_{c_l} \leq \bar{P}_{c_l} \leq \overline{OPT}_{c_l}$. For 0-1 KPIP, let $P_{c_l} := \frac{\bar{P}_{c_l}}{c_l}$, which is a $\frac{1}{2}$ solution for c_l with profits $\frac{p_j}{c_l}$ according to Lemma 4. Lemma 3 shows that $P_0 := \max_{l\in\{1,\ldots,M\}} P_{c_l}$ is also a $\frac{1}{2}$ approximation of OPT with $\frac{1}{2}OPT \leq P_0 \leq OPT$.

Theorem 8. *For 0-1 KPIP, we have* $\bar{P}_{c_l} \geq \frac{1}{2}\overline{OPT}_{c_l}$ *and* $P_{c_l} \geq \frac{1}{2}OPT_{c_l}$. *Furthermore,* P_0 *is a* $\frac{1}{2}$ *approximation of the global optimum* OPT.

Corollary 9. *For 0-1 KPIP, we can discard knapsacks* c_l *with* $P_{c_l} = \frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$.

The $\bar{P}_{c_l}$ can be constructed without having to sort the items according to their efficiency by a median-based divide-and-conquer strategy (similar to [10]).

Lemma 10. *All* $\bar{P}_{c_l}$ *and therefore* P_0 *can be found in time* $O(M \cdot n)$ *and space* $O(M + n)$ *without sorting the items according to their efficiency.*

3.3 Scaling and Dividing: The Basic FPTAS

We introduce the basic algorithm, an adaptation of Lawler's algorithm [10], which also uses ideas by Sahni [14] as well as Ibarra and Kim [5]. The basic idea of Lawler's FPTAS for the normal 0-1 KP with knapsack size c is as follows:

First, a threshold T, which will be defined later, is introduced: items a_i with $p(a_i) \geq T$ are *large*, the other items *small*. For ease of notation, let $a_1, \ldots, a_{n_L}$ be the large items. They are scaled as follows: if $p(a_j) = p_j \in (2^k T, 2^{k+1}T]$, the item has the scaled profit $q(a_j) = q_j := 2^k \left\lfloor \frac{p_j}{2^k K} \right\rfloor$, where K will be defined later. (We assume that $q(a_j)$ can be computed in time $O(1)$ because k can be found in $O(1)$ with the logarithm.) Dynamic programming is applied to the large items with profits q_j, but unchanged sizes s_j, and dominated $F_{n_L}(i)$ are discarded.

Moreover, let $\phi(c - F_{n_L}(i))$ be the profit obtained by greedily filling up the remaining capacity $c - F_{n_L}(i)$ with small items: as in Subsection 3.2 items are added in non-increasing order of their efficiency $\frac{p_j}{s_j}$ (and starting with the item of largest efficiency) until adding the next item would result in a packing of size more than $c - F_{n_L}(i)$.

Lawler's algorithm for 0-1 KP then returns $\max_{F_{n_L}(i)\leq c} K \cdot i + \phi(c - F_{n_L}(i))$. We now want to use the same principle for KPIP and combine it with the idea from Subsection 3.1: first, the $F_{n_L}(i)$ are determined for the scaled large items and the dominated values discarded. For every c_l, we take

$$\bar{P}_l^{(1)} := \max_{F_{n_L}(i)\leq c_l} K \cdot i + \phi_l(c_l - F_{n_L}(i)) \ , \tag{1}$$

where ϕ_l only uses small items in $\bar{S}_l = \{a_i \mid s(a_i) \leq c_l\}$ to fill the remaining capacity $c_l - F_{n_L}(i)$. Therefore, the $F_{n_L}(i)$ are calculated only once and used to determine all $\bar{P}_l^{(1)}$ similar to Subsection 3.1.

Lemma 11. *Let* $\tilde{V}_{c_l}$ *be the items chosen by the algorithm for knapsack* c_l, *and let* $\overline{APP}_{c_l} := \sum_{a\in\tilde{V}_{c_l}} p(a)$ *be their profit. Let* $c_{\min} = c_1$ *be the smallest knapsack size.*

By setting $T := \frac{1}{2}\varepsilon \bar{P}_{c_{\min}}$ *and* $K := \frac{1}{4}\varepsilon^2 \bar{P}_{c_{\min}}$*, we have* $\overline{APP}_{c_l} \geq (1-\varepsilon)\overline{OPT}_{c_l}$ *for all* c_l*. Moreover,* $\overline{APP}_{c_l} \geq \bar{P}_l^{(1)} \geq (1-\varepsilon)\overline{OPT}_{c_l}$ *holds, and the algorithm also works for non-integral* $p_j \in \mathbb{R}_{>0}$*.*

The lemma, i.e. the correctness of the algorithm, is proved similar to Lawler's proof [10]. Algorithm *MaxSolution* then yields a $(1-\varepsilon)$ solution for 0-1 KPIP.

Let us now determine the running time. We will see how to optimize it by a change to the calculation of the $F_{n_L}(i)$.

Running Time for the Large-Item Computation. We have already seen in Theorem 6 the general running time of $O(n \cdot i_{\max})$ for the dynamic program, where $i_{\max}$ is the largest profit i of scaled items we have to consider. Therefore, we have to bound $i_{\max}$.

It is not difficult to see that $i_{\max} \leq \frac{\overline{OPT}}{K} = \frac{c_{\max} OPT_{c_{\max}}}{K} \overset{\text{Thm. 8}}{\leq} \frac{2c_{\max}P_0}{K}$ so that time in $O\left(\frac{2c_{\max}P_0}{K} \cdot n_L\right)$ is needed for the large-item computation.

We have $K = \frac{\varepsilon^2}{4}\bar{P}_{c_{\min}} = \frac{\varepsilon^2}{4} c_{\min} P_{c_{\min}}$ as seen in Lemma 11 and Subsection 3.2. Since we assume without loss of generality that $P_{c_l} \geq \frac{P_0}{2}$ (see Corollary 9), we get $i_{\max} \leq \frac{2c_{\max}P_0}{K} = \frac{2c_{\max}P_0}{\frac{\varepsilon^2}{4}c_{\min}P_{c_{\min}}} \leq \frac{8c_{\max}P_0}{\varepsilon^2 c_{\min}\frac{P_0}{2}} = \frac{16}{\varepsilon^2}\frac{c_{\max}}{c_{\min}}$. Hence, the large-item computation can be done in time $O(\frac{2c_{\max}P_0}{K} \cdot n_L) = O(\frac{16}{\varepsilon^2}\frac{c_{\max}}{c_{\min}} n_L)$.

The ratio $\frac{c_{\max}}{c_{\min}}$ may be quite large. Fortunately, there is a method to control it. Partition $[c_{\min}, c_{\max}]$ into intervals $C_b := (c_{\min}^{w\cdot(b+1)}, c_{\min}^{w\cdot b}]$ for $w > 0$ and $b \in \{0, \ldots, \lfloor \frac{1}{w} \rfloor\}$ (with the exception of the last interval, which is $[c_{\min}, c_{\min}^{w\cdot\lfloor\frac{1}{w}\rfloor}]$). The value for w will be chosen later.

For every C_b, we only consider the bin sizes $c_l \in C_b$. Therefore, we have an adapted threshold $T_b := \frac{1}{2}\varepsilon \bar{P}_{c_{b,\min}}$ and scaling factor $K_b := \frac{1}{4}\varepsilon^2 \bar{P}_{c_{b,\min}}$, where $c_{b,\min}$ is the smallest knapsack size in C_b. The disadvantage is obviously that we have to partition the items into large and small ones as well as scale the large items again if we consider another C_b. The advantage is a decreased running time: if $c_{b,\max}$ is the largest knapsack size in C_b, we have $\frac{c_{b,\max}}{c_{b,\min}} \leq \frac{c_{\min}^{w\cdot b}}{c_{\min}^{w\cdot(b+1)}} = \frac{1}{c_{\min}^{w}}$.

Should $M \leq \lfloor \frac{1}{w} \rfloor + 1$ hold, we instead set $C_b = C_l := \{c_l\}$ for $l \in \{1, \ldots, M\}$: we have $\frac{c_{b,\max}}{c_{b,\min}} = 1$ and less C_b than for partitioning into $(c_{\min}^{w\cdot(b+1)}, c_{\min}^{w\cdot b}]$.

Theorem 12. *Let* n_b *be the number of large items for knapsack interval* C_b*, and let* n_L *be an upper bound for all* n_b*. The overall time and space bound for the large-item computation in one knapsack interval* C_b *is in* $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}} n_b)$*, which is in* $O(\frac{1}{\varepsilon^2}\frac{1}{c_{\min}^{w}} n_L)$ *for* $M > \lfloor \frac{1}{w} \rfloor + 1$ *and in* $O(\frac{1}{\varepsilon^2} n_L)$ *otherwise.*

Note that the running time for re-partitioning and re-scaling is not considered yet, as well as the optimal choice of w.

Adding the Small Items Efficiently. Let C_b be a knapsack interval and $F_{n_b}(i)$ the values $F_j(i)$ that consider all n_b large items. For every c_l and $F_{n_b}(i) \leq c_l$,

we have to determine $\phi_l(c_l - F_{n_b}(i))$. One way to do this would be sorting all items according to their efficiency $\frac{p(a)}{s(a)}$. It is however possible to avoid sorting: we use median finding in a divide-and-conquer strategy similar to the idea by Lawler [10]. Details can be found in the technical report [7]. The algorithm also returns item sets $\tilde{J}_i$ from which the item choice for every $c_l - F_{n_b}(i)$ can be reconstructed.

Theorem 13. *For one $c_l \in C_b$, the values $\phi_l(c_l - F_{n_b}(i))$ can be determined in time $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}} + n \cdot \log(\frac{1}{\varepsilon}\frac{c_{b,\max}}{c_{b,\min}}))$ and in space $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}} + n)$. The item set J_i for the profit $\phi_l(c_l - F_{n_b}(i))$ can be constructed from the sets $\tilde{J}_{i'}$ by $J_i = \bigcup_{i'=1}^{i} \tilde{J}_{i'}$.*

Putting the Basic FPTAS Together. The algorithm works as follows.

(1) **if** $M > \lfloor \frac{1}{w} \rfloor + 1$ **then** Partition the c_l into intervals $C_b = (c_{\min}^{w\cdot(b+1)}, c_{\min}^{w\cdot b}]$ **else** create $C_b = C_l = \{c_l\}$ for $l \in \{1, \ldots, M\}$
(2) Create the sets S_l for $l \in \{1, \ldots, M\}$.
(3) Calculate the $\bar{P}_{c_l}$ and P_0, discard bin sizes with $\frac{\bar{P}_{c_l}}{c_l} < \frac{1}{2}P_0$.
(4) Let $P_1 := 0$ and $I := \emptyset$.
(5) **For** every C_b **do**
  (5.1) Determine $c_{b,\min}, c_{b,\max}$ and compute T_b and K_b.
  (5.2) Partition the items in every S_l for $c_l \le c_{b,\max}$ into large and small items and scale the large ones.
  (5.3) Calculate the $F_j(i)$. Discard dominated $F_{n_b}(i)$.
  (5.4) **For** every $c_l \in C_b$ **do**
    (5.4.1) Find $\bar{P}_l^{(1)} = \max_{F_{n_b}(i) \le c_l} K_b \cdot i + \phi_l(c_l - F_{n_b}(i))$
    (5.4.2) **if** $\frac{1}{c_l}\bar{P}_l^{(1)} > P_1$ **then** $P_1 := \frac{1}{c_l}\bar{P}_l^{(1)}$
      Determine the item set I_t for solution $\bar{P}_l^{(1)}$ and set $I := I_t$ **end if**
    end do
  end do
(6) **Return** P_1 and I.

The algorithm is an implementation of Algorithm *MaxSolution*. Its correctness follows from Lemma 3, Lemma 4 and Lemma 11. Note that we keep the structure of the S_l to faster get the small items in $\bar{S}_l$ for the calculations in Step (5.4.1).

Except for the space $O(M)$ needed in Steps (2) and (3) and to store all c_l, all operations are dominated in time and space by Sub-steps (5.3) and (5.4.1) over all C_b. Fix one C_b. Calculating and saving the $F_j(i)$ in Step (5.3) needs time and space $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}} n_b)$ according to Theorem 12, and discarding the dominated $F_{n_b}(i)$ is done in time $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}})$ as seen in Lemma 5 because the largest profit $i_{b,\max}$ that has to be considered is in $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}})$. In Step (5.4.1), the $F_{n_b}(i) \le c_l$ have to be found as well as the $\phi_l(c_l - F_{n_b}(i))$ determined, which can be done in $O(|C_b| \cdot (\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}} + n \cdot \log(\frac{1}{\varepsilon}\frac{c_{b,\max}}{c_{b,\min}})))$ (see Thm. 13). The space needed is bounded by $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}} + n)$. Finally, the for-loop (5) is executed $\min\{\lfloor \frac{1}{w} \rfloor + 1, M\}$ times.

The space of Steps (5.4) and (5) is bounded by the space of one iteration because the new values except of P_1 and I can be discarded after one iteration.

Theorem 14. *The basic FPTAS determines a solution of 0-1 KPIP in time* $O(\min\{(\lfloor\frac{1}{w}\rfloor+1)\frac{c_{b_0,\max}}{c_{b_0,\min}}, M\}\cdot\frac{1}{\varepsilon^2}n_L + M\cdot(\frac{1}{\varepsilon^2}\frac{c_{b_0,\max}}{c_{b_0,\min}} + n\cdot\log(\frac{1}{\varepsilon}\frac{c_{b_0,\max}}{c_{b_0,\min}})))$ *and in space* $O(\frac{1}{\varepsilon^2}\frac{c_{b_0,\max}}{c_{b_0,\min}}n_L + M + n)$*, where* $n_L \leq n$ *is an upper bound for the number of the large items and* $\frac{c_{b_0,\max}}{c_{b_0,\min}} := \max_{C_b}\frac{c_{b,\max}}{c_{b,\min}}$*. Note that* $\frac{c_{b_0,\max}}{c_{b_0,\min}} \leq \frac{1}{c_{\min}^w}$ *for* $M > \lfloor\frac{1}{w}\rfloor + 1$ *and* $\frac{c_{b_0,\max}}{c_{b_0,\min}} = 1$ *otherwise.*

3.4 Improved FPTAS: Reducing Running Time and Storage Space

Lawler [10] presented two techniques to decrease the running time and the storage space of the basic FPTAS algorithm.

Let a be a large item with $p(a) \in (2^k T_b, 2^{k+1}T_b]$ and therefore with scaled profit $q_j \in \tilde{I}_{k,b} := (\lfloor\frac{2}{\varepsilon}\rfloor 2^k, \lfloor\frac{4}{\varepsilon}\rfloor 2^k]$. An estimate similar to [10] shows that only a subset of $n_{L,j} \in O(\frac{2^{-k}}{\varepsilon}\frac{c_{b,\max}}{c_{b,\min}})$ items for every scaled profit $q_j \in \tilde{I}_{k,b}$ has to be kept. Moreover, an optimal solution using $n' \leq n_{L,j}$ items of profit q_j obviously uses the n' items of smallest size. Thus, the large items needed can be found by grouping them according to q_j and taking the $n_{L,j}$ items of smallest size by a median-based divide-and-conquer strategy.

Lemma 15. *For one* C_b*, there are at most* $O(\frac{2}{\varepsilon}\cdot\log(\frac{1}{\varepsilon}\frac{c_{b,\max}}{c_{b,\min}}))$ *different scaled profits* q_j*. We need at most* $n_b \leq n_L \in O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}})$ *items of large profit, which can be found in time and space* $O(n + \frac{2}{\varepsilon}\log(\frac{1}{\varepsilon}\frac{c_{b,\max}}{c_{b,\min}}))$*.*

From now on, let $F_j(i)$ represent the smallest size to get profit i with items of profits in $q_1, \ldots, q_j$. Hence, sort the $n_{L,j}$ smallest items of profit q_j in non-increasing order of their size. One $F_j(i)$ is then determined by checking similar to Subsection 3.1 with the $F_{j-1}(i')$ which number of the first $n' \leq n_{L,j}$ smallest items with profit q_j yields the smallest $F_j(i)$.

Lemma 16. *Redefine the* $F_j(i)$ *as above. By modifying the large-item computation, we need space* $O(\frac{1}{\varepsilon^3}\frac{c_{b,\max}}{c_{b,\min}})$ *to find and store the* $F_j(i)$ *of one* C_b*. The running time is in* $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}}n_L) \subseteq O(\frac{1}{\varepsilon^4}(\frac{c_{b,\max}}{c_{b,\min}})^2)$*.*

Lemma 15 and 16 can now be applied to the basic FPTAS for 0-1 KPIP. Let $\frac{c_{b_0,\max}}{c_{b_0,\min}} = \max_{C_b}\frac{c_{b,\max}}{c_{b,\min}}$. The running time of Step (5.3) decreases to $O(\min\{(\lfloor\frac{1}{w}\rfloor+1)(\frac{c_{b_0,\max}}{c_{b_0,\min}})^2, M\}\frac{1}{\varepsilon^4})$ and only needs space in $O(\frac{1}{\varepsilon^3}\frac{c_{b_0,\max}}{c_{b_0,\min}})$. The additional running time and space of Step (5.2) for reducing the large items and sorting them according to their size is dominated by Step (5.3) and the original operations in Step (5.2).

Lemma 17. *The improved FPTAS needs time in* $O(\min\{(\lfloor\frac{1}{w}\rfloor+1)(\frac{c_{b_0,\max}}{c_{b_0,\min}})^2, M\}\cdot\frac{1}{\varepsilon^4} + M\cdot(\frac{1}{\varepsilon^2}\frac{c_{b_0,\max}}{c_{b_0,\min}} + n\cdot\log\frac{1}{\varepsilon}\frac{c_{b_0,\max}}{c_{b_0,\min}}))$ *and space in* $O(\frac{1}{\varepsilon^3}\frac{c_{b_0,\max}}{c_{b_0,\min}} + M + n)$*.*

w has still to be chosen so that the running time is minimized. Note that $\frac{c_{b_0,\max}}{c_{b_0,\min}} \leq \frac{1}{c_{\min}^w}$ and that $c_{\min} \leq c_{\min}^w \leq 1$ so that $w \leq 1$, i.e. $\lfloor\frac{1}{w}\rfloor + 1 \leq \frac{2}{w}$. For simplicity, we minimize the expression $\min\{(\lfloor\frac{1}{w}\rfloor+1)(\frac{c_{b_0,\max}}{c_{b_0,\min}})^2, M\}\frac{1}{\varepsilon^4}$ by minimizing the

dominating expression $\frac{2}{w}\frac{1}{\varepsilon^4 c_{\min}^{2w}}$. A short calculation shows that the minimum is attained at $w = -\frac{1}{2\ln c_{\min}}$. Since $c_{\min}^{2w}|_{w=-\frac{1}{2\ln c_{\min}}} = e^{-\frac{2\ln c_{\min}}{2\ln c_{\min}}} = e^{-1} \in O(1)$, we get the following result:

Theorem 18. *0-1 KPIP can be solved in time* $O(\min\{M, \lfloor 2\log\frac{1}{c_{\min}}\rfloor + 1\}\frac{1}{\varepsilon^4} + M \cdot \left(\frac{1}{\varepsilon^2} + n\log\frac{1}{\varepsilon}\right))$ *and in space* $O\left(\frac{1}{\varepsilon^3} + M + n\right)$.

4 Variants of KPIP

4.1 The Unbounded KPIP

Here, it is allowed to take an arbitrary number of copies of each item a_j.

The computation of P_0 in Step (3) becomes much easier. For P_0, take for every c_l the most efficient item $a_{l,\text{eff}}$ and the most precious item $a_{l,\max}$ in $\bar{S}_l = \cup_{l'=1}^{l} S_{l'}$. Then $\bar{P}_{c_l} = \max\{\lfloor\frac{c_l}{s(a_{l,\text{eff}})}\rfloor \cdot p(a_{l,\text{eff}}), p(a_{l,\max})\}$, and since the items can be found in $O(n+M)$ by a single scan of the S_l, the time to determine P_0 is in $O(n+M)$. A similar result holds for the computation of the $\phi_l(c_l - F_{n_b}(i))$ in Step (5.4.1): as there are $O(\frac{1}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}})$ scaled profits i to be considered, the running time is in $O(\min\{\lfloor\frac{1}{w}\rfloor + 1, M\}n + \frac{M}{\varepsilon^2}\frac{c_{b,\max}}{c_{b,\min}})$ over all iterations of Step (5).

For Step (5.2), note that only one large item $\tilde{a}_j$ of smallest size is necessary for every scaled profit q_j. These large items are taken similar to Lemma 15. The problem is then transformed into a normal 0-1 KPIP problem by creating item copies $\tilde{a}_j^{(r)}$ with profit $2^r p(\tilde{a}_j)$ and size $2^r s(\tilde{a}_j)$ for $r \in \{1, \ldots, \lfloor\log_2(n_{L,j})\rfloor\}$, where $n_{L,j}$ is the maximal possible number of items with scaled profit q_j in a solution. Obviously, these copies are sufficient to represent any choice of large items in a feasible solution of UKPIP. We have the following lemma:

Lemma 19. *Let q_j be one fixed scaled profit. In UKPIP, it is sufficient to have only one item $\tilde{a}_{j_1}^{(r)}$ of smallest size with $q(\tilde{a}_{j_1}^{(r)}) = q_j$.*

The item copies are therefore reduced again. Hence, the running time of Step (5.3) decreases: since only one item for every scaled profit has to be considered, the time to determine one $F_j(i)$ is in $O(1)$ and the time for the large-item computation therefore equal to the space bound in Lemma 16. Step (5.3) therefore has a running time in $O(\min\{(\lfloor\frac{1}{w}\rfloor+1)\frac{c_{b_0,\max}}{c_{b_0,\min}}, M\} \cdot \frac{1}{\varepsilon^3})$ over all iterations, which dominates the running time of the algorithm together with Steps (2) and (5.4.1).

By changing the large-item computation, less space for UKPIP is needed: only the current entries $(F_j(i), j')$ are saved and the old entries $(F_{j-1}(i), j'')$ discarded, where the second value j' is the index of the scaled profit $q_{j'}$ with which the entry $F_j(i)$ was formed. This is enough to find a solution by backtracking with the final values $(F_{n_b}(i), j)$ so that the space bound decreases to $O(\frac{1}{\varepsilon^2}\frac{c_{b_0,\max}}{c_{b_0,\min}})$. (The ideas above are again taken from [10].)

Lemma 20. *UKPIP can be solved in time* $O(n\log M + \min\{\lfloor\frac{1}{w}\rfloor + 1, M\}n + \min\{(\lfloor\frac{1}{w}\rfloor + 1)\frac{c_{b_0,\max}}{c_{b_0,\min}}, M\}\frac{1}{\varepsilon^3} + \frac{M}{\varepsilon^2}\frac{c_{b_0,\max}}{c_{b_0,\min}})$ *and space* $O(M + n + \frac{1}{\varepsilon^2}\frac{c_{b_0,\max}}{c_{b_0,\min}})$.

Similar to Theorem 18, we get $w = -\frac{1}{\ln c_{\min}}$ and Theorem 1.

4.2 The Bounded KPIP

Contrary to UKPIP, only a bounded number $d_j \in \mathbb{N}$ of copies of every item a_j, $j \in \{1, \ldots, n\}$, can be taken. Thus, at most $m_{L,j} := \min\{d_j, n_{L,j}\}$ item copies may be used for any large item with scaled profit q_j. Similar to UKPIP, the Bounded KPIP (BKPIP) is solved with a slightly modified 0-1 KPIP FPTAS.

The median-based divide-and-conquer strategy (which is used to get the $\bar{P}_{c_l}$ and P_0 in Step (3), the subset of large items in Step (5.2) and the small-item values $\phi_l(c_l - F_{n_b}(i))$ in Step (5.4.1)) can be easily adapted to still run in time linear to the number of item sizes that are considered. As proposed in [13, p. 296], item copies of the large items can be taken similar to UKPIP. However, the additional modifications for UKPIP are not possible, i.e. the running time and space bound are identical to the improved 0-1 KPIP algorithm.

Theorem 21. *BKPIP can be solved in the same time and space as 0-1 KPIP, i.e. the bounds of Lemma 17 and Theorem 18 also apply to BKPIP.*

5 A Faster AFPTAS for Variable-Sized Bin Packing

Let $\varepsilon > 0$ and take a Variable-Sized Bin Packing (VBP) instance with n items I and M bin sizes C. As mentioned in the introduction, a part of our AFPTAS [6] for VBP solves UKPIP instances. More precisely, it has to find solutions of value at least $(1-\frac{\bar{\varepsilon}}{6})OPT$ for $\bar{\varepsilon} = \Theta(\varepsilon)$. The instances have $d_1 \in O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ item sizes, a minimal knapsack size $c_{\min} \geq \varepsilon$ and $\overline{M} := \min\left\{M, \lfloor\frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\rfloor + 1\right\} \in O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ different knapsack sizes c_l. (If $M > \lfloor\frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\rfloor + 1$, the AFPTAS partitions the bin sizes $c \geq \varepsilon$ in C into intervals $((1+\varepsilon)^{-(l+1)}, (1+\varepsilon)^{-l}]$ and keeps only the largest size in every interval. This yields the number of bin sizes $\overline{M}$ and only slightly increases the approximation ratio.)

Let $KP(d_1, \bar{\varepsilon}, \overline{M}, c_{\min})$ be the running time for an FPTAS to solve one of these UKPIP instances. Then, the AFPTAS needs time in $O(KP(d_1, \bar{\varepsilon}, \overline{M}, c_{\min})$ $\frac{1}{\varepsilon^3}\log\frac{1}{\varepsilon} + \log(\frac{1}{\varepsilon})(M+n))$. So far, we used Algorithm $MaxSolution$ with the FPTAS by Lawler [10] for UKP s.t. we had $KP(d_1, \bar{\varepsilon}, \overline{M}, c_{\min}) \in O\left(\frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon}\right)$ because of the values of $d_1, \bar{\varepsilon}, \overline{M}$ and $c_{\min}$. Now, our algorithm yields a running time $KP(d_1, \bar{\varepsilon}, \overline{M}, c_{\min}) \in O\left(\frac{1}{\varepsilon^3}\log\frac{1}{\varepsilon}\right)$ (see Thm. 1). Hence, the overall running time of our AFPTAS for VBP decreases by $\Theta(\frac{1}{\varepsilon})$. Moreover, we assumed time in $O(\log(\frac{1}{\varepsilon})M)$ for the reduction of the number of bin sizes to $\lfloor\frac{2}{\varepsilon}\ln\frac{1}{\varepsilon}\rfloor + 1$. This can be done in $O\left(M + \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ with the technique for the large-item reduction of Lemma 15. We get Thm. 2.

6 Concluding Remarks

We have introduced the 0-1 Knapsack Problem with Inversely Proportional Profits (0-1 KPIP) and its variants, the Unbounded KPIP and the Bounded KPIP. The study of KPIP was motivated by our AFPTAS for Variable-Sized Bin Packing [6] where UKPIP instances have to be solved for column generation.

We have adapted Lawler's algorithm [10] to KPIP and improved the running time by using for large M the values $F_j(i)$ determined by dynamic programming for all knapsack sizes in $C_b = (c_{b,\min}^{(b+1)w}, c_{b,\min}^{bw}]$. The running time was minimized by finding a good balance between the number of intervals C_b and the ratio $\frac{c_{b,\max}}{c_{b,\min}} \leq \frac{1}{c_{\min}^{w}}$. This has yielded a faster algorithm for UKPIP and therefore decreased the running time of our AFPTAS for VBP by a factor of $\frac{1}{\varepsilon}$.

Better algorithms for UKPIP may be possible. For instance, it may be sufficient to consider only a subset of the M knapsacks. Moreover, Kellerer and Pferschy's algorithm for 0-1 KP [9, pp. 166–183] may be adapted to UKPIP.

References

1. Bellman, R.E.: Dynamic Programming. Princeton University Press (1957)
2. Garey, M., Johnson, D.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
3. Gilmore, P., Gomory, R.: A linear programming approach to the cutting stock problem. Operations Research 9(6), 849–859 (1961)
4. Gilmore, P., Gomory, R.: A linear programming approach to the cutting stock problem—Part II. Operations Research 11(6), 863–888 (1963)
5. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM 22, 463–468 (1975)
6. Jansen, K., Kraft, S.: An improved approximation scheme for variable-sized bin packing. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 529–541. Springer, Heidelberg (2012)
7. Jansen, K., Kraft, S.: An improved approximation scheme for variable-sized bin packing. Tech. Rep. 1301, Christian-Albrechts-Universität zu Kiel (2013) ISSN 2192-6247
8. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982), November 3-5, pp. 312–320. IEEE Computer Society, Chicago (1982)
9. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)
10. Lawler, E.L.: Fast approximation algorithms for knapsack problems. Mathematics of Operations Research 4(4), 339–356 (1979)
11. Magazine, M.J., Oguz, O.: A fully polynomial approximation algorithm for the 0-1 knapsack problem. European Journal of Operational Research 8(3), 270–273 (1981)
12. Murgolo, F.D.: An efficient approximation scheme for variable-sized bin packing. SIAM Journal on Computing 16(1), 149–161 (1987)
13. Plotkin, S.A., Shmoys, D.B., Tardos, E.: Fast approximation algorithms for fractional packing and covering problems. Mathematics of Operations Research 20(2), 257–301 (1995)
14. Sahni, S.: Approximate algorithms for the 0/1 knapsack problem. Journal of the ACM 22, 115–124 (1975)
15. Shachnai, H., Yehezkely, O.: Fast asymptotic FPTAS for packing fragmentable items with costs. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 482–493. Springer, Heidelberg (2007)

QuickHeapsort: Modifications and Improved Analysis

Volker Diekert and Armin Weiß

FMI, Universität Stuttgart, Universitätsstr. 38, D-70569 Stuttgart, Germany
{diekert,weiss}@fmi.uni-stuttgart.de

Abstract. We present a new analysis for QuickHeapsort. This enables us to consider samples of non-constant size for the pivot selection and leads to better theoretical bounds for the algorithm. Furthermore, we introduce some modifications of QuickHeapsort, both in-place and using n extra bits. We show that on every input the expected number of comparisons is $n \lg n - 0.03n + o(n)$ (in-place) respectively $n \lg n - 0.997n + o(n)$ (always $\lg n = \log_2 n$). Both estimates improve the previously known best results. (It is conjectured [17] that the in-place algorithm Bottom-Up-Heapsort uses at most $n \lg n + 0.4n$ on average and for Weak-Heapsort which uses n extra bits the average number of comparisons is at most $n \lg n - 0.42n$ [8].) Moreover, our non-in-place variant can even compete with index based Heapsort variants (e.g. Rank-Heapsort [15]) and Relaxed-Weak-Heapsort ($n \lg n - 0.9n + o(n)$ comparisons in the worst case) for which no $\mathcal{O}(n)$-bound on the number of extra bits is known.

Keywords: in-place sorting, heapsort, quicksort, analysis of algorithms.

1 Introduction

QuickHeapsort is a combination of Quicksort and Heapsort which was first described by Cantone and Cincotti [2]. It is based on Katajainen's idea for Ultimate Heapsort [11]. Its advantage is that it is very fast in the average case and hence not only of theoretical interest. In fact, it uses always less comparisons than Quicksort, but we did not aim to beat Quicksort by running time. In our implementations Quicksort is only slightly faster when the sample size for pivot selection is close to $\sqrt{n}$.

Both, QuickHeapsort and Ultimate Heapsort have in common that first the array is partitioned into two parts. Then, in one part a heap is constructed and the elements are successively extracted. Finally, the remaining elements are treated recursively. As a consequence the sift-down needs one comparison per level, whereas standard Heapsort needs two comparisons per level, which is one of the reasons why standard Heapsort cannot compete with Quicksort in practice (of course there are also other reasons like cache behavior). The difference between QuickHeapsort and Ultimate Heapsort lies in the choice of the pivot element for partitioning the array. While for Ultimate Heapsort the pivot is chosen as median of the whole array, for QuickHeapsort the pivot is selected as median of some smaller sample (e.g. as median of 3 elements).

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 24–35, 2013.

In [2] the basic version with fixed index as pivot is analyzed and – together with the median of three version – implemented and compared with other Quick- and Heapsort variants. In [8] Edelkamp and Stiegeler compare these variants with so called Weak-Heapsort [7] and some modifications of it (e.g. Relaxed-Weak-Heapsort). Weak-Heapsort beats basic QuickHeapsort with respect to the number of comparisons, however it needs $\mathcal{O}(n)$ bits extra-space, hence is not in place.

We split the analysis of QuickHeapsort into three parts: the partitioning phases, the heap construction and the heap extraction. This allows us to get better bounds for the running time, especially when choosing the pivot as median of a larger sample. It also simplifies the analysis. We introduce some modifications of QuickHeapsort, too. The first one is in-place and needs $n \lg n - 0.03n + o(n)$ comparisons on average what is to the best of our knowledge better than any other known in-place Heapsort variant. We also examine a modification using $\mathcal{O}(n)$ bits extra-space, which applies the ideas of MDR-Heapsort to QuickHeapsort. With this method we can bound the average number of comparisons to $n \lg n - 0.997n + o(n)$. Actually, a complicated, iterated in-place MergeInsertion uses only $n \lg n - 1.3n + \mathcal{O}(\lg n)$ comparisons, [14]. Unfortunately, for practical purposes this algorithm is not competitive.

Our contributions are as follows: 1. We give a simplified analysis which gives better bounds than previously known. 2. Our approach yields the first precise analysis of QuickHeapsort when the pivot element is taken from a larger sample. 3. We give a simple in-place modification of QuickHeapsort which saves $0.75n$ comparisons. 4. We give a modification of QuickHeapsort using n extra bits only and a bound of the expected number of comparisons. Our bound is better than all bounds previously known for the worst case of Heapsort variants using $\mathcal{O}(n \lg n)$ extra bits (for these algorithms, average and worst case are almost the same). 5. We have implemented QuickHeapsort, and our experiments confirm the theoretical predictions.

Due to lack of space most proofs are omitted. They can be found on ArXiv [6].

2 QuickHeapsort

A *two-layer-min-heap* is an array $A[1..n]$ of n elements together with a partition (G, R) of $\{1, \ldots, n\}$ into *green* and *red* elements such that for all $g \in G, r \in R$ we have $A[g] \leq A[r]$. Furthermore, the green elements g satisfy the heap condition $A[g] \leq \min\{A[2g], A[2g+1]\}$, and if g is red, then $2g$ and $2g+1$ are red, too. (The conditions are required to hold, only if the indices involved are in the range of 1 to n.) The green elements are called "green" because the they can be extracted out of the heap without caution, whereas the "red" elements are blocked. *Two-layer-max-heaps* are defined analogously. We can think of a two-layer-heap as rooted binary tree such that each node is either green or red. Green nodes satisfy the standard heap-condition, children of red nodes are red. Two-layer-heaps were defined in [11]. In [2] for the same concept a different language is used (they describe the algorithm in terms of External Heapsort). Now we are ready to describe the QuickHeapsort algorithm as it has been proposed in [2].

We intend to sort an array $A[1..n]$. First, we choose a *pivot* p. This is the randomized part of the algorithm. Then, just as in Quicksort, we rearrange the array according to p. That means, using $n-1$ comparisons the partitioning function returns an index k and rearranges the array A so that $A[i] \geq A[k]$ for $i < k$, $A[k] = p$, and $A[k] \geq A[j]$ for $k < j$. After the partitioning a two-layer-heap is built out of the elements of the smaller part of the array, either the part left of the pivot or right of the pivot. We call this smaller part *heap-area* and the larger part *work-area*. More precisely, if $k-1 < n-k$, then $\{1,\ldots,k-1\}$ is the heap-area and $\{k+1,\ldots,n\}$ is the work-area. If $k-1 \geq n-k$, then $\{1,\ldots,k-1\}$ is the work-area and $\{k+1,\ldots,n\}$ is the heap-area. Note that we know the final position of the pivot element without any further comparison. Therefore, we do not count it to the heap-area nor to the work-area. If the heap-area the part of the array left of the pivot, a two-layer-max-heap is built, otherwise a two-layer-min-heap is built.

At the beginning the heap-area is an ordinary heap, hence it is a two-layer-heap consisting of green elements, only. Now the heap extraction phase starts. We assume that we are in the case of a max-heap. The other case is symmetric. Let m denote the size of the heap-area. The m elements of the heap-area are moved to the work-area. The extraction of one element works as follows: the root of the heap is placed at the current position of the work-area (which at the beginning is its last position). Then, starting from the root the resulting "hole" is trickled down: always the larger child is moved up into the vacant position and then this child is treated recursively. This stops as soon as a leaf is reached. We call this the SpecialLeaf procedure according to [2]. Now, the element which before was at the current position in the work-area is placed as red element in this hole at the leaf in the heap-area. Finally the current position in the work-area is moved by one and the next element can be extracted.

The procedure sorts correctly, because after the partitioning it is guaranteed that all red elements are smaller than all green elements. Furthermore there is enough space in the work-area to place all green elements of the heap, since the heap is always the smaller part of the array. After extracting all green elements the pivot element it placed at its final position and the remaining elements are sorted recursively.

Actually we can improve the procedure, thereby saving $3n/4$ comparisons by a simple trick. Before the heap extraction phase starts in the heap-area with m elements, we perform at most $\frac{m+2}{4}$ additional comparisons in order to arrange all pairs of leaves which share a parent such that the left child is not smaller than its right sibling. Now, in every call of SpecialLeaf, we can save exactly one comparison, since we do not need to compare two leaves. For a max-heap we only need to move up the left child and put the right one at the place of the former left one. Summing up over all heaps during an execution of standard QuickHeapsort, we invest $\frac{n+2t}{4}$ comparisons in order to save n comparisons, where t is the number of recursive calls. The expected number of t is in $\mathcal{O}(\lg n)$. Hence, we can expect to save $\frac{3n}{4} + \mathcal{O}(\lg n)$ comparisons. We call this version the *improved* variant of QuickHeapsort.

3 Analysis of QuickHeapsort

This section contains the main contribution of the paper. We analyze the number of comparisons. By n we denote the number of elements of an array to be sorted. We use standard $\mathcal{O}$-notation where $\mathcal{O}(g)$, $o(g)$, and $\omega(g)$ denote classes of functions. In our analysis we do not assume any random distribution of the input, i.e. it is valid for every permutation of the input array. Randomization is used however for pivot selection. With $\Pr[\,e\,]$ we denote the probability of some event e. The expected value of a random variable T is denoted by $\mathbb{E}[\,T\,]$.

The number of assignments is bounded by some small constant times the number of comparisons. Let $T(n)$ denote the number of comparisons during QuickHeapsort on a fixed array of n elements. We are going to split the analysis of QuickHeapsort into three parts:

1. Partitioning with an expected number of comparisons $\mathbb{E}[\,T_{\text{part}}(n)\,]$ (average case).
2. Heap construction with at most $T_{\text{con}}(n)$ comparisons (worst case).
3. Heap extraction (sorting phase) with at most $T_{\text{ext}}(n)$ comparisons (worst case).

We analyze the three parts separately and put them together at the end. The partitioning is the only randomized part of our algorithm. The expected number of comparisons depends on the selection method for the pivot. For the expected number of comparisons by QuickHeapsort on the input array we obtain $\mathbb{E}[\,T(n)\,] \le T_{\text{con}}(n) + T_{\text{ext}}(n) + \mathbb{E}[T_{\text{part}}(n)]$.

Theorem 1. *The expected number $\mathbb{E}[\,T(n)\,]$ of comparisons by basic resp. improved QuickHeapsort with pivot as median of p randomly selected elements on a fixed input array of size n is $\mathbb{E}[\,T(n)\,] \le n \lg n + cn + o(n)$ with c as follows:*

p	c *basic*	c *improved*
1	+2.72	+1.97
3	+1.92	+1.17
$f(n)$	+0.72	−0.03

Here, $f \in \omega(1) \cap o(n)$ with $1 \le f(n) \le n$, e.g., $f(n) = \sqrt{n}$ and we assume that we choose the median of $f(n)$ randomly selected elements in time $\mathcal{O}(f(n))$.

As we see, the selection method for the pivot is very important. However, one should notice that the bound for fixed size samples for pivot selection are not tight. The proof of these results are postponed to Sect. 3.3. Note that it is enough to prove the results without the improvement, since the difference is always $0.75n$.

3.1 Heap Construction

The standard heap construction [9] needs at most $2m$ comparisons to construct a heap of size m in the worst case and approximately $1.88m$ in the average case.

For the mathematical analysis better theoretical bounds can be used. The best result we are aware of is due to Chen et al. in [5]. According to this result we have $T_{\mathrm{con}}(m) \leq 1.625m + o(m)$. Earlier results are of similar magnitude, by [4] it has been known that $T_{\mathrm{con}}(m) \leq 1.632m + o(m)$ and by [10] it has been known $T_{\mathrm{con}}(m) \leq 1.625m + o(m)$, but Gonnet and Munro used $\mathcal{O}(m)$ extra bits to get this result, whereas the new result of Chen et al. is in-place (by using only $\mathcal{O}(\lg m)$ extra bits).

During the execution of QuickHeapsort over n elements, every element is part of a heap only once. Hence, the sizes of all heaps during the entire procedure sum up to n. With the result of [5] the total number of comparisons performed in the construction of all heaps satisfies:

Proposition 1. $T_{\mathrm{con}}(n) \leq 1.625n + o(n)$.

3.2 Heap Extraction

For a real number $r \in \mathbb{R}$ with $r > 0$ we define $\{r\}$ by the following condition

$$r = 2^k + \{r\} \text{ with } k \in \mathbb{Z} \text{ and } 0 \leq \{r\} < 2^k \ .$$

This means that 2^k is largest power of 2 which is less than or equal to r and $\{r\}$ is the difference to that power, i.e. $\{r\} = r - 2^{\lfloor \lg r \rfloor}$. In this section we first analyze the extraction phase of one two-layer-heap of size m. After that, we bound the number of comparisons $T_{\mathrm{ext}}(n)$ performed in the worst case during all heap extraction phases of one execution of QuickHeapsort on an array of size n. Thm. 2 is our central result about heap extraction.

Theorem 2. $T_{\mathrm{ext}}(n) \leq n \cdot (\lfloor \lg n \rfloor - 3) + 2\{n\} + \mathcal{O}(\lg^2 n)$.

The proof of Thm. 2 covers almost the rest of Section 3.2. In the following, the *height* height(v) of an element v in a heap H is the maximal distance from that node to a leaf below it. The *height* of H is the height of its root. The *level* level(v) of v is its distance from the root. In this section we want to count the comparisons during SpecialLeaf procedures, only. Recall that a SpecialLeaf procedure is a cyclic shift on a path from the root down to some leaf, and the number comparisons is exactly the length of this path. Hence, an upper bound for one SpecialLeaf execution is the height of the heap. But there is a better analysis.

Let us consider a heap with m green elements which are all extracted by SpecialLeaf procedures. The picture is as follows: First, we color the green root red. Next, we perform a cyclic shift defined by the SpecialLeaf procedure. In particular, the leaf is now red. Moreover, red positions remain red, but there is exactly one position v which has changed its color from green to red. This position v is on the path defined by the SpecialLeaf procedure. Hence, the number of comparisons needed to color the position v red is bounded by height(v)+level(v).

The total number of comparisons $E(m)$ to extract all m elements of a Heap H is therefore bounded by $E(m) \leq \sum_{v \in H}(\text{height}(v) + \text{level}(v))$.

We have $\text{height}(H) - 1 \leq \text{height}(v) + \text{level}(v) \leq \text{height}(H) = \lfloor \lg m \rfloor$ for all $v \in H$. We now count the number of elements v where $\text{height}(v) + \text{level}(v) = \lfloor \lg m \rfloor$ and the number of elements v where $\text{height}(v) + \text{level}(v) = \lfloor \lg m \rfloor - 1$. Since there are exactly $\{m\} + 1$ nodes of level $\lfloor \lg m \rfloor$, there are at most $2\{m\} + 1 + \lg m$ elements v with $\text{height}(v) + \text{level}(v) = \lfloor \lg m \rfloor$. All other elements satisfy $\text{height}(v) + \text{level}(v) = \lfloor \lg m \rfloor - 1$. We obtain

$$\begin{aligned} E(m) &\leq 2 \cdot \{m\} \cdot \lfloor \lg m \rfloor + (m - 2 \cdot \{m\})(\lfloor \lg m \rfloor - 1) + \mathcal{O}(\lg m) \\ &= m \cdot (\lfloor \lg m \rfloor - 1) + 2 \cdot \{m\} + \mathcal{O}(\lg m) \ . \end{aligned} \tag{1}$$

Note that this is an estimate of the worst case, however this analysis also shows that the best case only differs by $\mathcal{O}(\lg m)$-terms from the worst case.

Now, we want to estimate the number of comparisons in the worst case performed during all heap extraction phases together. During QuickHeapsort over n elements we create a sequence $H_1, \ldots, H_t$ of heaps of green elements which are extracted using the SpecialLeaf procedure. Let $m_i = |H_i|$ be the size of the i-th Heap. The sequence satisfies $2m_i \leq n - \sum_{j<i} m_j$, because heaps are constructed and extracted on the smaller part of the array.

Here comes a subtle observation: Assume that $m_1 + m_2 \leq n/2$. If we replace the first two heaps with one heap H' of size $|H|' = m_1 + m_2$, then the analysis using the sequence $H', H_3, \ldots, H_t$ cannot lead to a better bound. Continuing this way, we may assume that we have $t \in \mathcal{O}(\lg n)$ and therefore $\sum_{1 \leq i \leq t} \mathcal{O}(\lg m_i) \subseteq \mathcal{O}(\lg^2 n)$. With (1) we obtain the bound

$$T_{\text{ext}}(n) \leq \sum_{i=1}^{t} E(m_i) = \left(\sum_{i=1}^{t} m_i \cdot \lfloor \lg m_i \rfloor + 2\{m_i\} \right) - n + \mathcal{O}(\lg^2 n) \ . \tag{2}$$

We will replace the m_i by other positive real numbers. Let $1 \leq \nu \in \mathbb{R}$. We say a sequence $x_1, x_2, \ldots, x_t$ with $x_i \in \mathbb{R}^{>0}$ is *valid* w.r.t. ν, if for all $1 \leq i \leq t$ we have $2x_i \leq \nu - \sum_{j<i} x_j$.

As just mentioned the initial sequence $m_1, m_2 \ldots, m_t$ is valid w.r.t. n. Let us define a continuous function $F : \mathbb{R}^{>0} \to \mathbb{R}$ by $F(x) = x \cdot \lfloor \lg x \rfloor + 2\{x\}$. It is piecewise differentiable with right derivative $\lfloor \lg x \rfloor + 2$. Furthermore, for $x \geq y > \delta \geq 0$ we have the inequalities:

$$F(x) + F(y) \leq F(x + \delta) + F(y - \delta) \text{ and } F(x) + F(y) \leq F(x + y).$$

Applying them we can prove the following (for details see [6]):

1. Let $1 \leq \nu \in \mathbb{R}$. For all sequences $x_1, x_2, \ldots, x_t$ with $x_i \in \mathbb{R}^{>0}$, which are valid w.r.t. ν, we have $\sum_{i=1}^{t} F(x_i) \leq \sum_{i=1}^{\lfloor \lg \nu \rfloor} F\left(\frac{\nu}{2^i}\right)$.
2. $\sum_{i=1}^{\lfloor \lg n \rfloor} F\left(\frac{n}{2^i}\right) \leq F(n) - 2n + \mathcal{O}(\lg n)$.

Combining these facts with (2) yields the proof of Thm. 2. Furthermore, we obtain the following corollary of Thm. 2 by using [16, Thm. 1].

Corollary 1. *We have* $T_{\text{ext}}(n) \leq n \lg n - 2.9139n + \mathcal{O}(\lg^2 n)$.

3.3 Partitioning

In the following $T_{\text{pivot}}(n)$ denotes the number of comparisons required to choose the pivot element in the worst case; and, as before, $\mathbb{E}[\, T_{\text{part}}(n)\,]$ denotes the expected number of comparisons performed during partitioning. We have the following recurrence:

$$\mathbb{E}[\, T_{\text{part}}(n)\,] \leq n - 1 + T_{\text{pivot}}(n) + \sum_{k=1}^{n} \Pr\left[\text{pivot } = k\right] \cdot \mathbb{E}[T_{\text{part}}(\max\{k-1, n-k\})] \ .$$

If we choose the pivot at random, we obtain by standard methods:

$$\mathbb{E}[\, T_{\text{part}}(n)\,] \leq n - 1 + \frac{1}{n} \cdot \sum_{k=1}^{n} \mathbb{E}[\, T_{\text{part}}(\max\{k-1, n-k\})\,] \leq 4n \ .$$

Similarly, if we choose the pivot with the median of three method, then we obtain:

$$\mathbb{E}[\, T_{\text{part}}(n)\,] \leq 3.2n + \mathcal{O}(\lg n) \ .$$

The proof of the first part of Thm. 1 follows from the above eqations, Thm. 2, and 1. Using a growing number of elements (as n grows) as sample for the pivot selection, we can do better. The second part of Thm. 1 follows from Thm. 2, 1, and Thm. 3.

Theorem 3. *Let $f \in \omega(1) \cap o(n)$ with $1 \leq f(n) \leq n$. When choosing the pivot as median of $f(n)$ randomly selected elements in time $\mathcal{O}(f(n))$ (e.g. with the algorithm of [1]), the expected number of comparisons used in all recursive calls of partitioning is in $2n + o(n)$.*

Thm. 3 is close to a well-known result in [12, Thm. 5] on Quickselect. Actually, in our approach it becomes a corollary.

Corollary 2 ([12]). *Let $f \in \omega(1) \cap o(n)$ with $1 \leq f(n) \leq n$. When implementing Quickselect with the median of $f(n)$ randomly selected elements as pivot, the expected number of comparisons is $2n + o(n)$.*

The following lemma is the key step in the proof of Thm. 3.

Lemma 1. *Let $0 < \delta < \frac{1}{2}$. If we choose the pivot as median of $2c+1$ elements such that $2c + 1 \leq \frac{n}{2}$, then we have $\Pr\left[\, \text{pivot } \leq \frac{n}{2} - \delta n\,\right] < (2c+1)\alpha^c$ where $\alpha = 4\left(\frac{1}{4} - \delta^2\right) < 1$.*

In [12] it is also proved that choosing the pivot as median of $\mathcal{O}(\sqrt{n})$ elements is optimal for Quicksort as well as for Quickselect. This suggests that we choose the same value in QuickHeapsort; what is backed by our experiments.

4 Modifications of QuickHeapsort Using Extra-Space

In this section we want to describe some modification of QuickHeapsort using n bits of extra storage. We introduce two bit-arrays. In one of them (the CompareArray) – which is actually two bits per element – we store the comparisons already done (we need two bits, because there are three possible values – right, left, unknown – we have to store). In the other one (the RedGreenArray) we store which element is red and which is green.

Since the heaps have maximum size $n/2$, the RedGreenArray only requires $n/2$ bits. The CompareArray is only needed for the inner nodes of the heaps, i.e. length $n/4$ is sufficient. Totally this sums up to n extra bits.

For the heap construction we do not use the algorithms described in Sect. 3.1. With the CompareArray we can do better by using the algorithm of McDiarmid and Reed [13]. The heap construction works similarly to Bottom-Up-Heapsort, i.e. the array is traversed backward calling for all inner positions i the Reheap procedure on i. The Reheap procedure takes the subheap with root i and restores the heap condition, if it is violated at the position i. First, the Reheap procedure determines a *special leaf* using the SpecialLeaf procedure as described in Sect. 2, but without moving the elements. Then, the final position of the former root is determined going upward from the special leaf (bottom-up-phase). In the end, the elements above this final position are moved up towards the root by one position. That means that all but one element which are compared during the bottom-up-phase, stay in their places. Since in the SpecialLeaf procedure these elements have been compared with their siblings, these comparisons can be stored in the CompareArray and can be used later.

With another improvement concerning the construction of heaps with seven elements as in [3] the benefits of this array can be exploited even more.

The RedGreenArray is used during the sorting phase, only. Its functionality is straightforward: Every time a red element is inserted into the heap, the corresponding bit is set to red. The SpecialLeaf procedure can stop as soon as it reaches an element without green children. Whenever a red and a green element have to be compared, the comparison can be skipped.

Theorem 4. *Let $f \in \omega(1) \cap o(n)$ with $1 \le f(n) \le n$, e.g., $f(n) = \sqrt{n}$, and let $\mathbb{E}[\,T(n)\,]$ be the expected number of comparisons by QuickHeapsort using the CompareArray with the improvement of [3] and the RedGreenArray on a fixed input array of size n. Choosing the pivot as median of $f(n)$ randomly selected elements in time $\mathcal{O}(f(n))$, we have*

$$\mathbb{E}[\,T(n)\,] \le n \lg n - 0.997n + o(n) \ .$$

Proof. We can analyze the savings by the two arrays separately, because the CompareArray only affects comparisons between two green elements, while the RedGreenArray only affects comparisons involving at least one red element.

First, we consider the heap construction using the CompareArray. With this array we obtain the same worst case bound as for the standard heap construction method. However, the CompareArray has the advantage that at the end of the

heap construction many comparisons may be stored in the array and can be reused for the extraction phase. More precisely: For every comparison except the first one made when going upward from the special leaf, one comparison is stored in the CompareArray. This is because for every additional comparison one element on the path defined by SpecialLeaf stays at its place. Since every pair of siblings has to be compared at one point during the heap construction or extraction, all these stored comparisons can be reused. Hence, we only have to count the comparisons in the SpecialLeaf procedure during the construction plus $\frac{n}{2}$ for the first comparison when going upward. Thus, we get an amortized bound for the comparisons during heap construction of $\frac{3n}{2}$.

In [3] the notion of *Fine-Heaps* is introduced. A Fine Heap is a heap with the additional CompareArray such that for every node the larger child is stored in the array. Such a Fine-Heap of size m can be constructed using the above method with $2m$ comparisons. In [3] Carlsson, Chen and Mattsson showed that a Fine-Heap of size m actually can be constructed with only $\frac{23}{12}m + \mathcal{O}(\lg^2 m)$ comparisons. That means we have to invest $\frac{23}{12}m + \mathcal{O}(\lg^2 m)$ for the heap construction and at the end there are $\frac{m}{2}$ comparisons stored in the array. All these comparisons stored in the array are used later. Summing up over all heaps during an execution of Quick-Heapsort, we can save another $\frac{1}{12}n$ comparisons additionally to the comparisons saved by the CompareArray with the result of [3]. Hence, for the amortized cost of the heap construction $T_{\text{con}}^{\text{amort}}$ (i.e. the number of comparisons needed to build the heap minus the number of comparisons stored in the CompareArray after the construction which all can be reused later) we have obtained:

Proposition 2. $T_{\text{con}}^{\text{amort}}(n) \leq \frac{17}{12}n + o(n)$.

This bound is slightly better than the average case for the heap construction with the algorithm of [13] which is $1.52n$.

Now, we want to count the number of comparisons we save using the RedGreenArray. We distinguish the two cases that two red elements are compared and that a red and a green element are compared. Every position in the heap has to turn red at one point. At that time, all nodes below this position are already red. Hence, for that position we save as many comparisons as it is above the bottom level. Summing over all levels of a heap of size m the saving results in $\approx \frac{m}{4} \cdot 1 + \frac{m}{8} \cdot 2 + \cdots = m \cdot \sum_{i \geq 1} i2^{-i-1} = m$. This estimate is exact up to $\mathcal{O}(\lg m)$-terms. Since the expected number of heaps is $\mathcal{O}(\lg n)$, we obtain for the overall saving the value $T_{\text{saveRR}}(n) = n + \mathcal{O}(\lg^2 n)$.

Another place where we save comparisons with the RedGreenArray is when a red element is compared with a green element. It occurs at least one time – when the node looses its last green child – for every inner node that we compare a red child with a green child. Hence, we save at least as many comparisons as there are inner nodes with two children, i.e. at least $\frac{m}{2} - 1$. Since every element – except the expected $\mathcal{O}(\lg n)$ pivot elements – is part of a heap exactly once, we save at least $T_{\text{saveRG}}(n) \geq \frac{n}{2} + \mathcal{O}(\lg n)$ comparisons when comparing green with red elements. In the average case the saving might be even slightly higher, since comparisons can also be saved when a node does not loose its last green child.

Summing up all our savings and using the median of $f(n) \in \omega(1) \cap o(n)$ as pivot we obtain the proof of Thm. 4:

$$\begin{aligned}\mathbb{E}[\,T(n)\,] &\le T_{\text{con}}^{\text{amort}}(n) + T_{\text{ext}}(n) + \mathbb{E}[T_{\text{part}}(n)] - T_{\text{saveRR}}(n) - T_{\text{saveRG}}(n) \\ &\le \frac{17}{12}n + n \cdot (\lfloor \lg n \rfloor - 3) + 2\,\{n\} + 2n - \frac{3n}{2} + o(n) \\ &\le n \lg n - 0.997n + o(n)\ .\end{aligned}$$

5 Experimental Results and Conclusion

In Fig. 1 we present the number of comparisons of the different versions of QuickHeapsort we considered in this paper, i.e. the basic version, the improved variant of Sect. 2, and the version using bit-arrays (however, without the modification by [3]) for different values of n. We compare them with Quicksort, Ultimate Heapsort and MDR-Heapsort. All algorithms are implemented with median of $\sqrt{n}$ elements as pivot (for Quicksort we show additionally the data with median of 3). For the heap construction we implemented the normal algorithm due to Floyd [9] as well as the algorithm using the extra bit-array (which is the same as in MDR-Heapsort).

As our theoretical estimates predict, QuickHeapsort with bit-arrays beats all other variants when implemented with median of $\sqrt{n}$ for pivot selection. In Table 1 on page 33 we present actual running times of the different algorithms for $n = 1000000$ with two different comparison functions. One of them is the normal integer comparison, the other one first applies four times the logarithm to both operands before comparing them. Like in [8], this simulates expensive comparisons.

Table 1. Running times for QuickHeapsort and other algorithms tested on 10^6, average over 10 runs elements

Sorting algorithm	integer data time $[s]$	$\log^{(4)}$-test-function time $[s]$
Basic QuickHeapsort, median of 3	0.1154	4.21
Basic QuickHeapsort, median of $\sqrt{n}$	0.1171	4.109
Improved QHS, median of 3	0.1073	4.049
Improved QHS, median of $\sqrt{n}$	0.1118	3.911
QHS with bit-arrays, median of 3	0.1581	3.756
QHS with bit-arrays, median of $\sqrt{n}$	0.164	3.7
Quicksort with median of 3	0.1181	3.946
Quicksort with median of $\sqrt{n}$	0.1316	3.648
Ultimate Heapsort	0.135	5.109
MDR-Heapsort	0.2596	4.129

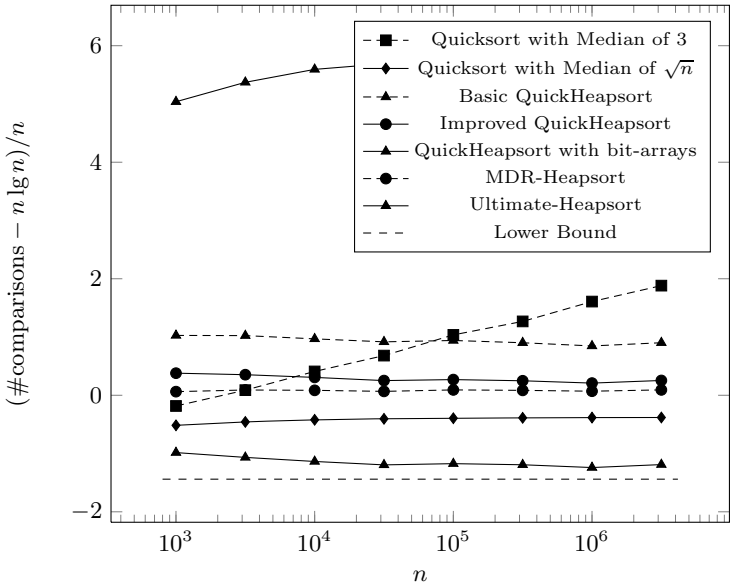


Fig. 1. Average number (over 100 runs) of comparisons of QuickHeapsort implemented with median of $\sqrt{n}$ compared with other algorithms

More experimental results with other pivot selection strategies can be found on ArXiv [6]. They also confirm that a sample size of $\sqrt{n}$ is optimal for pivot selection with respect to the number of comparisons and also that the $o(n)$-terms in Thm. 1 and Thm. 3 are not too big.

In this paper we have shown that with known techniques QuickHeapsort can be implemented with expected number of comparisons less than $n \lg n - 0.03n + o(n)$ and extra storage $O(1)$. On the other hand, using n extra bits we can improve this to $n \lg n - 0.997n + o(n)$, i.e. we showed that QuickHeapsort can compete with the most advanced Heapsort variants. These theoretical estimates were also confirmed by our experiments. We also considered different pivot selection schemes. For any constant size sample for pivot selection, QuickHeapsort beats Quicksort for large n, since Quicksort has a expected running time of $\approx Cn \lg n$ with $C > 1$. However, when choosing the pivot as median of $\sqrt{n}$ elements (i.e. with the optimal strategy) then our experiments show that Quicksort needs less comparisons than QuickHeapsort. However, using bit-arrays QuickHeapsort is the winner, again. In order to make the last statement rigorous, better theoretical bounds for Quicksort with sampling $\sqrt{n}$ elements are needed. For future work it would also be of interest to prove the optimality of $\sqrt{n}$ elements for pivot selection in QuickHeapsort, to estimate the lower order terms of the average running time of QuickHeapsort and also to find an exact average case analysis for the saving by the bit-arrays.

Acknowledgements. We thank Martin Dietzfelbinger, Stefan Edelkamp, Jyrki Katajainen and the anonimous referees for their helpful comments. We thank Simon Paridon for implementing the algorithms for our experiments.

References

1. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. Comput. Syst. Sci. 7(4), 448–461 (1973)
2. Cantone, D., Cincotti, G.: Quickheapsort, an efficient mix of classical sorting algorithms. Theor. Comput. Sci. 285(1), 25–42 (2002)
3. Carlsson, S., Chen, J., Mattsson, C.: Heaps with Bits. In: Du, D.-Z., Zhang, X.-S. (eds.) ISAAC 1994. LNCS, vol. 834, pp. 288–296. Springer, Heidelberg (1994)
4. Chen, J.: A Framework for Constructing Heap-like structures in-place. In: Ng, K.W., Balasubramanian, N.V., Raghavan, P., Chin, F.Y.L. (eds.) ISAAC 1993. LNCS, vol. 762, pp. 118–127. Springer, Heidelberg (1993)
5. Chen, J., Edelkamp, S., Elmasry, A., Katajainen, J.: In-place Heap Construction with Optimized Comparisons, Moves, and Cache Misses. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 259–270. Springer, Heidelberg (2012)
6. Diekert, V., Weiss, A.: Quickheapsort: Modifications and improved analysis. ArXiv e-prints, abs/1209.4214 (2012)
7. Dutton, R.D.: Weak-heap sort. BIT 33(3), 372–381 (1993)
8. Edelkamp, S., Stiegeler, P.: Implementing HEAPSORT with $n \log n - 0.9n$ and QUICKSORT with $n \log n + 0.2n$ comparisons. ACM Journal of Experimental Algorithmics 7, 5 (2002)
9. Floyd, R.W.: Algorithm 245: Treesort. Commun. ACM 7(12), 701 (1964)
10. Gonnet, G.H., Munro, J.I.: Heaps on Heaps. SIAM J. Comput. 15(4), 964–971 (1986)
11. Katajainen, J.: The Ultimate Heapsort. In: Lin, X. (ed.) CATS. Australian Computer Science Communications, vol. 20, pp. 87–96. Springer-Verlag Singapore Pte. Ltd. (1998)
12. Martínez, C., Roura, S.: Optimal Sampling Strategies in Quicksort and Quickselect. SIAM J. Comput. 31(3), 683–705 (2001)
13. McDiarmid, C., Reed, B.A.: Building Heaps Fast. J. Algorithms 10(3), 352–365 (1989)
14. Reinhardt, K.: Sorting *in-place* with a *worst case* complexity of $n \log n - 1.3n + o(\log n)$ comparisons and $\epsilon n \log n + o(1)$ transports. In: Ibaraki, T., Inagaki, Y., Iwama, K., Nishizeki, T., Yamashita, M. (eds.) ISAAC 1992. LNCS, vol. 650, pp. 489–498. Springer, Heidelberg (1992)
15. Wang, X.-D., Wu, Y.-J.: An Improved HEAPSORT Algorithm with $n \log n - 0.788928n$ Comparisons in the Worst Case. Journal of Computer Science and Technology 22, 898–903 (2007), doi:10.1007/s11390-007-9106-7
16. Wegener, I.: The Worst Case Complexity of McDiarmid and Reed's Variant of Bottom-Up-Heap Sort is Less Than $n \log n + 1.1n$. In: Jantzen, M., Choffrut, C. (eds.) STACS 1991. LNCS, vol. 480, pp. 137–147. Springer, Heidelberg (1991)
17. Wegener, I.: BOTTOM-UP-HEAPSORT, a new variant of HEAPSORT, beating, on an average, QUICKSORT (if n is not very small). Theoretical Computer Science 118(1), 81–98 (1993)

Alphabetic Minimax Trees in Linear Time

Paweł Gawrychowski[1,2,⋆]

[1] Institute of Computer Science, University of Wrocław, Poland
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
gawry@cs.uni.wroc.pl

Abstract. We develop a linear time algorithm for the following problem: given an ordered sequence of n real weights, construct a binary tree on n leaves labelled with those weights when read from left to right minimizing the maximum value of w_i plus the depth of the corresponding leaf. This improves the previously known $O(n \log n)$ time solutions [3,10,12]. Assuming that the integer and the fractional part of each weight is given separately, our solution works in the linear decision tree model, i.e., we use only the basic arithmetical operations on the input numbers. To decide (efficiently) which operations to perform we need the word RAM model, though. We provide a simplified $\mathcal{O}(nd)$ version of the algorithm, where d is the number of distinct integer parts, which does not require the full power of the word RAM model in order to decide which operations to perform. Nevertheless, it improves the previously known $\mathcal{O}(nd \log \log n)$ solution of Gagie [5].

Keywords: minimax tree, Yeung's inequality, linear decision tree.

1 Introduction

Trees are one of the most common and useful objects appearing in such areas as graph theory, data structures, and information theory, to name just a few. There exists an enormous amount of research devoted to investigating various aspects of constructing *optimal* trees, for different definitions of optimality. In particular, a lot of effort has been put into solving variants of the following problem: given a collection of n weights construct a (binary, ternary, or t-ary) tree with those weights stored in the leaves so that the sum (or the maximum) of all root-to-leaf paths weights is minimized, where the path weight depends only on its length and the leaf weight.

For the case when the path weight is simply the product of those two quantities, a well-known $\mathcal{O}(n \log n)$ time solution was given by Huffman [11]. If we additionally require that the collection of weights is ordered, and they must be assigned in the natural left-to-right order in the tree, we call the problem *alphabetic*. A significantly more complex $\mathcal{O}(n \log n)$ time solution for the alphabetic version of the problem was developed by Hu and Tucker [10]. In some

⋆ Supported by MNiSW grant number N N206 492638, 2010–2012 and START scholarship from FNP.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 36–48, 2013.

special cases better running time is known to be possible. For example, the non-alphabetic variant can be solved in linear time assuming the weights are sorted [14], and the alphabetic case admits an $\mathcal{O}(n\sqrt{\log n})$ time algorithm in the word RAM model [13].

For the case when we minimize the maximum path weight, and the path weight is the sum of its length and the leaf weight, Golumbic [7] modified the Huffman's algorithm to find the optimum tree, which has been then used [4,8] to restrict the fan-in and fan-out of a circuit without increasing its size too much, and recently a linear time solution in the word RAM model of computation for the problem was given [6]. Also the alphabetic version of this case was considered before. Hu, Kleitman and Tamaki [9] observed that a certain modification of the Hu-Tucker algorithm can be used to compute the ordered minimax cost in $\mathcal{O}(n \log n)$ time (actually, their algorithm minimizes $w_i 2^{l_i}$, but this is easily seen to be equivalent). Then Kirkpatrick and Klawe [12] considered the strict t-ary version and applied their $\mathcal{O}(n \log n)$ time solution to study the effects of fan-out constraint in planar logical circuits (for a more recent application of a similar formulation, see [1]). Later Coppersmith, Klawe and Pippinger [3] solved the non-strict version with the same complexity. If the weights are integer, a linear time solution is known [12], and if the number of different rounded down weights is bounded by d, running time of $\mathcal{O}(nd \log \log n)$ is possible [5]. This suggest the following natural question: is there a linear time solution for the alphabetic case? Or maybe when we minimize the maximum instead of the sum this additional requirement makes the problem more complex in terms of the best running time possible? In this paper we show that this is not the case.

A (very) high level idea of our algorithm is the same as in the non-alphabetic case, but we need to apply a significantly more complicated reasoning in order to deal with the alphabetic constraint. Nevertheless, we are able to achieve a linear running time in the linear decision tree model. More precisely, we assume that we are given the integer and the fractional part of each w_i separately, and are allowed to branch based on the sign of a linear function of all $\lfloor w_i \rfloor$ and $\mathrm{frac}(w_i)$. To actually implement the algorithm in linear time, i.e., to quickly decide which expression should be computed next, we need the word RAM model of computation, though.

We start with a simple linear time algorithm for the case when all w_i are integer. While this is not a new result, the characterization of ordered binary search trees we use there can be applied to develop an $\mathcal{O}(nd)$ time algorithm for the more general case when w_i are arbitrary real numbers, with d being the cardinality of $\{\lfloor w_i \rfloor : i = 1, 2, \ldots, n\}$. Then we use the power of the word RAM model more extensively in order to improve the complexity to linear. Of course the interest in achieving such complexity is mostly theoretical, as in practice one would be fine with either suboptimal constructions, or slower algorithm. Nevertheless, we believe that determining the exact complexity of the problem is an intriguing question on its own. Furthermore, our computationally effective use of the Yeung's inequality, while maybe heavily tailored to this specific application, seems to be a direction unexplored before.

2 Preliminaries

Given a sequence of n real weights $w_1, w_2, \ldots, w_n$, we want to construct an ordered binary search tree on n leaves labelled with those weights. The labeling is ordered: the leftmost leaf should correspond to w_1, the second leftmost to w_2, and so on. Our goal is to minimize the maximum value of w_i plus depth of the i-th leaf. This quantity will be called the ordered minimax cost $M(w_1, w_2, \ldots, w_n)$.

We assume that for each w_i we are given its integer part $\lfloor w_i \rfloor$ and fractional $\mathrm{frac}(w_i)$ part separately. The only operation concerning those parts will be branching based on the sign of a linear function of all $\lfloor w_i \rfloor$ and $\mathrm{frac}(w_i)$. To quickly decide which expression should be computed next we assume the usual word RAM model with words of size $\Omega(\log n)$, though.

We are going to work with ordered binary trees, meaning that each node can have a left and a right child. The *shape* of such tree is a sequence of n integers, the depths of its leaves when read from left to right. We need an efficient way of checking whether a given shape corresponds to at least one tree. When the tree is not meant to be ordered, this is possible thanks to the well-known Kraft's inequality. In the ordered case the inequality is no longer useful, though. We use a different and not so well-known characterization instead.

Theorem 1 (Yeung's inequality [15]). *$\langle l_1, l_2, \ldots, l_n \rangle$ is a shape of some ordered binary tree if and only if*

$$f_{l_1} \circ f_{l_2} \circ \ldots \circ f_{l_{n-1}} \circ f_{l_n}(0) \leq 1 \tag{1}$$

where $f_a(x) = \frac{\lceil x 2^a \rceil + 1}{2^a}$.

The intuition behind the above formula is that to construct a tree of a given shape, one can always use a simple greedy method: add leaves from left to right, putting each of them as deeply as it is possible. A n-tuple $\langle l_1, l_2, \ldots, l_n \rangle$ for which the above lemma holds will be called a *valid shape*.

Having the above lemma, we can formulate the problem of computing the ordered minimax cost $M(w_1, w_2, \ldots, w_n)$ as follows: minimize $\max_i w_i + l_i$ among all $l_1, l_2, \ldots, l_n$ such that $\langle l_1, l_2, \ldots, l_n \rangle$ is a valid shape. From now on we will work with such formulation of the problem. Furthermore, we will assume that the weights are normalized so that $\min_i w_i = 0$ and $\max_i w_i \leq n$, as we can simply replace weights smaller than $\max_i w_i - n$ with $\max_i w_i - n$ and then subtract the same value from all of them at once.

Finally, notice that having the minimax cost allows us to actually output the whole tree in linear time. For this we first compute the depth of each node, and add them one-by-one starting from the leftmost. We maintain the rightmost path in the current (partial) tree on a stack, where maximal sequences of unary nodes are compressed into single objects. Then adding new leaf requires just (amortized) constant time.

3 Linear Time Algorithm for Integer Weights

We begin with a rather simple linear time algorithm for the case when all w_i are integers. While it neither improves or simplifies already known solutions, it does help to understand the general real weight case algorithm.

We begin with modifying the formulation of Lemma 1 to make it more convenient to use. First of all, we do not want to use fractions. Define $g_a(x) = \left\lceil \frac{x}{2^a} \right\rceil 2^a + 2^a$ and observe that any tree can be rebuilt so that the depths of all leaves do not exceed n and the depth of any leaf does not increase. Hence we can assume that $l_i \leq n$ and rewrite (1) as

$$g_{n-l_1} \circ g_{n-l_2} \circ \ldots \circ g_{n-l_{n-1}} \circ g_{n-l_n}(0) \leq 2^n \tag{2}$$

Lemma 1. *If all w_i are integers, $M(w_1, w_2, \ldots, w_n)$ can be calculated in linear time.*

Proof. First observe that the i-th leaf must be created at depth not exceeding $l_i = M(w_1, w_2, \ldots, w_n) - w_i$ and define:

$$A = g_{w_1} \circ g_{w_2} \circ \ldots \circ g_{w_n}(0)$$

Then if c is the smallest possible integer such that $A \leq 2^c$ (or, in other words, c is A rounded up to the nearest power of 2), the ordered minimax cost is c.

To calculate A, first recall that all w_i are between 0 and n. We start the computation with $x = 0$ and successively apply $g_{w_1}, g_{w_2}, \ldots, g_{w_n}$ to the current x. Note that because of the bounds on all w_i and the structure of all g_{w_i}, the current x will be always an integer between 0 and $n2^n$. We store it as a sorted list L of bits set to 1, so for example if the current value is 100110_2, we store $[1, 2, 5]$. Computing $g_{w_i}(x) = \left\lceil \frac{x}{2^a} \right\rceil 2^a + 2^a$ consists of two steps:

1. removing the prefix of L consisting of elements less than a,
2. adding 2^a twice or once to the current value of x, depending on whether we removed at least one element in the previous step or not, respectively.

For a detailed description of the procedure see INTEGER-ORDERED-MINIMAX. To bound its running time by $\mathcal{O}(n)$ we assign one credit to each element of L. □

4 $\mathcal{O}(nd)$ Time Algorithm for Real Weights

To deal with the case of non-integer weights we follow the approach of [12], who reduced the general case to a sequence of integers instances. We restate their method in a different form, which will be more convenient for our purposes.

Algorithm 1. INTEGER-ORDERED-MINIMAX$(w_1, w_2, \ldots, w_n)$

```
L ← []
for i ← 1 to n do
    t ← 1
    while L = [a, ...] and a < w_i do
        remove a from L
        t ← 2
    end while
    for k ← 1 to t do
        b ← w_i
        while L = [b, ...] do
            remove b from L
            b ← b + 1
        end while
        prepend w_i to L
    end for
end for
c ← last element of L
return c + [|L| > 1]
```

Lemma 2. $\lfloor M(w_1, w_2, \ldots, w_n) \rfloor = M(\lfloor w_1 \rfloor, \lfloor w_2 \rfloor, \ldots, \lfloor w_n \rfloor)$.

Proof. Left side minimizes $\lfloor \max_i w_i + l_i \rfloor$ among all valid shapes, and right side minimizes $\max_i \lfloor w_i \rfloor + l_i$. This immediately gives the claim. □

Lemma 3. *$M(w_1, w_2, \ldots, w_n) \leq X$ if and only if $M(w'_1, w'_2, \ldots, w'_n) \leq \lfloor X \rfloor$ where w'_i is $\lfloor w_i \rfloor$ if $\mathrm{frac}(w_i) \leq \mathrm{frac}(X)$ and $\lfloor w_i \rfloor + 1$ otherwise.*

Proof. $M(w_1, w_2, \ldots, w_n) \leq X$ if and only if for some valid shape $\langle l_1, l_2, \ldots, l_n \rangle$ inequality $w_i + l_i \leq X$ holds for all i. As l_i is integer, there are two cases:

1. $\mathrm{frac}(w_i) \leq \mathrm{frac}(X)$, the inequality is equivalent to $\lfloor w_i \rfloor + l_i \leq \lfloor X \rfloor$,
2. $\mathrm{frac}(w_i) > \mathrm{frac}(X)$, the inequality is equivalent to $\lfloor w_i \rfloor + 1 + l_i \leq \lfloor X \rfloor$.

Hence the whole claim follows. □

This gives a simple $\mathcal{O}(n \log n)$ time algorithm, as observed in [12]. First use Lemma 1 and Lemma 2 to compute $\lfloor M(w_1, w_2, \ldots, w_n) \rfloor$. Then note that the fractional part of the answer must be equal to some $\mathrm{frac}(w_i)$. We can compute it using a binary search inside which we use Lemma 1 and Lemma 3 (again) to check if a chosen $\mathrm{frac}(w_i)$ is bigger than $\mathrm{frac}(M(w_1, w_2, \ldots, w_n))$.

To accelerate the $\mathcal{O}(n \log n)$ time procedure, we must somehow reuse the information found by the successive steps of binary search. We split the whole $\{1, 2, \ldots, n\}$ into three parts L, C, and R. L contains indices i such that we

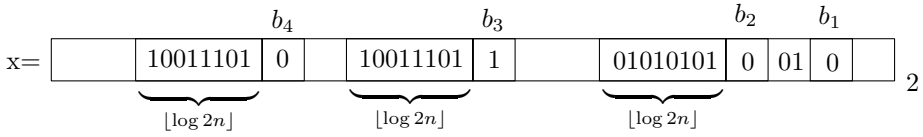


Fig. 1. Succinct representation of x is $[010_2, 010101010_2, 100111011_2, 100111010_2]$

already know that $\text{frac}(w_i) \leq \text{frac}(M(w_1, w_2, \ldots, w_n))$, R contains i for which we already know that $\text{frac}(w_i) > \text{frac}(M(w_1, w_2, \ldots, w_n))$, and C consists of all the remaining indices. At each step we select the median of $\{\text{frac}(w_i) : i \in C\}$ and compare it with $\text{frac}(M(w_1, w_2, \ldots, w_n))$ using Lemma 1 and Lemma 3. Depending on the outcome of this comparison we move half of the elements of C into L or R. Observe that at each step of the computation values of w_i with $i \in L \cup R$ are permanently round up or down. This suggest that if a whole segment $i, i+1, i+2, \ldots, j$ belongs to $L \cup R$ already, we could try to somehow preprocess the function $g_{w'_i} \circ g_{w'_{i+1}} \circ g_{w'_{i+2}} \circ \ldots \circ g_{w'_j}$, where each w'_i is either $\lfloor w_i \rfloor$ or $\lfloor w_i \rfloor + 1$, and apply the whole compositions at once instead of processing their elements one-by-one. Hence we need to take a closer look at how such composition $g_{a_1,a_2,\ldots,a_k}(x) = g_{a_1} \circ g_{a_2} \circ \ldots \circ g_{a_k}(x)$ looks like.

Lemma 4. *If $a_1, a_2, \ldots, a_k$ are nonnegative integers, $g_{a_1,a_2,\ldots,a_k}(x) = \lceil \frac{x+r}{2^a} \rceil 2^a + c$, where $a = max_i\ a_i$ and r, c are of the form $\sum_{i=1}^{k} \alpha_i 2^{a_i}$ with $\alpha_i \in \{0, 1, 2\}$ for all i.*

Now if there are just d different values of $\lfloor w_i \rfloor$, any $g_{w'_i} \circ g_{w'_{i+1}} \circ \ldots \circ g_{w'_j}$ depends on just $\mathcal{O}(d \log n)$ bits. More specifically, assume those rounded down values are $b_1 < b_2 < \ldots < b_d$. For any position k which does not belong to any block of the form $\{d_i, d_i + 1, \ldots, d_i + \lfloor \log 2n \rfloor\}$, the k-th bit of all $g_{w'_1} \circ g_{w'_2} \circ \ldots \circ g_{w'_i}(0)$ is set to zero. Hence while the numbers x, r, c we operate on might be as large as $n2^n$, there are just d blocks of $\mathcal{O}(\log n)$ consecutive indices which (potentially) correspond to nonzero bits. Thus x, r and c can be actually described in just d machine words, each word storing the bits from a single block, see Figure 1. It is easy to see that given such *succinct* representation of x we can compute any $g_{w'_i}(x)$ in $\mathcal{O}(d)$ arithmetical operations on whole words. More generally, given the succinct representations of x, r, and c we can compute $\lceil \frac{x+r}{2^a} \rceil 2^a + c$ in $\mathcal{O}(d)$ time as well.

The last building block for the $\mathcal{O}(nd)$ time algorithm is a method for computing the description of $g_{a_1,a_2,\ldots,a_k} \circ g_{b_1,b_2,\ldots,b_\ell}$ given the descriptions of $g_{a_1,a_2,\ldots,a_k}$ and $g_{b_1,b_2,\ldots,b_\ell}$. In other words, we need to describe how a composition of two functions $h_1(x) = \left\lceil \frac{x+r_1}{2^a} \right\rceil 2^a + c_1$ and $h_2(x) = \left\lceil \frac{x+r_2}{2^b} \right\rceil 2^b + c_2$ looks like.

Lemma 5. *Let $h_1(x) = \left\lceil \frac{x+r_1}{2^a} \right\rceil 2^a + c_1$ and $h_2(x) = \left\lceil \frac{x+r_2}{2^b} \right\rceil 2^b + c_2$. If $b \leq a$ then $h_2(h_1(x)) = \left\lceil \frac{x+r_1}{2^a} \right\rceil 2^a + \left\lceil \frac{c_1+r_2}{2^b} \right\rceil 2^b + c_2$, otherwise $h_2(h_1(x)) = \left\lceil \frac{x+r_1+\left\lceil \frac{c_1+r_2}{2^a} \right\rceil 2^a}{2^b} \right\rceil 2^b + c_2$.*

Using the above lemma, given the succinct representations of r_1, c_1, r_2, c_2 we can compute the representation of $h_1 \circ h_2$ in $\mathcal{O}(d)$ time. The procedure which calculates $M(w_1, w_2, \ldots, w_n)$ using such operations will be called Slow-ordered-minimax.

Theorem 2. Slow-ordered-minimax *computes* $M(w_1, w_2, \ldots, w_n)$ *in* $\mathcal{O}(nd)$ *time, where* d *is the number of different values of* $\lfloor w_i \rfloor$.

Proof. We start with computing $M(\lfloor w_1 \rfloor, \lfloor w_2 \rfloor, \ldots, \lfloor w_n \rfloor)$. Initially we do not know whether w_i should be rounded up or down for any i. During the execution of Slow-ordered-minimax we repeatedly choose the median of all $\mathrm{frac}(w_i)$ with i belonging to the set of indices which we do not know how to round yet. By a well-known result the selection can be performed in $\mathcal{O}(|C|)$ time [2]. Then we temporarily round all w_i with $\mathrm{frac}(w_i)$ not exceeding this median down, and all remaining w_i up, and compute the ordered minimax cost for the rounded weights w'_i. If $M(w'_1, w'_2, \ldots, w'_n) = M(\lfloor w_1 \rfloor, \lfloor w_2 \rfloor, \ldots, \lfloor w_n \rfloor)$ we permanently move all indices corresponding to the rounded down w_i from C to L. Otherwise we move all indices corresponding to the rounded up w_i from C to R. In either case, corresponding w_i stay rounded till the end of the procedure.

To evaluate $M(w'_1, w'_2, \ldots, w'_n)$ efficiently, for each maximal segment of indices $i, i+1, \ldots, j$ belonging $L \cup R$ we store a succinct description of the corresponding function $g_{w'_i, w'_{i+1}, \ldots, w'_j}$. More precisely, we store an ordered collection S of maximal segments $[\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_s, r_s]$ consisting of indices from $L \cup R$, and for each such segment we keep the corresponding function. Then computing $M(w'_1, w'_2, \ldots, w'_n)$ reduces to evaluating a composition of $|S| + |C|$ functions given by their succinct descriptions. Note that because the segments in S are maximal, $|S| \leq |C| + 1$, so this evaluation can be performed in $\mathcal{O}(d|C|)$ time. After computing $M(w'_1, w'_2, \ldots, w'_n)$ we shrink C by moving half of its elements to either L or R. As a consequence we must update S by first adding singleton segments $[i, i]$ for all i removed from C and then gluing together all pairs of neighboring segments of the form $[\ell_i, r_i]$, $[\ell_{i+1}, r_{i+1}]$ with $r_i + 1 = \ell_{i+1}$. Both operations require just $\mathcal{O}(|C|)$ operations on succinct descriptions of either integers or functions, and hence the total running time is $\mathcal{O}(d(n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots)) = \mathcal{O}(nd)$. □

While we do use arithmetical operations on whole words, the only computation directly concerning the input numbers is comparing two fractional parts, computing the sorted list of all b_i, and finding the position of each $\lfloor w_i \rfloor$ on this list. As the allowed time per element is $\mathcal{O}(d)$, we can afford to simply traverse the list for each i to either update it or find the corresponding position.

5 Linear Time Algorithm for Real Weights

At a very high level, the idea behind the fully linear time algorithm is the same as in Slow-ordered-minimax$(w_1, w_2, \ldots, w_n)$. We iteratively select the median fractional part of all w_i with $i \in C$ and after computing $g_{w'_1} \circ g_{w'_2} \circ \ldots \circ g_{w'_n}(0)$ remove half of C. The bottleneck is clearly the evaluation of g.

Note that a running time of order $\mathcal{O}(\frac{n}{\log n})$ would be perfectly acceptable here in order to get a linear overall bound. Before we describe how to speed up the evaluation, we need to take a closer look at our computational model. While the only operation on the fractional parts of the weights will be comparison, for each integer part (which is between 0 and n by the assumptions described in the preliminaries) we need to have its value available in a machine word in order to facilitate indirect addressing. As we are interested in the linear decision tree model, we cannot simply say that we store a part of the input in our working memory. Nevertheless, we can easily apply binary search to compute $\lfloor w_i \rfloor$ in $\mathcal{O}(\log \log n)$ time. To decrease this complexity, we observe that in some cases we do not need the exact value of each $\lfloor w_i \rfloor$. For example, the exact value of $\lfloor w_1 \rfloor$ is not important as long as $\lfloor w_1 \rfloor + 1 < \lfloor w_2 \rfloor$. A more general version of this observation is formulated below.

Lemma 6. *We can compute in linear time a sequence of n machine words with the property that replacing each $\lfloor w_i \rfloor$ with the integer stored in the i-th word does not change the ordered minimax cost. The computation accesses the input numbers only by branching based on the sign of a linear function of all $\lfloor w_i \rfloor$.*

Now we can focus on how to accelerate computing g. For this we split all indices $\{1, 2, \ldots, n\}$ into groups of consecutive $\lfloor \log n \rfloor$ elements, and call such group a *package*. Consider the function $g_{w'_i, w'_{i+1}, \ldots, w'_{i+\lfloor \log n \rfloor - 1}}$ corresponding to such package. Each w'_i is either $\lfloor w_i \rfloor$ or $\lfloor w_i \rfloor + 1$, so by Lemma 4 this function is of the form $\lceil \frac{x+r}{2^a} \rceil 2^a + c$ with both r and c possibly containing nonzero bits just on a specified set of $2 \lfloor \log n \rfloor$ positions, no matter how the corresponding w_i are rounded. Let the sorted list of those positions be $[t_1, t_2, \ldots, t_{2\lfloor \log n \rfloor}]$. Note that we can easily construct all such sorted lists in linear time by preprocessing all the packages at once in the very beginning. Additionally for any index i we store the positions of bits corresponding to $\lfloor w_i \rfloor$ and $\lfloor w_i \rfloor + 1$ on its package list. Observe that composing any sequence of $g_{w'_i} \circ g_{w'_{i+1}} \circ \ldots \circ g_{w'_j}$ with all indices belonging to the same package results in a function $\lceil \frac{x+r}{2^a} \rceil 2^a + c$ with both r and c containing nonzero bits only on positions from the package list. This allows us to store a succinct description of such function in a constant amount of machine words. Furthermore, given such descriptions of two functions corresponding to consecutive fragments of the same package, we can compute the description of their composition in constant time by Lemma 5.

During the execution of the algorithm some w'_i are permanently rounded up, some are permanently rounded down, and some are yet undecided. For each package we consider its maximal fragments consisting of indices with already known values of w'_i. We store succinct descriptions of all corresponding functions and update them accordingly whenever any w'_i becomes fixed. We claim that this allows us to construct succinct descriptions of all functions corresponding to the whole packages.

Lemma 7. *Given succinct descriptions of all functions corresponding to maximal fragments of packages consisting of indices with already fixed values of w'_i*

and the current value of t, we can compute succinct descriptions of all functions corresponding to whole packages in $\mathcal{O}(|C| + \frac{n}{\log n})$ time.

Proof. We consider the packages one by one. Consider a single package. First for any index i for which w'_i is not fixed yet, we construct a succinct description of the function $g_{\lfloor w_i \rfloor + [\mathrm{frac}(w_i) > t]}$. This requires just constant time as during the preprocessing stage we found the position of $\lfloor w_i \rfloor + [\mathrm{frac}(w_i) > t]$ on the package list. Then we must compute a succinct description of the composition of all functions corresponding to the fragments of the current package. Due to Lemma 5 this requires time proportional to the number of those fragments. There are $\frac{n}{\log n}$ packages and $|C|$ not permanently rounded indices so the total running time is as claimed. □

Given succinct descriptions of functions corresponding to all packages, we still have to somehow evaluate their composition at 0. We would like to start with $x = 0$ and apply the functions one by one. This is not that simple to perform quickly, though. While there are just $\frac{n}{\log n}$ functions, and we already have a succinct description of each of them, lists of different packages might consist of completely different elements. Thus as a result of applying them one by one we might get x with more than just $2 \lfloor \log n \rfloor$ bits set to one, and so we cannot simply claim that applying a single function takes constant time.

By Lemma 4 as a result of applying the functions we get $0 \leq x \leq n2^n$. The obvious method of storing the current value of x would be to keep a list of its bit set to one as we did in INTEGER-ORDERED-MINIMAX which would require $\Theta(n)$ time to operate on. An obvious improvement would be to store the values of $\log n$ consecutive bits in a single machine word. While it improves the storage requirements to $\mathcal{O}(\frac{n}{\log n})$, it is not clear how to apply the functions efficiently using such representation. We switch to a more complicated hybrid storage method.

Definition 1. *A hybrid representation of a n-bit integer x is an ordered list of objects. There are two types of objects:*

chunk *a description of at most $\log n$ consecutive bits,*
scattered chunk *a description of a range of consecutive bits, some of them corresponding to a contiguous fragment of a single package list, and all remaining set to zero.*

For a chunk we store the values of those bits in a single machine word together with the position of the first and last bit in x. For a scattered chunk we store the corresponding package number and the values of the potentially nonzero bits in a single machine word.

Note that the positions of the extreme bits on the package list in fact give us the corresponding positions in x. We call a representation valid if the ranges of bits of x referred to by the objects are disjoint and sorted. See Figure 2 for an example of such valid representation.

Consider evaluating $h(x) = \left\lceil \frac{x+r}{2^a} \right\rceil 2^a + c$ given a valid hybrid representation of x and a succinct representation of r and c. It requires performing a few steps:

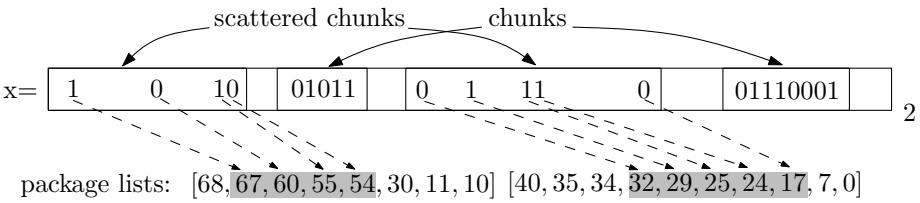


Fig. 2. Example of a valid hybrid representation

1. check if the current value of x modulo 2^a plus r exceeds 2^a,
2. erase all bits on positions less than a,
3. add 2^a once or twice to the current value of x,
4. add $\left\lfloor \frac{c}{2^a} \right\rfloor 2^a$ to the current value of x,
5. add $c \bmod 2^a$ to the current value of x.

Some of those steps are fairly simple to perform efficiently assuming we can use the hybrid representation. Consider step **5**. Succinct representation of c can be immediately converted into a scattered chunk, and after step **2** there will be no bits set to one at positions $0, 1, \ldots, a-1$, so this new scattered chunk can be simply appended to the current representation. Note that a is either a' or $a'+1$, where a' is the maximum value of w_i with i belonging to the corresponding package. We will need some preprocessing depending on those values of a and thus we would like a to be the same no matter how the w_i are rounded. This can be ensured by replacing step **5** with:

5(a). if $a = a'+1$ and the a'-th bit of c is set, add $2^{a'}$ to the current value of x,
5(b). add $c \bmod 2^{a'}$ to the current value of x.

Let the functions we are applying be $h_1, h_2, \ldots, h_{\frac{n}{\log n}}$. Assuming step **5** is the only place a new scatter chunk can be created, the above modification ensures the following property:

Lemma 8. *For any i and j there exists at most one package such that the i-th bit of $h_1 \circ h_2 \circ \ldots h_j(0)$ belongs to a scattered chunk referring to this package, no matter how all w_i were rounded.*

Proof. Induction on i. For i there are no scattered chunks so the claim holds. Assuming it holds for some i, applying h_{i+1} creates one new scattered chunk describing bits 0,1,...,$a'-1$ and for all higher positions the claim holds by the induction hypothesis. □

Lemma 9. *Given a scattered chunk describing bits at positions $i, i+1, \ldots, j$ we can construct a chunk describing bits at positions $i, i+1, \ldots, i+\log n-1$ and a scattered chunk describing bits at positions $i+\log n, \ldots, j$ in constant time.*

Proof. For any position k in a package list we construct a word with the i-th bit (with $0 \leq i \leq \log n$) set if and only if $i+k$ belongs to the list as well.

Then given a scattered chunk we retrieve the preprocessed word. By counting the number of bits set there we calculate the shift necessary to construct the new scattered chunk. Constructing the chunk reduces to the following problem: given $b = \sum_i 2^{j_i}$ with j_i strictly increasing and $c = \sum_i \beta_i 2^i$ (both given in single words) compute the value of $\sum_i \beta_i 2^{j_i}$. This can be answered in constant time after a simple preprocessing. □

Lemma 10. *Given hybrid representations of two integers $0 \leq x, y < 2^a$ we can check if $x + y \geq 2^a$ in linear time.*

Proof. To compute the carry we process the representations from right to left. Instead of performing the processing bit-by-bit, we go through whole chunks at once. The only problem is that we must be able to add two scattered chunks and a chunk to a scattered chunk efficiently. For the latter we apply Lemma 9 and simply add two chunks. For the former let the scattered chunks refer to potentially nonzero bits at positions $i_1, i_1+1, \ldots, j_1$ and $i_2, i_2+1, \ldots, j_2$, respectively. Note that if $j_1 > i_1 + 2\log n$ and $j_2 > i_2 + 2\log n$ there will be no carry no matter which bits are set. Otherwise we apply Lemma 9 twice and remove at least one scattered chunk from the representations. □

Lemma 11. *Steps* **1** *and* **2** *can be performed in amortized constant time.*

Proof. We apply Lemma 10 with $x \bmod 2^a$ and r. The hybrid representation of $x \bmod 2^a$ is a suffix of the current representation of x, with the exception that we might need to split a chunk (which is simple to perform in constant time) or a scattered chunk into two. To perform the latter in constant time, note that due to Lemma 8 we can preprocess the predecessor of a' on the package list corresponding to this scattered chunk, as the list is the same no matter how we round the w_i. Using such preprocessing we can locate the predecessor of a and erase all bits on its right. This allows us to construct hybrid representations of $x \bmod 2^a$ and $\left\lfloor \frac{x}{2^a} \right\rfloor 2^a$ in time proportional to the size of the former. After applying Lemma 10 we replace x with $\left\lfloor \frac{x}{2^a} \right\rfloor 2^a$ so the time can be amortized by the decrease in the size of the representation. □

Lemma 12. *Given $0 \leq t \leq n$ and a hybrid representation of x we can add t to x in amortized constant time.*

Proof. We convert t into a chunk and go through the representation of x from right to left using Lemma 9 as long as there is a carry. Note that we stop as soon as we encounter a scattered chunk referring to a sufficiently large range of bits, namely at least $2\log n$. All smaller scattered chunks end up converted to chunks. To amortize to constant time we assign one credit to each chunk and two credits to each scattered chunk. □

Lemma 13. *Steps* **3** *and* **4** *can be performed in amortized constant time.*

Proof. After step **2** there are no bits at positions $0, 1, \ldots, a-1$ and thus while both 2^a and $\left\lfloor \frac{c}{2^a} \right\rfloor 2^a$ can be much larger than n, we will try to apply Lemma 12

as if the numbers were divided by 2^a. It is easy to see that we can do that with 2^a but $\left\lfloor \frac{c}{2^a} \right\rfloor 2^a$ requires more attention. By Lemma 4, $\left\lfloor \frac{c}{2^a} \right\rfloor \leq \lfloor \log n \rfloor$. Unfortunately, c is given in a succinct representation, and we would like to compute $\left\lfloor \frac{c}{2^a} \right\rfloor$ as an integer. This can be done by preprocessing the position of a on the corresponding package list (note that we actually have to preprocess the positions of a' and $a'+1$ which are the possible values of a), which allows us to construct a scattered chunk describing $\left\lfloor \frac{c}{2^a} \right\rfloor 2^a$ with a constant number of bitwise operations. Then we apply Lemma 9 to compute $\left\lfloor \frac{c}{2^a} \right\rfloor$, and Lemma 12 to add it to the current x. □

Lemma 14. *Step* **5** *can be performed in constant time.*

Proof. After precomputing the position of a' on the corresponding package list we can check if the a'-th bit is set in c in constant time. If so, we append a new chunk containing just one bit at the a'-th position to the hybrid representation of x. Then we construct and append a new scattered chunk describing $c \bmod 2^{a'}$ by simply erasing bits at higher position from the succinct representation of c and converting it into the scattered chunk in constant time. □

Finally we combine Lemma 11, Lemma 13 and Lemma 14.

Theorem 3. *$M(w_1, w_2, \ldots, w_n)$ can be calculated in $\mathcal{O}(n)$ time.*

Acknowledgements. Many thanks to Travis Gagie for helpful discussions concerning (alphabetic and not) minimax trees.

References

1. Bartoschek, C., Held, S., Maßberg, J., Rautenbach, D., Vygen, J.: The repeater tree construction problem. Inf. Process. Lett. 110(24), 1079–1083 (2010)
2. Blum, M., Floyd, R., Pratt, V., Rivest, R., Tarjan, R.: Time bounds for selection. Journal of Computer and System Sciences 7(4), 448–461 (1973)
3. Coppersmith, D., Klawe, M., Pippenger, N.: Alphabetic Minimax Trees of Degree at Most t. SIAM Journal on Computing 15, 189 (1986)
4. Parker Jr., D.S.: Combinatorial Merging and Huffman's Algorithm. IEEE Transactions on Computers, 365–367 (1979)
5. Gagie, T.: A new algorithm for building alphabetic minimax trees. Fundam. Inf. 97, 321–329 (2009), http://portal.acm.org/citation.cfm?id=1735991.1735995
6. Gawrychowski, P., Gagie, T.: Minimax trees in linear time with applications. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 278–288. Springer, Heidelberg (2009)
7. Golumbic, M.C.: Combinatorial merging. IEEE Trans. Comput. 25, 1164–1167 (1976), http://dx.doi.org/10.1109/TC.1976.1674574
8. Hoover, H.J., Klawe, M.M., Pippenger, N.J.: Bounding fan-out in logical networks. J. ACM 31, 13–18 (1984), http://doi.acm.org/10.1145/2422.322412
9. Hu, T., Kleitman, D., Tamaki, J.: Binary trees optimum under various criteria. SIAM Journal on Applied Mathematics 37(2), 246–256 (1979)
10. Hu, T., Tucker, A.: Optimal computer search trees and variable-length alphabetical codes. SIAM Journal on Applied Mathematics 21(4), 514–532 (1971)

11. Huffman, D.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE 40(9), 1098–1101 (1952)
12. Kirkpatrick, D., Klawe, M.: Alphabetic minimax trees. SIAM Journal on Computing 14, 514 (1985)
13. Larmore, L.L., Przytycka, T.M.: The optimal alphabetic tree problem revisited. J. Algorithms 28(1), 1–20 (1998)
14. van Leeuwen, J.: On the construction of Huffman trees. In: ICALP, pp. 382–410 (1976)
15. Yeung, R.: Alphabetic codes revisited. IEEE Transactions on Information Theory 37(3), 564–572 (2002)

Decidability and Enumeration for Automatic Sequences: A Survey

Jeffrey Shallit

School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
shallit@cs.uwaterloo.ca

Abstract. In this talk I will report on some recent results concerning decidability and enumeration for properties of automatic sequences. This is work with Jean-Paul Allouche, Émilie Charlier, Narad Rampersad, Dane Henshall, Luke Schaeffer, Eric Rowland, Daniel Goč, and Hamoon Mousavi.

1 Introduction

An infinite sequence $\mathbf{a} = (a_n)_{n \geq 0}$ over a finite alphabet is said to be *k-automatic* if there exists a deterministic finite automaton (with outputs associated with the states) such that after completely processing the input n expressed in base k, the automaton reaches some state q with output a_n [17,5]. A typical example of such a sequence is the Thue-Morse sequence

$$\mathbf{t} = t_0 t_1 t_2 \cdots = \mathtt{011010011001} \cdots ,$$

which is generated by the automaton in Figure 1. Here the input is n, expressed in base 2.

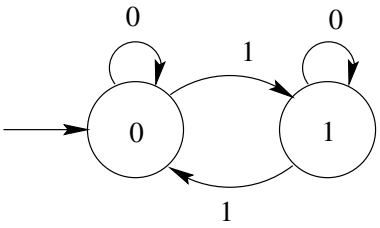


Fig. 1. Finite automaton generating the Thue-Morse sequence **t**

The Thue-Morse sequence is named after the Norwegian mathematician Axel Thue [45,46,6], who discovered it in 1912, although it also appears if one reads "between the lines" in an 1851 paper of Prouhet [39], and it has since been rediscovered many times (e.g., [34,22]). For more information about **t**, see the survey [4].

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 49–63, 2013.

Below we list just a few of the properties of **t** that people have studied. By a *factor* we mean a contiguous block of symbols inside another word.

1. **t** is not ultimately periodic.
2. **t** contains no factor that is an *overlap*, that is, a word of the form $axaxa$, where a is a single letter and x is an arbitrary finite word [45,46,6].
3. **t** has infinitely many distinct palindromic factors and infinitely many distinct antipalindromic factors. (A *palindrome* is a word equal to its reverse; an example of a palindrome in Russian is доход ("income"). An *antipalindrome* is a word of the form $x\overline{x^R}$, where x^R denotes the reverse of x and $\overline{0} = 1$, $\overline{1} = 0$.)
4. The number $p(n)$ of distinct palindromic factors of length n in **t** is given by

$$p(n) = \begin{cases} 0, & \text{if } n \text{ odd and } n \geq 5; \\ 1, & \text{if } n = 0; \\ 2, & \text{if } 1 \leq n \leq 4, \text{ or } n \text{ even and } 3 \cdot 4^k + 2 \leq n \leq 4^{k+1} \text{ for } k \geq 1; \\ 4, & \text{if } n \text{ even and } 4^k + 2 \leq n \leq 3 \cdot 4^k \text{ for } k \geq 1; \end{cases}$$

   see [7]. A similar expression exists for the number $p'(n)$ of distinct antipalindromic factors of length n.
5. **t** contains infinitely many distinct square factors xx, but for each such factor we have $|x| = 2^n$ or $3 \cdot 2^n$, for $n \geq 1$. Examples of squares in Russian include дядя ("uncle") and кускус ("couscous").
6. **t** is *mirror-invariant*: if x is a finite factor of **t**, then so is its reverse x^R.
7. **t** is *recurrent*, that is, every factor that occurs, occurs infinitely often [34].
8. **t** is *uniformly recurrent*, that is, for all factors x occurring in **t**, there is a constant $c(x)$ such that two consecutive occurrences of x are separated by at most $c(x)$ symbols [35, pp. 834 et seq.].
9. **t** is *linearly recurrent*, that is, it is uniformly recurrent and furthermore there is a constant C such that $c(x) \leq C|x|$ for all factors x [35, pp. 834 et seq.]. In fact, the optimal bound is given by $c(1) = 3$, $c(2) = 8$, and $c(n) = 9 \cdot 2^e$ for $n \geq 3$, where $e = \lfloor \log_2(n-2) \rfloor$.
10. The lexicographically least sequence in the orbit closure of **t** is $\overline{t_1}\ \overline{t_2}\ \overline{t_3}\ \cdots$, which is also 2-automatic [2].
11. The *subword complexity* $\rho(n)$ of **t**, which is the function counting the number of distinct factors of **t**, is given by

$$\rho(n) = \begin{cases} 2^n, & \text{if } 0 \leq n \leq 2; \\ 2n + 2^{t+2} - 2, & \text{if } 3 \cdot 2^t \leq n \leq 2^{t+2} + 1; \\ 4n - 2^t - 4, & \text{if } 2^t + 1 \leq n \leq 3 \cdot 2^{t-1}; \end{cases}$$

    see [8,32].
12. **t** has an unbordered factor of length n if $n \not\equiv 1 \pmod 6$ [19]. (Here by an *unbordered* word y we mean one with no expression in the form $y = uvu$ for words u, v with u nonempty.)

Recently I and my co-authors J.-P. Allouche, E. Charlier, D. Goč, D. Henshall, N. Rampersad, E. Rowland, and L. Schaeffer, have developed and implemented a decision procedure by which all these assertions, and many others, can be feasibly verified and/or generated in a purely mechanical fashion. In this talk I will explain how our method works, what has been done so far, and what remains to be done.

2 Logic

By $\mathrm{Th}(\mathbb{N}, +, 0, 1, <)$ I mean the set of all true first-order sentences in the logical theory of the natural numbers with addition. In such a theory, for example, we can express the so-called "Chicken McNuggets" theorem [47, Lesson 5.8, Problem 1] to the effect that 43 is the largest integer that cannot be represented as a non-negative integer linear combination of 6, 9, and 20, as follows:

$$(\forall n > 43\ \exists x, y, z \geq 0 \text{ such that } n = 6x + 9y + 20z) \wedge \\ \neg(\exists x, y, z \geq 0 \text{ such that } 43 = 6x + 9y + 20z). \quad (1)$$

Here, of course, "$6x$" is shorthand for the expression "$x + x + x + x + x + x$", and similarly for $9y$ and $20z$.

Thanks to the work of Presburger [37,38] we know that $\mathrm{Th}(\mathbb{N}, +, 0, 1, <)$ is *decidable*: that is, there exists an algorithm that, given a sentence in the theory, will decide its truth.

In fact, there is a relatively simple proof of this fact, based on finite automata, and due to Büchi [11,12], Elgot [21], and Hodgson [27]. More recently it has appeared (without attribution) in the textbook of Sipser [44, §6.2] and progress has been made on its complexity (e.g., [29]). The idea is to represent integers in a integer base $k \geq 2$ using the alphabet $\Sigma_k = \{0, 1, \ldots, k-1\}$. We can then represent n-tuples of integers as words over the alphabet Σ_k^n, padding with leading zeroes, if necessary. Thus, for example, the pair $(21, 7)$ can be represented in base 2 by the word

$$[1, 0][0, 0][1, 1][0, 1][1, 1].$$

Then the relation $x + y = z$ can be checked by a simple 2-state automaton depicted in Figure 2, where transitions not depicted lead to a nonaccepting "dead state".

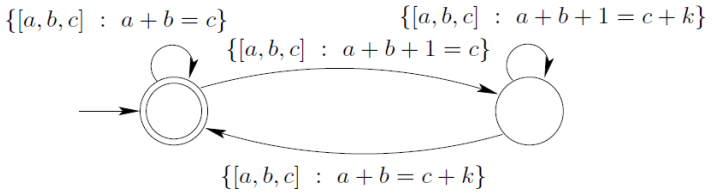


Fig. 2. Checking addition in base k

Relations like $x = y$ and $x < y$ can be checked similarly.

Given a formula with free variables $x_1, x_2, \ldots, x_n$, we construct an automaton accepting the base-k expansion of those n-tuples $(x_1, \ldots, x_n)$ for which the proposition holds. If a formula is of the form $\exists x_1, x_2, \ldots x_n\ p(x_1, \ldots, x_n)$, then we use nondeterminism to "guess" the x_i and check them. If the formula is of the form $\forall p$, we use the equivalence $\forall p \equiv \neg\exists\neg p$; this may require using the subset construction to convert an NFA to a DFA and then flipping the "finality" of states. Finally, the truth of a formula can be checked by using the usual depth-first search techniques to see if any final state is reachable from the start state.

However, even more is true. If we add the function $V_k : \mathbb{N} \to \mathbb{N}$ to our logical theory, where $V_k(x) = k^n$, and k^n is the largest power of k dividing x, it is still decidable by a similar automaton-based technique [10]. By doing so, we gain the capability of deciding many questions about automatic sequences. Thus we have

Theorem 1. *There is an algorithm that, given a predicate phrased using only the universal and existential quantifiers, indexing into a given automatic sequence* **a***, addition, subtraction, logical operations, and comparisons, will decide the truth of that proposition.*

We call such a predicate an *automatic predicate*.

Although the worst-case running time of our algorithm is bounded above by

$$2^{2^{\cdot^{\cdot^{\cdot^{2^{p(N)}}}}}},$$

where the number of 2's in the exponent is equal to the number of quantifiers, p is a polynomial, and N is the number of states needed to describe the underlying automatic sequence, it turns out that in practice, significantly better running times are usually achieved.

3 Periodicity

An infinite word **a** is *periodic* if it is of the form $x^\omega = xxx\cdots$ for a finite nonempty word x. It is *ultimately periodic* if it is of the form yx^ω for a (possibly empty) finite word y.

Honkala [28] was the first to prove that ultimate periodicity is decidable for automatic sequences. Later, Leroux [31], and, more recently, Marsault and Sakarovitch [33] gave efficient algorithms for the problem.

Using our approach, we can easily see that periodicity is decidable for k-automatic sequences [3]. It suffices to express ultimately periodicity as an automatic predicate:

$$\exists p \geq 1, N \geq 0\ \forall i \geq N\ \mathbf{a}[i] = \mathbf{a}[i + p].$$

When we run this on the Thue-Morse sequence, we discover (as expected) that **t** is not ultimately periodic.

4 Repetitions

Repetitions in sequences have been studied for over a hundred years. We defined overlaps above in § 1. Other classic repetitions include *squares* (factors of the form xx, where x is nonempty) and *cubes* (factors of the form xxx).

Thue [46] proved that $\mathbf{t}$ contains no overlaps; that is, $\mathbf{t}$ is overlap-free. Using our technique, we can express the property of having an overlap $axaxa$ beginning at position N with $|ax| = p$, as follows: $\mathbf{a}[N..N+p] = \mathbf{a}[N+p..N+2p]$. So the corresponding automatic predicate for $\mathbf{t}$ is

$$\exists p \geq 1, N \geq 0 \ \mathbf{t}[N..N+p] = \mathbf{t}[N+p..N+2p],$$

or, in other words,

$$\exists p \geq 1, N \geq 0 \ \forall i, 0 \leq i \leq p \ \mathbf{t}[N+i] = \mathbf{t}[N+p+i].$$

From now on, we will abbreviate predicates like the one above by writing the first form only.

We programmed up our decision procedure and verified that indeed $\mathbf{t}$ is overlap-free [3].

We can also ask about the lengths and positions of squares in the Thue-Morse sequence. Here we can create an automaton to accept

$$\{(N,p)_2 \ : \ p \geq 1 \text{ and } N \geq 0 \text{ and } \mathbf{t}[N..N+p-1] = \mathbf{t}[N+p..N+2p-1]\}.$$

When we do so, we get the automaton depicted below in Figure 3 (computed by Daniel Goč).

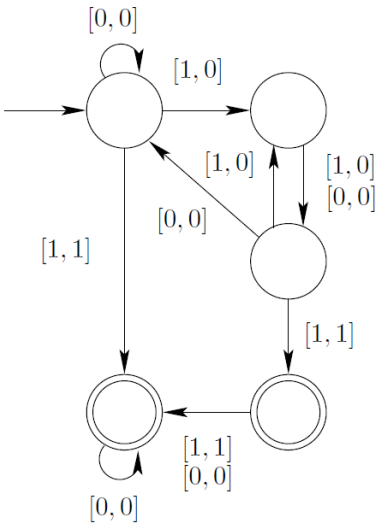


Fig. 3. Positions and lengths of squares in Thue-Morse

From this automaton, we easily recover the results of [36,8] that the only squares xx that occur have $|x| = 2^n$ or $|x| = 3 \cdot 2^n$ for $n \geq 0$, and all those lengths occur. The positions where these squares occur were previously given by Brown et al. [9].

5 Critical Exponent

We can define more general repetitions as follows: a word x is an α-power for $\alpha \geq 1$ if we can write $x = y^e y'$ where $e = \lfloor \alpha \rfloor$ and y' is a prefix of y and $|x| = \alpha |y|$. Thus, for example, `abracadabra` is an $\frac{11}{7}$-power and the Russian word люблю ("love") is a $\frac{5}{3}$-power. The techniques above suffice to check if a k-automatic sequence has α-powers, using the following predicate:

$$\exists N \geq 0, p, q \geq 1 \ \mathbf{a}[N..N+p-q-1] = \mathbf{a}[N+q..N+p-1] \text{ and } p = \alpha q.$$

However, this observation alone does not suffice to compute the so-called *critical exponent* of $\mathbf{a}$, which is the supremum over all rational α such that $\mathbf{a}$ has α-power factors.

It turns out that the critical exponent is also computable for automatic sequences [43,42]. More generally, we can extend the concept of k-automatic sets of natural numbers to k-automatic sets of non-negative rational numbers, as follows. Given a word $x \in (\Sigma_k \times \Sigma_k)^*$, define

$$\mathrm{quo}_k(x) = \frac{[\pi_1(x)]_k}{[\pi_2(x)]_k},$$

where $\pi_i(x)$ is the projection of x onto its i'th coordinate ($i = 1, 2$), and $[x]_k$ is the integer represented by the word x in base k. This is extended to languages L in the usual way:

$$\mathrm{quo}_k(L) = \{\mathrm{quo}_k(x) \ : \ x \in L\}.$$

Then we have

Theorem 2. *If M is a DFA, then* $\sup \mathrm{quo}_k(L(M))$ *is either rational or infinite, and it is computable.*

We can now apply this theorem to our problem. Let $\mathbf{a}$ be a k-automatic sequence. Using the techniques above, we can compute a DFA M accepting the language

$$L = \{(p, q) \ : \ \exists N \ \mathbf{a}[N..N+p-q-1] = \mathbf{a}[N+q..N+p-1]\},$$

which represents all fractional powers p/q occurring in $\mathbf{a}$. Now, applying Theorem 2, we get the desired result.

As an application, we considered an old construction of Leech [30] for squarefree words: consider the fixed point $\mathbf{L}$ of the 13-uniform morphism φ given by

$$0 \to 0121021201210$$
$$1 \to 1202102012021$$
$$2 \to 2010210120102$$

Using our method, we proved that the critical exponent of L is actually $\frac{15}{8}$. Furthermore, if x is a $\frac{15}{8}$ power occurring in $\mathbf{L}$, then $|x| = 15 \cdot 13^i$ for some $i \geq 0$. See [23].

6 Mirror Invariance

We can express the property that $\mathbf{a}$ is mirror-invariant as follows:

$$\forall N \geq 0, \ell \geq 1 \ \exists N' \geq 0 \ \mathbf{a}[N..N+\ell-1] = \mathbf{a}[N'..N'+\ell-1]^R,$$

which is the same as

$$\forall N \geq 0, \ell \geq 1 \ \exists N' \geq 0 \ \forall i, \ 0 \leq i < \ell \ \mathbf{a}[N+i] = \mathbf{a}[N'+\ell-i-1],$$

which can be easily checked by our method.

7 Recurrence

We can express the property that $\mathbf{a}$ is recurrent by saying that for each factor, and each integer M there exists a copy of that factor occurring at a position after M in $\mathbf{a}$. This corresponds to the following predicate:

$$\forall N, M \geq 0, \ell \geq 1 \ \exists M' \geq M \ \mathbf{a}[N..N+\ell-1] = \mathbf{a}[M'..M'+\ell-1].$$

An easy argument shows that an infinite word $\mathbf{a}$ is recurrent if and only if each finite factor occurs at least twice. This means that the following simpler predicate suffices:

$$\forall N \geq 0, \ell \geq 1 \ \exists M \neq N \ \mathbf{a}[N..N+\ell-1] = \mathbf{a}[M..M+\ell-1].$$

For uniform recurrence, we need to express the fact that two consecutive occurrences of each factor are separated by no more than C positions. Since there are only finitely many factors of each length, we can take C to be the maximum of the constants corresponding to each factor of that length. Thus we get the following predicate:

$$\forall \ell \geq 1 \ \exists C \geq 1 \ \forall N \geq 0 \ \exists M \text{ with } N < M \leq N+C \ \mathbf{a}[N..N+\ell-1] = \mathbf{a}[M..M+\ell-1].$$

For linear recurrence, we have to work harder, since at first glance knowing if there is a factor at distance $C\ell$ seems to require multiplication, which we cannot perform. Instead, we construct a DFA accepting the language

$$\begin{aligned} L = \{(n,\ell)_k \ : \ & \exists \ i \geq 0 \text{ s. t. } \forall \ j, 0 \leq j < \ell \text{ we have } a[i+j] = a[i+n+j] \text{ and} \\ & \nexists \ t, 0 < t < n \text{ s. t. } \forall \ j, 0 \leq j < \ell \text{ we have } a[i+j] = a[i+t+j] \ \}. \end{aligned}$$

Note that $(n,\ell)_k \in L$ iff there exists some factor of length ℓ for which the next occurrence is at distance n. Then linear recurrence corresponds to $\text{quo}_k(L) < \infty$, which we can test using Theorem 2.

8 Orbit Closure

The *orbit* of a sequence $\mathbf{a} = a_0 a_1 a_2 \cdots$ is the set of all sequences under the shift, that is, the set $\mathcal{S} = \{a_i a_{i+1} a_{i+2} \cdots : i \geq 0\}$. The *orbit closure* is the topological closure $\overline{\mathcal{S}}$ under the usual topology. In other words, a sequence $\mathbf{b} = b_0 b_1 b_2 \cdots$ is in $\overline{\mathcal{S}}$ if and only if, for each $j \geq 0$, the prefix $b_0 \cdots b_j$ is a factor of $\mathbf{a}$.

In general, the cardinality of the orbit closure is uncountable. On the other hand, the k-automatic sequences are countable. Hence most sequences in the orbit closure of a k-automatic sequence are not automatic themselves. However, we can use our method to show that two distinguished sequences, the lexicographically least and lexicographically greatest sequences in the orbit closure, are indeed k-automatic.

For example, Currie [18] showed that the lexicographically least sequence in the orbit closure of the Rudin-Shapiro sequence

$$\mathbf{r} = r_0 r_1 r_2 \cdots = \mathtt{0001001000011101000100101110000}\cdots$$

is $\mathtt{0}\mathbf{r}$, thus confirming a conjecture in [3].

9 Unbordered Factors

Recall that a word is *bordered* if it can be expressed as uvu for words u, v with u nonempty, and otherwise it is unbordered. Currie and Saari [19] proved that $\mathbf{t}$ has an unbordered factor of length n if $n \not\equiv 1 \pmod 6$. However, these are not the only lengths with an unbordered factor; for example,

$$\mathtt{0011010010110100110010110100101}$$

is an unbordered factor of length 31. We can express the property that $\mathbf{t}$ has an unbordered factor of length ℓ as follows:

$$\exists N \geq 0 \ \forall j, 1 \leq j \leq \ell/2 \ \mathbf{t}[N..N+j-1] \neq \mathbf{t}[N+\ell-j..N+\ell-1].$$

Using our technique, we can create a DFA to accept the base-2 representations of all such ℓ. Using our method, we were able to prove [23]

Theorem 3. *There is an unbordered factor of length ℓ in $\mathbf{t}$ if and only iff $(\ell)_2 \notin 1(01^*0)^*10^*1$.*

10 Enumeration

Up to now we have focused on deciding properties of automatic sequences. In many cases, however, we can actually count the number $T(n)$ of length-n factors of an automatic sequence having a particular property P. Here by "count" we mean, give an algorithm A to compute $T(n)$ efficiently, that is, in time bounded

by a polynomial in $\log n$. Although *finding* the algorithm A may not be particularly efficient (and indeed, has a "tower-of-2's" running time depending on the predicate to express P), but once we have it, we can compute $T(n)$ quickly.

One example is subword complexity, the number of distinct length-n factors of a sequence. To count these factors, we create a DFA M accepting the language

$$\begin{aligned}&\{(n,\ell)_k \ : \ \mathbf{a}[n..n+\ell-1] \text{ is the first occurrence of the given factor}\}\\ =\ &\{(n,\ell)_k \ : \ \forall n' < n \ \mathbf{a}[n..n+\ell-1] \neq \mathbf{a}[n'..n'+\ell-1]\}.\end{aligned}$$

Once we have M, the number of ℓ corresponding to a given n is just the subword complexity. It then turns out [16] that this number can be expressed as the product

$$vM_{a_1}\cdots M_{a_i}w$$

for suitable vectors v, w and matrices $M_0, \ldots, M_{k-1}$, where $a_1 \cdots a_i$ is the base-k representation of n, thus giving an efficient algorithm to compute it.

In a similar way, we can handle

- palindrome complexity (the number of distinct length-n palindromic factors) [1];
- the number of words whose reversals are also factors;
- the number of squares of a given length;
- the number of unbordered factors [24];

and so forth.

For this last example, the number $f(n)$ of unbordered factors of length n, we carried out an explicit computation for the Thue-Morse sequence. The resulting computation allowed us to prove that $f(n) \leq n$ for $n \geq 4$ and $f(n) = n$ infinitely often [24].

11 Synchronization

Sometimes even more is true: we can build a DFA to accept the language

$$\{(n, T(n))_k \ : \ n \geq 0\},$$

where $T(n)$ counts some interesting property about an automatic sequence. In this case we say, following Carpi [15,13,14], that the function T is k-synchronized. When a function T is k-synchronized, we have $T(n) = O(n)$ and further, we can compute it in $O(\log n)$ time [25].

Many enumerations about automatic sequences are now known to be k-synchronized. These include

- the separator sequence [15];
- the repetitivity index [13];
- the recurrence function [16];
- the appearance function [16];

- the subword complexity function [25];
- the number of factors of length n that are primitive [25].

Here is a novel example of synchronization. Blondin-Massé et al. studied the *longest palindromic suffix* of a finite word w, defined to be the unique longest word x such that $x = x^R$ and there exists y such that $w = yx$; it is denoted $\mathrm{LPS}(w)$. Given an infinite word $\mathbf{a} = a_0 a_1 \cdots$, they defined the related function $\mathrm{LLPS}_{\mathbf{a}}(n) = |\mathrm{LPS}(\mathbf{a}[0..n])|$, which measures the length of the longest palindromic suffix of each prefix.

We can see that $\mathrm{LLPS}_{\mathbf{a}}(n)$ is k-synchronized, as we can build an automaton to accept

$$\{(n,i)_k : \mathbf{a}[n-i+1..n] = \mathbf{a}[n-i+1..n]^R \text{ and } \exists j, 0 \le j \le n-i \ \mathbf{a}[j..n] \ne \mathbf{a}[j..n]^R\}.$$

Blondin-Massé et al. also studied a related function, given by

$$H_{\mathbf{a}}(n) = \begin{cases} \mathrm{LLPS}_{\mathbf{a}}(n), & \text{if } \mathrm{LPS}(\mathbf{a}[0..n]) \text{ does not occur in } \mathbf{a}[0..n-1]\ ; \\ 0, & \text{otherwise.} \end{cases}$$

This sequence $H_{\mathbf{a}}$ is also k-synchronized, as we can express it as

$$\begin{aligned} \{(n,i)_k : \mathbf{a}[n-i+1..n] &= \mathbf{a}[n-i+1..n]^R \text{ and} \\ \exists j, 0 \le j \le n-i \quad \mathbf{a}[j..n] &\ne \mathbf{a}[j..n]^R \text{ and} \\ \forall \ell, 0 \le \ell \le n-i \quad \mathbf{a}[\ell..\ell+i-1] &\ne \mathbf{a}[n-i+1..n]\} \\ \cup \{(n,0)_k : \mathbf{a}[n-i+1..n] &= \mathbf{a}[n-i+1..n]^R \text{ and} \\ \exists j, 0 \le j \le n-i \quad \mathbf{a}[j..n] &\ne \mathbf{a}[j..n]^R \text{ and} \\ \exists \ell, 0 \le \ell \le n-i \quad \mathbf{a}[\ell..\ell+i-1] &= \mathbf{a}[n-i+1..n]\}. \end{aligned}$$

12 Paperfolding

Up to now we have only applied our decision procedure to a single automatic sequence. Sometimes, however, it is desirable to talk about the properties of a *family* of such sequences. A famous example of such a family is the set of *paperfolding sequences*. Given a sequence of *unfolding instructions* $\mathbf{f} = f_0 f_1 f_2 \cdots$ over the alphabet $\{0,1\}$, the paperfolding sequence $\mathbf{P_f} = p_1 p_2 p_3 \cdots$ is defined as the limit of the finite sequences given by

$$\begin{aligned} x_0 &= f_0 \\ x_{n+1} &= x_i \ f_i \ \overline{x_i^R}, \end{aligned}$$

where, as before, $\overline{0} = 1$ and $\overline{1} = 0$. The *regular paperfolding sequence*

$$\mathtt{001001100011011}\cdots$$

corresponds to the unfolding instructions $\mathtt{000}\cdots$.

It turns out that many properties of these sequence are also decidable using our method. The key observation is due to Luke Schaeffer: a known formula to compute the n'th term of a paperfolding sequence [20] can be implemented by the following automaton of 5 states (depicted below in Figure 4) that takes, as input, a prefix of a sequence of unfolding instructions in parallel with the base-2 expansion of n (starting with the least significant digit), and computes the n'th term of the corresponding paperfolding sequence.

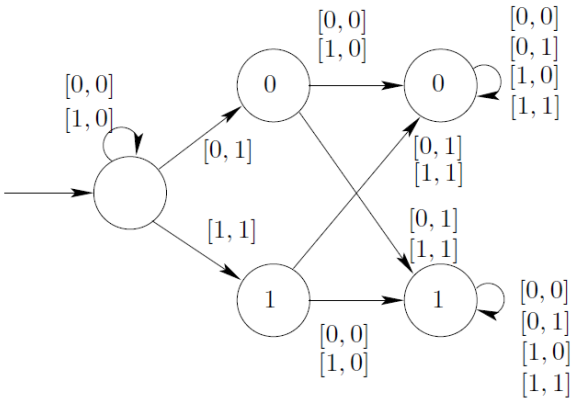


Fig. 4. Automaton for the paperfolding sequences

This makes it possible to prove many of the known results about paperfolding sequences, and some new ones, in a purely mechanical fashion. We just mention one new result, answering a question of Narad Rampersad [26]:

Theorem 4. *If* $\mathbf{f} = f_0 f_1 f_2 \cdots$ *and* $\mathbf{g} = g_0 g_1 g_2 \cdots$ *are two different sequences of unfolding instructions, and the smallest index where they differ is* $f_i = g_i$*, then* $P_{\mathbf{f}}$ *and* $P_{\mathbf{g}}$ *have no factors of length* $\geq 14 \cdot 2^i$ *in common.*

13 Implementation

As mentioned previously, the extraordinary upper bound on the running time of the decision procedure means that care has to be taken during the implementation. Dane Henshall and my master's student Daniel Goč independently wrote code that takes a description of an automatic sequence and a predicate as input and translates the predicate to the appropriate automaton. In Goč's algorithm, DFA minimization is done using Brzozowski's algorithm, which often seems to outperform the usual methods. With these implementations we have been able to find new machine proofs of many old results and also some new ones.

14 Inexpressible Predicates

It is natural to wonder if other kinds of properties of automatic sequences are solvable using our method. One natural candidate that seems difficult is testing abelian squarefreeness. We say that a nonempty word x is an *abelian square* if it of the form ww' with $|w| = |w'|$ and w' a permutation of w. (An example in English is the word `reappear`, and three examples in Russian are крекер ("cracker") отлетело ("(it) flew away") and ротатора ("(of) rotator", genitive case).)

Recently my student Luke Schaeffer has shown that the predicate for abelian squarefreeness is indeed inexpressible, in general [41]. To do so, he considers the regular paperfolding sequence

$$\mathbf{f} = f_1 f_2 f_3 \cdots = 00100110001101100010011100110 11 \cdots,$$

which is 2-automatic, and then the language

$$L = \{(n,i)_2 \ : \ \mathbf{f}[i..i+n-1] \text{ is a permutation of } \mathbf{f}[i+n..i+2n-1]\}.$$

If abelian squarefreeness were expressible, then L would be regular, but he shows it is not [41].

15 Open Questions

There are still many interesting questions that are unresolved. For example, it is known that, given a DFA M, we can decide if $\mathrm{quo}_k(L(M)) \subseteq \mathbb{N}$ [40]. However, the following related problems are still open:

Open Question 1. *Are any of the following problems recursively solvable? Given a DFA M accepting $L \subseteq (\Sigma_k \times \Sigma_k)^*$,*

(a) Does there exist $x \in L$ such that $\mathrm{quo}_k(x) \in \mathbb{N}$?
(b) Do there exist infinitely many $x \in L$ such that $\mathrm{quo}_k(x) \in \mathbb{N}$?
(c) Is there an infinite subset $S \subseteq \mathbb{N}$ such that $S \subseteq \mathrm{quo}_k(L)$?

Similarly, if L is represented by a pushdown automaton instead of a DFA, we can ask.

Open Question 2. *Is $\sup \mathrm{quo}_k(L)$ computable for context-free languages L?*

References

1. Allouche, J.P., Baake, M., Cassaigne, J., Damanik, D.: Palindrome complexity. Theoret. Comput. Sci. 292, 9–31 (2003)
2. Allouche, J.P., Currie, J., Shallit, J.: Extremal infinite overlap-free binary words. European J. Combinatorics 5, R27 (1998), `http://www.combinatorics.org/ojs/index.php/eljc/article/view/v5i1r27`

3. Allouche, J.P., Rampersad, N., Shallit, J.: Periodicity, repetitions, and orbits of an automatic sequence. Theoret. Comput. Sci. 410, 2795–2803 (2009)
4. Allouche, J.P., Shallit, J.: The ubiquitous Prouhet-Thue-Morse sequence. In: Ding, C., Helleseth, T., Niederreiter, H. (eds.) Proceedings of Sequences and Their Applications, SETA 1998, pp. 1–16. Springer (1999)
5. Allouche, J.P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press (2003)
6. Berstel, J.: Axel Thue's work on repetitions in words. In: Leroux, P., Reutenauer, C. (eds.) Séries Formelles et Combinatoire Algébrique, Publications du LaCim, Université du Québec à Montréal, vol. 11, pp. 65–80 (1992)
7. Blondin Massé, A., Brlek, S., Garon, A., Labbé, S.: Combinatorial properties of f-palindromes in the Thue-Morse sequence. Pure Math. Appl. 19(2-3), 39–52 (2008), http://www.mat.unisi.it/newsito/puma/public_html/19_2_3/4.pdf
8. Brlek, S.: Enumeration of factors in the Thue-Morse word. Disc. Appl. Math. 24, 83–96 (1989)
9. Brown, S., Rampersad, N., Shallit, J., Vasiga, T.: Squares and overlaps in the Thue-Morse sequence and some variants. RAIRO Inform. Théor. App. 40, 473–484 (2006)
10. Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bull. Belgian Math. Soc. 1, 191–238 (1994); Corrigendum, Bull. Belg. Math. Soc. 1, 577 (1994)
11. Büchi, J.R.: Weak secord-order arithmetic and finite automata. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 6, 66–92 (1960); reprinted in Mac Lane, S., Siefkes, D. (eds.): The Collected Works of J. Richard Büchi, pp. 398–424. Springer (1990)
12. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.), pp. 1–11. Stanford University Press (1962)
13. Carpi, A., D'Alonzo, V.: On the repetitivity index of infinite words. Internat. J. Algebra Comput. 19, 145–158 (2009)
14. Carpi, A., D'Alonzo, V.: On factors of synchronized sequences. Theoret. Comput. Sci. 411, 3932–3937 (2010)
15. Carpi, A., Maggi, C.: On synchronized sequences and their separators. RAIRO Inform. Théor. App. 35, 513–524 (2001)
16. Charlier, E., Rampersad, N., Shallit, J.: Enumeration and decidable properties of automatic sequences. Internat. J. Found. Comp. Sci. 23, 1035–1066 (2012)
17. Cobham, A.: Uniform tag sequences. Math. Systems Theory 6, 164–192 (1972)
18. Currie, J.: Lexicographically least words in the orbit closure of the Rudin-Shapiro word. Theoret. Comput. Sci. 41, 4742–4746 (2011)
19. Currie, J.D., Saari, K.: Least periods of factors of infinite words. RAIRO Inform. Théor. App. 43, 165–178 (2009)
20. Dekking, F.M., Mendès France, M., van der Poorten, A.J.: Folds! Math. Intelligencer 4, 130–138, 173–181, 190–195 (1982); erratum 5, 5 (1983)
21. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Amer. Math. Soc. 98, 21–51 (1961)
22. Euwe, M.: Mengentheoretische Betrachtungen über das Schachspiel. Proc. Konin. Akad. Wetenschappen, Amsterdam 32, 633–642 (1929)
23. Goč, D., Henshall, D., Shallit, J.: Automatic theorem-proving in combinatorics on words. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 180–191. Springer, Heidelberg (2012)

24. Goč, D., Mousavi, H., Shallit, J.: On the number of unbordered factors. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 299–310. Springer, Heidelberg (2013)
25. Goč, D., Schaeffer, L., Shallit, J.: The subword complexity of k-automatic sequences is k-synchronized (2012) (submitted), preprint available at `http://arxiv.org/abs/1206.5352`
26. Goč, D., Schaeffer, L., Shallit, J.: A new approach to the paperfolding sequences (manuscript in preparation, 2013)
27. Hodgson, B.: Décidabilité par automate fini. Ann. Sci. Math. Québec 7, 39–57 (1983)
28. Honkala, J.: A decision method for the recognizability of sets defined by number systems. RAIRO Inform. Théor. App. 20, 395–403 (1986)
29. Klaedtke, F.: Bounds on the automata size for Presburger arithmetic. ACM Trans. Comput. Logic 9(2), article 11 (March 2008), `http://doi.acm.org/10.1145/1342991.1342995`
30. Leech, J.: A problem on strings of beads. Math. Gazette 41, 277–278 (1957)
31. Leroux, J.: A polynomial time Presburger criterion and synthesis for number decision diagrams. In: 20th IEEE Symposium on Logic in Computer Science (LICS 2005), pp. 147–156. IEEE Press (2005)
32. Luca, A.D., Varricchio, S.: Some combinatorial properties of the Thue-Morse sequence and a problem in semigroups. Theoret. Comput. Sci. 63, 333–348 (1989)
33. Marsault, V., Sakarovitch, J.: Ultimate periodicity of b-recognisable sets: a quasi-linear procedure (2013), preprint available at `http://arxiv.org/abs/1301.2691`
34. Morse, M.: Recurrent geodesics on a surface of negative curvature. Trans. Amer. Math. Soc. 22, 84–100 (1921)
35. Morse, M., Hedlund, G.A.: Symbolic dynamics. Amer. J. Math. 60, 815–866 (1938)
36. Pansiot, J.J.: The Morse sequence and iterated morphisms. Inform. Process. Lett. 12, 68–70 (1981)
37. Presburger, M.: Über die Volständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Sparawozdanie z I Kongresu Matematyków Krajów Slowianskich, Warsaw, pp. 92–101 (1929)
38. Presburger, M.: On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. Hist. Phil. Logic 12, 225–233 (1991)
39. Prouhet, E.: Mémoire sur quelques relations entre les puissances des nombres. C. R. Acad. Sci. Paris 33, 225 (1851)
40. Rowland, E., Shallit, J.: k-automatic sets of rational numbers. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 490–501. Springer, Heidelberg (2012)
41. Schaeffer, L.: Abelian powers in automatic sequences are not always automatic. Talk at CanadaDAM 2013 Conference, St. John's, Newfoundland (June 2013)
42. Schaeffer, L., Shallit, J.: The critical exponent is computable for automatic sequences. Int. J. Found. Comput. Sci. (2012) (to appear)
43. Shallit, J.: The critical exponent is computable for automatic sequences. In: Ambroz, P., Holub, S., Másaková, Z. (eds.) Proceedings 8th International Conference Words 2011. Elect. Proc. Theor. Comput. Sci, vol. 63, pp. 231–239 (2011)
44. Sipser, M.: Introduction to the Theory of Computation. Cengage Learning, 3rd edn. (2013)

45. Thue, A.: Über unendliche Zeichenreihen. Norske vid. Selsk. Skr. Mat. Nat. Kl. 7, 1–22 (1906); reprinted in Nagell, T. (ed.): Selected Mathematical Papers of Axel Thue, Universitetsforlaget, Oslo, pp. 139–158 (1977)
46. Thue, A.: Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. Norske vid. Selsk. Skr. Mat. Nat. Kl 1, 1–67 (1912); reprinted in Nagell, T. (ed.): Selected Mathematical Papers of Axel Thue, Universitetsforlaget, Oslo, pp. 413–478 (1977)
47. Wah, A., Picciotto, H.: Algebra: Themes, Tools, Concepts. Creative Publications, Mountain View (1994), http://www.mathedpage.org/attc/attc.html

Walking on Data Words*

Amaldev Manuel, Anca Muscholl, and Gabriele Puppis

LaBRI, University of Bordeaux, France

Abstract. We see data words as sequences of letters with additional edges that connect pairs of positions carrying the same data value. We consider a natural model of automaton walking on data words, called Data Walking Automaton, and study its closure properties, expressiveness, and the complexity of paradigmatic problems. We prove that deterministic DWA are strictly included in non-deterministic DWA, that the former subclass is closed under all boolean operations, and that the latter class enjoys a decidable containment problem.

1 Introduction

Data words arose as a generalization of strings over finite alphabets, where the term 'data' denotes the presence of elements from an infinite domain. Formally, data words are modelled as finite sequences of elements chosen from a set of the form $\Sigma \times \mathbb{D}$, where Σ is a finite alphabet and $\mathbb{D}$ is an infinite alphabet. Elements of Σ are called *letters*, while elements of $\mathbb{D}$ are called *data values*. Sets of data words are called *data languages*.

It comes natural to investigate reasonable mechanisms (e.g., automata, logics, algebras) for specifying languages of data words. Some desirable features of such mechanisms are the decidability of the paradigmatic problems (i.e., emptiness, universality, containment) and effective closures of the recognized languages under the usual boolean operations and projections. The often-used idea is to enhance a finite state machine with data structures to provide some ability to handle data values. Examples of these structures include registers to store data values [5,6], pebbles to mark positions in the data word [7], hash tables to store partitions of the data domain [1]. In [4] the authors introduced the novel idea of composing a finite state transducer and a finite state automaton to obtain a so-called Data Automaton. Remarkably, the resulting class of automata captures the data languages definable in two-variable first-order logic over data words. For all models except Pebble Automata and Two-way Register Automata the non-emptiness problem is decidable; universality and, by extension, equivalence and inclusion problems are undecidable for all non-deterministic models.

In this work we consider data words as sequences of letters with additional edges that connect pairs of positions carrying the same data value. This idea is

* The research leading to these results has received funding from the ANR project 2010 BLAN 0202 01 FREC and from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 259454.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 64–75, 2013.

consistent with the fact that as far as a data word is concerned the actual data value at a position is not relevant, but only the relative equality and disequality of positions with respect to data values. It is also worth noting that none of the above automaton models makes any distinction between permutations of the data values inside data words. Our model of automaton, called Data Walking Automaton, is naturally two-way: it can roughly be seen as a finite state device whose head moves along successor and predecessor positions, as well as along the edges that connect any position to the closest one having the same data value, either to the right or to the left. Remarkably, emptiness, universality, and containment problems are decidable for Data Walking Automata. Our automata capture, up to letter-to-letter renamings, all data languages recognized by Data Automata. The deterministic subclass of Data Walking Automata is shown to be closed under all boolean operations (closure under complementation is not immediate since the machines may loop). Finally, we deduce from results about Tree Walking Automata [2,3] that deterministic Data Walking Automata are strictly less powerful than non-deterministic Data Walking Automata, which in turn are subsumed by Data Automata.

2 Preliminaries

We use $[n]$ to denote the subset $\{1, ..., n\}$ of the natural numbers. Given a data word $w = (a_1, d_1) \ ... \ (a_n, d_n)$, a *class* of w is a maximal set of positions with identical data value. The set of classes of w forms a partition of the set of positions and is naturally defined by the equivalence relation $i \sim j$ iff $d_i = d_j$.

The *global successor* and *global predecessor* of a position i in a data word w are the positions $i+1$ and $i-1$ (if they exist). The *class successor* of a position i is the least position after i in its class (if it exists) and is denoted by $i \oplus 1$. The *class predecessor* of a position i is the greatest position before i in its class (if it exists) and is denoted by $i \ominus 1$. The global and class successors of a position are collectively called successors, and similarly for the predecessors.

Using the above definitions we can identify any data word $w \in (\Sigma \times \mathbb{D})^*$ with a directed graph whose vertices are the positions of w, each one labelled with a letter from Σ, and whose edges are given by the successors and predecessor functions $+1$, -1, $\oplus 1$, $\ominus 1$. This graph is represented in space $\Theta(|w|)$.

Local Types. Given a data word w and a position i in it, we introduce local types $\overrightarrow{\mathsf{type}}_w(i)$ and $\overleftarrow{\mathsf{type}}_w(i)$ to describe if each of the successors and predecessors of i exist and whether they coincide. Formally, when considering the successors of a position i, four scenarios are possible: (1) i is the rightmost position and neither the global successor nor the class successor are defined (for short we denote this by $\overrightarrow{\mathsf{type}}_w(i) = \mathsf{max}$), (2) i is not the rightmost position, but it is the greatest in its class, in which case the global successor exists but not the class successor ($\overrightarrow{\mathsf{type}}_w(i) = \mathsf{cmax}$), (3) both global and class successors of i are defined and they coincide, i.e. $i + 1 = i \oplus 1$ ($\overrightarrow{\mathsf{type}}_w(i) = \mathsf{1succ}$), or (4) both successors of i are defined and they diverge, i.e. $i + 1 \neq i \oplus 1$ ($\overrightarrow{\mathsf{type}}_w(i) = \mathsf{2succ}$). We define $\overrightarrow{\mathsf{Types}} = \{\mathsf{max}, \mathsf{cmax}, \mathsf{1succ}, \mathsf{2succ}\}$ to be the set of possible right types

of positions of data words. The analogous scenarios for the predecessors of i are determined by the left type $\overleftarrow{\mathsf{type}}_w(i) \in \overleftarrow{\mathsf{Types}} = \{\mathsf{min}, \mathsf{cmin}, \mathsf{1pred}, \mathsf{2pred}\}$. Finally, we define $\mathsf{type}_w(i) = \big(\overleftarrow{\mathsf{type}}_w(i), \overrightarrow{\mathsf{type}}_w(i)\big) \in \mathsf{Types} = \overleftarrow{\mathsf{Types}} \times \overrightarrow{\mathsf{Types}}$.

Class-Memory Automata. We depend on Data Automata [4] for our decidability results. For convenience we use an equivalent model called Class-Memory Automata [1]. Class-Memory Automata are finite state automata enhanced with memory-functions from $\mathbb{D}$ to a fixed finite set $[k]$. On encountering a pair (a, d), a transition is non-deterministically chosen from a set that may depend on the current state of the automaton, the memory-value $f(d)$, and the input letter a. When a transition on (a, d) is executed, the current state and the memory-value of d are updated. Below we give a formal definition of Class-Memory Automata and observe that this model is similar to that of Tiling Automata [9].

A *Class-Memory Automaton* (*CMA*) is formally defined as a tuple $\mathcal{A} = (Q, k, \Sigma, \Delta, I, F, K)$, where Q is the finite set of states, $[k]$ is the set of memory-values, Σ is the finite alphabet, $\Delta \subseteq Q \times \Sigma \times (\{0\} \cup [k]) \times Q \times [k]$ is the transition relation, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of accepting states, and $K \subseteq [k]$ is the set of accepting memory-values. Configurations are pairs (q, f), with $q \in Q$ and f partial function from $\mathbb{D}$ to $[k]$ (for the sake of brevity, we write $f(d) = 0$ whenever f is undefined on d). Transitions are of the form $(q, f) \xrightarrow{(a,d)} (q', f')$, with $(q, a, f(d), q', h) \in \Delta$, $f'(d) = h$, and $f'(e) = f(e)$ for all $e \in \mathbb{D} \setminus \{d\}$. Sequences of transitions are called runs. The initial configurations are the pairs (q_0, f_0), with $q_0 \in I$ and $f_0(d) = 0$ for all $d \in \mathbb{D}$; the final configurations are the pairs (q, f), with $q \in F$ and $f(d) \in \{0\} \cup K$ for all $d \in \mathbb{D}$. The recognized language $\mathscr{L}(\mathcal{A})$ contains all data words $w = (a_1, d_1) \ldots (a_n, d_n) \in (\Sigma \times \mathbb{D})^*$ that admit runs of the form $(q_0, f_0) \xrightarrow{(a_1,d_1)} \ldots \xrightarrow{(a_n,d_n)} (q_n, f_n)$, starting in an initial configuration and ending in a final configuration.

It is known that CMA-recognizable languages are effectively closed under union, intersection, letter-to-letter renaming, but not under complementation. Their emptiness problem is decidable and reduces to reachability in vector addition systems, which is not known to be of elementary complexity. Inclusion and universality problems are undecidable. The following result, paired with closure under intersection, allows us to assume that the information about local types of positions of a data word is available to CMA:

Proposition 1 (Björklund and Schwentick [1]). *Let L be the set of all data words $w \in (\Sigma \times \mathsf{Types} \times \mathbb{D})^*$ such that, for all positions i, $w(i) = (a, \tau, d)$ implies $\tau = \mathsf{type}_w(i)$. The language L is recognized by a CMA.*

Tiling Automata. We conclude this preliminary section by observing that CMA are similar to the model of *Tiling Automata on directed graphs* [9], restricted to a subclass of graphs, namely data words. We fix a finite set Γ of colours to be used in tiles. Given a type $\tau = (\overleftarrow{\tau}, \overrightarrow{\tau}) \in \mathsf{Types}$, a τ-tile associates colours to each position and to its neighbours (as specified by the type). For instance, a $(\mathsf{1pred}, \mathsf{2pred})$-tile is a tuple of the form $t = (\gamma_0, \gamma_{-1}, \gamma_{+1}, \gamma_{\oplus 1}) \in \Gamma^4$ such that, when associated to a position i in w with type $\mathsf{type}_w(i) = (\mathsf{1pred}, \mathsf{2pred})$, implies

that the colour of i is γ_0, the colour of $i-1$ $(= i \ominus 1)$ is γ_{-1}, the colour of $i+1$ is γ_{+1}, and the colour of $i \oplus 1$ is $\gamma_{\oplus 1}$. A *Tiling Automaton* consists of a family $\mathcal{T} = (T_{a,\tau})_{a \in \Sigma, \tau \in \mathsf{Types}}$ of τ-tiles for each letter $a \in \Sigma$ and each type $\tau \in \mathsf{Types}$. A tiling by $\mathcal{T}$ of a data word $w = (a_1, d_1) \ldots (a_n, d_n)$ is a function $\widetilde{w} : [n] \to \Gamma$ such that, for all types τ and all positions i of type τ, the τ-tile that is formed by i and its neighbours belongs to the set $T_{a_i,\tau}$. The language recognized by the Tiling Automaton $\mathcal{T}$ consists of all data words that admit a valid tiling by $\mathcal{T}$.

The following result depends on the fact that CMA can compute the types of the positions in a data word and is obtained by simple translations of automata:

Proposition 2. *CMA and Tiling Automata on data words are equivalent.*

3 Automata Walking on Data Words

An automaton walking on data words is a finite state acceptor that processes a data word by moving its head along the successors and predecessors of positions. We let $\mathsf{Axis} = \{0, +1, \oplus 1, -1, \ominus 1\}$ be the set of the five possible directions of navigation in a data word (0 stands for 'stay in the current position').

Definition 1. *A* Data Walking Automaton *(*DWA *for short) is defined as a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, where Q is the finite set of states, Σ is the finite alphabet, $\Delta \subseteq Q \times \Sigma \times \mathsf{Types} \times Q \times \mathsf{Axis}$ is the transition relation, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states.*

Let $w = (a_1, d_1) \ldots (a_n, d_n) \in (\Sigma \times \mathbb{D})^*$ be a data word. Given $i \in [n]$ and $\alpha \in \mathsf{Axis}$, we denote by $\alpha(i)$ the position that is reached from i by following the axis α (for instance, if $\alpha = 0$ then $\alpha(i) = i$, if $\alpha = \oplus 1$ then $\alpha(i) = i \oplus 1$, provided that i is not the last element in its class). A configuration of $\mathcal{A}$ is a pair consisting of a state $q \in Q$ and a position $i \in [n]$. A transition is a tuple of the form $(p, i) \xrightarrow{w} (q, j)$ such that $(p, a_i, \tau, q, \alpha) \in \Delta$, with $\tau = \mathsf{type}_w(i)$ and $j = \alpha(i)$. The initial configurations are the pairs (q_0, i_0), with $q_0 \in I$ and $i_0 = 1$. The halting configurations are those pairs (q, i) on which no transition is enabled; such configurations are said to be *final* if $q \in F$. The language $\mathscr{L}(\mathcal{A})$ recognized by $\mathcal{A}$ is the set of all data words $w \in (\Sigma \times \mathbb{D})^*$ that admit a run of $\mathcal{A}$ that starts in an initial configuration and halts in a final configuration.

We will also consider *deterministic* versions of DWA, in which the set I of initial states is a singleton and the transition relation Δ can be seen as a partial function from $Q \times \Sigma \times \mathsf{Types}$ to $Q \times \mathsf{Axis}$.

Example 1. Let L_1 be the set of all data words that contain at most one occurrence of each data value (this language is equally defined by the formula $\forall x \forall y \; x \sim y \;\rightarrow\; x = y$). A deterministic DWA can recognize L_1 by reading the input data word from left to right (along axis $+1$) and by checking that all positions except the last one have type $(\mathsf{cmin}, \mathsf{cmax})$. When a position with type $(\mathsf{cmin}, \mathsf{max})$ or $(\mathsf{min}, \mathsf{max})$ is reached, the machine halts in an accepting state.

Example 2. Let L_2 be the set of all data words in which every occurrence of a is followed by an occurrence of b in the same class (this is expressed by the formula $\forall x\ a(x) \rightarrow \exists y\ b(y) \wedge x < y \wedge x \sim y$). A deterministic DWA can recognize L_2 by scanning the input data word along the axis $+1$. On each position i with left type cmin, the machine starts a sub-computation that scans the entire class of i along the axis $\oplus 1$, and verifies that every a is followed by a b. The sub-computation terminates when a position with right type cmax is reached, after which the machines traverses back the class, up to the position i with left type cmin, and then resumes the main computation from the successor $i+1$. Intuitively, the automaton traverses the data word from left to right in a 'class-first' manner.

Example 3. Our last example deals with the set L_3 of all data words in which every occurrence of a is followed by an occurrence of b that is *not* in the same class (this is expressed by the formula $\forall x\ a(x) \rightarrow \exists y\ b(y) \wedge x < y \wedge x \nsim y$). This language is recognized by a deterministic DWA, although not in an obvious way. Fix a data word w. It is easy to see that $w \in L_3$ iff one following cases holds:

1. there is no occurrence of a in w,
2. w contains a rightmost occurrence of b, say in position ℓ_b, and all occurrences of a are before ℓ_b; in addition, we require that either the class of ℓ_b does not contain an a, or the class of ℓ_b contains a rightmost occurrence of a, say in position ℓ_a, and another b appears after ℓ_a but outside the class of ℓ_b.

It is easy to construct a deterministic DWA that verifies the first case. We show how to verify the second case. For this, the automaton reaches the rightmost position $|w|$ and searches backward, following the axis -1, the first occurrence of b: this puts the head of the automaton in position ℓ_b. From position ℓ_b the automaton searches along the axis $\ominus 1$ an occurrence of a. If no occurrence of a is found before seeing the left type cmin, then the automaton halts by accepting. Otherwise, as soon as a is seen (necessarily at position ℓ_a), a second phase starts that tries to find another occurrence of b after ℓ_a and outside the class of ℓ_b (we call such an occurrence a *b-witness*). To do this, the automaton moves along the axis $+1$ until it sees a b, say at position i. After that, it scans the class of i along the axis $\oplus 1$. If the right type cmax is seen before b, this means that the class of i does not contain a b: in this case, the automaton goes back to position i (which is now the first position along axis $\ominus 1$ that contains a b) and accepts iff b is seen along axis $+1$ (thanks to the previous test, that occurrence of b must be outside the class of ℓ_b and hence a b-witness). Otherwise, if a b is seen in position j before the right type cmax, this means that the class of i contains a b: in this case, the automaton backtracks to position i and resumes the search for another occurrence of b to the right of i (note that if i is a b-witness, then j is also a b-witness, which will be eventually processed by the automaton).

Closure Properties. Closure of non-deterministic DWA under union is easily shown by taking a disjoint union of the state space of the two automata. Closure

under intersection is shown by assuming that one of the two automata accepts only by halting in the leftmost position and by coupling its final states with the initial states of the other automaton.

Closure properties for deterministic DWA rely on the fact that one can remove loops from deterministic computations. The proof of the following result is an adaptation of Sipser's construction for eliminating loops on configurations of deterministic space-bounded Turing machines [8].

Proposition 3. *Given a deterministic DWA $\mathcal{A}$, one can construct a deterministic DWA $\mathcal{A}'$ equivalent to $\mathcal{A}$ that always halts.*

Proposition 4. *Non-deterministic DWA are effectively closed under union and intersection. Deterministic DWA are effectively closed under union, intersection, and complementation.*

4 Deterministic vs Non-deterministic DWA

We aim at proving the following separation results:

Theorem 1. *There exist data languages recognized by non-deterministic DWA that cannot be recognized by deterministic DWA. There also exist data languages recognized by CMA that cannot be recognized by non-deterministic DWA.*

Intuitively, the proof of the theorem exploits the fact that one can encode binary trees by suitable data words and think of deterministic DWA (resp. non-deterministic DWA, CMA) as deterministic Tree Walking Automata (resp. non-deterministic Tree Walking Automata, classical bottom-up tree automata). One can then use the results from [2,3] that show that (i) Tree Walking Automata cannot be determinized and (ii) Tree Walking Automata, even non-deterministic ones, cannot recognize all regular tree languages. We develop these ideas below.

Encodings of trees. Hereafter we use the term 'tree' (resp. 'forest') to denote a generic finite tree (resp. forest) where each node is labelled with a symbol from a finite alphabet Σ and has either 0 or 2 children. To encode trees/forests by data words, we will represent the node-to-left-child and the node-to-right-child relationships via the predecessor functions $\ominus 1$ and -1, respectively. In particular, a *leaf* will correspond to a position of the data word with *no class predecessor*, an *internal node* will correspond to a position where *both class and global predecessors* are defined (and are distinct), and a *root* will be represented either by the *rightmost position* in the word or by a position with no class successor that is immediately followed by a position with no class predecessor. As an example, given pairwise different data values d, e, f, g, the complete binary tree of height 2 can be encoded by the following data word:

$$w \;=\; d \quad e \leftarrow d \quad f \quad g \leftarrow f \leftarrow d$$

(to ease the understanding of the example, we drew only the instances of the predecessor functions $\ominus 1$ and -1 that represent left and right edges of the tree).

A formal definition of encoding of a tree/forest follows:

Definition 2. *We say that a data word* $w \in (\Sigma \times \mathbb{D})^+$ *is a* forest encoding *if there is no position* i *such that* $\overleftarrow{\mathsf{type}}_w(i) = \mathsf{1pred}$ *and no pair of consecutive positions* i *and* $i+1$ *such that* $\overrightarrow{\mathsf{type}}_w(i) = \mathsf{2succ} \wedge \overleftarrow{\mathsf{type}}_w(i+1) = \mathsf{2pred}$.

Given a forest encoding w*, we denote by* $\mathsf{forest}(w)$ *the directed binary forest that has for nodes the positions of* w*, labelled over* Σ*, and such that:*

- *if* $\overleftarrow{\mathsf{type}}_w(i) \in \{\mathsf{min}, \mathsf{cmin}\}$*, then* i *is a leaf in* $\mathsf{forest}(w)$*,*
- *if* $\overleftarrow{\mathsf{type}}_w(i) = \mathsf{2pred}$*, then* $i \ominus 1$ *and* $i-1$ *are left and right children of* i*,*
- *if* $\overrightarrow{\mathsf{type}}_w(i) = \mathsf{max}$ *or* $\overrightarrow{\mathsf{type}}_w(i) = \mathsf{cmax} \wedge \overleftarrow{\mathsf{type}}_w(i+1) = \mathsf{cmin}$*, then* i *is a root*

($\mathsf{forest}(w)$ *is clearly an acyclic directed graph; the fact that each node* i *has at most one parent follows from a case distinction based on the types of* i *and* $i+1$*).*

We let $\mathsf{tree}(w) = \mathsf{forest}(w)$ *if the forest encoded by* w *contains a single root, namely, it is a tree, otherwise, we let* $\mathsf{tree}(w)$ *be undefined.*

We remark that there exist several encodings of the same tree/forest that are not isomorphic even up to permutations of the data values. For instance, the two data words below encode the same complete binary tree of height 2:

Among all possible encodings of a tree/forest, we identify special ones, called *canonical encodings*, in which the nodes are listed following the *post-order* visit. Each tree t has a unique canonical encoding up to permutations of the data values, which we denote by $\mathsf{enc}(t)$.

Separations of Tree Automata. We briefly recall the definition of a tree walking automaton and the separation results from [2,3]. In a way similar to DWA, we first introduce local types of nodes inside trees. These can be seen as pairs of labels from the finite sets $\mathsf{Types}^{\downarrow} = \{\mathsf{leaf}, \mathsf{internal}\}$ and $\mathsf{Types}^{\uparrow} = \{\mathsf{root}, \mathsf{leftchild}, \mathsf{rightchild}\}$, and they allow us to distinguish between a leaf and an internal node as well as between a root, a left child, and a right child. We envisage a set $\mathsf{TAxis} = \{0, \uparrow, \swarrow, \searrow\}$ of four navigational directions inside a tree: 0 is for staying in the current node, $\uparrow$ is for moving to the parent, $\swarrow$ is for moving to the left child, and $\searrow$ is for moving to the right child. A *non-deterministic Tree Walking Automaton* (*TWA*) is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, where Σ is the finite alphabet, Q is the final set of states, $\Delta \subseteq Q \times \Sigma \times \mathsf{Types}^{\downarrow} \times \mathsf{Types}^{\uparrow} \times Q \times \mathsf{TAxis}$ is the transition relation, and $I, F \subseteq Q$ are the sets of initial and final states. Runs of these automata are defined in a way similar to the runs of DWA. The sub-class of *deterministic* TWA is obtained by replacing the transition relation Δ with a partial function from $Q \times \Sigma \times \mathsf{Types}^{\downarrow} \times \mathsf{Types}^{\uparrow}$ to $Q \times \mathsf{TAxis}$ and by letting I consist of a single initial state q_0.

Theorem 2 (Bojanczyk and Colcombet [2,3]). *There exist languages recognized by non-deterministic TWA that cannot be recognized by deterministic TWA. There also exist regular languages of trees that cannot be recognized by non-deterministic TWA.*

Translations between TWA and DWA. Hereafter, given a tree language L, we define L^{enc} to be the language of all data words that encode (possibly in a non-canonical way) trees in L, that is, $L^{\mathsf{enc}} = \{w : \mathsf{tree}(w) \in L\}$. To derive from Theorem 2 analogous separation results for data languages, we provide translations between TWA and DWA, as well as from tree automata to CMA:

Lemma 1. *Given a deterministic (resp. non-deterministic) TWA $\mathcal{A}$ recognizing L, one can construct a deterministic (resp. non-deterministic) DWA $\mathcal{A}^{\mathsf{enc}}$ recognizing L^{enc}. Conversely, given a deterministic (resp. non-deterministic) DWA $\mathcal{A}$, one can construct a deterministic (resp. non-deterministic) TWA $\mathcal{A}^{\mathsf{tree}}$ such that, for all trees t, $\mathcal{A}^{\mathsf{tree}}$ accepts t iff $\mathcal{A}$ accepts the canonical encoding $\mathsf{enc}(t)$.*

The proof of the first claim is almost straightforward: the DWA $\mathcal{A}^{\mathsf{enc}}$ is obtained by first transforming $\mathcal{A}$ into a DWA $\mathcal{A}'$ that mimics $\mathcal{A}$ when the input is a valid encoding of a tree, and then intersecting $\mathcal{A}'$ with a deterministic DWA $\mathcal{U}$ that accepts all and only the valid encodings of trees. For the proof of the second claim, we observe that the navigational power of a DWA is generally greater than that of a TWA: when the input is a non-canonical encoding of a tree, a DWA may choose to move from a position i to the position $i+1$ even if i does not represent a right child; on the other hand, a TWA is only allowed to move from node i to node $i+1$ when the former is a right child of the latter. Nonetheless, when restricting to *canonical* encodings of trees, the successor $i+1$ of a position represents the node that immediately follows i in the post-order visit of the tree; in this case, any move of a DWA from i to $i+1$ can be mimicked by a maximal sequence of TWA moves of the form $\uparrow\searrow\swarrow \ldots \swarrow$.

Lemma 2. *Given a tree automaton $\mathcal{A}$ recognizing a regular language L, one can construct a CMA $\mathcal{A}^{\mathsf{enc}}$ recognizing L^{enc}.*

We are now ready to transfer the separation results to data languages:

Proof (of Theorem 1). Let L_1 be a language recognized by a non-deterministic TWA $\mathcal{A}_1$ that cannot be recognized by deterministic TWA (cf. first claim of Theorem 2). Using the first claim of Lemma 1, we construct a non-deterministic DWA $\mathcal{A}_1^{\mathsf{enc}}$ such that $\mathscr{L}(\mathcal{A}_1^{\mathsf{enc}}) = L_1^{\mathsf{enc}}$. Suppose by way of contradiction that there is a deterministic DWA $\mathcal{B}_1$ that also recognizes L_1^{enc}. We apply the second claim of Lemma 1 and we obtain a deterministic TWA $\mathcal{B}_1^{\mathsf{tree}}$ that accepts all and only the trees whose canonical encodings are accepted by $\mathcal{B}_1$. Since $L_1^{\mathsf{enc}} = \{w : \mathsf{tree}(w) \in L_1\}$ is invariant under equivalent encodings of trees (that is, $w \in L_1^{\mathsf{enc}}$ iff $w' \in L_1^{\mathsf{enc}}$ whenever $\mathsf{tree}(w) = \mathsf{tree}(w')$), we have that $t \in L_1$ iff $\mathsf{enc}(t) \in L_1^{\mathsf{enc}}$, iff $t \in \mathscr{L}(\mathcal{B}_1^{\mathsf{tree}})$. We have just shown that the deterministic TWA $\mathcal{B}_1^{\mathsf{tree}}$ recognizes the language L_1, which contradicts the assumption on L_1.

By applying similar arguments to a regular tree language L_2 that is not recognizable by non-deterministic TWA (cf. second claim of Theorem 2), one can separate CMA from non-deterministic DWA. □

We conclude by observing that if non-deterministic TWA were not closed under complementation, as one would reasonably expect, then, by Lemma 1, non-deterministic DWA would not be closed under complementation either.

5 Decision Problems on DWA

We analyse in detail the complexity of the decision problems on DWA. We start by considering the simpler acceptance problem, which consists of deciding whether $w \in \mathscr{L}(\mathcal{A})$ for a given a DWA $\mathcal{A}$ and data word w. Subsequently, we move to the emptiness and universality problems, which consist of deciding, respectively, whether a given DWA accepts at least one data word and whether a given DWA accepts all data words. We will show that these problems are decidable, as well as the more general problems of containment and equivalence.

Acceptance. Compared to other classes of automata on data words (e.g. CMA, Register Automata), deterministic DWA enjoy an acceptance problem of very low time/space complexity, and the problem does not get much worse if we consider non-deterministic DWA:

Proposition 5. *The acceptance problem for a deterministic DWA $\mathcal{A}$ and a data word w is decidable in time $\mathcal{O}(|w| \cdot |\mathcal{A}|)$ and is* LOGSPACE*-complete under* NC^1 *reductions. The acceptance problem for a non-deterministic DWA is* NLOGSPACE*-complete.*

Emptiness. We start by reducing the emptiness of CMA to the emptiness of deterministic DWA (or, equivalently, to universality of deterministic DWA). For this purpose, it is convenient to think of a CMA $\mathcal{A}$ as a Tiling Automaton over a finite set Γ of colours and accordingly identify the set of all runs of $\mathcal{A}$ with the set $\mathsf{Tilings}(\mathcal{A}) \subseteq (\Sigma \times \Gamma \times \mathbb{D})^*$ of all valid tilings of data words. Given a data word $\widetilde{w} \in (\Sigma \times \Gamma \times \mathbb{D})^*$, checking whether $\widetilde{w}$ belongs to $\mathsf{Tilings}(\mathcal{A})$ reduces to checking constraints on neighbourhoods of positions. Since this can be done by a deterministic DWA, we get the following result:

Proposition 6. *Given a CMA $\mathcal{A}$, one can construct a deterministic DWA $\mathcal{A}^{\mathsf{tiling}}$ that recognizes the data language* $\mathsf{Tilings}(\mathcal{A})$.

Two important corollaries follow from this observation:

Corollary 1. *Data languages recognized by CMA are projections of data languages recognized by deterministic DWA.*

Corollary 2. *Emptiness and universality of deterministic DWA is at least as hard as emptiness of CMA, which in turn is equivalent to reachability in VASS.*

We turn now to showing that languages recognized by non-deterministic DWA are also recognized by CMA, and hence emptiness of DWA is reducible to emptiness of CMA. Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be a non-deterministic DWA. Without loss of generality, we can assume that $\mathcal{A}$ has a single initial state q_0 and a single final state q_f. We can also assume that whenever $\mathcal{A}$ accepts a data word w, it does so by halting in the rightmost position of w. For the sake of brevity, given a transition $\delta = (p, a, \tau, q, \alpha) \in \Delta$, we define $\mathsf{source}(\delta) = p$, $\mathsf{target}(\delta) = q$, $\mathsf{letter}(\delta) = a$, $\mathsf{type}(\delta) = \tau$, and $\mathsf{reach}(\delta) = \alpha$. Below, we introduce the concept of min-flow, which can be thought of as a special form of tiling that witnesses acceptance of a data word w by $\mathcal{A}$.

Definition 3. *Let* $w = (a_1, d_1)\ ...\ (a_n, d_n)$ *be a data word of length* n*. A* min-flow on w *is any map* $\mu : [n] \to 2^{\Delta}$ *that satisfies the following conditions:*

1. *There is a transition* $\delta \in \mu(1)$ *such that* $\mathsf{source}(\delta) = q_0$*;*
2. *There is a transition* $\delta \in \mu(n)$ *such that* $\mathsf{target}(\delta) = q_f$*;*
3. *For all* $i \in [n]$*, if* $\delta \in \mu(i)$*, then* $\mathsf{letter}(\delta) = a_i$ *and* $\mathsf{type}(\delta) = \mathsf{type}_w(i)$*;*
4. *For each* $i \in [n]$ *and each* $q \in Q$*, there is at most one transition* $\delta \in \mu(i)$ *such that* $\mathsf{source}(\delta) = q$*;*
5. *For each* $i \in [n]$ *and each* $q \in Q$*, there is at most one position* $j \in [n]$ *for which there is* $\delta \in \mu(j)$ *such that* $\mathsf{target}(\delta) = q$ *and* $i = \mathsf{reach}(\delta)(j)$*;*
6. *For each* $i \in [n]$*, let* $\mathsf{exiting}(i)$ *be the set of all states of the form* $\mathsf{source}(\delta)$ *for some* $\delta \in \mu(i)$*; similarly, let* $\mathsf{entering}(i)$ *be the set of all states of the form* $\mathsf{target}(\delta)$ *for some* $\delta \in \mu(j)$ *and some* $j \in [n]$ *such that* $i = \mathsf{reach}(\delta)(j)$*; our last condition states that for all positions* $i \in [n]$*,*
   (a) if $i = 1$*, then* $\mathsf{entering}(i) = \mathsf{exiting}(i) \setminus \{q_0\}$*,*
   (b) if $i = n$*, then* $\mathsf{exiting}(i) = \mathsf{entering}(i) \setminus \{q_f\}$*,*
   (c) otherwise, $\mathsf{exiting}(i) = \mathsf{entering}(i)$*.*

Lemma 3. $\mathcal{A}$ *accepts* w *iff there is a min-flow* μ *on* w*.*

Proof. Let $w = (a_1, d_1)...(a_n, d_n)$ be a data word of length n and let ρ be a successful run of $\mathcal{A}$ on w of the form $(q_0, i_0) \xrightarrow{w} (q_1, i_1) \xrightarrow{w} ...(q_m, i_m)$ obtained by the sequence of transitions $\delta_1, ..., \delta_m$. Without loss of generality, we can assume that no position in ρ is visited twice with the same state (indeed, if $i_k = i_h$ and $q_k = q_h$ for different indices k, h, ρ would contain a loop that could be eliminated without affecting acceptance). We associate with each position $i \in [n]$ the set $\mu(i) = \{\delta_k : 1 \leq k \leq m,\ i_k = i\}$. One can easily see that μ is a min-flow on w.

For the other direction, we assume that there is a min-flow μ on w. We construct the edge-labelled graph G_μ with vertices in $Q \times [n]$ and edges of the form $\big((p, i), (q, j)\big)$ labelled by a transition δ, where $i \in [n]$, $\delta \in \mu(i)$, $p = \mathsf{source}(\delta)$, $q = \mathsf{target}(\delta)$, and $j = \mathsf{reach}(\delta)(i)$. By construction, every vertex of G_μ has the same in-degree as the out-degree (either 0 or 1), with the only exceptions being the vertex $(q_0, 1)$ of in-degree 0 and out-degree 1, and the vertex (q_f, n) of in-degree 1 and out-degree 0. One way to construct a successful run of $\mathcal{A}$ on w is to repeatedly choose the *only* vertex x in G_μ with in-degree 0 and out-degree 1, execute the transition δ that labels the *only* edge departing from x, and remove that edge from G_μ. This procedure terminates when no edge of G_μ can be removed and it produces a successful run on w. □

Since min-flows are special forms of tilings, CMA can guess them and hence:

Theorem 3. *Given a DWA, one can construct an equivalent CMA.*

Universality. Here we show that the complement of the language recognized by a DWA is also recognized by a CMA, and hence universality of DWA is reducible to emptiness of CMA. As usual, we fix a DWA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, with $I = \{q_0\}$ and $F = \{q_f\}$, and we assume that $\mathcal{A}$ halts only on rightmost positions. Below we define max-flows, which, dually to min-flows, can be seen as a special forms of tilings witnessing non-acceptance.

Definition 4. *Let* $w = (a_1, d_1) \dots (a_n, d_n)$ *be a data word of length* n*. A* max-flow on w *is any map* $\nu : [n] \to 2^Q$ *that satisfies the following conditions:*

1. $q_0 \in \nu(1)$ *and* $q_f \notin \nu(n)$,
2. *for all positions* $i \in [n]$ *and all transitions* $\delta \in \Delta$*, if* $\mathsf{source}(\delta) \in \nu(i)$, $\mathsf{letter}(\delta) = a_i$*, and* $\mathsf{type}(\delta) = \mathsf{type}_w(i)$*, then* $\mathsf{target}(\delta) \in \nu\mathsf{reach}(\delta)(i)$.

Lemma 4. *$\mathcal{A}$ rejects w iff there is a max-flow ν on w.*

Theorem 4. *Given a non-deterministic DWA $\mathcal{A}$ recognizing L, one can construct a CMA $\mathcal{A}'$ that recognizes the complement of L.*

Containment and Other Problems. We conclude by mentioning a few interesting decidability results that follow directly from Theorems 3 and 4 and from the closure properties of CMA under union and intersection. The first result concerns the decidability of containment/equivalence of DWA. The second result concerns the property of language of being *invariant under tree encodings*, namely, of being of the form L^{enc} for some language L of trees.

Corollary 3. *Given two non-deterministic DWA $\mathcal{A}$ and $\mathcal{B}$, one can decide whether $\mathscr{L}(\mathcal{A}) \subseteq \mathscr{L}(\mathcal{B})$.*

Corollary 4. *Given a non-deterministic DWA $\mathcal{A}$, one can decide whether $\mathscr{L}(\mathcal{A})$ is invariant under tree encodings.*

6 Discussion

We showed that the model of walking automaton can be adapted to data words in order to define robust families of data languages. We studied the complexity of the fundamental problems of word acceptance, emptiness, universality, and containment (quite remarkably, all these problems are shown to be decidable). We also analysed the relative expressive power of the deterministic and non-deterministic models of Data Walking Automata, comparing them with other classes of automata appeared in the literature (most notably, Data Automata and Class-Memory Automata). In this respect, we proved that deterministic DWA, non-deterministic DWA, and CMA form a strictly increasing hierarchy of data languages, where the top ones are projections of the bottom ones.

It follows from our results that DWA satisfy properties analogous to those satisfied by Tree Walking Automata – for instance deterministic DWA, like deterministic TWA, are effectively closed under under all boolean operations, and are strictly less expressive than non-deterministic DWA. It turns out that DWA are also incomparable with one-way Register Automata [5]: on the one hand, DWA can check that all data values are distinct, whereas Register Automata cannot; on the other hand, Register Automata can recognize languages of data strings that do not encode valid runs of Turing machines, while Data Walking Automata cannot, as otherwise universality would become undecidable.

Since moving along the axis $\oplus 1$ (resp. $\ominus 1$) can be simulated by storing the current data value or putting a pebble at the current position and moving along the axis $+1$ (resp. -1) searching for the nearest position with the stored data value or marked data value, it follows that DWA are subsumed by two-way 1-Register Automata and 2-Pebble Automata (note that in Pebble Automata one pebble is always used by the head). Other variants of DWA could have been considered, for instance, by adding registers, pebbles, alternation, or nesting. Unfortunately, none of these extensions yield a decidable containment problem. For instance, equipping DWA with a single pebble would enable encoding positive instances of the Post Correspondence Problem, thus implying undecidability of emptiness.

We leave open the following problems:

- Are non-deterministic DWA closed under complementation? (a similar separation result remains open for Tree Walking Automata [2,3]).
- Do DWA capture all languages definable by two-variable first-order formulas using the predicates $<$ and $\sim$.

As a matter of fact, we can easily show that DWA capture FO^2 logic with predicates $+1$ and $\oplus 1$ (the proof relies on a variant of Gaifman's locality theorem).

Acknowledgments. The first author thanks Thomas Colcombet for detailed discussions and acknowledges that some of the ideas were inspired during these. The second author acknowledges Mikołaj Bojańczyk and Thomas Schwentick for discussions about the relationship between DWA and Data Automata.

References

1. Björklund, H., Schwentick, T.: On notions of regularity for data languages. Theoretical Computer Science 411(4-5), 702–715 (2010)
2. Bojańczyk, M., Colcombet, T.: Tree-walking automata cannot be determinized. Theoretical Computer Science 350(2-3), 164–173 (2006)
3. Bojańczyk, M., Colcombet, T.: Tree-walking automata do not recognize all regular languages. SIAM Journal 38(2), 658–701 (2008)
4. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. ACM Transactions on Computational Logic 12(4), 27 (2011)
5. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134(2), 329–363 (1994)
6. Libkin, L., Vrgoč, D.: Regular expressions for data words. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18 2012. LNCS, vol. 7180, pp. 274–288. Springer, Heidelberg (2012)
7. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic 5(3), 403–435 (2004)
8. Sipser, M.: Halting space-bounded computations. Theoretical Computer Science 10, 335–338 (1980)
9. Thomas, W.: Elements of an automata theory over partial orders. In: Partial Order Methods in Verification, pp. 25–40. Americal Mathematical Society (1997)

Careful Synchronization of Partial Automata with Restricted Alphabets

Pavel V. Martyugin

Ural Federal University, Ekaterinburg, Russia
martuginp@gmail.com

Abstract. We consider the notion of careful synchronization for partial finite automata as a natural generalization of the notion of synchronization for complete finite automata. We obtain a lower bound for the length of the shortest carefully synchronizing words of an automaton with a fixed number of states and a fixed number of letters. In particular, we consider this bound for automata over a binary alphabet. Our results improve previously known bounds.

Keywords: careful synchronization, partial automata, synchronizing words, Černý conjecture.

1 Introduction

A *deterministic finite automaton* (DFA for short) is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet and δ is a totally defined transition function. Denote by Σ^* the free Σ-generated monoid and by λ the empty word. The function δ extends in a natural way to an action $Q \times \Sigma^* \to Q$. This extension is also denoted by δ. A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action *resets* $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in Q it starts at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any such word w is said to be a *reset* or *synchronizing* word for the automaton $\mathscr{A}$.

A conjecture proposed by Černý in [2] states that every synchronizing DFA with n states can be synchronized by a word of length at most $(n-1)^2$. There have been made many attempts to prove it but they all have failed so far. The conjecture was proved only for some partial classes of DFA (see [3,1,5,13]). The best upper bound known up to date is $n(7n^2 + 6n + 16)/48$, see [11]. A survey of results concerning synchronizing words can be found in [12].

The notion of a synchronizing word may be generalized to the case of automata with a partial transition function (PFA). *A partial finite automaton* (PFA) is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet and δ is a partial function from $Q \times \Sigma$ to Q. The function δ can be undefined on some pairs from the set $Q \times \Sigma$. The function δ can be first extended to an action from $Q \times \Sigma^*$ to Q as follows. We put $\delta(q, \lambda) = q$ for every $q \in Q$. Let $q \in Q$, $a \in \Sigma$, $w \in \Sigma^*$. If the transitions $\delta(q, w)$ is defined, $p = \delta(q, w)$ and the value

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 76–87, 2013.

$\delta(p,a)$ is defined, then we put $\delta(q,wa)=\delta(p,a)$. Next, the function δ can be extended to $2^Q\times\Sigma^*$, where 2^Q denotes the set of all subsets of Q. Take $S\subseteq Q$ and $w\in\Sigma^*$. If the values $\delta(q,w)$ are defined for all states $q\in S$, then we put $\delta(S,w)=\{\delta(q,w)\mid q\in S\}$.

A PFA $\mathscr{A}=(Q,\Sigma,\delta)$ is called *carefully synchronizing*, if there exists a word $w\in\Sigma^*$ such that the value $\delta(Q,w)$ is defined and $|\delta(Q,w)|=1$. Any such word w is said to be *carefully synchronizing word (c.s.w.)* for the PFA $\mathscr{A}$. It may turns out that a c.s.w. synchronizes a PFA and does not "break" it in the sense that no undefined transition is ever used. Clearly, as each DFA is also a PFA, a c.s.w. for a DFA is also a synchronizing word for it. Therefore, the notion of careful synchronization for PFA is a natural generalization of the notion of synchronization for DFA.

The Černý-type problem may be also considered for PFA. Let $car(n)$ be the maximal possible length of the shortest c.s.w. of a carefully synchronizing PFA with n states. The value $car(n)$ was initially studied in [6], [7] and [8] (in these papers this value was considered in general way and was denoted by $d_3(n)$). In particular, it was obtained that $car(n)=\Omega(2^{n/2})$. The lower bound was improved in [10]. Namely, it was proved that $car(n)=\Omega(3^{n/3})$. The best known up to now upper bound $car(n)=O(n^2\cdot 4^{n/3})$ was found in [4].

Observe that known bounds on the length of the shortest synchronizing words are polynomial in n. By contrast the length of the shortest c.s.w may be an exponential function of n. But there is an essential feature of the series of PFA used in the proof of the previous lower bounds. In these series the input alphabet size grows with the number of states.

The above mentioned examples contrast with Černý's examples [2], in which the alphabet size is independent of the number of states in the DFA. Thereby, we may propose the following natural problem: determine the maximum length of the shortest carefully synchronizing word for an n-state PFA over an input alphabet of fixed size.

For any integer m we may define the value $car_m(n)$ as the maximal length of the shortest c.s.w. among all carefully synchronizing PFA with n states and an m-letter input alphabet. In [9] the following lower bounds were obtained:

$$car_2(n)>2e^{0{,}9\sqrt[2+\xi]{n}},\ car_3(n)>e^{0{,}9\sqrt[2+\xi]{2n}}.$$

In this paper we improve these bounds. Let $m\geq 3$. Then for infinitely many n

$$car_m(n)>3^{\frac{n}{3\log_{m-1}n}}$$

For a binary alphabet the relation

$$car_2(n)>3^{\frac{n}{6\log_2 n}}$$

holds for infinitely many n. These inequalities improve the previous lower bounds for n large enough.

We now give some auxiliary notations. Let p and q be integers and $p\leq q$. We denote the set $\{p,p+1,\ldots,q\}$ by $p..q$. Let w be a word over some alphabet Σ.

We denote by $|w|$ the length of the word w. And for $1 \le i \le |w|$ by $w[i]$ the i-th letter of w. More generally, for given $i, k \in 1..|w|$ we denote by $w[i,k]$ the word $w[i]w[i+1]\cdots w[k]$. For a DFA $\mathscr{A} = (Q, \Sigma, \delta)$ and $q \in Q, a \in \Sigma$, we will write $q.a$ instead of $\delta(q,a)$, when δ is clear from the context. For any subset $S \subseteq Q$ and for any word $w \in \Sigma^*$ we also write $S.w$ instead of $\delta(S,w)$.

2 Arbitrary Alphabet

Our goal is to construct series of PFA with an increasing number of states, a fixed alphabet and a large length for the shortest c.s.w. First we introduce a simple series of a PFA with an increasing alphabet size and a sufficiently long shortest c.s.w. After that we will use this PFA to construct more complicated PFA with a restricted alphabet size. Our first construction is a simplification of the construction from [10]. The lower bound from [10] is better but this example is useful to construct required examples of automata over fixed alphabets.

Theorem 1. *The inequality $car(n) \ge 3^{n/3}$ holds for infinitely many n.*

Proof. Take a positive integer $n = 3k$, where k is a natural number. We define a PFA $\mathscr{A}(k) = (Q, \Sigma, \delta)$ with n states and $k+1$ letters such that the length of the shortest c.s.w. for $\mathscr{A}(k)$ is $3^k = 3^{n/3}$. Let Q be the disjoint union $\bigcup_{i=1}^{k} Q_i$, where $Q_i = \{q(i,j) \mid j \in 0..2\}$ and let $\Sigma = \{x_0, \ldots, x_{k-1}, y\}$. The set Q can be seen as a table with 3 columns and k rows. We now define a partial function δ as follows. Let $i \in 0..k-1$, $j \in 0..2$ and $m \in 0..k-1$. Then

$$q(i,j).x_m = \begin{cases} q(i,j) & \text{if } m < i; \\ q(i,j-1) & \text{if } m = i,\ j \neq 0; \\ q(i,0) & \text{if } m = i, j = 0; \\ q(i,2) & \text{if } m > i, j = 0; \\ \text{undefined} & \text{if } m > i, j \neq 0; \end{cases}$$

$$q(i,j).y = \begin{cases} q(0,0) & \text{if } j = 0; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let Q_i be the i-th row of the set Q. The restriction of $\mathscr{A}(k)$ to the set Q_i leads to three possible actions $-, +$ and $\odot$ of letters $x_0, \ldots, x_{k-1}$. The action "$-$" associated with the letters $x_0, \ldots, x_{i-1}$ is the identity action $0.- = 0$, $1.- = 1, 2.- = 2$. The action "$\odot$" associated with the letter x_i is given by $0.\odot = 0, 1.\odot = 0, 2.\odot = 1$. The action "$+$" corresponding to the remaining letters $x_{i+1}, \ldots, x_{k-1}$ is $0.+ = 2, 1.+$ and $2.+$ undefined. We denote by $\mathscr{K} = (\{0,1,2\}, \{\odot, -, +\}, .)$ the resulting PFA. This 3-state PFA is a basis of all constructions from this paper. Figure 1 represents the automaton $\mathscr{K}$ and the automaton $\mathscr{A}(k)$ for $k = 3$.

We are going to prove that the length of the shortest c.s.w. of $\mathscr{A}(k)$ is 3^k. Let $v_0 = x_0^2$; $v_1 = v_0x_1v_0x_1v_0$; $\ldots$; $v_{k-1} = v_{k-2}x_{k-1}v_{k-2}x_{k-1}v_{k-2}$. It is easy to check by induction on m that for $m \in 0..k-1$

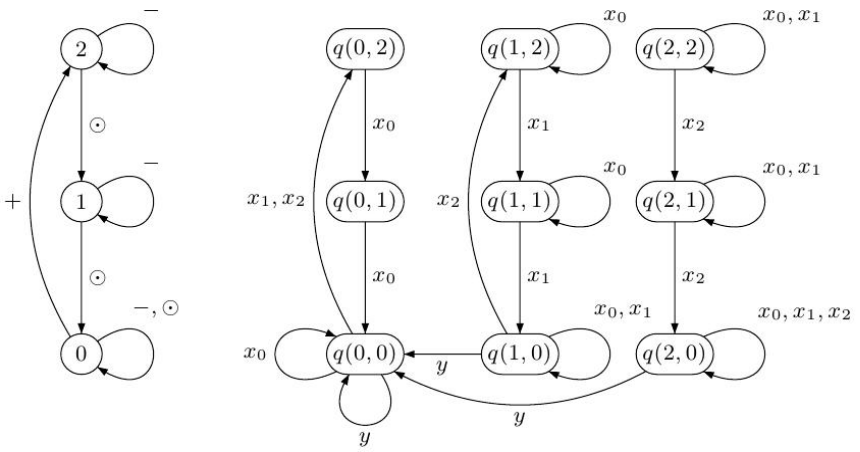


Fig. 1. The automata $\mathscr{K}$ and $\mathscr{A}(3)$

$$Q.v_m = Q_k \cup Q_{k-1} \cup \cdots \cup Q_{m+1} \cup \{q(m,0), \ldots, q(0,0)\}$$

Therefore, $Q.v_{k-1} = \{q(k-1,0), \ldots, q(0,0)\}$ and the word $w = v_{k-1}y$ carefully synchronizes the automaton $\mathscr{A}(k)$ to the state $q(0,0)$. It is not hard to calculate the length of the word w. It equals 3^k.

We are going to prove that the word w is the shortest c.s.w for the PFA $\mathscr{A}(k)$. Note the following simple properties of the automaton $\mathscr{A}(k)$.

1. Let $i \in 0..k-1$ and let $S \subseteq Q_i$, then for any $m \in 0..k-1$, one has $Q_i \supseteq S.x_m \neq \emptyset$;
2. For any word $u \in (\Sigma \setminus \{y\})^*$ and for any $i \in 0..k-1$, one has $Q.u \cap Q_i \neq \emptyset$;
3. Only the letter y can merge states from different Q_i. Hence every shortest c.s.w. contains y;
4. The letter y is defined only on the states of the set $T_0 = \{q(i,0) \mid i \in 0..k-1\}$ and $T_0.y = \{q(0,0)\}$.

These properties imply, that the shortest c.s.w. w is equal to $w'y$, where $w' \in (\Sigma \setminus y)^*$ and $Q.w' = T_0$. Furthermore for any prefix u of w' and for any $i \in 0..k-1$, one has $Q.u \cap Q_i \neq \emptyset$.

Next we define a weight function μ for an arbitrary subset $S \subseteq Q$ such that $S \cap Q_i \neq \emptyset$ for $i \in 0..k-1$. Let $T \subseteq Q_i$ for some $i \in 0..k-1$. We put

$$\mu(T) = \begin{cases} 2 \cdot 3^i & \text{if } q(i,2) \in T; \\ 3^i & \text{if } q(i,2) \notin T, q(i,1) \in T; \\ 0 & \text{if } T = \{q(i,0)\}. \end{cases}$$

For each $S \subseteq Q$ such that $S \cap Q_i \neq \emptyset$ for $i \in 0..k-1$, we set $\mu(S) = \sum_{i=0}^{k-1} \mu(S \cap Q_i)$. Let $S \subseteq Q$ be such that the value $\mu(S)$ is defined and for

$m \in 0..k-1$ the set $S.x_m$ is defined. In this case from the definition of μ and the transition function δ we obtain

1. For any $i \in 0..m-1$ the values $q(i,1).x_m$ and $q(i,2).x_m$ are undefined. Hence we have $S \cap Q_i = \{q(i,0)\}$. Hence $S.x_m \cap Q_i = \{q(i,2)\}$. Therefore, $\mu(S.x_m \cap Q_i) = \mu(S \cap Q_i) + 2 \cdot 3^i$.
2. For $i \in m+1..k-1$ the letter x_m fixes the states $q(i,0), q(i,1)$ and $q(i,2)$. Hence we have $S.x_m \cap Q_i = S \cap Q_i$. Therefore, $\mu(S.x_m \cap Q_i) = \mu(S \cap Q_i)$.
3. Let $i = m$. If $\mu(S \cap Q_m) = 2 \cdot 3^m$, then $\mu(S.x_m \cap Q_m) = 3^m$. If $\mu(S \cap Q_m) = 3^m$, then $\mu(S.x_m \cap Q_m) = 0$. If $\mu(S \cap Q_m) = 0$, then $\mu(S.x_m \cap Q_m) = 0$. Therefore, $\mu(S.x_m \cap Q_m) \geq \mu(S \cap Q_m) - 3^m$.

Thus $\mu(S.x_m) \geq \mu(S) - 3^m + \sum_{m-1}^{i=0} 3^i = \mu(S) - 1$. Furthermore, $\mu(Q) = 3^k - 1$ and $\mu(T_0) = 0$. Therefore, the length of the shortest word to reach the set T_0 from the set Q is not less than $3^k - 1$. Thus the length of c.s.w. in not less than $3^k = 3^{n/3}$. The theorem is proved.

3 Fixed Alphabet Size

The n-state automaton from the previous section has $\frac{n}{3} + 1$ letters in the input alphabet. Suppose we can only use an alphabet of fixed size. Can we construct a series of slowly carefully synchronizing PFA in this case? The answer is positive. In this section for any integer $m \geq 3$ we construct a series of PFA over an m-letter alphabet such that the shortest c.s.w. of the PFA from this series are exponentially long.

Theorem 2. *Let $m \geq 3$. The inequality $car_m(n) > 3^{\frac{n}{3\log_{m-1} n}}$ holds for infinitely many n.*

Proof. Let $n = 3(r+1)(m-1)^r$ for some integer $r > 0$. For any such n we construct an automaton $\mathscr{B}(m,r) = (P, \Sigma, \gamma)$ with m letters $0, 1, \ldots, m-2, c$ and n states such that the shortest c.s.w. for $\mathscr{B}(m,r)$ has length not less than $3^{\frac{n}{3\log_{m-1} n}}$. We denote the set of letters $0..m-2$ by Ψ. Let $k = (m-1)^r$ and

$$P = \{p(i,j,t) \mid i \in 0..k-1, j \in 0..2, t \in 0..r\}.$$

The states of the PFA $\mathscr{B}(m,r)$ can be drawn as k blocks (numbered from 0 to $k-1$) where each block is a table with $r+1$ rows (numbered from 0 to r) and 3 columns (numbered from 0 to 2).

The upper part of the Figure 2 represents the PFA $\mathscr{B}(m,r)$ for $m = 3$ and $r = 2$. In this case $k = 4$ and $n = 36$. The action of the letter 1 is drawn by solid lines, the action of the letter 0 is drawn by dashed lines, the action of the letter c is drawn by dotted lines. Let us define the function γ. First, we define the action of the letter c. For any $i \in 0..k-1,\ t \in 0..r,\ j \in 0..2$ we put

$$\gamma(p(i,j,t), c) = \begin{cases} p(i,j,0) & \text{if } t = r; \\ p(0,0,0) & \text{if } t = 0, j = 0; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let $i \in 0..k-1$, and let $(a_0, \ldots, a_{r-1})$ be the $(m-1)$-ary presentation of the number i (i.e. the presentation of i in the numeric system with $m-1$ digits $0, \ldots, m-2$). If $i < 2^{r-1}$ then we add leading zeros to the presentation. For any $\alpha \in \Psi, j \in 0..2, t \in 0..r$ we put

$$\gamma(p(i,j,t),\alpha) = \begin{cases} p(i,j,t+1) & \text{if } a_t = \alpha, t < r-1; \\ p(i,r,j-1) & \text{if } a_t = \alpha, j > 0, t = r-1; \\ p(i,r,0) & \text{if } a_t = \alpha, j = 0, t = r-1; \\ \text{undefined} & \text{if } a_t > \alpha, j > 0, t < r; \\ p(i,r,2) & \text{if } a_t > \alpha, j = 0, t < r; \\ p(i,r,j) & \text{if } a_t < \alpha, t < r; \\ p(i,t,j) & \text{if } t = r. \end{cases}$$

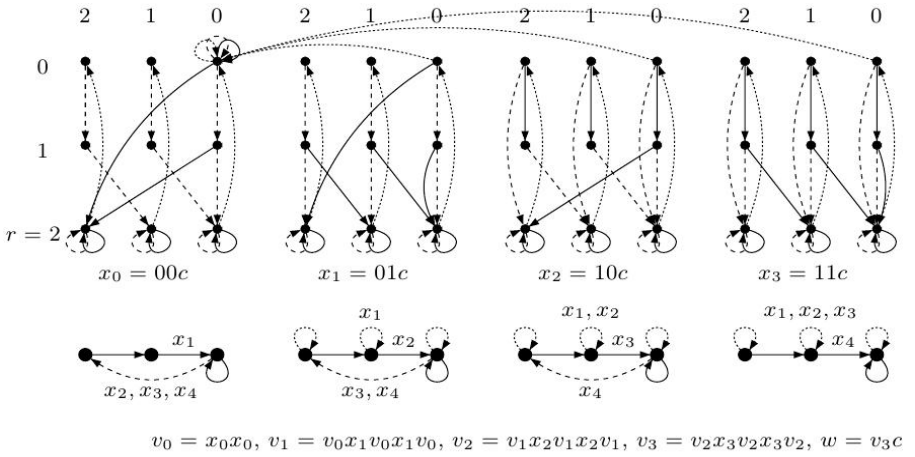


Fig. 2. The automaton $\mathscr{B}(3,2)$ and the corresponding automaton $\mathscr{A}(4)$

Let $Q = \{p(i,j,0) \mid i \in 0..k-1, j \in 0..2\}$ (it is the first row in Figure 2). Let $\Psi^r c$ be the set of all $r+1$-letter words having letters from Ψ in the first r positions and the letter c in the last position. The following lemma clears up the complicated construction of the automaton $\mathscr{B}(m,r)$ and its correlation with the automaton $\mathscr{A}(k)$ from Theorem 1.

Lemma 1. *For any word $u \in (\Psi^r c \cup \{c\})^*$ and for any state $q \in Q$ such that $q.u$ is defined, one has $q.u \in Q$, and the PFA $(Q, \Psi^r c \cup \{c\}, \gamma)$ is exactly the PFA $\mathscr{A}(k) = \mathscr{A}(m^{r-1})$.*

Proof. The upper part and the lower part of Figure 2 illustrate this lemma. The lower part of the figure represents the action of all words from the set $(\Psi^r c \cup \{c\})^*$

on the set Q (which is the first row of the states in the picture above) for PFA $\mathscr{B}(3,2)$ from upper part of the picture. This automaton is isomorphic to $\mathscr{A}(4)$ with letters $x_0 = 00c, x_1 = 01c, x_2 = 10c$ and $x_3 = 11c$. It consists of four parts. Each part is isomorphic to the automaton $\mathscr{K}$ form the previous section. In the i-th part (if we number them from 0 to 3) the action of the letter x_i is drawn by solid lines, the actions of letters $x_h, h < i$ is drawn by dotted lines and the actions of letters $x_h, h > i$ is drawn by dashed lines.

The first statement of this lemma can be easily checked using the definition of the function γ. So we prove that $\mathscr{A}(k) = \mathscr{A}(m^{r-1})$. Let $(a_0, a_1, \ldots, a_{r-1})$ where $a_0, ..., a_{r-1} \in 0..m-2$ be the $(m-1)$-ary presentation of some integer $\mu \in 0..k-1$. We may consider any digit $a_s \in \{0,1\}$ as the letter from the alphabet of the PFA $\mathscr{B}(m,r)$. We prove that the word $a_0 \ldots a_{r-1}c$ plays the role of the letter x_μ for the PFA $\mathscr{A}(k)$, the letter c plays the role of the letter y and, for any $i \in 0..k-1,\ j \in 0..2$, the state $p(i,j,0)$ plays the role of the state $q(i,j)$ in Q.

First, if the letter c is defined on some state $q \in Q$, then $\gamma(q,c) = p(0,0,0) = q(0,0)$. Therefore, the letter c acts on the states from the set Q as the letter y from $\mathscr{A}(k)$.

Let $i = \mu$. In this case by the definition of the function γ, we have $p(i,j,0).a_0 = p(i,j,1)$, for any j. Further $p(i,j,1).a_1 = p(i,j,2)$, $\ldots$, $p(i,j,r-2).a_{r-2} = p(i,j,r-1)$. Finally

$$p(i,j,r-1).a_{r-1} = \begin{cases} p(i,j-1,r) & \text{if } j > 0; \\ p(i,j,r) & \text{if } j = 0. \end{cases}$$

Thus,

$$\gamma(p(i,j,0), a_0 \ldots a_{r-1}c) = \begin{cases} p(i,j-1,0) & \text{if } j > 0; \\ p(i,j,0) & \text{if } j = 0 \end{cases} = \delta(q(i,j), x_\mu).$$

Let $i > \mu$ and let $(b_0, \ldots, b_{r-1})$ be the binary presentation of the number i. In this case there exists $t \in 0..r-1$ such that $a_t < b_t$. Let t be the minimal number with this property. In this case from the definition of γ we obtain $p(i,j,0).a_0 \ldots a_{t-1} = p(i,j,t)$ and $p(i,j,t).a_t = p(i,j,r)$. Further
$p(i,j,r).a_{t+1} \ldots a_{r-1} = p(i,j,r)$. Thus,

$$\gamma(p(i,j,0), a_0 \ldots a_{r-1}c) = p(i,j,0) = \delta(q(i,j), x_\mu).$$

Let $i < \mu$ and let $(b_0, \ldots, b_{r-1})$ be the binary presentation of the number i. In this case there exists $t \in 0..r-1$ such that $a_t > b_t$. Let t be the minimal number with this property. As in the previous case, we get $p(i,j,0).a_0 \ldots a_{t-1} = p(i,j,t)$. If $j > 0$, then $p(i,j,t).a_t$ is undefined. If $j = 0$, then $p(i,j,t).a_t = p(i,2,r)$. Further, $p(i,2,r).a_{t+1} \ldots a_{r-1} = p(i,2,r)$. Thus,

$$\gamma(p(i,j,0), a_0 \ldots a_{r-1}c) = \begin{cases} \text{undefined} & \text{if } j > 0; \\ p(i,2,0) & \text{if } j = 0. \end{cases} = \delta(q(i,0), x_\mu).$$

The lemma is proved.

Lemma 2. *Let w be a shortest c.s.w. for the PFA $\mathscr{B}(m,r)$. Then $w \in (\Psi^r c \cup \{c\})^*$.*

Proof. Only the transition 0 is defined on the state $p(0,2,0)$ and $p(0,2,0).0 = p(0,2,1)$. Only the transition 0 is defined on the state $p(0,2,1)$ and $p(0,2,1).0 = p(0,2,2)$, e.t.c. only the transition 0 is defined on the state $p(0,2,r-1)$. Therefore, w starts with 0^r. Furthermore, $Q.0^r \subseteq \overline{Q} = \{p(i,r,j) \mid i \in 0..k-1,\ j \in 0..2\}$ ($\overline{Q}$ is the last row of states in Figure 2). All states from the set $\overline{Q}$ are fixed by the letters $0..m-2$. Therefore, we should apply the letter c after 0^r to move something. Hence the word w starts with $0^r c$. Denote by Q_i the set $\{p(i,j,0) \mid j = 0..2\}$. Note that $0^r c \in \Psi^r c$ and, for $i \in 0..k-1$, $Q.0^r c \cap Q_i \neq \emptyset$. Further $Q.0^r c.c$ is not defined. We call a subset $S \subseteq Q$ a Q-set if for any $i \in 0..k-1$, one has $S \cap S_i \neq \emptyset$ and $\gamma(S,c)$ is not defined. Note, that $Q.0^r c$ is a Q-set.

Let S be a Q-set. Now we prove that for any natural $r_0 < r$ and for any word $u \in \Psi^*$ of length $r_0 < r$ the word uc is not defined on the set S. Let $u = a_0 \dots a_{r_0-1}$. Let $i \in 0..k-1$ be an $(m-1)$-ary r-digit integer with $(m-1)$-ary presentation $(a_0, \dots, a_{r_0-1}, 0, ..., 0)$. It is easy to check by the definition of γ, that $Q_i.u$ belongs to the set $\{p(i,j,r_0) \mid j \in 0..2\}$. The letter c is not defined on any state of this set. Hence the word uc is not defined on any state from Q_i. Therefore, the word uc is undefined on the set S.

It can be also proved that for any word $u \in \Psi^*$ of length $r_0 > r$ the relations $S.u \subseteq \overline{Q}$ and $S.u = S.u[1,r]$ holds, where $u[1,r]$ is the r-letter prefix of the word u. Moreover, for any word $u \in \Psi^r c$ and any $i \in 0..k-1$, we have $S.u \cap Q_i \neq \emptyset$.

Thus, we have the following structure of the word w. The word w starts with $w_1 = 0^r c$ and $P.w_1$ is a Q-set. The word w can be extended only by some word $w_2 \in \Psi^r c$. In this case $P.w_1 w_2$ is also the Q-set. And so on, the word w starts with $w_1 w_2 \dots w_z$, where $w_1, \dots, w_z \in \Psi^r c$ and for any $y \in 1..z-1$ we have $P.w_1 w_2 \dots w_y$ is a Q-set, but $P.w_1 \dots w_z$ is not a Q-set. In this case the letter c should be defined on the set $P.w_1 \dots w_z$. We also have $P.w_1 \dots w_z c = p(0,0,0)$. Thus, $w = w_1 \dots w_j c \in (\Psi^r c \cup \{c\})^*$. The lemma is proved.

We now come back to the proof of Theorem 2. Let w be the shortest c.s.w. for $\mathscr{B}(m,r)$. By Lemma 2 we know that $w = w_1 \dots w_z c$ for some integer $z > 0$, where $w_1, \dots, w_z \in \Psi^r c$. Moreover, it follows from the proof of Lemma 2 that $w_1 = 0^r c$. Therefore, it can be checked that $\gamma(P, w_1) = Q \backslash \{p(0,2,0)\} = \gamma(Q, w_1)$. So $\gamma(P, w_1 \dots w_y) = \gamma(Q, w_1 \dots w_y)$ for any $y \in 1..z$. Hence w, being a word over the alphabet $\Psi^r c \cup \{c\}$, is the shortest c.s.w. for the PFA $\mathscr{A}(k)$. Therefore, by Theorem 1, the word w in the PFA $\mathscr{A}(k)$ has 3^k letters, and just the last letter of w is c. So $|w| = (3^k - 1)(r+1) + 1$. Recall that $n = 3(m-1)^r(r+1)$. Finally, we get

$$\log_{m-1} n = \log_{m-1} 3 + r + \log_{m-1}(r+1) > r+1,\ (m-1)^r = \frac{n}{3(r+1)} > \frac{n}{3\log_{m-1} n}.$$

$$|w| = (3^k - 1)(r+1) + 1 \geq 3^{(m-1)^r} mr > 3^{\frac{n}{3\log_{m-1} n}}.$$

The theorem is completely proved.

4 Binary Alphabet

In this section we construct a series of PFA with 2 letters. This series still give us an exponential lower bound of the length of the shortest c.s.w. The same construction can be implemented for an alphabet with more than two letters, but such a construction gives worse lower bound than the bound obtained from the construction of the corresponding PFA $\mathscr{B}(m,r)$.

Theorem 3. *The inequality* $car_2(n) > 3^{\frac{n}{6\log_2 n}}$ *holds for infinitely many* n.

Proof. Let $n = 3(2r+1)2^r$ for some integer $r > 0$. We will construct an automaton $\mathscr{C}(r) = (P, \{0,1\}, \gamma)$ over a binary alphabet with n states such that the length of its shortest c.s.w. is not less than $3^{\frac{n}{6\log_2 n}}$. Let $k = 2^r$ and let

$$P = p(i,j,t) \mid i \in 0..k-1, j \in 0..2, t \in \{-1,0,r\} \cup$$

$$\cup \{p(i,j,t,e) \mid i \in 0..k-1, j \in 0..2, t \in 1..r-1, e \in \{0,1\}\}$$

Sometimes we will denote the states of the kind $p(i,j,0)$ by $p(i,j,0,0)$ and the states of the kind $p(i,j,r)$ by $p(i,j,r,1)$. The set of states of the PFA $\mathscr{C}(r)$ can be drawn as k blocks (numbered from 0 to k-1), where each block is a table with $r+2$ rows (from -1 to r) and 3 columns (from 0 to 2), where the middle part of any column splits into two columns (0 and 1). Figure 3 represents the PFA $\mathscr{C}(3)$. For this automaton $k = 8$ and $n = 168$. The action of the letter 0 is drawn by dashed lines, the action of the letter 1 is drawn by solid lines. Any of the eight gray states represents a state of the form $p(0,0,-1)$. Let us define the function γ. For any $i \in 0..k-1$, and $j \in 0..2$ we put

$$\gamma(p(i,j,-1),0) = p(i,j,0);\ \gamma(p(i,j,-1),1) = \begin{cases} p(0,0,-1) & \text{if } j = 0; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\gamma(p(i,j,r),0) = p(i,2,r), \gamma(p(i,j,r),1) = p(i,j,-1).$$

Let $i \in 0..k-1$ and let $(a_0,\ldots,a_{r-1})$ be the binary presentation of the number i. If $i < 2^{r-1}$ then we add leading zeros. For any $t \in -1..r-1,\ j \in 0..2, e \in \{0,1\}, \alpha \in \{0,1\}$ we put

$$\gamma(p(i,j,t,e),\alpha) = \begin{cases} p(i,j,t+1,0) & \text{if } a_t = \alpha, t < r-1, e = 0; \\ p(i,j-1,r) & \text{if } a_t = \alpha, j > 0, t = r-1, e = 0; \\ p(i,j,r) & \text{if } a_t = \alpha, j = 0, t = r-1, e = 0; \\ \text{undefined} & \text{if } a_t > \alpha, j > 0, e = 0; \\ p(i,2,t+1,1) & \text{if } a_t > \alpha, j = 0, e = 0; \\ p(i,j,t+1,1) & \text{if } a_t < \alpha, e = 0; \\ p(i,j,t+1,1) & \text{if } e = 1. \end{cases}$$

The proof of this theorem is similar to the proof of the previous theorem. We just give a sketch of it. Let $Q = \{p(i,j,-1) \mid i \in 0..k-1, j \in 0..2\}$ (it is the first row on the Figure 3). The construction of the PFA $\mathscr{C}(r)$ is also related to the PFA $\mathscr{A}(k)$ from Theorem 1 (as the construction of the PFA $\mathscr{B}(m,r)$).

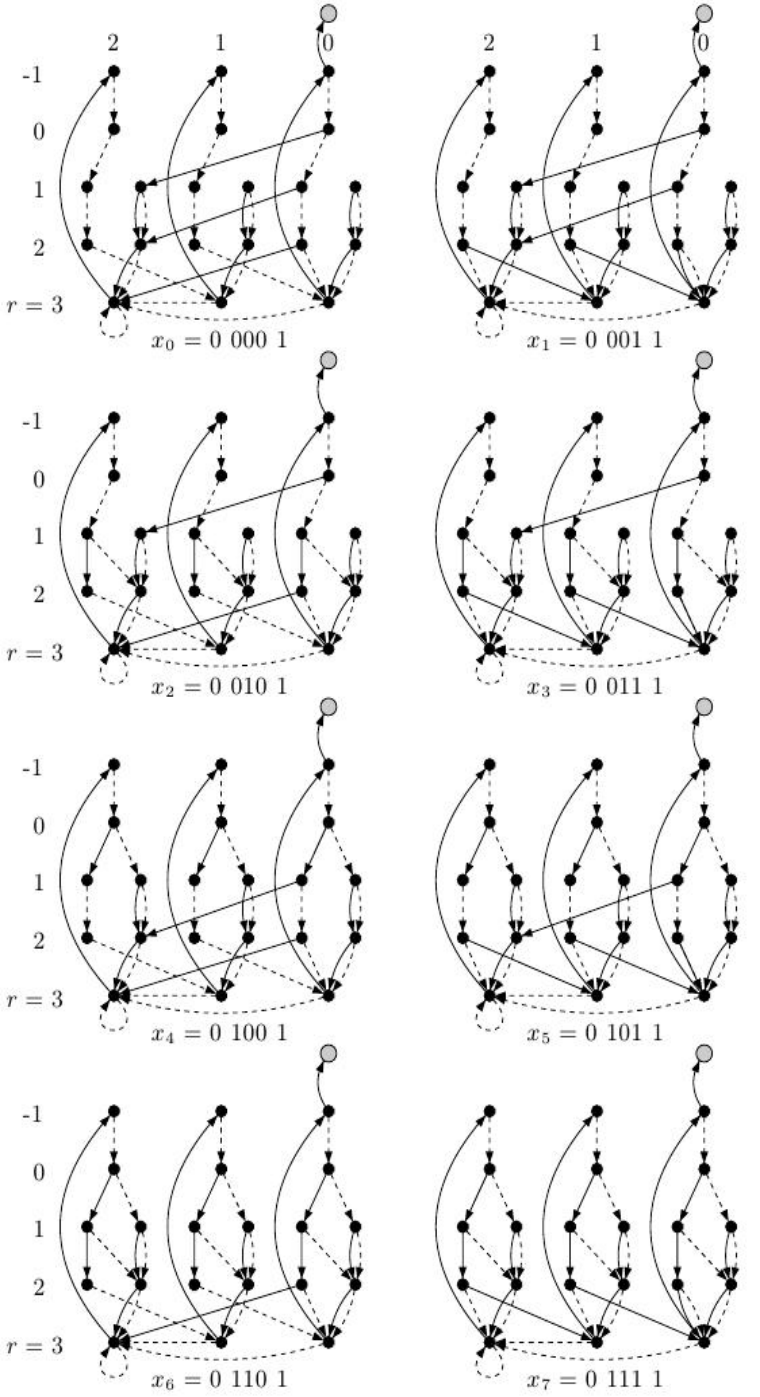


Fig. 3. The automaton $\mathscr{C}(3)$

Lemma 3. *For any word $u \in 0\{0,1\}^r1 \cup \{1\}$ and for any state $q \in Q$ such that $q.u$ is defined, one has $q.u \in Q$, and the PFA $(Q, 0\{0,1\}^r1 \cup \{1\}, \gamma)$ is exactly the PFA $\mathscr{A}(k) = \mathscr{A}(2^r)$.*

Proof. The proof of this lemma is similar to the proof of Lemma 1. For any $i \in 0..k-1, j \in 0..2$ the state $p(i,j,-1)$ plays the role of the state $q(i,j)$ in the PFA $\mathscr{A}(k)$ (as the state $p(i,j,0)$ in $\mathscr{B}(m,r)$) Let $a_0 \in \{0,1\}, \ldots, a_{r-1} \in \{0,1\}$ and let $\mu \in 0..k-1$ be a number with a $m-1$-ary record $(a_0, \ldots, a_{r-1})$. The word $0a_0 \ldots a_{r-1}1$ in PFA $\mathscr{C}(r)$ plays the role of the letter x_μ in $\mathscr{A}(k)$ (as the word $a_0 \ldots a_{r-1}c$ in $\mathscr{B}(m,r)$). The letter 1 from $\mathscr{C}(r)$ plays the role of the letter y from $\mathscr{A}(k)$ on the set Q.

Thus, Lemma 3 can be proved in the same way as Lemma 1 by using the definition of the function γ. We skip details here.

Lemma 4. *Let w be the shortest c.s.w. for PFA $\mathscr{C}(r)$, then $w \in (0\{0,1\}^r1)^*1$.*

Proof. The proof of this lemma is similar to the proof of Lemma 2. We give a sketch of it.

Let w be the shortest c.s.w. for $\mathscr{C}(r)$. It not difficult to prove, that the word w starts with $0^{r+1}1$ and $P.0^{r+1}1 \subseteq Q$. Let p be the state from the set P without the last row (i.e. except all states of form $p(i,j,r)$). Then the letters 0 and 1 move this state to the next row. Furthermore, the letter 1 is not defined on the -1-th row (states of form $p(i,j,-1)$). The letter 1 moves the states from the last row to the -1-th row. It is not useful to apply the letter 0 to any state of the last row, because it moves it back to the subcolumn with number 2, but we need to obtain the states of the form $p(i,0,-1)$ to use the last letter 1 and reach the one-element set $\{p(0,0,-1)\}$. Thus, any "iteration" of the shortest synchronization except the last letter starts with 0, continues with some $r-1$ letters and finishes with 1. Therefore, the shortest c.s.w. consists of words from the set $0\{0,1\}^r1$ and the last letter 1. The lemma is proved.

Let us finish the proof of Theorem 3. Let w be the shortest c.s.w. of $\mathscr{C}(r)$. Similarly to the proof of Theorem 2 we obtain that $w = w_1 \ldots w_z 1$, where $w_1, \ldots, w_z \in 0\{0,1\}^r1$ and $\gamma(P, w_1 \ldots w_h) = \gamma(Q, w_1 \ldots w_h)$ for any $h \in 1..z$. Hence w, being a word over $0\{0,1\}^r1 \cup \{1\}$, is the shortest c.s.w. of the PFA $\mathscr{A}(k)$. Therefore, the word w in the PFA $\mathscr{A}(k)$ has 3^k letters and just the last letter of w is equal to c. Therefore, $|w| = (3^k-1)(r+2)+1$. Recall that $n = 3(2r+1)2^r$. Finally,

$$\log_2 n = \log_2 3 \ + r + \log_{m-1}(2r+1) \ > (2r+1)/2, \ 2^r = \frac{n}{3(2r+1)} > \frac{n}{6\log_2 n}.$$

$$|w| = (3^k-1)(r+2)+1 \geq 3^{2^r} r > 3^{\frac{n}{6\log_2 n}} \ .$$

The theorem is proved.

Acknowledgement. The author acknowledges support from the Presidential Programm for young researchers, grant MK-266.2012.1. The author also acknowledges Jean-Eric Pin for a careful reading and useful recommendations.

References

1. Ananichev, D.S., Volkov, M.V.: Synchronizing monotonic automata. Theoret. Comput. Sci. 327, 225–239 (2004)
2. Černý, J.: Poznámka k homogénnym experimentom s konečnými automatmi. Mat.-Fyz. Cas. Slovensk. Akad. Vied. 14, 208–216 (1964) (in Slovak)
3. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19, 500–510 (1990)
4. Gazdag, Z., Ivan, S., Nagy-Gyorgy, J.: Improved upper bounds on synchronizing nondeterministic automata. Information Processing Letters 109(17), 986–990 (2009)
5. Kari, J.: Synchronizing finite automata on eulerian digraphs. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 432–438. Springer, Heidelberg (2001)
6. Imreh, B., Steinby, M.: Directable nondeterministic automata. Acta Cybernetica 14, 105–115 (1999)
7. Ito, M., Shikishima-Tsuji, K.: Some results on directable automata. In: Karhumäki, J., Maurer, H., Păun, G., Rozenberg, G. (eds.) Theory Is Forever. LNCS, vol. 3113, pp. 125–133. Springer, Heidelberg (2004)
8. Ito, M.: Algebraic Theory of Automata and Languages. World Scientific, Singapore (2004)
9. Martyugin, P.V.: Lower bounds for the length of the shortest carefully synchronizing words for two- and three-letter partial automata. Diskretn. Anal. Issled. Oper. 15(4), 44–56 (2008)
10. Martyugin, P.V.: A Lower Bound for the Length of the Shortest Carefully Synchronizing Words. Russian Mathematics (Iz. VUZ) 54(1), 46–54 (2010)
11. Trahtman, A.N.: Modifying the Upper Bound on the Length of Minimal Synchronizing Word. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 173–180. Springer, Heidelberg (2011)
12. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
13. Volkov, M.V.: Synchronizing automata preserving a chain of partial orders. Theoret. Comput. Sci. 410, 3513–3519 (2009)

Random Generation of Deterministic Acyclic Automata Using the Recursive Method*

Sven De Felice and Cyril Nicaud

LIGM, Université Paris-Est, 77454 Marne-la-Vallée Cedex 2, France
defelic@univ-mlv.fr, nicaud@univ-mlv.fr

Abstract. In this article, we propose a uniform random generator for accessible deterministic acyclic automata with n states, which is based on the recursive method. The generator has a preprocessing that requires $\mathcal{O}(n^3)$ arithmetic operations, and, once it is done, can generate acyclic automata using $\mathcal{O}(n)$ arithmetic operations for each sample. We also propose a lazy version of the algorithm that takes advantage of the typical shape of random acyclic automata to reduce experimentally the preprocessing. Using this algorithm, we provide some statistics on acyclic automata with up to 1000 states.

1 Introduction

The field of random generation has been very active in the last two decades, developing general techniques based on combinatorics and probabilities to produce efficient random samplers for combinatorial structures that appear in computer science. The main focus is to build an algorithm for a given combinatorial set E, i.e. a set together with a size function, which takes an integer n as input and produces an element of E of size n uniformly at random. The interest in random generators comes from both practice and theory: they are commonly used as an alternative to benchmarks, in order to test the efficiency of implementations; they are also widely used by theorists in their works on describing the typical properties of large random objects, which is a cornerstone in average case analysis of algorithms [7].

This article deals with the random generation of acyclic deterministic automata with n states on a fixed finite alphabet. Acyclic automata are automata recognizing finite languages, and, as such, form an important subclass of automata. They are especially used in applications such as in linguistics where the languages of interest are essentially finite. A better understanding of their combinatorics, of their typical behavior and the average cases analysis of algorithms that handle that kind of structures is a long term goal of our work.

In random generation standards, we aim at designing algorithms that can produce experimental statistics on objects of size at least 1000, in a reasonable

* This work is supported by the French National Agency (ANR) through ANR-10-LABX-58, through ANR-2010-BLAN-0204 and through ANR-JCJC-12-JS02-012-01.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 88–99, 2013.

amount of time: though probabilistic and combinatorial in nature, random generator are above all algorithms and the main concern when designing them is to optimize their complexity, in order to allow the generation of objects that are large enough.

In the sequel, we present how deterministic acyclic automata can be generated using a technique known as the *recursive method*. It is based on an inductive specification of the objects of interest and consists in two steps: a preprocessing which is costly but need to be done only once, and the random generation itself. This method originates in [10] and was systematized in [8] to classes of combinatorial structures that can be described using the symbolic method [7, Ch. I]. Acyclic automata cannot be described in such a way, and the aim of this article is to show that the technique applies anyway, though not automatically. Our algorithm has a preprocessing step using $\mathcal{O}(n^3)$ arithmetic operations, and can then generate deterministic acyclic automata with n states using a linear number of arithmetic operations. We also propose an improved version that is very efficient in practice to lower the complexity of the preprocessing. This efficiency relies on the typical shape of a random acyclic automaton: we formulate its complexity as a function of a parameter of its output (its width, defined in Section 4.5) which seems to grow very slowly in practice. Though requiring a detailed probabilistic analysis for this observation to be mathematically established, we were able to compute some statistics on automata with up to 1000 states within a few minutes using our improved algorithm. This is an example where information on typical structures helps in designing better algorithms.

One of our motivation for studying acyclic automata comes from the theory of *automaton groups*. Consider a letter-by-letter deterministic transducer, and the different functions from A^* to A^* it realizes when taking all the possibilities for the initial state. Assume that some conditions ensure that this functions are permutations of A^* and consider the group they generate. That kind of groups are especially rich and have been studied by both mathematicians and computer scientists (see [1] for more details). In particular, Antonenko [3] gave some conditions on the shape of the transducer that ensure that the generated group is finite, and these conditions make use of acyclic automata. The quantitative study of Antonenko automata therefore requires more knowledge on deterministic acyclic automata, and could be a first step toward understanding the combinatorics behind that kind of groups.

Related works. In [4], the first author and Carnino proposed a solution to the same problem based on Markov chain techniques. This yields an elegant and easily adaptable algorithm[1], that produces an acyclic automaton with almost uniform distribution. The major drawback of this method is that there is no estimation of the bias induced when halting the algorithm after a given number of iterations, making the design of the algorithm difficult when uniformity is important[2].

[1] For instance, they described a Markov chain on minimal acyclic automata only.

[2] Using Markov chain terminology: the mixing time of the chain is not known and seems to be difficult to estimate.

The recursive method is based on the combinatorial properties of the objects to be generated. In [9], Liskovets presented inclusion-exclusion results on the number of deterministic acyclic automata and on some related quantities. We will use his work at several stages of the design of our algorithm. Prior to this work, Domaratzki, Kisman and Shallit proposed lower and upper bounds on the number of acyclic automata in [5,6].

In [2], Almeida, Moreira and Reis gave a solution to a related problem: they proposed an algorithm that generate *exhaustively* all minimal automata with a given number of states. Because the number of minimal acyclic automata grows very fast, such an exhaustive generator is limited to small number of states (there are more than $7 \cdot 10^{14}$ minimal acyclic automata with 15 states on a two-letter alphabet), but it has the virtue of checking every object unlike a random generator. Both solutions are relevant, in different manners, when testing conjectures.

2 Definition and Notations

For any integer $n \geq 1$, let $[n] = \{1, \ldots, n\}$ be the set of integer from 1 to n. If n and m are two non-negative integers, we denote by $\mathrm{Surj}(n, m)$ the number of surjections from $[n]$ onto $[m]$. The value of $\mathrm{Surj}(n, m)$ can be computed recursively using the formula: $\mathrm{Surj}(n, m) = m \cdot \mathrm{Surj}(n-1, m-1) + m \cdot \mathrm{Surj}(n-1, m)$, with initial conditions $\mathrm{Surj}(n, 1) = 1$ and $\mathrm{Surj}(n, m) = 0$ if $m > n$.

Let A be a finite alphabet, a *deterministic automaton* (or *automaton* for short) on A is a tuple (Q, δ, q_0), where Q is a finite set of *states*, δ is the *transition function*, a possibly partial mapping from $Q \times A$ to Q, and $q_0 \in Q$ is the *initial state*. If $p, q \in Q$ and $a \in A$ are such that $\delta(p, a) = q$, then (p, a, q) is the *transition* from p to q labelled by a, and is denoted by $p \xrightarrow{a} q$. An automaton $\mathcal{A} = (A, Q, \delta)$ is classically seen as a labelled directed graph whose set of vertices is Q and whose edges are the transitions of $\mathcal{A}$.

An automaton is *accessible* (or *initially connected*) when for every state p there exists a path starting from the initial state that ends at p. The transition function is extended to $Q \times A^*$ by morphism, setting $\delta(p, \varepsilon) = p$ for every $p \in Q$ and $\delta(p, ua) = \delta(\delta(p, u), a)$ when everything is defined, and undefined otherwise.

An automaton is *acyclic* when its graph is acyclic. A *source* of an automaton $\mathcal{A}$ is a state with no incoming transition. An acyclic automaton always has at least one source, and accessible acyclic automata are acyclic automata with exactly one source.

In the sequel, the set of states of an automaton with n states will almost always be $[n]$. If $\mathcal{A}$ is an automaton of set of states $[n]$ and X is a set of n positive integers, the *relabelling of $\mathcal{A}$ using X* is the automaton obtained from $\mathcal{A}$ when changing the states labels by elements of X, while respecting their relative order: if $p < q$ in $\mathcal{A}$ then the new label of p is smaller than the one of q. Notice that there is only one way to do so.

Important : We are not interested in final states except in the experimental section of this article (Section 5). We will therefore call "automaton" a determin-

istic automaton without final states throughout the article, and denote classical deterministic automata by "automata with final states".

3 Combinatorics of Acyclic Automata

3.1 Liskovet's Formula

In [9], Liskovets establishes formulas to count the number of acyclic automata. These results relies on the inclusion-exclusion method [11], which is a classical and elegant technique yielding formulas with alternating sums. One such result is the following (set $r = 1$ in Eq. (3) of [9]): if $a_k(n)$ denote the number of labelled acyclic automata with n states on a k-letter alphabet, then

$$a_k(n) = \sum_{t=0}^{n-1} \binom{n}{t} (-1)^{n-t-1} (t+1)^{k(n-t)} a_k(t). \tag{1}$$

Let $\alpha_k(n, s)$ denote the number of labelled acyclic automata with n states on a k-letter alphabet that have exactly s sources. Using almost the same proof as Liskovets' one can obtain the following formula:

$$\alpha_k(n, s) = \binom{n}{s} \sum_{i=0}^{n-s} \binom{n-s}{i} (-1)^i a_k(n-s-i) \cdot (n-s-i+1)^{k(s+i)}. \tag{2}$$

Lemma 1. *The values of $a_k(m)$ for every $m \in [n]$ can be computed using $\mathcal{O}(n^2)$ arithmetic operations and storing $\mathcal{O}(n^2)$ integers. The values of $\alpha_k(m, t)$ for every $m \in [n]$ and $t \in [s]$, with $s \leq n$, can be computed using $\mathcal{O}(sn^2)$ arithmetic operations and storing $\mathcal{O}(n^2)$ integers.*

Though originating from a combinatorial description, inclusion-exclusion formulas are not always very useful when designing efficient random generators[3], because of the complications inherent to the exclusions of subsets, which correspond to the minus signs in the formulas.

We will however use Liskovets' formulas later in our algorithms, as a shortcut to compute the required values more efficiently. For now, we need a more straightforward combinatorial decomposition that is amenable to the recursive method; this is the purpose of next section.

3.2 Decomposition Using Sources and Secondary Sources

Our decomposition consists intuitively in repeatedly pruning the automaton by removing its sources. This is a classical idea coming from the enumeration of acyclic directed graphs. If $\mathcal{A}$ is an acyclic automaton, a state of $\mathcal{A}$ is a *secondary source* if it is not a source and if all its incoming transitions come from sources.

[3] There is a noticeable exception when rejection techniques can be applied, but it does not appear to be the case for acyclic automata.

In other words, p is a secondary source if it has no more incoming transition when the sources of $\mathcal{A}$ are removed.

Let $\mathcal{A}$ be an acyclic automaton with n states and s sources on a k-letter alphabet; when the s sources and their outgoing transitions are removed from $\mathcal{A}$, what remains is an acyclic automaton $\mathcal{B}$ with $n-s$ states. We obtain a formula by partitioning the possibilities according to the number u of sources of $\mathcal{B}$, that is, the number of secondary sources of $\mathcal{A}$. Thinking backward, for any given $\mathcal{B}$ with $n-s$ states and u sources, one can reconstruct an automaton $\mathcal{A}$ by doing the following:

1. Choose the set of labels $Y \subseteq [n]$ for the s sources of $\mathcal{A}$ and relabel $\mathcal{B}$ following $[n] \setminus Y$.
2. For every transition starting from one of the s sources, choose whether it is undefined or not, and if it is not, where it ends amongst the $n-s$ possibilities. This must be done in such a way that every secondary source has at least one incoming transition.

There are $\binom{n}{s}$ ways to choose the set of labels. Let $\beta_k(n,s,u)$ be the number of possibilities for the second item. We count the number of valid configurations for the ks transitions starting from a source using the number i of transitions that end in a secondary source as a parameter: at fixed i, a valid configuration is obtained by choosing the i transitions amongst the ks possibilities, how they are mapped to their u possible ending states in a surjective way (since each secondary source must have an incoming transition from a source) and the ending state of each of the $ks-i$ remaining transitions, which can be either undefined or one of the $n-s-u$ states that are neither a source nor a secondary source. Hence, the number of valid configurations is given by:

$$\beta_k(n,s,u) = \sum_{i=u}^{ks} \binom{ks}{i} \cdot \mathrm{Surj}(i,u) \cdot (n-s-u+1)^{ks-i}. \tag{3}$$

This yields the following formula for the number of acyclic automata with $s < n$ sources:

$$\alpha_k(n,s) = \sum_{u=1}^{\min(ks,n-s)} \binom{n}{s} \cdot \beta_k(n,s,u) \cdot \alpha_k(n-s,u), \tag{4}$$

since there are $\alpha_k(n-s,u)$ ways to choose $\mathcal{B}$. Of course, we also have $\alpha_k(n,n) = 1$.

Remark that for computational purposes, Eq. (4) is not as good as Liskovet's formula, which can be computed in time $\mathcal{O}(n^2)$ according to Lemma 1. The gain is the combinatorial description that can be directly turned into a random generator, provided all the required quantities are already computed.

3.3 Another Description for $\beta_k(n,s,u)$

For $\mathcal{T}$, $\mathcal{U}$ and $\mathcal{R}$ three finite sets such that $\mathcal{U}$ and $\mathcal{R}$ are disjoint, consider the family $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$ of partial functions from $\mathcal{T}$ to $\mathcal{U} \cup \mathcal{R}$ such that every $q \in \mathcal{U}$

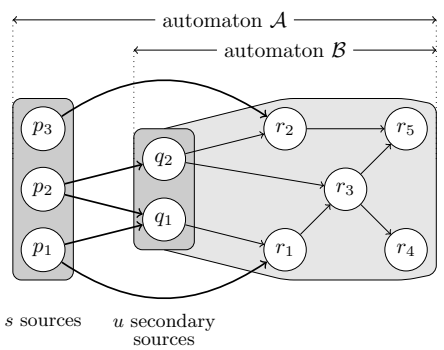


Fig. 1. Main decomposition on an automaton with $n = 10$ states on a two-letter alphabet: To build an acyclic automaton $\mathcal{A}$ with $s = 3$ sources and $u = 2$ secondary sources, one has to choose $\mathcal{B}$, an acyclic automaton with 2 sources, and to set the transitions starting from the sources of $\mathcal{A}$: they can either be defined or undefined, cannot end in a source of $\mathcal{A}$, and must cover all the sources of $\mathcal{B}$. The number of ways to do so is $\beta_2(10, 3, 2) = \gamma(6, 2, 5)$.

has at least one preimage. Let $\gamma(t, u, r)$ be the cardinality of $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$ when $|\mathcal{T}| = t$, $|\mathcal{U}| = u$ and $|\mathcal{R}| = r$.

Recall that $\beta_k(n, s, u)$ counts the number of ways to define the transitions starting from the s sources such that each of the u secondary sources has at least one incoming transition, with a total of n states. Let $\mathcal{T}$ denote the pairs (s, a) where s is a source and a is a letter, let $\mathcal{U}$ denote the set of secondary sources, and let $\mathcal{R}$ denote the set of states that are neither a source nor a secondary source. The function that maps each (s, a) to its ending state, when it exists, is a partial function from $\mathcal{T}$ to $\mathcal{U} \cup \mathcal{R}$ such that every $q \in \mathcal{U}$ has at least one preimage: it is therefore an element of $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$. Conversely, every such function corresponds to a valid choice for the transitions starting from a source. Hence we have the identity $\beta_k(n, s, u) = \gamma(ks, u, n - s - u)$.

The inductive description of $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$ is the following. Let x be any element of $\mathcal{T}$. The functions in $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$ fall in two categories: those such that x is the unique preimage of an element of $\mathcal{U}$ and the other ones. A function of the first category restricted to $\mathcal{T}\setminus\{x\}$ is exactly an element of $\mathcal{G}(\mathcal{T}\setminus\{x\},\mathcal{U}\setminus\{q\},\mathcal{R})$: there are $u \cdot \gamma(t - 1, u - 1, r)$ such possibilities. Otherwise, the restriction to $\mathcal{T} \setminus \{x\}$ is exactly an element of $\mathcal{G}(\mathcal{T} \setminus \{x\},\mathcal{U},\mathcal{R})$ and there are $(u + r + 1)\gamma(t - 1, u, r)$ possibilities, $u + r - 1$ being the number of different ways to choose the image of x including the case when it is undefined. We therefore obtained that:

$$\gamma(t, u, r) = u \cdot \gamma(t - 1, u - 1, r) + (r + u + 1) \cdot \gamma(t - 1, u, r). \tag{5}$$

Moreover this formula has a combinatorial meaning, since it is a discussion on the different possibilities for the image of a given element of $\mathcal{T}$. The boundary conditions are: $\gamma(t, 0, r) = (r + 1)^t$ and $\gamma(t, u, r) = 0$ for $t < u$.

3.4 Remark on Labelled Combinatorial Structures

The combinatorial study of labelled structures, i.e. when the n vertices are labelled with the elements of $[n]$, is often easier than the one of unlabelled structures, since symmetries that can appear in the unlabelled case usually make the counting more complicated.

The situation is different for structures that are *rigid*, that is, structures with no symmetry[4], since the number of labelled structures is exactly $n!$ times the number of unlabelled ones. Fortunately, this is the case for deterministic automata when they are accessible, as explained in [9]. In particular, the number of unlabelled accessible acyclic automata is $\frac{1}{n!}\alpha_k(n,1)$ and they can be randomly generated as labelled structures and still being uniform as unlabelled automata.

4 Random Generator

4.1 The Recursive Method

As stated in the introduction, acyclic automata cannot not be directly described using the symbolic method [7, Ch. I]. Therefore, we cannot use the automatic translation into a random generator proposed in [8]. The purpose of this section is to adapt the method to our specific formulas of Section 3 in order to get such a generator. Remark informally that our formulas are always of the form $\lambda_n = \sum_i \lambda_{n,i}$, where parameter i has a combinatorial meaning. Assume that λ_n and all the $\lambda_{n,i}$ have already been computed, it is then easy to generate the value of parameter i for a uniform object of size n, since $\mathbb{P}_n(\text{parameter} = i) = \frac{\lambda_{n,i}}{\lambda_n}$. The idea is to choose i with correct probability, reducing the problem to the uniform random generation of smaller objects. Some additional constructions can be required to finally build the result, depending on the combinatorial construction that leads to the formula for λ_n.

4.2 Application to Acyclic Automata

The method described above can directly be applied to generate uniformly at random accessible acyclic automata. The first unoptimized version of the algorithm for an acyclic automaton with s sources is the following:

1. Compute all the values of $a_k(m)$, $\alpha_k(m,s)$, $\beta_k(m,s,u)$ and also of $\binom{ks}{i}$ and $\mathrm{Surj}(i,u)$ for every $m \in [n]$, every $s \in [m]$, every $u \in [m-s]$ and every $i \in [m]$.
2. Use Eq. (4) with s sources to generate the value of u with correct probability: The number of secondary sources takes value u with probability
$$\frac{\binom{n}{s} \cdot \beta_k(n,s,u) \cdot \alpha_k(n-s,u)}{\alpha_k(n,s)}. \tag{6}$$
3. Recursively generate an acyclic automaton $\mathcal{B}$ of size $n-s$ having u sources.
4. Choose the set of source labels X, and relabel $\mathcal{B}$ following $[n] \setminus X$.
5. Use Eq. (3) to generate the number of transitions starting from a source and ending in a secondary source. The number of such transitions takes value i with probability
$$\frac{\binom{ks}{i} \cdot \mathrm{Surj}(i,u) \cdot (n-s-u+1)^{ks-i}}{\beta_k(n,s,u)}. \tag{7}$$

[4] Formally, the group of structure automorphisms is trivial.

6. Generate the transitions starting from sources with correct probability, by choosing the i transitions ending in a secondary source, the surjective way they are associated to secondary sources, and by choosing the ending state of the other ones (or whether they are undefined).

Lemma 2. *The method above produces a random acyclic automaton with n states and s sources uniformly at random.*

The proof is done by induction on n and follows from the unicity of our decomposition. A direct computation of the probabilities using Eq. (6), Eq. (7), the induction hypothesis and the probability associated with step 6 yields that $\mathcal{A}$ is produced with uniform probability.

This straightforward way to turn the formulas of Section 3 into a random generator is constitutive of the recursive method. Notice that a random generator for accessible acyclic automata of size n is obtained by setting $s = 1$.

Also remark that the first step is the main limitation of this method, since it requires quite some time and space to compute and store all the needed results. Using Eq. (3) and Eq. (4) to perform the computations require $\mathcal{O}(n^4)$ arithmetic operations. However, it is important to notice that this preprocessing must be done only once. Thereafter, as will be explained in Section 4.4, the random generation of an acyclic automaton with n states is done in time $\mathcal{O}(n)$.

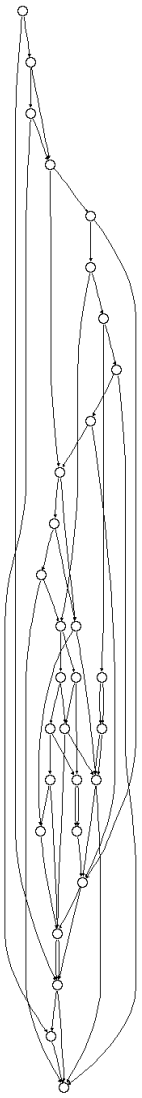
In the sequel we will show how to improve the complexity of the algorithm.

4.3 Using γ Instead of β

In Section 3.3 is explained that $\beta_k(n, s, u) = \gamma(ks, u, r)$, where both quantities describe how to link the sources to the secondary sources, in two different ways. The formula for $\gamma(ks, u, r)$ is more advantageous in terms of time complexity, since one can compute all the $\mathcal{O}(n^3)$ needed values for γ using $\mathcal{O}(n^3)$ arithmetic operations with Eq. (5). Using γ instead of β_k, we also do not need to compute the values of $\mathrm{Surj}(i, u)$ anymore, since the combinatorial decomposition that leads to Eq. (5) can be directly turned into an algorithm: in order to generate a random element f of $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$, start from any $x \in \mathcal{T}$ then pick a random number d in $[\gamma(t, u, r)]$. If $d \leq u\cdot\gamma(t-1, u-1, r)$, choose uniformly an element q of $\mathcal{U}$ and set that x is the unique preimage of q by f; it remains to recursively draw the restriction of f to $\mathcal{T}\setminus\{x\}$ in $\mathcal{G}(\mathcal{T}\setminus\{x\},\mathcal{U}\setminus\{q\},\mathcal{R})$. If $d > u\cdot\gamma(t-1, u-1, r)$, choose uniformly the image of x by f in $\mathcal{U}\cup\mathcal{R}\cup\{\perp\}$, where $f(x) = \perp$ means that $f(x)$ is undefined, and recursively draw the restriction of f to $\mathcal{T}\setminus\{x\}$ in $\mathcal{G}(\mathcal{T}\setminus\{x\},\mathcal{U},\mathcal{R})$. The complexity of generating an element of $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$ is therefore linear in $|\mathcal{T}|$, using adapted data structures.

4.4 Algorithms and Complexity

Our main algorithms are given in Fig. 2, page 96. As explain before, one must first compute the values for α_k, γ and the binomial coefficients that are needed in the process. In particular, γ having three parameters that can all three be



RandomNumberOfSecondarySources(n,s)

if $n = s$ **then**
  return 0
$d \leftarrow$Random($[\alpha_k(n, s)]$)
$u \leftarrow 0$
while $d > 0$ **do**
  $u \leftarrow u + 1$
  $d \leftarrow d - \binom{n}{s} \cdot \gamma(ks, u, n - s - u) \cdot \alpha_k(n - s, u)$
return u

RandomlySetTransitionsFromSources($\mathcal{T}$,$\mathcal{U}$,$\mathcal{R}$,δ)

// Transitions are added to δ during the process
if $\mathcal{T} = \emptyset$ **then**
  return
$(p, a) \leftarrow$Remove the first element of $\mathcal{T}$
if Random($[\gamma(|\mathcal{T}|, |\mathcal{U}|, |\mathcal{R}|)]$) $\leq u \cdot \gamma(|\mathcal{T}| - 1, |\mathcal{U}| - 1, |\mathcal{R}|)$ **then**
  $q \leftarrow$Remove the first element of $\mathcal{U}$
  $\delta(p, a) \leftarrow q$
else
  $q \leftarrow$ Random($\mathcal{U} \cup \mathcal{R} \cup \{\perp\}$)
  if $q \neq \perp$ **then**
    $\delta(p, a) = q$
RandomlySetTransitionsFromSources($\mathcal{T}$,$\mathcal{U}$,$\mathcal{R}$,δ)

RandomAcyclicAutomaton(n,s,δ)

if $n = s$ **then**
  $\delta =$empty function
  return
$u \leftarrow$ RandomNumberOfSecondarySources(n,s)
$\mathcal{B} \leftarrow$RandomAcyclicAutomaton($n - s$,u,δ)
$\mathcal{T} \leftarrow \emptyset$
for $p \in \{n - s + 1, \ldots, n\}$ and $a \in A$ **do**
  Add (p, a) in $\mathcal{T}$
$\mathcal{U} \leftarrow \{n - s - u + 1 \ \ldots, n - s\}$
$\mathcal{R} \leftarrow [n - s - u]$
RandomlySetTransitionsFromSources($\mathcal{T}$,$\mathcal{U}$,$\mathcal{R}$,δ)

Fig. 2. On the left a random acyclic automaton with 30 states on a two-letter alphabet. On the right our main algorithms. The states are labelled in a specific way during the process, but this does not change the uniformity of the unlabelled result, since we follow the correct counting numbers for the sources, secondary sources and transitions between them.

proportional to n, there are $\Theta(n^3)$ numbers to store. Thanks to Eq. (5), each new value of γ is computed in a constant number of arithmetic operations, giving the following result.

Theorem 1 (Preprocessing). *The preprocessing step of the algorithm, where all the possibly needed values of α_k, γ and $\binom{n}{s}$ are computed can be done using $\mathcal{O}(n^3)$ arithmetic operations and the memory to store $\mathcal{O}(n^3)$ numbers.*

Once the preprocessing is done, the random generation can be performed efficiently, as stated in the following theorem.

Theorem 2 (Generation). *After the preprocessing, the random generation of an acyclic automaton with n state can be done in a linear number of arithmetic operations and random generations of integers.*

Notice that, as it is usually the case when using the recursive method, the numbers involved in the computations are huge. This is why our theorems are stated in terms of number of arithmetic operations: one cannot consider that such operations can be done in constant time in real implementations. Alternative algorithms for the recursive method that use floating point arithmetic have been studied, but this is beyond the scope of this article.

4.5 A Lazy Strategy

A common strategy for that kind of algorithms is the lazy strategy, which consists in computing the values for α_k and γ only when needed. They are still stored, but the computations are done on the fly. This strategy proves to be very efficient in practice in our case, because of the specific shape of a uniform random acyclic automata (as depicted in Fig. 3).

If $\mathcal{A}$ is an acyclic automaton, let sources($\mathcal{A}$) be its number of sources and let pruned($\mathcal{A}$) be the acyclic automaton obtained when removing the sources and their outgoing transitions. We define the *width* width($\mathcal{A}$) of an acyclic automaton $\mathcal{A}$ by $\text{width}(\mathcal{A}) = \max\{\text{sources}(\mathcal{A}), \text{width}(\text{pruned}(\mathcal{A}))\}$ if $\mathcal{A}$ has at least one state and $\text{width}(\mathcal{A}) = 0$ otherwise. The width of an acyclic automaton $\mathcal{A}$ is therefore the maximum size of a layer of sources obtained when repeatedly pruning $\mathcal{A}$.

We aim at using the width of the output automaton as a parameter for the complexity of our algorithm. The main motivation for this is the typical flat shape of a random acyclic automaton (Fig. 3). Assume that the algorithm produces an automaton $\mathcal{A}$ of width w. Then the various values taken by u in the algorithms are always smaller than or equal to w, and those taken by $|\mathcal{T}|$ smaller than or equal to $k \cdot w$. Therefore, the lazy strategy only computes values for $\gamma(t, u, r)$ for $t \leq k \cdot w$, $u \leq w$ and $r \leq n$, which requires $\mathcal{O}(n \cdot w^2)$ time and space. However, the idea would not work without a shortcut to compute the values of $\alpha_k(n, s)$, which is needed in the algorithms; indeed, the variable u in Eq. (4) takes values that are bigger than w, especially considering the inductive nature of this equation. Fortunately, we can use the inclusion-exclusion formula of Eq. (2) instead, which can be computed using $\mathcal{O}(n^2 \cdot w)$ arithmetic operations according to Lemma 1. This gives the following result.

Fig. 3. The shape of a random acyclic automata with 200 states on a two-letter alphabet. The initial state is on the left, and we have represented the number of sources seen at each step when repeatedly pruning the automaton. The width of this automaton is 6, and its shape is typical of what is observed under the uniform distribution.

Theorem 3 (Lazy Strategy). *Using the lazy strategy in the random generator, the generation algorithm (including the computation of the required values) needs $\mathcal{O}(n^2 \cdot w)$ arithmetic operations, where w is the width of the generated acyclic automaton.*

Theorem 3 is not enough to prove the generic efficiency of the lazy strategy: one would also needs to show that, with high probably, a random uniform acyclic automaton with n states has a small width with respect to n. Experimentation indicates that it should be true, and trying to prove this is ongoing work. However, there is no reason to avoid this strategy, which is at least as good as the classical one even when the generated acyclic automaton has a large width.

5 Conclusion and Experiments

Using some optimizations on an inductive decomposition of acyclic automata, we proposed a random generator that is efficient enough to make experimental statistics on automata of size up to 1000 or a bit more. We relied on the alternative description of β_k by γ to lower the initial complexity, and used inclusion-exclusion formulas to make the lazy strategy possible.

We have implemented our random generator in the interpreted language Python, which is clearly not the best choice for computational speed. However, we could easily generate accessible automata of size 1000 in a reasonable amount of time (a few minutes to generate 100 automata on a personal computer).

In our experiments, we considered that each state is final with probability $\frac{1}{2}$. In our first test, we computed experimentally the probability that a random acyclic automaton with final states is minimal. Notice that minimality is easier to check for acyclic automata than for general automata (see [2] for the details). The results for alphabets of size 2 and 3 are depicted in Fig. 4.

We also computed the number of words in the finite language recognized by a random acyclic automaton with final states. Since it grows very fast, we switch to logarithms by calculating logarithm of the geometric mean of the number of recognized words. It seems to indicate that random acyclic automata are a very compact way to describe huge random sets of words (see Fig. 5).

Acknowledgments. we would like to thanks Arnaud Carayol for the very fruitful discussions we had during the preparation of this article.

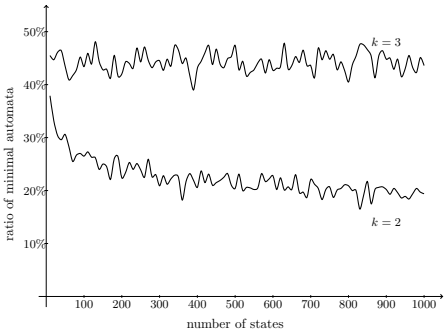
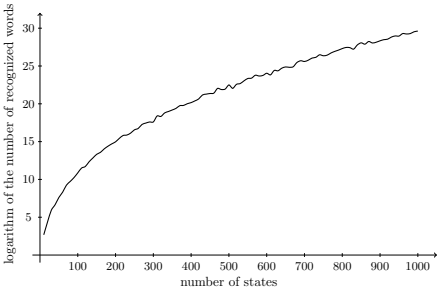


Fig. 4. The ratio of minimal automata for alphabet of size 2 and 3. Each curve has been obtained by generating 1000 acyclic automata for 101 different sizes, from 10 to 1000. Note that one can significantly increase the ratio by forcing states with no outgoing transition to be final (hoping that there is only one such state, which is often the case experimentally).

Fig. 5. The natural logarithm of the number of recognized words for an alphabet of size 2. The curve has been obtained by generating 1000 acyclic automata for 101 different sizes, from 10 to 1000. It is difficult to guess the function behind that kind of experimental curves, but subexponential growth like $x \mapsto e^{\sqrt{x}}$ is a possibility.



References

1. Akhavi, A., Klimann, I., Lombardy, S., Mairesse, J., Picantin, M.: On the finiteness problem for automaton (semi)groups. IJAC 22(6) (2012)
2. Almeida, M., Moreira, N., Reis, R.: Exact generation of minimal acyclic deterministic finite automata. Int. J. Found. Comput. Sci. 19(4), 751–765 (2008)
3. Antonenko, A.S.: On transition function of mealy automata with finite growth. Matematychni Studii 29(1) (2008)
4. Carnino, V., De Felice, S.: Sampling different kinds of acyclic automata using Markov chains. TCS 450, 31–42 (2012)
5. Domaratzki, M.: Improved bounds on the number of automata accepting finite languages. In: Ito, M., Toyama, M. (eds.) DLT 2002. LNCS, vol. 2450, pp. 209–219. Springer, Heidelberg (2003)
6. Domaratzki, M., Kisman, D., Shallit, J.: On the number of distinct languages accepted by finite automata with n states. Journal of Automata, Languages and Combinatorics 7(4), 469–486 (2002)
7. Flajolet, P., Sedgewick, R.: Analytic combinatorics. Cambridge Univ. Pr. (2009)
8. Flajolet, P., Zimmermann, P., Cutsem, B.V.: A calculus for the random generation of labelled combinatorial structures. TCS 132(2), 1–35 (1994)
9. Liskovets, V.A.: Exact enumeration of acyclic deterministic automata. Discrete Applied Mathematics 154(3), 537–551 (2006)
10. Nijenhuis, A., Wilf, H.: Combinatorial algorithms. Computer Science and Applied Mathematics. Academic Press (1975), `http://bks9.books.google.fr/books?id=09BQAAAAMAAJ`
11. Stanley, R.P.: Enumerative Combinatorics, Cambridge Univ. Pr. (2000)

Boolean Language Operations on Nondeterministic Automata with a Pushdown of Constant Height

Viliam Geffert[1,⋆], Zuzana Bednárová[1,⋆],
Carlo Mereghetti[2], and Beatrice Palano[2]

[1] Dep. Computer Sci., P. J. Šafárik Univ., Jesenná 5, 04154 Košice, Slovakia
viliam.geffert@upjs.sk, ivazuzu@eriv.sk
[2] Dip. Informatica, Univ. degli Studi di Milano, v. Comelico 39, 20135 Milano, Italy
mereghetti@di.unimi.it, palano@di.unimi.it

Abstract. We study the size-cost of Boolean operations on *constant height nondeterministic pushdown automata*, i.e. on pushdown automata with a constant limit on the size of the pushdown. For *intersection*, we show an exponential simulation and prove that the exponential blow-up is necessary. For *union*, instead, we provide a linear trade-off while, for *complement*, we show a double-exponential simulation and prove a single-exponential lower bound.

Keywords: descriptional complexity, finite state automata, regular languages, nondeterministic pushdown automata.

1 Introduction

A primary task in the area of descriptional complexity is the analysis of how succinctly a given device is able to describe a certain class of languages. Quite often, languages that are more "complex" are obtained from "simpler" ones by the use of some "standard" language operations in the class, which requires evaluating the cost of implementing these language operations by the given device.

The largest amount of results devoted to descriptional complexity is related to *regular languages*. Among others, these languages are representable by regular grammars, expressions, and several variants of automata, starting from the original model of a *deterministic finite state automaton* (DFA) and ranging over enhanced models with additional features, like nondeterminism, alternation, probabilism, quantum or two-way versions... For a brief survey, see, e.g., [6,12].

In this paper, we study the descriptional power of a *constant height nondeterministic pushdown automaton* (*constant height* NPDA). Such machine is a traditional pushdown automaton (see, e.g., [7]) with a constant limit on the height of the pushdown, not depending on the input length. This model was introduced in [4], together with its *deterministic version* (*constant height* DPDA). It is easy to see that such devices accept regular languages. However, a representation by

⋆ Supported by the Slovak grant contracts VEGA 1/0479/12 and APVV-0035-10.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 100–111, 2013.

constant height pushdown automata can be more succinct. In [4], an optimal exponential gap was shown between the sizes of NPDAs and of *nondeterministic finite state automata* (NFAs). The same gap was found for the deterministic case, between DPDAs and DFAs. Converting a constant height NPDA into an equivalent DPDA uses, and also requires, a double-exponential blow-up [1].

Here we concentrate on a classical problem, the *cost of Boolean operations*, on constant height NPDAs. There exists a wide literature on this issue with respect to, e.g., finite state automata [6,13], regular expressions [3,5], or grammars [8,9]. The cost of these operations for constant height DPDAs was also investigated, in [2] (see also Tab. 1 in Sect. 5).

First, we analyze the cost of *intersection*. Given two constant height NPDAs A and B with respective sets of states Q_A, Q_B, pushdown alphabets Γ_A, Γ_B, and pushdown heights h_A, h_B, we design an NPDA for $L(A) \cap L(B)$ with at most $\|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\| \cdot \|Q_B\|$, states, $\|\Gamma_B\|$ pushdown symbols, and the pushdown height h_B. Since the roles of A and B can be swapped, the number of states is actually exponential in $h = \min\{h_A, h_B\}$. In the worst case, an exponential blow-up is necessary: for each fixed $c \geq 2$, we exhibit $\{L'_n\}_{n\geq 1}$ and $\{L''_n\}_{n\geq 1}$, two families of languages over a $(c+1)$-letter alphabet, such that: *(i)* both L'_n and L''_n are accepted by NPDAs with $O(c)$ states, c pushdown symbols, and the pushdown height n, but *(ii)* their intersection cannot be accepted by an NPDA in which both the number of states and the pushdown height are below $c^{n/3-O(\log n)}$.

The *union* operation, instead, turns out to be easy. We propose an NPDA for $L(A) \cup L(B)$ with a linear trade-off, namely, with at most $\max\{1, |h_A - h_B|\} + \|Q_A\| + \|Q_B\|$ states, $1 + \max\{\|\Gamma_A\|, \|\Gamma_B\|\}$ pushdown symbols, and the pushdown height bounded by $\max\{h_A, h_B\}$.

Finally, for the *complement* $L(A)^{\mathrm{c}}$, we provide an NPDA with $2^{\|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\|}$ many states, actually a DFA with the pushdown height equal to zero. Although we leave as open the problem of showing the optimality of such double-exponential blow-up, we prove a single-exponential lower bound for the cost, by providing $\{\tilde{L}_n\}_{n\geq 1}$, a family of languages over a $(c+1)$-letter alphabet, such that: *(i)* $\tilde{L}_n$ is accepted by an NPDA with $n + O(c)$ states, $c+1$ pushdown symbols, and the pushdown height $n+1$, but *(ii)* its complement cannot be accepted by an NPDA in which both the number of states and the pushdown height are below $c^{n/3-O(\log n)}$.

These lower bounds required some new techniques, for several reasons: *(i)* Already a deterministic machine with a polynomial pushdown height can use exponentially many different pushdown contents and hence exponential lower bounds cannot be obtained directly, by standard pigeonhole arguments. *(ii)* Moreover, our machines are *nondeterministic* and hence, after reading the first i symbols of an input $a_1 \cdots a_\ell$, the state and the pushdown content do not depend only on $a_1 \cdots a_i$, but, using a guess-and-verify fashion, on the entire input.

2 Preliminaries

We assume the reader is familiar with the standard models of deterministic and nondeterministic *finite state automata* (DFA and NFA, for short) and *pushdown*

automata (DPDA and NPDA, see, e.g., [7]). Here we briefly recall these models. By Σ^*, we denote the set of words over an alphabet Σ. For a word $\varphi = a_1 \cdots a_\ell \in \Sigma^*$, let $\varphi^{\mathrm{R}} = a_\ell \cdots a_1$ denote its reversal and $|\varphi| = \ell$ its length. The set of words of length i is denoted by Σ^i, with $\Sigma^{\leq h} = \bigcup_{i=0}^{h} \Sigma^i$. By $\|S\|$, we denote the cardinality of a set S and by S^{c} its complement.

For technical reasons, the NPDAs are introduced here in the form that clearly distinguishes instructions manipulating the pushdown store from those reading the input tape [4]. An NPDA is a sextuplet $A = \langle Q, \Sigma, \Gamma, H, q_{\mathrm{I}}, F\rangle$, where Q is the finite set of states, Σ the input alphabet, Γ the pushdown alphabet, $q_{\mathrm{I}} \in Q$ the initial state, $F \subseteq Q$ the set of accepting (final) states, and $H \subseteq Q \times (\{\varepsilon\} \cup \Sigma \cup \{-,+\}\cdot\Gamma) \times Q$ the *transition relation*, with the following meaning:

(i) $(q, \varepsilon, q') \in H$: A gets from the state q to the state q' without using the input tape or the pushdown store.
(ii) $(q, a, q') \in H$: if the next input symbol is a, A gets from q to q' by reading the symbol a, not using the pushdown store.
(iii) $(q, -X, q') \in H$: if the symbol on top of the pushdown is X, A gets from q to q' by popping X, not using the input tape.
(iv) $(q, +X, q') \in H$: A gets from q to q' by pushing the symbol X onto the pushdown, not using the input tape.

An *accepting computation* begins in the state q_{I} with the empty pushdown store, and ends in an accepting state $q' \in F$ after reading the entire input. As usual, $L(A)$ denotes the language accepted by the NPDA A. A *deterministic pushdown automaton* (DPDA) is obtained by claiming that the transition relation does not allow executing more than one transition at a time. (See [2] for a more formal definition.) The following "normal form" of NPDAs will be required later.

Lemma 1 ([4, Lem. 1]). *For any* NPDA $A = \langle Q, \Sigma, \Gamma, H, q_{\mathrm{I}}, F\rangle$, *there exists an equivalent* NPDA $A' = \langle Q \cup \{q_{\mathrm{F}}\}, \Sigma, \Gamma, H', q_{\mathrm{I}}, \{q_{\mathrm{F}}\}\rangle$, *such that* A' *accepts by entering the unique* q_{F} *with empty pushdown store at the end of the input.*

Given a constant $h \geq 0$, the NPDA A is *of pushdown height* h if, for any $\varphi \in L(A)$, there exists an accepting computation along which the pushdown store never contains more than h symbols. Such a machine will be denoted by a septuplet $A = \langle Q, \Sigma, \Gamma, H, q_{\mathrm{I}}, F, h\rangle$, where $h \geq 0$ is a constant denoting the pushdown height, and all other elements are defined as above. By definition, the meaning of the transitions in the form (iv) is modified as follows:

(iv') $(q, +X, q') \in H$: *if the current pushdown store height is smaller than* h, then A gets from the state q to the state q' by pushing the symbol X onto the pushdown, not using the input tape; *otherwise* A aborts and rejects.

A fair *descriptional complexity measure* takes into account all the components the device consists of, i.e., *(i)* the number of finite control states, *(ii)* the size of the pushdown alphabet, and *(iii)* the height of the pushdown store [4].

Lemma 2 ([2, Lem. 2]). *For each constant height* NPDA $A = \langle Q, \Sigma, \Gamma, H, q_{\mathrm{I}}, F, h\rangle$, *there exists an equivalent* NFA $A' = \langle Q', \Sigma, H', q'_{\mathrm{I}}, F'\rangle$ *with* $\|Q'\| \leq \|Q\| \cdot \|\Gamma^{\leq h}\|$ *states.*

We conclude this section by a combinatorial lemma required later. The lemma says that each sufficiently large subset of $A \times B$ (where A and B are some finite sets) must contain a trio of elements forming a "rectangular triangle".

Lemma 3. *Let A and B be arbitrary two finite sets satisfying $\|A\| \geq 2$ and $\|B\| \geq 2$, and let C be a subset of $A \times B$ satisfying $\|C\| \geq \|A\| + \|B\| - 1$. Then there must exist some elements $\dot{a}, \ddot{a} \in A$ and $\dot{b}, \ddot{b} \in B$, with $\dot{a} \neq \ddot{a}$ and $\dot{b} \neq \ddot{b}$, such that $[\dot{a}, \dot{b}]$, $[\dot{a}, \ddot{b}]$, $[\ddot{a}, \dot{b}]$ are all in C.*

3 Intersection for Constant Height NPDAs

Here we consider the amount of computational resources that are sufficient and necessary for a constant height NPDA accepting the intersection $L(A) \cap L(B)$, for the given constant height NPDAs A and B. After transforming both A and B into the equivalent NFAs, such machine can be built easily. However, by exploiting the power of pushdown storage, we obtain a better construction:[1]

Theorem 1. *Given two constant height NPDAs $A = \langle Q_A, \Sigma, \Gamma_A, H_A, q_A, F_A, h_A \rangle$ and $B = \langle Q_B, \Sigma, \Gamma_B, H_B, q_B, F_B, h_B \rangle$, there exists a constant height NPDA C accepting the intersection $L(A) \cap L(B)$ with the number of states bounded by $\|Q_C\| \leq \|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\| \cdot \|Q_B\|$, using $\|\Gamma_C\| = \|\Gamma_B\|$ pushdown symbols and the pushdown height $h_C = h_B$.*

The main idea is to turn the machine A into an NFA A' (by Lem. 2) and then construct C simulating A' and B simultaneously, using the pushdown store for the simulation of B. Final states are chosen so that C accepts if and only if the input is accepted by both machines. Since the roles of A and B can be swapped, the number of states is actually exponential in $h = \min\{h_A, h_B\}$.

We shall now show that the exponential cost cannot be avoided. To this purpose, for arbitrary fixed input alphabet Σ, we define two families of languages $\{L'_n\}_{n \geq 1}$ and $\{L''_n\}_{n \geq 1}$ accepted by constant height NPDAs with $O(\|\Sigma\|)$ states, $\|\Sigma\|$ pushdown symbols, and the pushdown height $h_n = n$, but with a lower bound $\|\Sigma\|^{\Omega(n)}$ for accepting $\{L'_n \cap L''_n\}_{n \geq 1}$. First, fix a special symbol $\$ \notin \Sigma$. Then, for each $n \geq 1$, define the following two languages:

$$\begin{aligned} L'_n &= \{u_1 \$ v_1 \$ u_2^{\mathrm{R}} \$ v_2^{\mathrm{R}} : u_1, v_1, u_2, v_2 \in \Sigma^*,\ |u_1| \leq n,\ \ u_2 \text{ is a suffix of } u_1\}, \\ L''_n &= \{u_1 \$ v_1 \$ u_2^{\mathrm{R}} \$ v_2^{\mathrm{R}} : u_1, v_1, u_2, v_2 \in \Sigma^*,\ |v_1| \leq n,\ \ v_2 \text{ is a suffix of } v_1\}. \end{aligned} \tag{1}$$

Lemma 4. *For any given Σ and $n \geq 1$, the languages L'_n and L''_n can be accepted by DPDAs (hence, also by NPDAs) A'_n and A''_n, respectively, with $2 \cdot \|\Sigma\| + 4 \leq O(\|\Sigma\|)$ states, $\|\Sigma\|$ pushdown symbols, and the pushdown height $h_n = n$.*

[1] Using the transition function in a form introduced in standard textbooks, only a single state would be required, since the language $L(A) \cap L(B)$ is regular, and hence context free. (See e.g. [7, Sect. 6.3.1].) This indicates that the "traditional" transition function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ (combining input and pushdown operations into a single step) is not realistic if the state-set size is at stake.

Proof. On input $u_1\$v_1\$u_2^{\mathrm{R}}\$v_2^{\mathrm{R}}$, the DPDA A'_n compares the pushdown content filled while reading u_1 with u_2^{R} in order to check whether v_2 is a suffix of v_1, which leaves the first $|u_1| - |u_2^{\mathrm{R}}|$ symbols of u_1 in the pushdown. (The machine A''_n runs in a similar way.) The tricky detail is that, to reduce the number of states from $\Omega(n{\cdot}\|\Sigma\|)$ to $O(\|\Sigma\|)$, we do allow both A'_n and A''_n to reject in the middle of the input by pushdown overflow. □

Denote now the intersection of L'_n and L''_n as

$$L_n = L'_n \cap L''_n = \{u_1\$v_1\$u_2^{\mathrm{R}}\$v_2^{\mathrm{R}} : u_1, v_1, u_2, v_2 \in \Sigma^*,\ |u_1| \le n,\ |v_1| \le n,\\ u_2 \text{ is a suffix of } u_1 \quad \text{and} \quad v_2 \text{ is a suffix of } v_1\}.$$

Clearly, if $|u_1| = |v_1| = |u_2| = |v_2| \le n$, the conditions for membership are simplified: $u_1\$v_1\$u_2^{\mathrm{R}}\$v_2^{\mathrm{R}}$ is in L_n if and only if $u_2 = u_1$ and $v_2 = v_1$.

Theorem 2. *Let $\{A_n\}_{n\ge1}$ be constant height* NPDA*s accepting the languages $\{L_n\}_{n\ge1}$, for some non-unary alphabet Σ, and let Q_n and h_n be, respectively, the number of states and the pushdown height in A_n. Then $(\|Q_n\|{+}1)^2{\cdot}(h_n{+}1) > \|\Sigma\|^n/(4n^2+6n)$, for each $n \ge 1$. Consequently, in $\{A_n\}_{n\ge1}$, the number of states and the pushdown height cannot be both polynomial in n; either $\|Q_n\|+1$ or h_n+1 (or both values) are above $\|\Sigma\|^{n/3}/\sqrt[3]{4n^2+6n} \ge \|\Sigma\|^{n/3-O(\log n)}$.*

Proof. Let $A_n = \langle Q_n, \Sigma, \Gamma_n, H_n, q_{\mathrm{I},n}, F_n, h_n\rangle$ be a constant height NPDA accepting L_n. For contradiction, assume first that the NPDA A_n is in the "normal form" of Lem. 1, that is, it accepts each input by entering the unique final state $q_{\mathrm{F},n}$ with *empty pushdown store* at the end of the input (hence, $F_n = \{q_{\mathrm{F},n}\}$), and that $p_n = \|Q_n\|^2{\cdot}(h_n{+}1) \le \|\Sigma\|^n/(4n^2+6n)$, for some n. From now on, for the sake of readability, we simply write p instead of p_n, as well as $A, Q, \Gamma, H, q_{\mathrm{I}}, q_{\mathrm{F}}, h$ instead of $A_n, Q_n, \Gamma_n, H_n, q_{\mathrm{I},n}, q_{\mathrm{F},n}, h_n$. From these assumptions we get that

$$p = \|Q\|^2{\cdot}(h+1) \le \|\Sigma\|^n/(4n^2+6n). \tag{2}$$

Next, define the following set of pairs:

$$V_0 = \Sigma^n \times \Sigma^n = \{[u,v] : u, v \in \Sigma^*,\ |u| = |v| = n\}.$$

Consider now the computation on the input $z = u\$v\$u^{\mathrm{R}}\$v^{\mathrm{R}}$, for each $[u,v] \in V_0$. It is clear that $z \in L_n$, and hence there must exist at least one accepting computation of A on this input. From among all possible accepting computations for this input, let us fix the "leftmost" accepting computation path. (That is, each time the machine gets into a configuration from which several nondeterministic choices lead to successful acceptance, take the leftmost choice, using some lexicographical ordering on H, the transition relation.) Now, let us fix some significant parameters for this leftmost path (see either side of Fig. 1):

- $y_\ell \in \{0, \ldots, h\}$, the lowest height of pushdown store in the course of reading the substring $\$v\$u^{\mathrm{R}}\$$,
- $q_\ell \in Q$, the state in which the height y_ℓ is attained for the last time, along $\$v\$u^{\mathrm{R}}\$$,

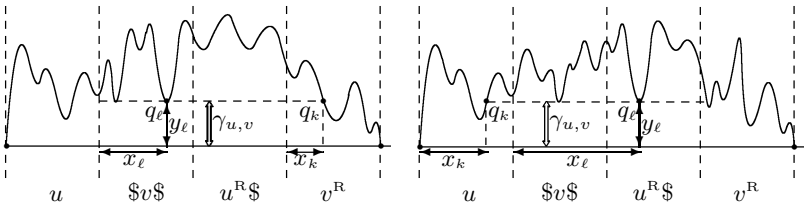


Fig. 1. Parameters y_ℓ, q_ℓ, x_ℓ and the pushdown content $\gamma_{u,v}$ along the computation (either side). Parameters q_k, x_k depend on whether q_ℓ is reached in the course of reading $\$v\$$ (shown on the left), or in the course of reading $u^{\mathrm{R}}\$$ (shown on the right).

- $x_\ell \in \{1, \ldots, |\$v\$u^{\mathrm{R}}\$|\} = \{1, \ldots, 2n+3\}$, the distance from the beginning of $\$v\$u^{\mathrm{R}}\$$ to the input position in which q_ℓ is entered, and
- $\gamma_{u,v}$, the pushdown content at this moment.

The values for the next two parameters, namely, for $q_k \in Q$ and $x_k \in \{1, \ldots, n\}$, depend on whether $x_\ell \leq |\$v\$| = n+2$ or $x_\ell > n+2$:

If $x_\ell \leq |\$v\$| = n+2$, that is, if the computation reaches the state q_ℓ in the course of reading $\$v\$$ (see the left part of Fig. 1), then

- $q_k \in Q$ is the state at the moment when the machine is going to decrease, for the first time, the pushdown height from y_ℓ to $y_\ell - 1$, in the course of reading v^{R} (because our automaton always accepts with empty pushdown store — by Lem. 1, such situation must happen), and
- $x_k \in \{1, \ldots, |v^{\mathrm{R}}|\} = \{1, \ldots, n\}$ is the distance from the beginning of v^{R} to the input position in which q_k is entered.

If $x_\ell > |\$v\$| = n+2$, that is, if the computation reaches the state q_ℓ in the course of reading $u^{\mathrm{R}}\$$ (see the right part of Fig. 1), then

- $q_k \in Q$ is the state at the moment when the machine has just increased, for the last time, the pushdown height from $y_\ell - 1$ to y_ℓ, in the course of reading u (because our automaton always starts with empty pushdown store — by definition, such situation must happen), and
- $x_k \in \{1, \ldots, |u|\} = \{1, \ldots, n\}$ is the distance from the beginning of u to the input position in which q_k is entered.

It is easy to see that, independent of whether $x_\ell \leq |\$v\$| = n+2$ or $x_\ell > n+2$, we have $y_\ell \in \{0, \ldots, h\}$, $q_\ell \in Q$, $x_\ell \in \{1, \ldots, 2n+3\}$, $q_k \in Q$, and $x_k \in \{1, \ldots, n\}$. Therefore, the number of different quintuples $[y_\ell, q_\ell, x_\ell, q_k, x_k]$ is bounded by $\|Q\|^2 \cdot (h+1) \cdot (2n+3) \cdot n = \|Q\|^2 \cdot (h+1) \cdot (2n^2+3n)$. Thus, by using also (2), the number of such quintuples can be bounded by $\|Q\|^2 \cdot (h+1) \cdot (2n^2+3n) \leq \|\Sigma\|^n / (4n^2+6n) \cdot (2n^2+3n) = \|\Sigma\|^n / 2$.

In conclusion, for each $[u, v] \in V_0$, we took the input $u\$v\$u^{\mathrm{R}}\$v^{\mathrm{R}}$, and fixed the unique leftmost accepting computation path, which gives the unique quintuple of parameters $[y_\ell, q_\ell, x_\ell, q_k, x_k]$. Thus, each pair $[u, v] \in V_0$ can be associated with exactly one quintuple $[y_\ell, q_\ell, x_\ell, q_k, x_k]$. Hence, a simple pigeonhole argument

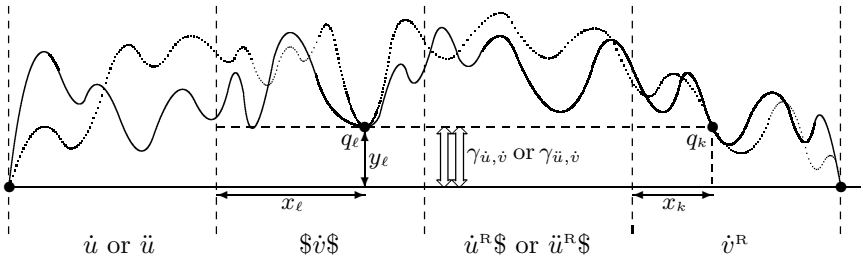


Fig. 2. Computation paths for the inputs $\dot{z} = \dot{u}\$\dot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}}$, $\ddot{z}_u = \ddot{u}\$\dot{v}\$\ddot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}}$, and $\delta_u = \dot{u}\$\dot{v}\$\ddot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}} \notin L_n$, for the case of $x_\ell \le |\$\dot{v}\$| = n{+}2$

proves the existence of a set $V_1 \subseteq V_0$, such that all $[u, v] \in V_1$ share the same $[y_\ell, q_\ell, x_\ell, q_k, x_k]$ and, moreover, the cardinality of such set is

$$\|V_1\| \ge \frac{\|V_0\|}{\|Q\|^2 \cdot (h+1) \cdot (2n^2+3n)} \ge \frac{\|\Sigma\|^{2n}}{\|\Sigma\|^n/2} = 2 \cdot \|\Sigma\|^n > 2 \cdot \|\Sigma\|^n - 1\,. \qquad (3)$$

Realize that $V_1 \subseteq \Sigma^n \times \Sigma^n$ and $\|\Sigma^n\| = \|\Sigma\|^n \ge 2$, for each $\|\Sigma\| \ge 2$ and $n \ge 1$. Hence, taking into account (3), the sets $A = \Sigma^n$, $B = \Sigma^n$, and $C = V_1$ satisfy the assumptions of Lem. 3. Therefore, there must exist some strings $\dot{u}, \ddot{u}, \dot{v}, \ddot{v} \in \Sigma^n$, with $\dot{u} \ne \ddot{u}$, $\dot{v} \ne \ddot{v}$, such that $[\dot{u}, \dot{v}]$, $[\dot{u}, \ddot{v}]$, and $[\ddot{u}, \dot{v}]$ are all in V_1. Consequently, they all share the same parameters $[y_\ell, q_\ell, x_\ell, q_k, x_k]$ on the corresponding accepting paths. Now we have to distinguish between the two cases, depending on the value x_ℓ.

Case i: $x_\ell \le n{+}2 = |\$\dot{v}\$|$. This means that, for $[u, v] \in \{[\dot{u}, \dot{v}], [\dot{u}, \ddot{v}], [\ddot{u}, \dot{v}]\}$, all fixed leftmost computations for the inputs $u\$v\$u^{\mathrm{R}}\$v^{\mathrm{R}}$ visit the same state $q_\ell \in Q$, with the same pushdown height y_ℓ, and at the same position x_ℓ, in the course of reading $\$v\$$. Thus, for all these inputs, the parameter $q_k \in Q$ is taken as the state at the moment when the height is going to be decreased below y_ℓ for the first time, along v^{R}, at a position x_k. Also the values q_k and x_k are the same for all these inputs. This situation is depicted in Fig. 2. Consider now $\dot{z} = \dot{u}\$\dot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}}$ and $\ddot{z}_u = \ddot{u}\$\dot{v}\$\ddot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}}$, together with their crossbreed $\delta_u = \dot{u}\$\dot{v}\$\ddot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}} \notin L_n$.

First, on the inputs $\dot{z}$ and $\ddot{z}_u$, at the moment when the machine A reaches the state q_ℓ at the position x_ℓ, the pushdown store contains, respectively, the string $\gamma_{\dot{u},\dot{v}}$ or $\gamma_{\ddot{u},\dot{v}}$, consisting of y_ℓ symbols loaded in the course of reading $\dot{u}$ (or $\ddot{u}$, respectively). On both $\dot{z}$ and $\ddot{z}_u$, these deepest y_ℓ symbols will stay unchanged in the pushdown until the moment when A reaches the same state q_k at the same position x_k, along $\dot{v}^{\mathrm{R}}$.

Now, for the input δ_u, one of the possible computations can start by following the trajectory for $\dot{z}$, reading $\dot{u}$ and the first x_ℓ symbols of $\$\dot{v}\$$, until it reaches the state q_ℓ. At this moment, the pushdown store contains the string $\gamma_{\dot{u},\dot{v}}$. Now the machine switches to the computation path for $\ddot{z}_u$, until it gets into the state q_k. Along this path, the computation does not visit the deepest y_ℓ symbols in the pushdown store, reading the remaining $|\$\dot{v}\$| - x_\ell$ symbols of $\$\dot{v}\$$, the entire block $\ddot{u}^{\mathrm{R}}\$$, and the first x_k symbols of $\dot{v}^{\mathrm{R}}$. From this point forward, the

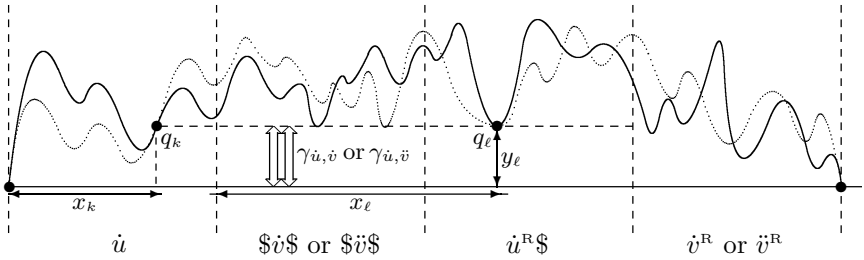


Fig. 3. Computation paths for the inputs $\dot{z} = \dot{u}\$\dot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}}$, $\ddot{z}_v = \dot{u}\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$\ddot{v}^{\mathrm{R}}$, and $\delta_v = \dot{u}\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}} \notin L_n$, for the case of $x_\ell > |\$\dot{v}\$| = n+2$

computation on δ_u can switch back to the trajectory for $\dot{z}$, working with the same content in the pushdown store and reading the remaining $|\dot{v}^{\mathrm{R}}| - x_k$ symbols of $\dot{v}^{\mathrm{R}}$. Clearly, such computation path stops with the empty pushdown in the accepting state q_{F}. Thus, A accepts $\delta_u = \dot{u}\$\dot{v}\$\ddot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}} \notin L_h$, which is a contradiction.

Case ii: $x_\ell > n+2 = |\$\dot{v}\$|$. Again, the three leftmost computations on the inputs $u\$v\$u^{\mathrm{R}}\$v^{\mathrm{R}}$, for $[u,v] \in \{[\dot{u},\dot{v}],[\dot{u},\ddot{v}],[\ddot{u},\dot{v}]\}$, visit the same state $q_\ell \in Q$, with the same pushdown height y_ℓ, and at the same position x_ℓ, this time in the course of reading $u^{\mathrm{R}}\$$. For all of them, the parameter $q_k \in Q$ is now taken as the state reached at the moment when the height has been increased to y_ℓ for the last time, along u, at a position x_k, but also here the values q_k and x_k are the same for all these inputs. This situation is depicted in Fig. 3. This time we consider the inputs $\dot{z} = \dot{u}\$\dot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}}$ and $\ddot{z}_v = \dot{u}\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$\ddot{v}^{\mathrm{R}}$, together with their crossbreed $\delta_v = \dot{u}\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}} \notin L_n$.

First, on the inputs $\dot{z}$ and $\ddot{z}_v$, at the moment when A gets to the state q_k at the position x_k, the pushdown store contains, respectively, the string $\gamma_{\dot{u},\dot{v}}$ or $\gamma_{\dot{u},\ddot{v}}$, consisting of y_ℓ pushdown symbols loaded in the course of reading the first x_k symbols of $\dot{u}$. On both $\dot{z}$ and $\ddot{z}_v$, these deepest y_ℓ symbols will stay unchanged in the pushdown until the moment when A reaches the same state q_ℓ at the same position x_ℓ, along $\dot{u}^{\mathrm{R}}\$$.

Now, for the input δ_v, one of the possible computations can start by following the trajectory for $\dot{z}$, reading first x_k symbols of $\dot{u}$, until it reaches the state q_k. At this moment, the pushdown store contains the string $\gamma_{\dot{u},\dot{v}}$. Here A switches to the computation path for $\ddot{z}_v$, until it gets into the state q_ℓ. Along this path, the computation does not visit the deepest y_ℓ symbols in the pushdown store, reading the remaining $|\dot{u}| - x_k$ symbols of $\dot{u}$ and the first x_ℓ symbols of $\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$$ (which includes, among others, traversing across the entire block $\$\ddot{v}\$$). From this point forward, the computation on δ_v can switch back to the trajectory for $\dot{z}$, working with the same content in the pushdown store and reading the remaining $|\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$| - x_\ell$ symbols of $\dot{u}^{\mathrm{R}}\$$ and the entire block $\dot{v}^{\mathrm{R}}$. Again, such path stops in q_{F}. Thus, A accepts $\delta_v = \dot{u}\$\ddot{v}\$\dot{u}^{\mathrm{R}}\$\dot{v}^{\mathrm{R}} \notin L_n$, which is a contradiction.

In conclusion, if $A = A_n$ accepts L_n, the inequality (2) must be reversed. Thus, the value $p = p_n$ must satisfy $p_n = \|Q_n\|^2 \cdot (h_n+1) > \|\Sigma\|^n/(4n^2+6n)$. However, recall that we have derived this lower bound for the NPDA A_n in the

"normal form" of Lem. 1. For unrestricted NPDAs (not assuming this form), the lower bound changes to $(\|Q_n\|+1)^2 \cdot (h_n+1) > \|\Sigma\|^n/(4n^2+6n)$, since converting a general NPDA into the normal form does not cost more than one state, keeping the same pushdown height. Consequently, either $\|Q_n\|+1 > \|\Sigma\|^{n/3}/\sqrt[3]{4n^2+6n}$, or else $h_n+1 > \|\Sigma\|^{n/3}/\sqrt[3]{4n^2+6n} \geq \|\Sigma\|^{n/3-O(\log n)}$. □

By combining Lem. 4 with Thm. 2, we get the following blow-up:

Theorem 3. *For each fixed constant $c \geq 2$, there exist $\{L'_n\}_{n\geq 1}$ and $\{L''_n\}_{n\geq 1}$, some families of regular languages built over a $(c+1)$-letter alphabet, such that:*

- *(i) there exist, respectively, $\{A'_n\}_{n\geq 1}$ and $\{A''_n\}_{n\geq 1}$, sequences of constant height* NPDA*s accepting these languages with $O(c)$ states, c pushdown symbols, and the pushdown height $h_n = n$, but*
- *(ii) for any constant height* NPDA*s $\{A_n\}_{n\geq 1}$ accepting the family of their intersections $\{L_n\}_{n\geq 1} = \{L'_n \cap L''_n\}_{n\geq 1}$, either the number of states in A_n or else the pushdown height must be above $c^{n/3-O(\log n)}$, independently of the size of the used pushdown alphabet.*

For comparison, by the use of Thm. 1 for NPDAs from Lem. 4, we get that $\{L'_n \cap L''_n\}_{n\geq 1}$ can be accepted[2] by NPDAs with $\|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\| \cdot \|Q_B\| \leq O(c^n)$ states, $\|\Gamma_B\| = c$ pushdown symbols, and the pushdown height $h_B = n$.

4 Union and Complement for Constant Height NPDAs

Now we shall deal with another two basic Boolean operations, union and complement. First, given two constant height NPDAs A and B, we construct a constant height NPDA C accepting the union $L(A) \cup L(B)$. The size of C is linear in all "reasonable" complexity measures. This allows us to derive an exponential lower bound for the complement $L(A)^{\mathrm{c}}$.

An NPDA C accepting $L(A) \cup L(B)$ is simple: C nondeterministically chooses which of the given two machines it will simulate. However, the pushdown heights h_A, h_B may be different. If the chosen machine uses a lower pushdown height than the other one, the difference in the pushdown limit must be repaired, by filling $|h_A - h_B|$ copies of some extra symbol at the bottom of the pushdown.

Theorem 4. *Given two constant height* NPDA*s $A = \langle Q_A, \Sigma, \Gamma_A, H_A, q_A, F_A, h_A\rangle$ and $B = \langle Q_B, \Sigma, \Gamma_B, H_B, q_B, F_B, h_B\rangle$, there exists a constant height* NPDA *C accepting the union $L(A) \cup L(B)$ with the number of states bounded by $\|Q_C\| \leq \max\{1, |h_A - h_B|\} + \|Q_A\| + \|Q_B\|$, using $\|\Gamma_C\| \leq 1+\max\{\|\Gamma_A\|, \|\Gamma_B\|\}$ pushdown symbols and the pushdown height $h_C = \max\{h_A, h_B\}$.*

[2] Alternatively, one can get a different NPDA for $\{L'_n \cap L''_n\}_{n\geq 1}$, using only $O(c^{n/2})$ states, with a pushdown of height $3/2 \cdot n$. The idea is to load the first half of u_1 and the entire v_1 in the pushdown, but to store the second half of u_1 in the finite state control. After checking the second half of u_1 against the first half of u_2^{R}, we store the second half of u_2^{R} in the finite state control, to be checked later, after comparing v_2^{R} with v_1. This indicates that — without using a different witness language — the gap from Thm. 3 cannot be raised higher than to $\Omega(c^{n/2})$.

The last operation is complement. A trivial double-exponential upper bound is obtained by the use of Lem. 2, coding the pushdown content of the given NPDA A in the finite state control, which gives a classical NFA with at most $\|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\|$ states. Then we make this machine deterministic, by the standard power set construction [7,11], with $2^{\|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\|}$ states. Finally, we obtain a DFA B for $L(A)^{\mathrm{c}}$ by swapping the roles of accepting and rejecting states.

Theorem 5. *Given a constant height* NPDA $A = \langle Q_A, \Sigma, \Gamma_A, H_A, q_A, F_A, h_A \rangle$, *there exists a* DFA B *(hence, also a constant height* NPDA*) accepting the complement* $L(A)^{\mathrm{c}}$ *with the number of states bounded by* $\|Q_B\| \leq 2^{\|Q_A\| \cdot \|\Gamma_A^{\leq h_A}\|}$ *(hence, using no pushdown symbols and the pushdown height equal to zero).*

At this point, one can easily combine the exponential lower bound obtained for intersection with the linear upper bound for union and conclude that the lower bound for complementing is at least exponential, by application of De Morgan's laws. Nevertheless, to see some growth rate for the gap, we shall consider a specific witness language. For this purposes, recall the languages L'_n and L''_n, introduced by (1). For the fixed alphabet Σ and each $n \geq 1$, let

$$\tilde{L}_n = {L'_n}^{\mathrm{c}} \cup {L''_n}^{\mathrm{c}}. \tag{4}$$

Lemma 5. *For any given* Σ *and* $n \geq 1$, *the language* $\tilde{L}_n$ *can be accepted by an* NPDA $\tilde{A}_n$ *with* $\|Q_n\| = n + 4 \cdot \|\Sigma\| + 11 \leq n + O(\|\Sigma\|)$ *states,* $\|\Sigma\| + 1$ *pushdown symbols, and the pushdown height* $h_n = n + 1$.

Proof. Recall that, by Lem. 4, both L'_n and L''_n are accepted by the respective DPDAs A'_n and A''_n using $2 \cdot \|\Sigma\| + 4$ states, $\|\Sigma\|$ pushdown symbols, and the pushdown height n. Since both A'_n and A''_n are *deterministic*, an NPDA $\tilde{A}_n$ for $\tilde{L}_n$ can nondeterministically choose which of these two machines it will simulate, to verify that its unique computation *rejects*. The tricky detail is that, to reduce the number of states from $\Omega(n \cdot \|\Sigma\|)$ to $n + O(\|\Sigma\|)$, we do not keep track of the current pushdown height during the simulation, and hence we do not detect pushdown *overflows*. However, A'_n and A''_n from Lem. 4 reject by pushdown overflows *only if* the length of some block (u_1 or v_1, respectively) exceeds n, that is, only if the input contains a substring of length n+1 not containing any \$-symbols. Therefore, $\tilde{A}_n$ proceeds as follows. First, $\tilde{A}_n$ stores some new initial symbol X_{I} at the bottom of the pushdown (to detect pushdown *underflows* during the simulation), and then nondeterministically chooses from among *(i)* testing whether A'_n rejects by a computation not blocked by a pushdown overflow, *(ii)* testing whether A''_n rejects by a computation not blocked by a pushdown overflow, and *(iii)* testing whether the input contains a substring φ of length n+1 without any \$-symbol, the starting position of φ is established nondeterministically. □

Conversely, by De Morgan's laws, the *complement* of the language introduced by (4) is $\tilde{L}_n^{\mathrm{c}} = ({L'_n}^{\mathrm{c}} \cup {L''_n}^{\mathrm{c}})^{\mathrm{c}} = L'_n \cap L''_n = L_n$. Recall that the lower bound derived for L_n in Thm. 2 is exponential. Combined with Lem. 5, this gives:

Theorem 6. *For each fixed constant $c \geq 2$, there exists $\{\tilde{L}_n\}_{n\geq 1}$, a family of regular languages built over a $(c+1)$-letter alphabet, such that:*

(i) there exists $\{\tilde{A}_n\}_{n\geq 1}$, a sequence of constant height NPDA*s accepting these languages with $Q_n \leq n + O(c)$ states, $c+1$ pushdown symbols, and the pushdown height $h_n = n+1$, but*

(ii) for any constant height NPDA*s $\{\tilde{A}^{\rm c}_n\}_{n\geq 1}$ accepting the family of their complements $\{\tilde{L}^{\rm c}_n\}_{n\geq 1}$, either the number of states in $\tilde{A}^{\rm c}_n$ or else the pushdown height must be above $c^{n/3-O(\log n)}$, independently of the size of the used pushdown alphabet.*

The above lower bound is far below the known conversion for complementing, presented in Thm. 5, using $2^{\|Q_A\|\cdot\|\Gamma_A^{\leq h_A}\|}$ states. It should also be pointed out that, in the case of our witness languages $\{\tilde{L}_n\}_{n\geq 1}$, their complements $\{\tilde{L}^{\rm c}_n\}_{n\geq 1} = \{L_n\}_{n\geq 1}$ *can* be accepted by NPDAs with only a *single*-exponential blow-up, namely, with $O(c^n)$ states, c pushdown symbols, and the pushdown height n, which is obtained by combining Lem. 4 with Thm. 1.

5 Concluding Remarks

We have analyzed the size cost of basic Boolean operations for *nondeterministic automata with a pushdown of constant height*. For intersection, a single-exponential cost is sufficient and, in the worst case, also necessary. On the other hand, the cost of union is only linear. Combining these results, we have shown that the lower bound for complement is single-exponential, but we have derived only a double-exponential upper bound, which leaves a large gap.

It was conjectured that the lower bound for the intersection from Thm. 3 can be improved to an exponential lower bound on the number of states, independently of the pushdown height. This would give two witness regular languages the intersection of which would be "expensive" with respect to the number of states even for an NPDA with unrestricted pushdown. However, using the "traditional" transition function, we can reduce the number of states in such a machine to one, with a large pushdown alphabet. (See also Ft. 1.)

The corresponding costs for constant height DPDAs, *deterministic versions* of NPDAs, are [2]: single-exponential for intersection and union, but polynomial for complement. These results are compared in Tab. 1. It turns out that the cost of studied Boolean operations, both for DPDAs and NPDAs with a constant height pushdown, reflects the closure properties for the corresponding machines with an *unrestricted pushdown* (hence, for deterministic and general context-free languages): the unrestricted version of the pushdown machine is closed under the given operation (see e.g. [7]) if and only if the cost of the same operation for the constant height version is at most polynomial. Therefore, it could be interesting to investigate the complexity of other language operations for constant height pushdown automata and compare them with unrestricted versions.

Table 1. Size-cost of Boolean operations on constant height DPDAs [2] and constant height NPDAs, studied in this paper.

Operation	constant height DPDAs	constant height NPDAs
Intersection	exponential	exponential
Union	exponential	linear
Complement	polynomial	exponential ... double-exponential

Similarly, we would like to emphasize the interest in *two-way versions* of these machines, and in some *more restricted versions*. For instance, one could study the cost for *unary* languages: the same investigation on unary NFAs [10] shows interesting differences from the general case.

References

1. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Removing nondeterminism in constant height pushdown automata. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 76–88. Springer, Heidelberg (2012)
2. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: The size-cost of Boolean operations on constant height deterministic pushdown automata. Theoret. Comput. Sci. 449, 23–36 (2012)
3. Ehrenfeucht, A., Zieger, P.: Complexity measures for regular expressions. J. Comput. System Sci. 12, 134–146 (1976)
4. Geffert, V., Mereghetti, C., Palano, B.: More concise representation of regular languages by automata and regular expressions. Inf. & Comp. 208, 385–394 (2010)
5. Gruber, H., Holzer, M.: Language operations with regular expressions of polynomial size. Theoret. Comput. Sci. 410, 3281–3289 (2009)
6. Holzer, M., Kutrib, M.: Descriptional complexity — an introductory survey. In: Martín-Vide, C. (ed.) Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)
7. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2001)
8. Kutrib, M., Malcher, A., Wotschke, D.: The Boolean closure of linear context-free languages. Acta Inform. 45, 177–191 (2008)
9. Meyer, A., Fischer, M.: Economy of description by automata, grammars, and formal systems. In: Proc. IEEE Symp. Switching & Automata Th., pp. 188–191 (1971)
10. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. Internat. J. Found. Comput. Sci. 13, 145–159 (2002)
11. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. 3, 114–125 (1959)
12. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer (1997)
13. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)

A Short Tutorial on Order-Invariant First-Order Logic

Nicole Schweikardt

Goethe-Universität Frankfurt am Main, Germany
schweika@informatik.uni-frankfurt.de

Abstract. This paper gives a short introduction to order-invariant first-order logic and arb-invariant first-order logic. We present separating examples demonstrating the expressive power, as well as tools for proving certain expressive weaknesses of these logics.

1 Introduction

Expressibility of logics over finite structures plays an important role in various areas of computer science. In descriptive complexity, logics are used to characterise complexity classes, and concerning databases, common query languages have well-known logical equivalents. Order-invariant and arb-invariant logics were introduced to capture the data independence principle in databases: An implementation of a database query may exploit the order in which the database elements are stored in memory, and thus identify the elements with natural numbers on which arithmetic can be performed. But the use of order and arithmetic should be restricted in such a way that the result of the query does not depend on the particular order in which the data is stored.

Arb-invariant queries are queries that can make use of an order predicate $<$ and of arithmetic predicates such as $+$ or $\times$, but only in such a way that the answer is independent of the particular interpretation of $<, +, \times$. Queries that only use the linear order, but no further arithmetic predicates, are called *order-invariant*.

It is known that order-invariant least fixed-point logic LFP precisely captures the polynomial time computable queries [12,26], while arb-invariant LFP and arb-invariant first-order logic capture the queries computable in $\mathrm{P}_{/\mathrm{poly}}$ [15] and AC^0 [13], respectively. Order-invariant queries and arb-invariant queries have been studied in depth, cf. e.g. [1,5,25,15,8,14,17,22,16,19,20,7,4,24,9,2]. A short overview of the state-of-the-art concerning these logics can be found in [23].

The aim of this paper is to give a short tutorial on order-invariant and arb-invariant first-order logic FO. In Section 2 we fix the basic notation. Section 3 gives the precise definition of order-invariant and arb-invariant FO, along with a few easy examples. Section 4 presents examples that separate order-invariant FO from plain FO. Section 5 shows how to prove that certain queries are not definable in order- or arb-invariant FO. Section 6 gives a list of open research questions.

2 Preliminaries

Basic Notation. We write $\mathbb{N}$ for the set of non-negative integers, and we let $\mathbb{N}_{\geq 1} := \mathbb{N} \setminus \{0\}$. For $n \in \mathbb{N}_{\geq 1}$ we write $[n]$ to denote the set $\{i \in \mathbb{N} : 0 \leq i < n\}$, i.e.,

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 112–126, 2013.

$[n] = \{0, \ldots, n-1\}$. For a positive real number r, the logarithm of r with respect to base 2 is denoted $\log r$.

For a finite set A we write $|A|$ to denote the cardinality of A. By 2^A we denote the power set of A, i.e., the set $\{Y : Y \subseteq A\}$. The set of all non-empty finite words built from symbols in A is denoted A^+. We write $|w|$ for the length of a word $w \in A^+$.

Structures. A *signature* σ is a set of relation symbols R, each of them associated with a fixed arity $ar(R) \in \mathbb{N}_{\geq 1}$. A *$\sigma$-structure* $\mathcal{A}$ consists of a non-empty set A called the *universe* of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^{ar(R)}$ for each relation symbol $R \in \sigma$.

The *cardinality* of a σ-structure $\mathcal{A}$ is the cardinality of its universe. *Finite* σ-structures are σ-structures of finite cardinality.

For σ-structures $\mathcal{A}$ and $\mathcal{B}$ and tuples $\overline{a} = (a_1, \ldots, a_k) \in A^k$ and $\overline{b} = (b_1, \ldots, b_k) \in B^k$ we write $(\mathcal{A}, \overline{a}) \cong (\mathcal{B}, \overline{b})$ to indicate that there is an isomorphism π from $\mathcal{A}$ to $\mathcal{B}$ that maps $\overline{a}$ to $\overline{b}$ (i.e., $\pi(a_i) = b_i$ for each $i \leq k$).

First-Order Logic. We assume that the reader is familiar with basic concepts and notations concerning first-order logic (cf., e.g., the textbooks [14,6]). We write FO(σ) to denote the class of all first-order formulas of signature σ. By *free*(φ) we denote the set of all free variables of an FO(σ)-formula φ. A *sentence* is a formula φ with *free*(φ) $= \emptyset$. We often write $\varphi(\overline{x})$, for $\overline{x} = (x_1, \ldots, x_k)$, to indicate that *free*(φ) $= \{x_1, \ldots, x_k\}$.

If $\mathcal{A}$ is a σ-structure and $\overline{a} = (a_1, \ldots, a_k) \in A^k$, we write $\mathcal{A} \models \varphi[\overline{a}]$ to indicate that the formula $\varphi(\overline{x})$ is satisfied in $\mathcal{A}$ when interpreting the free occurrences of the variables $x_1, \ldots, x_k$ with the elements $a_1, \ldots, a_k$.

If $\mathcal{A}$ and $\mathcal{B}$ are σ-structures and r is a natural number, we write $\mathcal{A} \equiv_r \mathcal{B}$ to indicate that $\mathcal{A}$ and $\mathcal{B}$ satisfy exactly the same FO(σ)-sentences of quantifier rank r.

Throughout the remainder of this paper, we will assume that σ is a fixed finite signature.

3 Order-Invariant Logic and Arb-Invariant Logic

The idea: Extend the expressive power of a logic by allowing formulas to use, apart from the relation symbols present in the signature σ, also a linear order $<$, arithmetic predicates such as $+$ or $\times$, or arbitrary numerical predicates.

Definition 3.1 (Numerical predicate)
For $r \in \mathbb{N}_{\geq 1}$, an r-ary *numerical predicate* is an r-ary relation on $\mathbb{N}$.

Three particular numerical predicates that will often be used in this paper are

- the linear order $<^{\mathbb{N}}$ consisting of all tuples $(a, b) \in \mathbb{N}^2$ with $a < b$,
- the addition predicate $+^{\mathbb{N}}$ consisting of all triples $(a, b, c) \in \mathbb{N}^3$ with $a + b = c$, and
- the multiplication predicate $\times^{\mathbb{N}}$ consisting of all triples $(a, b, c) \in \mathbb{N}^3$ with $a \times b = c$.

To allow logical formulas to use numerical predicates, we fix the following notation: For every $r \in \mathbb{N}_{\geq 1}$ and for every r-ary numerical predicate $P^{\mathbb{N}}$ (i.e., $P^{\mathbb{N}} \subseteq \mathbb{N}^r$), we let P be a new relation symbol of arity r — here, "new" means that P does not belong to σ. We write η_{arb} to denote the set of all the relation symbols P obtained this way, and

we let σ_{arb} be the disjoint union of σ and η_{arb} (the subscript "arb" stands for "arbitrary numerical predicates").

Next, we would like to allow FO(σ_{arb})-formulas to make meaningful statements about finite σ-structures. To this end, for a finite σ-structure $\mathcal{A}$, we consider embeddings ι of the universe of $\mathcal{A}$ into the initial segment of $\mathbb{N}$ of size $n = |A|$, i.e., the set $[n] = \{0, \ldots, n-1\}$.

Definition 3.2 (Embedding). Let $\mathcal{A}$ be a finite σ-structure, and let $n := |A|$.
An *embedding* ι of $\mathcal{A}$ is a bijection $\iota : A \to [n]$.

Given a finite σ-structure $\mathcal{A}$ and an embedding ι of $\mathcal{A}$, we can translate r-ary numerical predicates $P^{\mathbb{N}}$ into r-ary predicates on A as follows: The linear order $<^{\mathbb{N}}$ induces a linear order $<^{\iota}$ on A where for all $a, b \in A$ we have $a <^{\iota} b$ iff $\iota(a) < \iota(b)$. The addition predicate $+^{\mathbb{N}}$ induces an addition predicate $+^{\iota}$ on A where for all $a, b, c \in A$ we have $(a, b, c) \in +^{\iota}$ iff $\iota(a) + \iota(b) = \iota(c)$. In general, an arbitrary r-ary numerical predicate $P^{\mathbb{N}}$ induces the r-ary predicate P^{ι} on A, consisting of all r-tuples $\overline{a} = (a_1, \ldots, a_r) \in A^r$ where $\iota(\overline{a}) = (\iota(a_1), \ldots, \iota(a_r)) \in P^{\mathbb{N}}$.

The σ_{arb}-structure $\mathcal{A}^{\iota}$ associated with $\mathcal{A}$ and ι is the expansion of $\mathcal{A}$ by the predicates P^{ι} for all $P \in \eta_{\text{arb}}$. I.e., $\mathcal{A}^{\iota}$ has the same universe as $\mathcal{A}$, all relation symbols $R \in \sigma$ are interpreted in $\mathcal{A}^{\iota}$ in the same way as in $\mathcal{A}$, and every numerical symbol $P \in \eta_{\text{arb}}$ is interpreted by the relation P^{ι}.

To ensure that an FO(σ_{arb})-formula φ makes a meaningful statement about a σ-structure $\mathcal{A}$, we evaluate φ in $\mathcal{A}^{\iota}$, and we restrict attention to those formulas whose truth value is independent of the particular choice of the embedding ι. This is formalised by the following notion.

Definition 3.3 (Arb-invariance and arb-inv-FO)
Let $\varphi(\overline{x})$ be an FO(σ_{arb})-formula with k free variables, and let $\mathcal{A}$ be a finite σ-structure.

(a) The formula $\varphi(\overline{x})$ is *arb-invariant on* $\mathcal{A}$ if for *all* embeddings ι_1 and ι_2 of $\mathcal{A}$ and for all tuples $\overline{a} \in A^k$ we have: $\mathcal{A}^{\iota_1} \models \varphi[\overline{a}] \iff \mathcal{A}^{\iota_2} \models \varphi[\overline{a}]$.
(b) Let $\varphi(\overline{x})$ be arb-invariant on $\mathcal{A}$.
We write $\mathcal{A} \models \varphi[\overline{a}]$, if $\mathcal{A}^{\iota} \models \varphi[\overline{a}]$ for some (i.e., every) embedding ι of $\mathcal{A}$.
(c) $\varphi(\overline{x})$ is called *arb-invariant* if it is arb-invariant on every finite σ-structure $\mathcal{A}$.
(d) We write arb-inv-FO(σ) to denote the set of all arb-invariant FO(σ_{arb})-formulas.

Example 3.4. We present an arb-invariant FO(σ_{arb})-sentence φ_{even} which is satisfied by exactly those finite σ-structures that have even cardinality. The formula is chosen as follows:

$$\varphi_{\mathit{even}} \;:=\; \exists x \, \forall y \, \big(\, (\, y{<}x \;\vee\; y{=}x \,) \;\wedge\; \mathit{Odd}(x) \, \big),$$

where $\mathit{Odd}^{\mathbb{N}}$ is the unary numerical predicate consisting of all odd numbers.

Let us consider a finite σ-structure $\mathcal{A}$ of size $n = |A|$ and an embedding ι of $\mathcal{A}$ into the set $[n] = \{0, \ldots, n-1\}$. Obviously, $\mathcal{A}^{\iota} \models \varphi_{\mathit{even}}$ iff the maximum element in $[n]$, i.e., the number $n-1$, is odd, i.e., the number n is even. Thus, the formula φ_{even} is arb-invariant on $\mathcal{A}$, and it expresses that $\mathcal{A}$ is of even cardinality.

Note that $<^{\mathbb{N}}$ and $\mathit{Odd}^{\mathbb{N}}$ are the only numerical predicates used by the formula φ_{even}. Both predicates can be replaced by uses of the addition predicate $+^{\mathbb{N}}$, since $\mathit{Odd}(x)$ is equivalent to $\neg \exists z \; z{+}z{=}x$, and $y{<}x$ is equivalent to $(\, \neg \, y{=}x \;\wedge\; \exists z \; y{+}z{=}x \,)$.

Thus, "even cardinality" of finite σ-structures can also be expressed by an arb-invariant FO(σ_{arb})-sentence that only uses the numerical predicate $+^{\mathbb{N}}$. Recall that it is well-known that "even cardinality" can neither be expressed by FO(σ), nor by arb-invariant FO(σ_{arb})-sentences that only use the numerical predicate $<^{\mathbb{N}}$ (cf., e.g., the textbooks [14,6], where it is shown that "even cardinality of linear orders" is not definable in first-order logic).

Definition 3.5 (Order-invariance and addition-invariance)

(a) An arb-invariant formula that only uses the numerical predicate $<^{\mathbb{N}}$ is called *order-invariant*. By $<$-inv-FO(σ) we denote the set of all order-invariant FO($\sigma \cup \{<\}$)-formulas.
(b) An arb-invariant formula that only uses the numerical predicate $+^{\mathbb{N}}$ is called *addition-invariant*. By $+$-inv-FO(σ) we denote the set of all addition-invariant FO($\sigma \cup \{+\}$)-formulas.

Example 3.4 shows that $<$-inv-FO(σ) is less expressive than $+$-inv-FO(σ).

It is known that for any signature σ that contains at least one symbol of arity ≥ 2, there is no algorithm that decides whether an input FO($\sigma \cup \{<\}$)-sentence is order-invariant (this can be shown by an easy reduction using Trakhtenbrot's theorem, see e.g. [14]). However, if σ contains only unary relation symbols, order-invariance of an input sentence is decidable, since commutativity of regular languages is decidable (via checking if the language's syntactic monoid is commutative).

Definition 3.6. An arb-invariant formula that only uses numerical predicates that belong to a subset S of η_{arb} is called *S-invariant*. By S-inv-FO(σ) we denote the set of all S-invariant FO($\sigma \cup S$)-formulas.

The next two examples show that $+$-inv-FO(σ) is less expressive than $\{+, \times\}$-inv-FO(σ) which, in turn, is less expressive than arb-inv-FO(σ).

Example 3.7. Consider the formula φ_{even} from Example 3.4. Let φ_{square} be the formula obtained from φ_{even} by replacing the atom $Odd(x)$ by $Square'(x)$, where $Square'^{\mathbb{N}} := \{i^2{-}1 \,:\, i \in \mathbb{N}_{\geq 1}\}$. Obviously, φ_{square} is an arb-invariant sentence satisfied by exactly those finite σ-structures whose cardinality is a square number.

Note that $i^2{-}1 = (i{-}1)^2 + 2(i{-}1)$. Thus, $Square'(x)$ is equivalent to

$$\exists y\, \exists z_1\, \exists z_2\, \big(\, y{\times}y{=}z_1 \;\wedge\; y{+}y{=}z_2 \;\wedge\; z_1{+}z_2{=}x\,\big).$$

Therefore, "square number cardinality" can be expressed in $\{+, \times\}$-inv-FO(σ). It is well-known that this cannot be expressed in $+$-inv-FO(σ) (since, by the theorem of Ginsburg and Spanier, FO($+$)-definable subsets of $\mathbb{N}$ are semi-linear; see e.g. [21] for an overview).

Example 3.8. It is straightforward to see that $\{+, \times\}$-inv-FO(σ)-sentences can only define *decidable* properties of finite σ-structures. However, arb-inv-FO(σ) can define also undecidable properties. For example, let $U^{\mathbb{N}}$ be an undecidable subset of $\mathbb{N}$ (such a set exists, since there are uncountably many subsets of $\mathbb{N}$, but only a countable number

of decidable sets). Now let φ_U be the formula obtained from φ_{even} by replacing the atom $Odd(x)$ by $U'(x)$, where $U'^{\mathbb{N}} := \{i-1 \,:\, i \in U \text{ and } i \neq 0\}$. Clearly, φ_U is an arb-invariant sentence satisfied by exactly those finite σ-structures whose cardinality belongs to $U^{\mathbb{N}}$. As $U^{\mathbb{N}}$ is undecidable, also the class of finite σ-structures satisfying φ_U is undecidable.

The examples seen so far are simple in the sense that they only refer to the cardinality of structures and do not make use of the relation symbols present in σ. Showing that $<$-inv-FO(σ) is more expressive than plain FO(σ) requires much more sophisticated constructions and depends on the particular choice of the signature σ. In fact, if σ is the *empty* signature $\emptyset$ (as could have been chosen for the examples above), it is straightforward to see that $<$-inv-FO($\emptyset$) has exactly the same expressive power as FO($\emptyset$).

4 Three Examples Showing That Order-Invariant FO Is More Expressive Than FO

In the literature, basically only three examples are known that separate $<$-inv-FO(σ) from plain FO(σ), for various signatures σ. These examples go back to Gurevich (who did not publish this example; but it can be found in the textbooks [1,14]), Potthoff [18], and Otto [17].

Gurevich's Example. For a finite set X let $\mathcal{B}_X := (2^X, \subseteq)$ be the Boolean algebra over X. Thus, $\mathcal{B}_X$ is a finite σ-structure, where $\sigma := \{\subseteq\}$ is the signature consisting of a single binary relation symbol $\subseteq$.

Theorem 4.1 (Gurevich). *There is an order-invariant* FO($\sigma \cup \{<\}$)*-sentence* $\varphi_{Gurevich}$*, but no* FO(σ)*-sentence, such that for every finite set* X *we have:* $\mathcal{B}_X \models \varphi_{Gurevich} \iff |X|$ *is even.*

Proof sketch. Part 1: Construction of the $<$-inv-FO(σ)*-sentence* $\varphi_{Gurevich}$*.* An element $y \in 2^X$ is called an *atom* if it is a singleton set. Thus, $|X|$ is the number of atoms in 2^X. Obviously, the following FO(σ)-formula $atom(x)$ expresses that x is an atom:

$$atom(x) \;:=\; \big(\neg emptyset(x) \;\wedge\; \forall y \,\big(\, y \subseteq x \;\rightarrow\; (\, y{=}x \;\vee\; emptyset(y)\,)\big),$$

where $emptyset(y) := \forall z\; y \subseteq z$.

The order-invariant FO($\sigma \cup \{<\}$)-sentence $\varphi_{Gurevich}$ states that

(1) the underlying σ-structure is indeed a Boolean algebra $(2^X, \subseteq)$, and
(2) there exists a set $z \in 2^X$ that contains the first (w.r.t. $<$) atom of 2^X and every other atom (w.r.t. $<$) of 2^X, and that has the property that the last (w.r.t. $<$) atom of 2^X does not belong to z.

Note that the statements (1) and (2) ensure that $\varphi_{Gurevich}$ is order-invariant on the class of all finite σ-structures, and that a finite Boolean algebra $\mathcal{B}_X$ satisfies $\varphi_{Gurevich}$ if and only if $|X|$ is even. It is an easy exercise to express statement (1) by an FO(σ)-sentence and statement (2) by an FO($\sigma \cup \{<\}$)-sentence.

Part 2: Proof of the non-expressibility in FO(σ). By using a standard Ehrenfeucht-Fraïssé game argument one can show that $\mathcal{B}_{X_1} \equiv_r \mathcal{B}_{X_2}$ is true for all $r \in \mathbb{N}$ and all finite sets X_1 and X_2 of size at least 2^r. Thus, for every quantifier rank $r \in \mathbb{N}$, we can find sufficiently large finite sets X_1 and X_2 of odd and even cardinality, respectively, that cannot be distinguished by FO(σ)-sentences of quantifier rank r.

A detailed exposition of Gurevich's proof can be found in the textbook [14]. □

Potthoff's Example. We consider unordered finite binary trees T where every node is either a leaf or has exactly two children. The *height* of a leaf x of T is the length of the path from the root to x. The height of T is the largest height of a leaf of T. A tree T is *full* if all leaves are of the same height.

Let $\sigma := \{E, D\}$ be the signature consisting of two binary relation symobls E and D. We represent an unordered finite binary tree T by a σ-structure $\mathcal{A}_T$ whose universe is the set of nodes of T, and where E is the directed edge relation connecting every non-leaf node with its two children, and D is the descendant relation, i.e., the transitive closure of E.

Theorem 4.2 (Potthoff [18]). *There exists an order-invariant* FO($\sigma \cup \{<\}$)*-sentence* $\varphi_{\textit{Potthoff}}$*, but no* FO($\sigma$)*-sentence, such that for every full unordered finite binary tree* T *we have:* $\mathcal{A}_T \models \varphi_{\textit{Potthoff}} \iff T$ *is of even height.*

Proof sketch. Part 1: Construction of the $<$-inv-FO(σ)*-sentence* $\varphi_{\textit{Potthoff}}$. To keep the description of the formula simple, we here present a sentence $\varphi_{\textit{Potthoff}}$ that is order-invariant only on the class of all *full* unordered finite binary trees. A more sophisticated sentence that is order-invariant on all finite σ-structures is outlined below, after Lemma 4.3.

For constructing $\varphi_{\textit{Potthoff}}$ let us consider a full binary tree T of height h. We use the linear order $<$ to order the children of each node a of T: If b_1 and b_2 are a's children and $b_1 < b_2$, then b_1 is called *the 1-child*, and b_2 is called *the 2-child* of a. Now, we consider the *zig-zag-path* which starts in the root, visits the root's 1-child, that node's 2-child, that nodes 1-child, etc. I.e., the zig-zag-path is the path $(x_0, x_1, x_2, \ldots, x_h)$ where x_0 is the root, x_h is a leaf, and for odd $i \geq 1$, x_i is the 1-child of x_{i-1}, whereas for even $i \geq 1$, x_i is the 2-child of x_{i-1}.

As T is a *full* binary tree, the height h of T is even if and only if the last node of the zig-zag-path is a 2-child — and this is exactly the statement made by the formula $\varphi_{\textit{Potthoff}}$. Note that a formula making this statement will be order-invariant on the structure $\mathcal{A}_T$, for all *full* binary trees T.

The statement "the last node of the zig-zag-path is a 2-child" can be formalised by an FO($\sigma \cup \{<\}$)-sentence $\varphi_{\textit{Potthoff}}$, which states the following:

(1) There exists a node x_0 which is the root, and there exists a node x_h which is a leaf, such that
(2) x_h is the 2-child (w.r.t. $<$) of its parent,
(3) the node x_1 which satisfies $\big(E(x_0, x_1) \wedge D(x_1, x_h)\big)$ is the 1-child (w.r.t. $<$) of its parent, and
(4) for any three nodes u, v, w such that $E(u, v)$ and $E(v, w)$ and $\big(w{=}x_h \vee D(w, x_h)\big)$ we have that v is the 1-child of its parent iff w is the 2-child of its parent.

Part 2: Proof of the non-expressibility in FO(σ). By using a standard Ehrenfeucht-Fraïssé game argument, one can show that $\mathcal{A}_{T_1} \equiv_r \mathcal{A}_{T_2}$ is true for all $r \in \mathbb{N}$ and all full unordered finite binary trees T_1 and T_2 of height $\geq 2^{r+1}$ (the duplicator's winning strategy in the Ehrenfeucht-Fraïssé game is a straightforward generalisation of the winning strategy in the game on two linear orders, cf. e.g. [14]). Thus, for every $r \in \mathbb{N}$, we can find sufficiently big full unordered finite binary trees of odd and even height, respectively, that cannot be distinguished by FO(σ)-sentences of quantifier rank r. $\square$

For completeness, let us give the precise statement of Potthoff's result. Instead of constructing an order-invariant FO(σ)-formula, Potthoff constructs an FO(σ')-formula for the signature $\sigma' = \sigma \cup \{C_1, C_2\}$, where C_1 and C_2 are unary relation symbols. An *ordered* finite binary tree T is represented by the σ'-structure $\mathcal{B}_T$ which is the expansion of the structure $\mathcal{A}_T$ by unary relations C_1 and C_2, where C_1 consists of all nodes which are the first child of their parent, and C_2 consists of all nodes which are the second child of their parent.

Lemma 4.3 (Lemma 5.1.8 in [18]). *There is an* FO(σ')*-sentence* $\psi_{Potthoff}$ *such that for every ordered finite binary tree* T *we have:* $\mathcal{B}_T \models \psi_{Potthoff} \iff$ *every leaf of* T *is of even height.*

The order-invariant sentence $\varphi_{Potthoff}$ whose existence is claimed in Theorem 4.2 is now obtained as the conjunction of

- a straightforward FO(σ)-axiomatisation of unordered binary trees, and
- the formula obtained from $\psi_{Potthoff}$ by replacing atoms of the form $C_i(x)$ (for $i \in \{1, 2\}$) with an FO($E, <$)-formula stating that x is the i-child (w.r.t. $<$) of its parent.

Otto's Example. For every $n \in \mathbb{N}_{\geq 1}$ and every undirected graph G on $2n$ vertices, we consider a σ-structure $\mathcal{S}_{2n}(G)$ into which G is embedded. The signature $\sigma = \{E, \sim, \in, V, V', P'\}$ consists of three binary relation symbols $E, \sim, \in$ and three unary relation symbols V, V', P'.

We let $\sigma' := \sigma \setminus \{E\}$ and define, for each $n \in \mathbb{N}_{\geq 1}$, the σ'-structure $\mathcal{S}_{2n}$ as follows: The universe of $\mathcal{S}_{2n}$ is partitioned into three disjoint sets V, V', P', where $V = \{v_0, \ldots, v_{2n-1}\}$, $V' = \{v'_0, \ldots, v'_{2n-1}\}$, and $P' = 2^{V'}$. The relation $\in$ is the "element"-relation between V' and $2^{V'}$, connecting for each X in $2^{V'}$ every node $v' \in X$ with X. The relation $\sim$ is the equivalence relation on $V \cup V'$ whose equivalence classes are $\{v_i, v'_i, v_{n+i}, v'_{n+i}\}$ for all $i < n$. For every graph $G = (V, E)$, the σ-structure $\mathcal{S}_{2n}(G)$ is the expansion of the structure $\mathcal{S}_{2n}$ with the graph's edge relation E. An illustration can be found in Figure 1.

Theorem 4.4 (Otto [17]). *There is an order-invariant* FO($\sigma \cup \{<\}$)*-sentence* φ_{Otto}*, but no* FO(σ)*-sentence, such that for every* $n \in \mathbb{N}_{\geq 1}$ *and every graph* G *on* $2n$ *nodes we have:* $\mathcal{S}_{2n}(G) \models \varphi_{Otto} \iff G$ *is connected.*

Proof sketch. Part 1: Construction of the $<$-inv-FO(σ)*-sentence* φ_{Otto}. Otto's proof shows a stronger result, namely that every monadic second-order sentence Φ of signature $\{E\}$ can be translated into an order-invariant FO($\sigma \cup \{<\}$)-sentence φ_Φ, such that

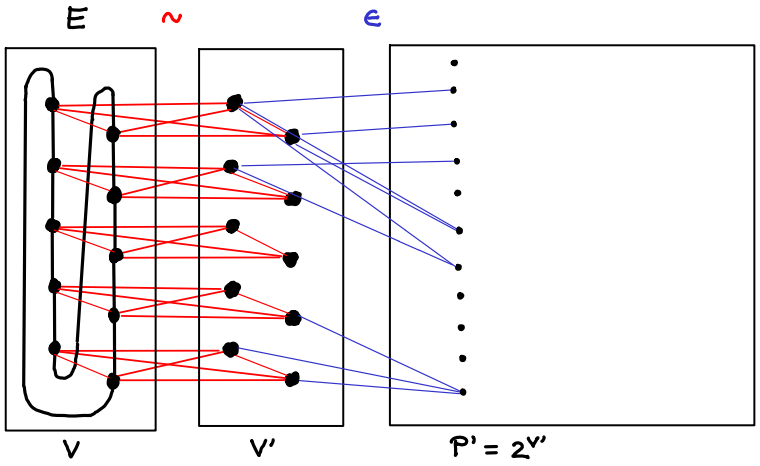


Fig. 1. The σ-structure $S_{2n}(G)$ where $G = G^1_{2n}$ is a cycle on $2n$ nodes $V = \{v_0, v_1, \ldots, v_{2n-1}\}$. G is represented in the leftmost box of the picture. The box in the middle contains the set $V' = \{v' : v \in V\}$. The 4-cliques between V and V' represent the equivalence relation $\sim$. The box on the right contains a node X for each element X in $P' = 2^{V'}$. The edges between the box in the middle and the box on the right represent the $\in$-relation connecting, for each X in $2^{V'}$, every node $v' \in X$ with the node X.

for every $n \in \mathbb{N}_{\geq 1}$ and every graph G on $2n$ nodes we have: $\mathcal{S}_{2n}(G) \models \varphi_\Phi \iff G \models \Phi$.

The claimed formula φ_{Otto} can then be chosen as φ_Φ where Φ is a monadic second-order formalisation of graph connectivity.
For constructing φ_Φ, the following observations are crucial:

(1) We can use the linear order $<$ to define a bijection $\beta_<$ from V to V' such that $v \sim \beta_<(v)$ for every $v \in V$. This bijection can be described by an FO$(V, V', \sim, <)$-formula.
(2) Using this bijection, we can identify V with V'. And using P' and the $\in$-relation between V' and P', we can simulate monadic second-order quantification over V by first-order quantification of elements in P'. Utilising this, it is straightforward to translate Φ into an FO$(\sigma \cup \{<\})$-sentence ψ_Φ.
(3) Finally, the formula φ_Φ is chosen as the conjunction of ψ_Φ with an FO(V, E)-sentence stating that E is a subset of $V \times V$, and an FO(σ')-axiomatisation of σ'-structures isomorphic to $\mathcal{S}_{2n}$ for some $n \in \mathbb{N}_{\geq 1}$.

The resulting formula φ_Φ is satisfied by $\mathcal{A}^\iota$, for a finite σ-structure $\mathcal{A}$ and an embedding ι of $\mathcal{A}$ if, and only if, $\mathcal{A}$ is isomorphic to $\mathcal{S}_{2n}(G)$ for some $n \in \mathbb{N}_{\geq 1}$ and some graph G on $2n$ vertices satisfying Φ. This also shows that φ_Φ is order-invariant on all finite σ-structures.

Part 2: Proof of the non-expressibility in FO(σ). For each $n \in \mathbb{N}_{\geq 1}$ let G^1_{2n} be the cycle $(v_0, v_1, \ldots, v_{2n-1}, v_0)$, and let G^2_{2n} be the disjoint union of the two cycles $(v_0, v_1, \ldots, v_{n-1}, v_0)$ and $(v_n, v_{n+1}, \ldots, v_{2n-1}, v_n)$. An illustration of G^1_{2n} is given in

the leftmost box of Figure 1. It suffices to show that for all $r \in \mathbb{N}$ and all sufficiently large n, the structures $\mathcal{S}_{2n}(G^1_{2n})$ and $\mathcal{S}_{2n}(G^2_{2n})$ cannot be distinguished by FO(σ)-sentences of quantifier rank r.

For each $b \in \{1,2\}$ let $\widehat{G}^b_{2n}$ be the expansion of G^b_{2n} where each node is labeled by its equivalence class with respect to $\sim$ in $\mathcal{S}_{2n}(G^b_{2n})$. An easy Hanf-locality argument (cf., [14]) shows that for every $r \in \mathbb{N}$ and all sufficiently large n, the structures $\widehat{G}^1_{2n}$ and $\widehat{G}^2_{2n}$ cannot be distinguished by first-order sentences of quantifier rank r.

A closer inspection of the structures $\mathcal{S}_{2n}(G^1_{2n})$ and $\mathcal{S}_{2n}(G^2_{2n})$ shows that the duplicator's winning strategy in the r-round Ehrenfeucht-Fraïssé game on $\widehat{G}^1_{2n}$ and $\widehat{G}^2_{2n}$ can be translated into a winning strategy on $\mathcal{S}_{2n}(G^1_{2n})$ and $\mathcal{S}_{2n}(G^2_{2n})$. Thus, the latter two structures cannot be distinguished by FO(σ)-sentences of quantifier rank r. □

5 Limitations of the Expressive Power of Arb-Invariant FO

The results stated in this section hold for arbitrary signatures σ. For simplicity of presentation, however, we let $\sigma = \{E\}$ be the signature consisting of a single binary relation symbol E. Thus, finite σ-structures are finite directed graphs.

Connections between arb-inv-FO(σ) and Circuit Complexity. For proving non-expressibility results for arb-inv-FO(σ), tools from circuit complexity are of major use. We assume that the reader is familiar with basic notions and results in circuit complexity (cf., e.g., the textbook [3]). We consider Boolean circuits consisting of AND- and OR-gates of unbounded fan-in, NOT-gates, input gates, and constant gates **0** and **1**. The size of a circuit is the number of its gates, and the depth is the length of the longest path from an input gate to the output gate.

Let C_m be a circuit with $m \in \mathbb{N}_{\geq 1}$ input gates, and let $w \in \{0,1\}^m$ be a bitstring of length m. We say that C_m *accepts* w if C_m evaluates to 1 when for every $i \leq m$ the i-th input gate of C_m is assigned the i-th symbol of w.

The non-expressibility proofs for arb-inv-FO(σ) presented in this section rely on Håstad's following well-known circuit lower bound.

Theorem 5.1 (Håstad [10]). *There exist numbers $\ell, m_0 > 0$ such that for every $d \in \mathbb{N}$ with $d \geq 2$ and every $m \in \mathbb{N}$ with $m \geq m_0$ the following is true: No circuit of depth d and size at most $2^{\ell \cdot \sqrt[d-1]{m}}$ accepts exactly those bitstrings $w \in \{0,1\}^m$ that contain an even number of ones.*

To establish the connection between circuits and arb-inv-FO(σ), we need to represent graphs by bitstrings. This is done in a straightforward way: Consider a directed graph $G = (V,E)$ on $|V| = n$ nodes. Let ι be an embedding of G into $[n]$, and let $(a_{i,j})_{0 \leq i,j < n}$ be the adjacency matrix of G with respect to ι, i.e., $a_{i,j} = 1$ if $(\iota^{-1}(i), \iota^{-1}(j)) \in E$, and $a_{i,j} = 0$ otherwise. The bitstring representation $\mathrm{Rep}^\iota(G)$ of G w.r.t. ι is then chosen as $\mathrm{Rep}^\iota(G) := a_{0,0} \cdots a_{0,n-1} \cdots a_{n-1,0} \cdots a_{n-1,n-1}$. I.e., $\mathrm{Rep}^\iota(G)$ is the concatenation of all rows of the adjacency matrix $(a_{i,j})_{0 \leq i,j < n}$. The connection between FO(σ_{arb}) and Boolean circuits is obtained by the following result.

Theorem 5.2 (Immerman [13]). *For every* FO(σ_{arb})*-sentence* φ *there exist numbers* $d, s \in \mathbb{N}$ *(with* $d \geq 2$*) such that for every* $n \in \mathbb{N}_{\geq 1}$ *there is a circuit* C_{n^2} *with* n^2 *input gates, depth* d*, and size* n^s *such that the following is true for all graphs* $G = (V, E)$ *with* $|V| = n$ *and all embeddings* ι *of* V *into* $[n]$*:* C_{n^2} *accepts* $\mathrm{Rep}^\iota(G) \iff G^\iota \models \varphi$.

Proof sketch. For every fixed n, we translate φ into a Boolean formula with n^2 Boolean variables:

(1) Replace every existential quantification "$\exists x$" of φ into a big disjunction $\bigvee_{0 \leq x < n}$,
(2) replace every universal quantification "$\forall x$" of φ into a big conjunction $\bigwedge_{0 \leq x < n}$.

After these two tranformation steps, the "atomic formulas" remaining in φ are either of the form $E(x, y)$ for $x, y \in [n]$, where E is the edge relation of the graph, or of the form $P(x_1, \ldots, x_r)$ for $x_1, \ldots, x_r \in [n]$, where P is a symbol for a numerical predicate $P^{\mathbb{N}}$ of arity r (here, equality of the form "$x_1 = x_2$" is also viewed as a numerical predicate).

(3) Replace every "atom" of the form $E(x, y)$ for $x, y \in [n]$ with the Boolean variable $a_{x,y}$ representing the edge from $\iota^{-1}(x)$ to $\iota^{-1}(y)$ in G^ι, and
(4) replace every "atom" of the form $P(x_1, \ldots, x_r)$ for $x_1, \ldots, x_r \in [n]$, where P is a symbol for a numerical predicate $P^{\mathbb{N}}$ by the constant **1** if $(x_1, \ldots, x_r) \in P^{\mathbb{N}}$, and by the constant **0** otherwise.

The result of this transformation is a Boolean formula with Boolean variables $a_{x,y}$ for $x, y \in [n]$. This Boolean formula can easily be turned into the desired circuit C_{n^2}. □

As an immediate consequence of the Theorems 5.2 and 5.1 one obtains the following.

Corollary 5.3. *There is no* arb-inv-FO(σ)*-sentence* φ *that is satisfied by exactly those finite directed graphs that consist of an even number of edges.*

Proof. For contradiction, assume that φ is an arb-inv-FO(σ)-sentence satisfied by exactly those finite directed graphs that consist of an even number of edges.

Let d, s and C_{n^2} (for every $n \in \mathbb{N}_{\geq 1}$) be chosen according to Theorem 5.2. It can easily be seen that the circuit C_{n^2} accepts exactly those bitstrings w of length n^2 that contain an even number of ones: Every $w \in \{0,1\}^{n^2}$ can be viewed as the bitstring representation $\mathrm{Rep}^\iota(G)$ of some graph $G = (V, E)$ on n nodes. Clearly, $G^\iota \models \varphi$ iff $G \models \varphi$ iff G contains an even number of edges. By Theorem 5.2 we furthermore know that $G^\iota \models \varphi$ iff $w = \mathrm{Rep}^\iota(G)$ is accepted by C_{n^2}.

Thus, for $m := n^2$, C_m is a circuit of depth d and size $n^s = m^{s/2}$ that accepts exactly those bitstrings $w \in \{0,1\}^m$ that contain an even number of ones. However, for any fixed ℓ and all sufficiently large m we have $m^{s/2} < 2^{\ell \cdot \sqrt[d-1]{m}}$, contradicting Theorem 5.1. □

Gaifman Locality of Arb-Invariant FO. A *k-ary query* q is a mapping that associates with every finite directed graph $G = (V, E)$ a k-ary relation $q(G) \subseteq V^k$, which is invariant under isomorphisms, i.e., if π is an isomorphism from a graph G to a graph H, then for all $\overline{a} = (a_1, \ldots, a_k) \in A^k$ we have $\overline{a} \in q(G)$ iff $\pi(\overline{a}) = (\pi(a_1), \ldots, \pi(a_k)) \in q(H)$. Every arb-inv-FO($\sigma$)-formula $\varphi(\overline{x})$ with k free variables defines a k-ary query q_φ via $q_\varphi(G) = \{\overline{a} \in V^k : G \models \varphi[\overline{a}]\}$.

The notion of *Gaifman locality* is a standard tool for showing that particular queries are not definable in certain logics (cf., e.g., the textbook [14] for an overview). For presenting the precise definition of Gaifman locality, we need the following notation.

The *Gaifman graph* of a directed graph $G = (V, E)$ is the undirected graph $\mathcal{G}(G)$ with the same vertex set as G, where for any $a, b \in V$ with $a \neq b$ there is an undirected edge between a and b iff $(a, b) \in E$ or $(b, a) \in E$. The *distance* $dist^G(a, b)$ between two nodes a, b of G is the length of the shortest path between a and b in $\mathcal{G}(G)$.

For every $r \in \mathbb{N}$, the *r-ball* $N_r^G(a)$ around a node a is the set of all nodes b with $dist^G(a, b) \leq r$. The r-ball $N_r^G(\overline{a})$ around a tuple $\overline{a} = (a_1, \ldots, a_k) \in V^k$ is the union of the r-balls around the nodes $a_1, \ldots, a_k$. The *r-neighborhood* of $\overline{a}$ is the induced subgraph $\mathcal{N}_r^G(\overline{a})$ of G on $N_r^G(\overline{a})$.

Definition 5.4 (Gaifman locality). Let $k \in \mathbb{N}_{\geq 1}$ and $f : \mathbb{N} \to \mathbb{N}$. A k-ary query q is *Gaifman $f(n)$-local* if there is an $n_0 \in \mathbb{N}$ such that for every $n \in \mathbb{N}$ with $n \geq n_0$ and every directed graph $G = (V, E)$ on n nodes, the following is true for all k-tuples $\overline{a}, \overline{b} \in V^k$ with $(\mathcal{N}_{f(n)}^G(\overline{a}), \overline{a}) \cong (\mathcal{N}_{f(n)}^G(\overline{b}), \overline{b})$: $\quad \overline{a} \in q(G) \iff \overline{b} \in q(G)$.

I.e., in a graph of size n, a query that is Gaifman $f(n)$-local cannot distinguish between k-tuples of nodes whose neighborhoods of radius $f(n)$ are isomorphic. Gaifman locality is a powerful tool for showing that certain queries cannot be defined by formulas of particular logics.

Example 5.5. Let F be a class of formulas such that every query q definable by a formula in F is Gaifman $f_q(n)$-local for a function $f_q : \mathbb{N} \to \mathbb{N}$ where $f_q(n) \leq n/5$ for all sufficiently large n. Then, none of the following queries is definable in F:

- *reach*$(G) := \{(a, b) : G$ contains a directed path from node a to node $b\}$,
- *cycle*$(G) := \{a : a$ is a node that lies on a cycle of $G\}$,
- *triangle-reach*$(G) := \{a : a$ is reachable from a triangle in $G\}$,
- *same-distance*$(G) := \{(a, b, c) : dist^G(a, c) = dist^G(b, c)\}$.

Assume, for contradiction, that *reach* is definable in F. By assumption, $f_{reach}(n) \leq n/5$ for all sufficiently large n. Now, consider for each n the graph G_n consisting of two disjoint directed paths of length $n/2$, and let a be the first node of the first path, let b be the last node of the first path, and let b' be the last node of the second path. Then, $\mathcal{N}_{n/5}^{G_n}(a, b)$ consists of two disjoint paths of length $n/5$, where a is the first node of the first path and b is the last node of the second path. Obviously, $(\mathcal{N}_{n/5}^{G_n}(a, b), (a, b)) \cong (\mathcal{N}_{n/5}^{G_n}(a, b'), (a, b'))$. Thus, due to the assumed Gaifman $f_{reach}(n)$-locality of the query *reach*, we have $(a, b) \in$ *reach*(G_n) iff $(a, b') \in$ *reach*(G_n). However, in G_n there is a directed path from a to b, but no directed path from a to b' — a contradiction.
Similar constructions can be used to show that none of the queries *cycle*, *triangle-reach*, *same-distance* is definable in F.

It is well-known that FO(σ)-definable queries are Gaifman local with constant locality radius, i.e., for every FO(σ)-definable query q there is a constant c such that q is Gaifman c-local [11]. This can be generalised to order-invariant FO:

Theorem 5.6 (Grohe, Schwentick [8]). *Order-invariant* FO *is Gaifman local with constant locality radius. I.e., for every* $<$-inv-FO(σ)*-definable query* q *there is a constant* c *such that* q *is Gaifman* c*-local.*

The result for *constant* locality radius (independent of the size of the graph) cannot be lifted to arb-invariant FO: In [2] it was shown that for every $d \in \mathbb{N}$ there is an $\{+,\times\}$-inv-FO(σ)-definable unary query q_d that is not Gaifman $(\log n)^d$-local. But still, for arb-invariant FO we get a Gaifman locality result for neighborhoods whose radius is bounded polylogarithmically in the size of the underlying graphs:

Theorem 5.7 (Anderson, Melkebeek, Schweikardt, Segoufin [2]). *Arb-invariant* FO *is Gaifman local with polylogarithmic locality radius. I.e., for every query* q *definable in* arb-inv-FO(σ) *there is a constant* c *such that* q *is Gaifman* $(\log n)^c$*-local.*

Note that this suffices to conclude that none of the queries mentioned in Example 5.5 is definable in arb-inv-FO(σ).

The proof of Theorem 5.6 relies on a sophisticated construction using Ehrenfeucht-Fraïssé games. A simplified proof a weaker version of Theorem 5.6 can be found in the textbook [14]. The proof of Theorem 5.7 exploits the connection between arb-invariant FO and Boolean circuits. In the following, we present the proof of a weaker version of Theorem 5.7 for the particular case of *unary* queries and the notion of *weak Gaifman locality* [14], where "$\overline{a} \in q(G) \iff \overline{b} \in q(G)$" needs to be true only for those tuples $\overline{a}$ and $\overline{b}$ whose $f(n)$-neighborhoods are disjoint.

Definition 5.8 (Weak Gaifman locality). Let $f : \mathbb{N} \to \mathbb{N}$. A unary query q is called *weakly Gaifman* $f(n)$*-local* if there is an $n_0 \in \mathbb{N}$ such that for every $n \in \mathbb{N}$ with $n \geq n_0$ and every directed graph $G = (V, E)$ on n nodes, the following is true for all nodes $a, b \in V$ with $(\mathcal{N}^G_{f(n)}(a), a) \cong (\mathcal{N}^G_{f(n)}(b), b)$ and $N^G_{f(n)}(a) \cap N^G_{f(n)}(b) = \emptyset$: $a \in q(G) \iff b \in q(G)$.

We give a proof of the following weaker version of Theorem 5.7:

Proposition 5.9. *For every unary query* q *definable in* arb-inv-FO(σ) *there is a constant* c *such that* q *is weakly Gaifman* $(\log n)^c$*-local.*

Proof. Let q be a unary query expressed by an arb-inv-FO(σ)-formula $\varphi(x)$. By using a variation of Theorem 5.2, there exist numbers $d, s \in \mathbb{N}$ such that for every $n \in \mathbb{N}_{\geq 1}$ there is a circuit C_{n^2+n} with n^2+n input gates, depth d, and size n^s such that the following is true for all graphs $G = (V, E)$ with $|V| = n$, for all nodes $a \in V$, and all embeddings ι of V into $[n]$:

$$C_{n^2+n} \text{ accepts } \mathrm{Rep}^\iota(G, a) \quad \iff \quad G \models \varphi[a]. \tag{1}$$

Here, $\mathrm{Rep}^\iota(G, a) = \mathrm{Rep}^\iota(G)\mathrm{Rep}^\iota(a)$ is the bitstring representation of (G, a), where $\mathrm{Rep}^\iota(a)$ is the bitstring $b_0 \cdots b_{n-1}$ with $b_{\iota(a)} = 1$ and $b_j = 0$ for all $j \neq \iota(a)$.

For contradiction, let us now assume that for every $c \in \mathbb{N}$ the query q defined by $\varphi(x)$ is *not* weakly Gaifman $(\log n)^c$-local. Thus, in particular for $c := 2(d-1)$ we obtain that for all $n_0 \in \mathbb{N}$ there exists an $n \geq n_0$, and

$(*)$: a graph $G = (V, E)$ on n nodes, and nodes $a, b \in V$ such that for $m := (\log n)^c = (\log n)^{2(d-1)}$ we have:

$$(\mathcal{N}_m^G(a), a) \cong (\mathcal{N}_m^G(b), b), \quad N_m^G(a) \cap N_m^G(b) = \emptyset, \quad G \models \varphi[a], \quad G \not\models \varphi[b].$$

Claim. *The circuit C_{n^2+n} can be transformed into a circuit $\tilde{C}_m$ on m input bits, such that $\tilde{C}_m$ has the same depth and size as C_{n^2+n} and accepts exactly those bitstrings $w \in \{0,1\}^m$ that contain an even number of ones.*

Before proving this claim, let us point out how it can be used to conclude the proof of Proposition 5.9. According to the claim, $\tilde{C}_m$ is a circuit of depth d and size n^s, which accepts exactly those bitstrings $w \in \{0,1\}^m$ that contain an even number of ones.

From Theorem 5.1 we know that the size n^s of $\tilde{C}_m$ must be bigger than $2^{\ell \cdot \sqrt[d-1]{m}}$. However, we had chosen $m = (\log n)^{2(d-1)}$, and hence $2^{\ell \cdot \sqrt[d-1]{m}} = 2^{\ell \cdot (\log n)^2} = n^{\ell \cdot \log n} > n^s$ for all sufficiently large n — a contradiction! Thus, for concluding the proof of Proposition 5.9, it suffices to prove the claim.

Proof of the claim. Let $G = (V, E)$ be the graph chosen according to $(*)$, and let ι be an arbitrary embedding of V into $[n]$. The idea is to define, for every bitstring $w \in \{0,1\}^m$, a graph G_w such that

$$(**): \qquad (G_w, a) \;\cong\; \begin{cases} (G, a) & \text{if } w \text{ contains an even number of ones,} \\ (G, b) & \text{otherwise.} \end{cases}$$

The circuit $\tilde{C}_m$ is constructed in such a way that on input $w \in \{0,1\}^m$ it does the same as circuit C_{n^2+n} does on input $\mathrm{Rep}^\iota(G_w, a)$. From (1) we then obtain that

$$\tilde{C}_m \text{ accepts } w \iff C_{n^2+n} \text{ accepts } \mathrm{Rep}^\iota(G_w, a) \iff G_w \models \varphi[a].$$

If w contains an even number of ones, $(G_w, a) \cong (G, a)$. As we know from $(*)$ that $G \models \varphi[a]$, we therefore obtain that $\tilde{C}_m$ accepts w.
If w contains an odd number of ones, $(G_w, a) \cong (G, b)$. As we know from $(*)$ that $G \not\models \varphi[b]$, we therefore obtain that $\tilde{C}_m$ does not accept w.
Thus, circuit $\tilde{C}_m$ accepts exactly those $w \in \{0,1\}^m$ that contain an even number of 1s.

Definition of G_w: From $(*)$ we know that there exists an isomorphism π from $\mathcal{N}_m^G(a)$ to $\mathcal{N}_m^G(b)$ with $\pi(a) = b$. Furthermore, we know that $N_m^G(a) \cap N_m^G(b) = \emptyset$, and thus G contains no edges that link vertices of $N_{m-1}^G(a)$ with vertices of $N_{m-1}^G(b)$. For $x \in \{a, b\}$, we partition $N_m^G(x)$ into *shells* $S_i(x) := \{y \in V : \mathit{dist}^G(x, y) = i\}$, for all $i \leq m$. Note that $\pi(S_i(a)) = S_i(b)$.
In the following, we write S_i for the set $S_i(a) \cup S_i(b)$.
For a bitstring $w = w_1 \cdots w_m \in \{0,1\}^m$ the graph G_w is defined as follows:

- G_w has the same vertex set V as the graph G.
- All edges of G that do *not* link a vertex of shell S_{i-1} with a vertex of shell S_i, for some $i \leq m$, are copied into G_w.
- Edges of G that link a vertex of shell S_{i-1} with a vertex of shell S_i, for some $i \leq m$ are modified depending on the i-th bit w_i of the bitstring w:

- If $w_i = 0$, then edges of G that link a vertex of shell S_{i-1} with a vertex of shell S_i are copied into G_w.
- If $w_i = 1$, then for every edge of G that links a vertex u of shell $S_{i-1}(a)$ with a vertex v of shell $S_i(a)$, we insert into G_w an edge that links vertex u (in shell $S_{i-1}(a)$) with vertex $\pi(v)$ (in shell $S_i(b)$), and we also insert the according edge that links vertex $\pi(u)$ (in shell $S_{i-1}(b)$) with the vertex v (in shell $S_i(a)$).

Thus, for every i with $w_i = 1$, the roles of the shells $S_i(a)$ and $S_i(b)$ are swapped. It is straightforward to see that the resulting graph G_w satisfies $(**)$; see Figure 2 for an illustration.

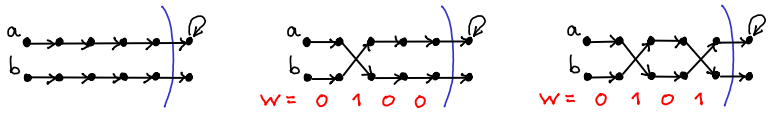


Fig. 2. Illustration of the graph G_w for neighborhoods of radius $m = 4$

Construction of $\tilde{C}_m$: Let us fix an embedding ι of V into $[n]$. The circuit $\tilde{C}_m$ is obtained from C_{n^2+n} by replacing the input gates of C_{n^2+n} as follows:

Let $u, v \in V$, and let $g_{\mu,\nu}$ for $\mu := \iota(u)$ and $\nu := \iota(v)$ be the input gate of C_{n^2+n} that corresponds to the entry $a_{\mu,\nu}$ of G's adjacency matrix w.r.t. ι (i.e., $a_{\mu,\nu} = 1$ iff $(u, v) \in E$).

In case that (u, v) does *not* belong to $(S_{i-1} \times S_i) \cup (S_i \times S_{i-1})$ for any $i \leq m$, the gate $g_{\mu,\nu}$ is replaced by the constant gate **1** if $(u, v) \in E$, and by the constant gate **0** if $(u, v) \notin E$.

In case that (u, v) belongs to $(S_{i-1}(a) \times S_i(a)) \cup (S_i(a) \times S_{i-1}(a))$ for some $i \leq m$, let $g := g_{\mu,\nu}$, and let $g', \tilde{g}, \tilde{g}'$ be the input gates of C_{n^2+n} corresponding to the potential edges $(\pi(u), \pi(v))$, $(u, \pi(v))$, and $(\pi(u), v)$, respectively.
If $(u, v) \notin E$, then $g, g', \tilde{g}, \tilde{g}'$ are replaced by the constant gate **0**.
If $(u, v) \in E$, then g and g' are replaced by $\neg w_i$, whereas $\tilde{g}$ and $\tilde{g}'$ are replaced by w_i, where w_i is the input gate for the i-th bit of the input bitstring of length m.

It is straightforward to see that on input $w \in \{0, 1\}^m$ the circuit $\tilde{C}_m$ does the same as circuit C_{n^2+n} does on input $\mathrm{Rep}^\iota(G_w, a)$. This completes the proof of Proposition 5.9. □

6 Some Open Questions

We conclude with a list of open research questions:

(1) Is addition-invariant FO Gaifman local with *constant* locality radius? (Cf., [8,14].)
(2) Can addition-invariant FO define string-languages that are not regular? (See [24] for details.)
(3) Are there analogues of the Theorems 5.6 and 5.7 for the notion of *Hanf locality*? (Cf. [14] for the definition of Hanf locality.)
(4) Does order-invariant FO have a zero-one law? (See [6,14] for zero-one laws.)
(5) Are there *decidable* characterisations of order-invariant FO, addition-invariant FO, or $\{+, \times\}$-invariant FO? (See [4,24,9] for related results.)

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Anderson, M., van Melkebeek, D., Schweikardt, N., Segoufin, L.: Locality from circuit lower bounds. SIAM Journal on Computing 41(6), 1481–1523 (2012)
3. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge Univ. Press (2009)
4. Benedikt, M., Segoufin, L.: Towards a characterization of order-invariant queries over tame structures. Journal of Symbolic Logic 74(1), 168–186 (2009)
5. Courcelle, B.: The monadic second-order logic of graphs X: Linear orderings. Theoretical Computer Science 160(1&2), 87–143 (1996)
6. Ebbinghaus, H.-D., Flum, J.: Finite model theory. Springer (1999)
7. Ganzow, T., Rubin, S.: Order-invariant MSO is stronger than Counting MSO in the finite. In: Proc. STACS 2008, pp. 313–324 (2008)
8. Grohe, M., Schwentick, T.: Locality of order-invariant first-order formulas. ACM Transactions on Computational Logic 1(1), 112–130 (2000)
9. Harwath, F., Schweikardt, N.: Regular tree languages, cardinality predicates, and addition-invariant FO. In: Proc. STACS 2012, pp. 489–500 (2012)
10. Håstad, J.: Computational limitations for small-depth circuits. PhD thesis. MIT (1986)
11. Hella, L., Libkin, L., Nurmonen, J.: Notions of locality and their logical characterizations over finite models. Journal of Symbolic Logic 64(4), 1751–1773 (1999)
12. Immerman, N.: Relational queries computable in polynomial time. Information and Control 68(1-3), 86–104 (1986)
13. Immerman, N.: Languages that capture complexity classes. SIAM Journal on Computing 16(4), 760–778 (1987)
14. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
15. Makowsky, J.A.: Invariant Definability and **P**/*poly*. In: Gottlob, G., Grandjean, E., Seyr, K. (eds.) CSL 1998. LNCS, vol. 1584, pp. 142–158. Springer, Heidelberg (1999)
16. Niemistö, H.: On locality and uniform reduction. In: Proc. LICS 2005, pp. 41–50 (2005)
17. Otto, M.: Epsilon-logic is more expressive than first-order logic over finite structures. Journal of Symbolic Logic 65(4), 1749–1757 (2000)
18. Potthoff, A.: Logische Klassifizierung regulärer Baumsprachen. PhD thesis, Christian-Albrechts-Universität Kiel (1994)
19. Rossman, B.: Successor-invariant first-order logic on finite structures. Journal of Symbolic Logic 72(2), 601–618 (2007)
20. Rossman, B.: On the constant-depth complexity of k-clique. In: Proc. STOC 2008, pp. 721–730 (2008)
21. Schweikardt, N.: Arithmetic, first-order logic, and counting quantifiers. ACM Transactions on Computational Logic 6(3), 634–671 (2005)
22. Schweikardt, N.: On the expressive power of monadic least fixed point logic. Theoretical Computer Science 350(2-3), 325–344 (2006)
23. Schweikardt, N.: On the expressive power of logics with invariant uses of arithmetic predicates. In: Ong, L., de Queiroz, R. (eds.) WoLLIC 2012. LNCS, vol. 7456, pp. 85–87. Springer, Heidelberg (2012)
24. Schweikardt, N., Segoufin, L.: Addition-invariant FO and regularity. In: Proc. LICS 2010, pp. 273–282 (2010)
25. Schwentick, T.: On Winning Ehrenfeucht Games and Monadic NP. Annals of Pure and Applied Logic 79(1), 61–92 (1996)
26. Vardi, M.: The complexity of relational query languages. In: Proc. STOC 1982, pp. 137–146 (1982)

Exponential Lower Bounds for Refuting Random Formulas Using Ordered Binary Decision Diagrams

Luke Friedman* and Yixin Xu**

Rutgers University, Piscataway, NJ, USA
{lbfried,yixinxu}@cs.rutgers.edu

Abstract. A propositional proof system based on ordered binary decision diagrams (OBDDs) was introduced by Atserias et al. in [3]. Krajíček proved exponential lower bounds for a strong variant of this system using feasible interpolation [14], and Tveretina et al. proved exponential lower bounds for restricted versions of this system for refuting formulas derived from the Pigeonhole Principle [20]. In this paper we prove the first lower bounds for refuting randomly generated unsatisfiable formulas in restricted versions of this OBDD-based proof system. In particular we consider two systems OBDD* and OBDD+; OBDD* is restricted by having a fixed, predetermined variable order for all OBDDs in its refutations, and OBDD+ is restricted by having a fixed order in which the clauses of the input formula must be processed. We show that for some constant $\epsilon > 0$, with high probability an OBDD* refutation of an unsatisfiable random 3-CNF formula must be of size at least $2^{\epsilon n}$, and an OBDD+ refutation of an unsatisfiable random 3-XOR formula must be of size at least $2^{\epsilon n}$.

1 Introduction

Propositional proof complexity is both an approach for attacking the famous P vs. NP problem, and also for obtaining a better theoretical understanding of algorithms for the satisfiability problem. A whole landscape of proof systems of varying strengths has been mapped out and studied – see for instance [17] for general background in this field. From a complexity theory standpoint the situation is similar to that of circuit complexity – for certain restricted systems such as the resolution system exponential lower bounds on the size of refuting many different families of unsatisfiable propositional formulas have been proved. However, for strong systems such as extended Frege, researchers have failed to prove even super-linear lower bounds for any family of unsatisfiable formulas, despite the fact that if such a system had polynomial-size refutations of all unsatisfiable formulas this would imply that NP = CO-NP.

* Partially supported by NSF grants CCF-0832787 and CCF-1064785.
** Partially supported by NSF grant CCF-0832787.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 127–138, 2013.

In this paper we prove the first lower bounds on the size of refuting randomly-generated unsatisfiable 3-CNF and 3-XOR formulas in proof systems based on ordered binary decision diagrams. Random CNF formulas have been studied extensively, both as a benchmark for measuring in some sense the average case performance of SAT solving algorithms, and also as a tool for proving proof complexity lower bounds. It is well-known that if a random 3-CNF formula on n variables is generated with Δn clauses for large enough constant Δ, then with high probability the formula will be unsatisfiable. The lack of structure in these formulas makes them hard to refute; indeed, it is conceivable that they require exponential size refutations in *any* proof system, and since even generating candidate hard formulas for strong proof systems can be difficult [16], they are a natural choice for lower bound proofs and developing new techniques for them is a worthwhile task.

Along with random 3-CNF formulas, we also consider random 3-XOR formulas, which are formulas whose clauses are satisfied if and only if one or three of its literals are satisfied. Unlike in the 3-CNF case, determining satisfiability of a 3-XOR formula is known to be computable in polynomial time, since such formulas can be equivalently represented as a system of linear equations over $\mathbf{F}_2$, and then an algorithm such as Gaussian elimination can be used to test the solvability of the system. However, random 3-XOR formulas retain a lot of the important properties of random 3-CNF formulas, and because they are easier to reason about they have been useful in proving lower bounds for weak proof systems (e.g. [2]).

Ordered Binary Decision Diagrams (OBDDs) are data structures for representing Boolean functions that were originally introduced in [5] and have found a wide variety of applications in areas of computer science such as VLSI design and model checking. They have also emerged as a basis for SAT solving algorithms that have been demonstrated to be competitive on certain classes of formulas with the state-of-the-art DPLL based solvers that are generally used in practice [15],[13]. Informally they are read-once branching programs where variables must be queried according to a fixed order. Part of what makes OBDDs so useful is that their relatively rigid structure makes it possible to manipulate them efficently: For any given Boolean function f on n variables and variable order π there is a unique (up to isomorphism) minimal OBDD computing f, and operations such as taking the conjunction of two OBDDs and determining whether an OBDD representing a function f_1 majorizes an OBDD representing a function f_2 (i.e for all x, $f_1(x) \geq f_2(x)$) are computable in polynomial time [5].

A refutation system based on OBDDs was introduced in [3]. The basic idea of such a system is simple: Given an unsatisfiable 3-CNF (or 3-XOR) formula $\mathcal{F}$, an OBDD refutation of $\mathcal{F}$ with respect to a variable order π is a sequence $\mathrm{OBDD}_1, \mathrm{OBDD}_2, \ldots \mathrm{OBDD}_t \equiv 0$, where each OBDD_i uses the variable order π and is either the OBDD representation of a clause from $\mathcal{F}$ (an axiom), or is the conjunction of two OBDDs derived earlier (i.e. $\mathrm{OBDD}_i = \mathrm{OBDD}_j \wedge \mathrm{OBDD}_k$ for some $j, k < i$). One can also include a weakening rule, so that OBDD_i may

also be an OBDD such that OBDD_i majorizes OBDD_j for some $j < i$. Such a refutation system is sound and complete, and because computing the conjunction of two OBDDs can be done in polynomial time (as well as determining whether one OBDD majorizes another in the case of a weakening), verifying whether a refutation is correct is also polynomial time computable. Thus these OBDD-based systems qualify as propositional proof systems in the formal sense introduced by Cook and Reckhow [7]. The OBDDs representing axioms in this type of refutation are small, as well as the final OBDD OBDD_t. Therefore, if a refutation in one of these OBDD-based systems has a polynomial number of steps, whether it is polynomial size or not depends only on whether one of the intermediate OBDDs computed along the way has super-polynomial size. The only non-deterministic choices the prover must make are which variable order π to use, and in what order to combine OBDDs. (If a weakening rule exists, the prover must also choose when and how to use it). These choices can be crucial however in determining the size of the refutation; for instance, it is a simple exercise to show that for certain functions the OBDD representation has size $O(n)$ according to one variable order yet size $\Omega(2^n)$ according to another order.

By restricting the options the prover has in making these choices, one can define different variants of this OBDD-based system that have varying strengths. One reason for doing so is that no current OBDD-based SAT solver takes full advantage of the power offered by the underlying OBDD proof system in its unrestricted form. This is a common phenomenon in SAT solving – basing solvers on more powerful proof systems does not necessarily make the solvers better. The reason is that as the proof systems become more powerful, trying to deterministically make the non-deterministic choices of the proof system becomes an increasingly difficult task. This is highlighted by the fact that the best general purpose SAT solvers in use today are variants of the DPLL algorithm, which is based on the resolution system, one of the weakest proof systems that has been studied. In the case of OBDD based systems, it is not clear how to best make use of the full weakening rule, and even determining the best variable order to use in an OBDD representation of a single function is an NP-complete problem [4]. Particularly when considering random formulas, because of the symmetry and lack of structure it seems unlikely that one variable order would be exponentially better than another, or even if such a good order did exist that it could be found efficiently. However, the sheer number of different possible variable orders make proving such a fact difficult from a technical standpoint.

From a theoretical point of view, restricting these OBDD systems creates interesting intermediate systems. It was proved in [3] that allowing unrestricted use of the weakening rule makes the OBDD proof system as strong as CP*, a variant of the cutting planes system where coefficients are represented in unary, that is strictly stronger than resolution and for which the only known lower bounds are based on feasible interpolation. However, if we do not allow weakening the story changes significantly – in this case there exist certain families of unsatisfiable formulas for which the smallest OBDD refutations are exponentially larger than the smallest resolution refutations [20]. Despite this apparent weakness, it has

not been proved that the Frege system, a powerful system that could even conceivably be optimal, can polynomially simulate this restricted OBDD system. The reason is that the lines of Frege systems are formulas, which cannot directly simulate the dag-like structure of OBDDs. Thus studying different variations of restricted OBDD-based systems is one possible route towards bridging the gap between systems we know to be weak and those for which we do not have lower bounds on natural families of formulas.

Krajíček gave exponential lower bounds for the OBDD-based system of [3] in its full generality using a form of the feasible interpolation method [14], and these are currently the only lower bounds known for this strongest variant. Tveretina et al. showed that if the weakening rule is disallowed, then an OBDD-based refutation of the pigeonhole principle must have exponential size [20], building upon a similar result from Groote and Zantema [11], who had also restricted the system to only consider specific variable orders. In this paper we take a first step towards understanding the limitations of OBDD-based systems to refute *random* formulas by proving exponential lower bounds for certain restricted variants.

In particular we consider two restricted OBDD-based systems, which we can denote by OBDD* and OBDD+. In both systems the weakening rule is excluded. In the OBDD* system the variable order that will be used for the refutation is fixed before the random formula is chosen. Because random formulas are generated symmetrically with respect to the variables, without loss of generality we can fix the identity order $\mathcal{I}$ that orders a set of variables $x_1, x_2, \ldots x_n$ as $x_1 < x_2 < \cdots < x_n$. In the OBDD* system the prover has the freedom to combine OBDDs during the refutation in an arbitrary way. In the OBDD+ system, the prover has the freedom to choose any variable order π *after* seeing the random formula ϕ that is to be refuted. However, during the refutation, the clauses of ϕ (represented as OBDDs) must be combined in a predetermined fashion corresponding to some canonical ordering of the clauses in ϕ.

The following two theorems are our main results:

Theorem 1. *Let Δ be a sufficiently large constant. There exists an $\epsilon > 0$, such that with high probability when ϕ is a random 3-CNF formula on n variables with clause density Δn, ϕ is unsatisfiable and any OBDD* refutation of ϕ must have size at least $2^{\epsilon n}$.*[1]

Theorem 2. *Let Δ be a sufficiently large constant. There exists an $\epsilon > 0$ such that with high probability when ϕ is a random 3-XOR formula on n variables with clause density Δn, ϕ is unsatisfiable and any OBDD+ refutation of ϕ must have size at least $2^{\epsilon n}$.*

The progress we have made in this paper is summarized in Figure 1.

[1] This theorem can be proved almost identically in the case where we consider a random 3-XOR formula as well. Also, a close inspection of the proof shows that for either the 3-CNF or 3-XOR case, if instead of fixing the variable order $\mathcal{I}$ we allow the prover to fix any set S of $2^{\delta n}$ variable orders for sufficiently small δ before seeing the random formula ϕ, then to choose one of the variable orders from S after seeing ϕ, the theorem still holds in this scenario as well.

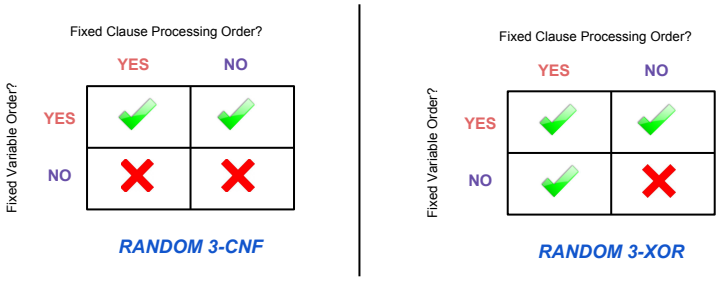


Fig. 1. A summary of the results from this paper. We consider different OBDD-based proof systems, none of which include a weakening rule. The systems differ according to two possible restrictions: (1) Is the variable order that will be used in the refutation fixed before the random formula is chosen? (2) Are the clauses processed in the refutation according to a canonical order in which they appear in the input formula? We consider both random 3-CNF and random 3-XOR formulas. A check mark appears in the box corresponding to a given proof system and type of random formula if we prove exponential lower bounds for this combination in this paper, and an X appears in the box if proving lower bounds in this case is still open.

2 Preliminaries and Notations

We will denote a set of n Boolean variables as $\{x_1, \ldots, x_n\}$. A literal x_i^j, $j \in \{0,1\}$, is either a variable or its negation. An assignment α to a set of n variables is a function $[n] \to \{0,1\}$, where $[n]$ denotes the set $\{1,2,\ldots,n\}$. α satisfies a literal x_i^j if and only if $\alpha(i) = j$.

A clause C is a set of literals. An assignment α satisfies C as a CNF clause if and only if α satisfies some literal in C. α satisfies C as an XOR clause if and only if α satisfies an odd number of literals in C. A 3-CNF (3-XOR) formula $\mathcal{F}$ over n variables is a list of clauses $(C_1, \ldots, C_m)$, where each of the clauses contains three literals from variables in the set $\{x_1, \ldots x_n\}$. It is satisfied by an assignment α if and only if every clause in $\mathcal{F}$ is satisfied by α as a CNF (XOR) clause. If it is irrelevant whether we are referring to a 3-CNF formula or a 3-XOR formula, we will often refer to the formula simply as a 3-formula.

Definition 1 (Random 3-formula). *A random 3-formula ϕ on n variables with clause density Δ is a 3-formula $(C_1, \ldots, C_{\Delta n})$, where each clause C_i is chosen uniformly at random from all of the $2^3\binom{n}{3}$ possible clauses.*

Let π be a total order on a set of variables $\{x_1, \ldots, x_n\}$. We will refer to π simply as an *order*. Alternatively, we can view π as a permutation such that $\pi(i) = j$ if and only if the i-th variable in the order of π is x_j. We will also write $\pi^{-1}(j) = i$ to indicate that $\pi(i) = j$. We also define the identity order $\mathcal{I}$ such that for all i, $\mathcal{I}(i) = i$.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function on n variables and let $\mathbf{z} \in \{0,1\}^t$ for $t \leq n$. We define $f|_{\pi,\mathbf{z}}$ to be the function $f' : \{0,1\}^{n-t} \to \{0,1\}$ that

is the function f restricted so that for each $1 \leq i \leq t$, if $\pi(i) = j$, then x_j is fixed to the constant value $\mathbf{z}_i$.

Definition 2 (OBDD). *Given an order π on $\{x_1, \ldots, x_n\}$, an ordered binary decision diagram with respect to π, denoted by OBDD_π, is a branching program with the following structure. An OBDD_π is a layered directed acyclic graph with layers 1 through $n+1$. Layer 1 contains a single root node, and layer $(n+1)$ contains two final nodes, one labeled with the value 0 and the other labeled with the value 1. Every node in layers 1 through n has outdegree two: such a node v on level i has one outgoing edge to a node on level $i+1$ labeled with the value 0, and another outgoing edge to a node on level $i+1$ labeled with the value 1.*

An OBDD_π defines a Boolean function $\{0,1\}^n \to \{0,1\}$ in the following way. For an assignment α on n variables, we start at the root node, and for $i = 1$ to n, advance along the edge labeled with $\alpha(\pi(i))$. When this process is complete, we will have arrived at one of the final nodes. If this final node is labeled with 0, then we define $\mathrm{OBDD}_\pi(\alpha) = 0$, and otherwise we define $\mathrm{OBDD}_\pi(\alpha) = 1$, where now we are associating α with an n bit string in the natural way.

$|\mathrm{OBDD}_\pi|$ denotes the size (the number of nodes) of the OBDD.

An important property of OBDDs is that for a given Boolean function $f : \{0,1\}^n \to \{0,1\}$ and an ordering π, there is a unique minimal OBDD_π up to isomorphism computing f [5]. Thus for a given f we can safely refer to $\mathrm{OBDD}_\pi(f)$ as *the* OBDD computing f according to π.

The following simple theorem (and corollary) provide general techniques for proving lower bounds on $|\mathrm{OBDD}_\pi(f)|$.

Theorem 3 ([18]). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function on n variables and π an order. Let $k = |\{f|_{\pi,\mathbf{z}} : \mathbf{z} \in \{0,1\}^t\}|$ (i.e., k counts the number of distinct subfunctions of f that can be produced by fixing the first t variables according to π). Then the t-th level of $OBDD_\pi(f)$ contains k nodes.*

Corollary 1 ([19]). *Let f be a Boolean function on n variables and π an order. Suppose the following conditions hold*

1. *$x_1, \cdots, x_t$ are the least t variables according to π for some $t < n$.*
2. *$B \subseteq \{1, \ldots, t\}$.*
3. *$\mathbf{z} \in \{0,1\}^t$.*
4. *For all $\mathbf{x}, \mathbf{x}' \in \{0,1\}^t$, if $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{z}_i$ for all $i \notin B$, then there exists $\mathbf{y} \in \{0,1\}^{n-t}$ such that $f(\mathbf{x}, \mathbf{y}) \neq f(\mathbf{x}', \mathbf{y})$.*

Then $|\mathrm{OBDD}_\pi(f)| \geq 2^{|B|}$.

Definition 3 ($\mathbf{OBDD}^*_\pi$ refutation). *Given an unsatisfiable 3-formula $\mathcal{F}$ and an order π, an $OBDD^*_\pi$ refutation of $\mathcal{F}$ is a sequence $OBDD_\pi(f_1), OBDD_\pi(f_2), \cdots, OBDD_\pi(f_t \equiv 0)$ such that for each f_i one of the following conditions is satisfied:*

1. f_i *is a clause of* $\mathcal{F}$. *(In this case we say that* f_i *is an* axiom.*)*
2. $f_i = f_j \wedge f_k$ *for some* $j, k < i$.

The size of the OBDD$^*_\pi$ *refutation is defined as* $\sum_{i=1}^{t} |OBDD_\pi(f_i)|$.

We define $S^*_\pi(\mathcal{F})$ *to be the minimum size of any* OBDD^*_π *refutation of* $\mathcal{F}$. *In this paper we focus on* $\pi = \mathcal{I}$ *and thus will refer to* $S^*_{\mathcal{I}}(\mathcal{F})$.

Definition 4 (OBDD$^+_\pi$ refutation). *An* OBDD^+_π *refutation of an unsatisfiable 3-formula* $\mathcal{F} = (C_1, \ldots, C_m)$ *is an* OBDD^*_π *refutation where the clauses of* $\mathcal{F}$ *are processed one at a time in order. Precisely, an* OBDD^+_π *refutation of* $\mathcal{F}$ *is a sequence* $OBDD_\pi(f_1), OBDD_\pi(f_2), \cdots, OBDD_\pi(f_{2m} = 0)$ *where for* $1 \le i \le m$, $f_i = C_i$, $f_{m+1} = C_1$, *and for* $m + 2 \le j \le 2m$, $f_j = f_{j-1} \wedge f_{j-m}$. *We define* $S^+_\pi(\mathcal{F})$ *to be the size of the unique* OBDD^+_π *refutation of* $\mathcal{F}$, *and we define* $S^+(\mathcal{F})$ *to be the minimum over* π *of* $S^+_\pi(\mathcal{F})$.

We will make use of the following bounds related to satisfiability thresholds.

Theorem 4. *[8] There exists* $\Delta^* \le 4.51$ *such that for large* n, *w.h.p a random 3-CNF formula with* n *variables and clause density* $\Delta > \Delta^*$ *will be unsatisfiable.*

Theorem 5. *[9] There exists* $\Delta^* \le 0.91$ *such that for large* n, *w.h.p a random 3-XOR formula with* n *variables and clause density* $\Delta > \Delta^*$ *will be unsatisfiable.*

We will also need the following lemma, which is a restatement of a result that appeared in [6]. For S a subset of the clauses of a 3-formula $\mathcal{F}$, let $var(S)$ be the set of all variables that appear in at least one of the clauses of S (ignoring the sign of the literal). We call a 3-formula $\mathcal{F}$ on n variables an (x, y)-expander if for all subsets S of the clauses of $\mathcal{F}$ such that $|S| \le xn$, $|var(S)| \ge y|S|$.

Lemma 1. *[6] For all* $y < 2$ *and* $\Delta > 0$, *there exists positive* x *such that w.h.p a random 3-formula on* n *variables with clause density* Δ *will be an* (x, y)*-expander.*

Finally, we need two results on systems of distinct representatives that follow from Hall's marriage theorem. For a clause C, let $var(C)$ be the set of variables appearing in C, and for a set of clauses S, let $var(S) = \cup_{C \in S} var(C)$. We say a subset S of clauses has a system of distinct representatives (SDR) if there is a one-to-one function $\sigma : S \to var(S)$ such that for all $C \in S$, $\sigma(C) \in var(C)$.

Lemma 2. *[12] Let* S *be a subset of clauses.* S *has an SDR if and only if for all* $S' \subseteq S$, $|var(S')| \ge |S'|$.

Lemma 3. *[6] Let* S *be a set of clauses and* V *a set of variables.* S *has an SDR* σ *with at most* t *elements of* V *in the range of* σ *if and only if it has an SDR and for all* $S' \subseteq S$, $|S'| - |var(S') \setminus V| \le t$.

3 Proof of the OBDD* Case

The purpose of this section is to prove Theorem 1, which we now restate.

Theorem 6 (restatement of Theorem 1). *Let $\Delta > 4.51$. There exists a constant $\epsilon > 0$ such that, with high probability when ϕ is a random 3-CNF formula on n variables with clause density Δ, ϕ is unsatisfiable and $S^*_{\mathcal{I}}(\phi) \geq 2^{\epsilon n}$.*

The main work in our proof of Theorem 6 is proving the following lemma.

Lemma 4. *Let $\Delta > 4.51$. There exist constants $\delta, \epsilon > 0$ such that, with high probability when ϕ is a random 3-CNF formula on n variables with clause density Δ, ϕ is a $(\delta, 1.9)$ expander and the following holds: Let S be any subset of the clauses of ϕ such that $\delta n/2 \leq |S| \leq \delta n$, and let f_S be the conjunction of these clauses. Then $|OBDD_{\mathcal{I}}(f_S)| \geq 2^{\epsilon n}$.*

Proof (of Theorem 6). Because $\Delta > 4.51$, by Theorem 4 with high probability ϕ will be unsatisfiable. Let $P = OBDD_{\mathcal{I}}(f_1), OBDD_{\mathcal{I}}(f_2), \cdots, OBDD_{\mathcal{I}}(f_t = 0)$ be an $\text{OBDD}^*_{\mathcal{I}}$ refutation of ϕ. Each f_i is a conjunction of some subset of clauses S of ϕ. Let $|f_i|$ denote $|S|$.

By Lemma 1, there exists a constant δ such that with high probability ϕ is a $(\delta, 1.9)$ expander. By Lemma 2 this means that any subset S of clauses of ϕ with $|S| \leq \delta n$ has an SDR. Any set of clauses S that has an SDR σ is satisfiable, since an assignment that for each clause $C \in S$ sets $\sigma(C)$ to the value that satisfies C will satisfy S. Therefore, since f_t is the constant 0 function, which is trivially unsatisfiable, $|f_t| \geq \delta n$. For every f_i that is an axiom, we have $|f_i| = 1$. If $f_i = f_j \wedge f_k$ for some $j, k < i$, then $|f_i| \leq |f_j| + |f_k|$. Therefore, for each $i \in t$, $|f_i| \leq 2 \max_{j<i} |f_j|$. This implies that there exists $i \in [t]$ such that $\delta n/2 \leq |f_i| \leq \delta n$. By Lemma 4, $|OBDD_{\mathcal{I}}(f_i)| \geq 2^{\epsilon n}$, so P has size at least $2^{\epsilon n}$.

The remainder of this section is devoted to proving Lemma 4. First we prove a few other lemmas that will be useful towards this goal. Some of these proofs are omitted for space reasons (but can be seen in the full version [10]).

Lemma 5. *Let $\Delta > 0$ and $0 < \delta < \Delta$ be some constant. There exists $\epsilon > 0$, such that with high probability when ϕ is a random 3-formula on n variables with clause density Δ, for any set T of ϵn variables, the number of clauses from ϕ that contain a variable from T is less than δn.*

Lemma 6. *Let $\Delta > 0$ and $0 < \delta < \Delta$ be some constant. There exists $\epsilon > 0$, such that with high probability when ϕ is a random 3-formula on n variables with clause density Δ, the following property holds: For all sets S of clauses from ϕ with $|S| \geq \delta n$, there exists a set of clauses $T \subseteq S$ with $|T| = \epsilon n$ such that the clauses in T are disjoint (i.e. no two clauses of T share a common variable).*

Definition 5 (splits). *Let t be a positive integer less than n and $\mathcal{F}$ a 3-formula. For a clause $C \in \mathcal{F}$, we say that t left-splits C according to an order π if there is exactly one variable $x_i \in var(C)$ such that $\pi^{-1}(i) \leq t$. In this case we define $\text{left}_{C,t,\pi} = x_i$. Similarly, we say that t right-splits C according to an order π if there is exactly one variable $x_i \in var(C)$ such that $\pi^{-1}(i) > t$, and in this case define $\text{right}_{C,t,\pi} = x_i$. If t either right-splits or left-splits C, then we will sometimes simply say that t splits C.*

Lemma 7. *Let $\Delta, \delta > 0$ be any constants, and for some $0 < \epsilon < 1$, let $\Gamma_\epsilon = \{\lceil \epsilon n \rceil, \lceil 2\epsilon n \rceil, \lceil 3\epsilon n \rceil, \cdots, \lceil (1-\epsilon)n \rceil\}$. Then with high probability when ϕ is a random 3-formula on n variables with clause density Δ, for any set of clauses S from ϕ, with $|S| \geq \delta n$, there exists $t \in \Gamma_\epsilon$ such that at least $(\delta - 7\epsilon\Delta)\epsilon n$ of the clauses are left-split by t according to $\mathcal{I}$.*

Lemma 8. *Let $\Delta > 4.51$, and let δ' be the constant that comes out of Lemma 1 such that with high probability a random 3-formula with clause density Δ is a $(\delta', 1.9)$ expander. Let $\delta < \delta'$ be some constant. There exists constants $\gamma, \epsilon > 0$, such that with high probability when ϕ is a random 3-formula on n variables with clause density Δ, the following property holds: For all sets T of clauses from ϕ, with $\delta n \leq |T| \leq \delta' n$, there exists $S \subseteq T$ such that*

1. *$|S| = \gamma n$.*
2. *There exists $t \in \Gamma_\epsilon$ such that every clause $C \in S$ is left-split by t according to $\mathcal{I}$.*
3. *The clauses of S are disjoint.*
4. *T has an SDR σ, such that for every clause $C \in S$, exactly one variable in C is in the range of σ.*

Proof. Suppose the conclusions of Lemma 1, Lemma 6, and Lemma 7 hold with respect to ϕ (which occurs with high probability). Let T be a set of clauses from ϕ such that $\delta n \leq |T| \leq \delta' n$. By Lemma 7, there exists a constant ϵ such that for $\lambda = (\delta - 7\epsilon\Delta)\epsilon > 0$, at least λn clauses from T are left-split by $t \in \Gamma_\epsilon$ according to $\mathcal{I}$. Call this set of λn clauses U.

By Lemma 6, for some constant λ' we can find a set of $\lambda' n$ disjoint clauses $U' \subseteq U$. Now we invoke Lemma 3 to show that there exists an SDR σ for T such that at most $1.6\lambda' n$ of the $3\lambda' n$ variables in $var(U')$ are in the range of σ. To do this it suffices to show that for any set of clauses $S' \subseteq T$, $|S'| - |var(S') \setminus var(U')| \leq 1.6\lambda' n$. If $|S'| \leq 1.6\lambda' n$ then trivially the inequality is satisfied. Otherwise, if $|S'| > 1.6\lambda' n$, then because ψ is a $(\delta', 1.9)$ expander, $|var(S')| \geq 1.9|S'|$, so

$$|S'| - |var(S') \backslash var(U')| \leq -0.9|S'| + 3\lambda' n \leq -1.44\lambda' n + 3\lambda' n \leq 1.6\lambda' n$$

Because there are at most $1.6\lambda' n$ variables from $var(U')$ in the range of σ, there must exist a set of clauses $S \subseteq U'$, with $|S| = 0.4\lambda' n$, such that for every clause $C \in S$, exactly one variable in C is in the range of σ. This set S satisfies the requirements of the lemma.

We are now ready to prove Lemma 4.

Proof (of Lemma 4).

By Lemma 1, there exists $\delta > 0$ such that with high probability ϕ is a $(\delta, 1.9)$ expander, and also with high probability the conclusion of Lemma 8 holds.

Let S be a subset of the clauses of ϕ such that $\delta n/2 \leq |S| \leq \delta n$. Let $S' \subseteq S$ be the set guaranteed to exist by Lemma 8 with the four properties from that lemma, and σ the corresponding SDR for S. Let $\text{left}_{S'} = \{\text{left}_{C,t,\mathcal{I}} : C \in S'\}$.

In order to prove the lemma we will make use of Corollary 1 to show that $|OBDD_{\mathcal{I}}(f_S)| \geq 2^{|\text{left}_{S'}|} \geq 2^{\epsilon n}$ for some constant $\epsilon > 0$. Our set B from that theorem will be $\text{left}_{S'}$. Define $\mathbf{z} \in \{0,1\}^t$ as follows. For each $1 \leq i \leq t$ such that $x_i = \sigma(C)$ for some clause C, let $\mathbf{z}_i$ be the value that satisfies the clause C. Assign all other values of $\mathbf{z}$ arbitrarily.

To finish the proof of the lemma, we need that for all $\mathbf{x}, \mathbf{x}' \in \{0,1\}^t$, if $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{z}_i$ for all $i \notin B$, then there exists $\mathbf{y} \in \{0,1\}^{n-t}$ such that $\phi(\mathbf{x}, \mathbf{y}) \neq \phi(\mathbf{x}', \mathbf{y})$.

Let $\mathbf{x}, \mathbf{x}' \in \{0,1\}^t$ such that $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{z}_i$ for all $i \notin B$. Let j be an index such that $\mathbf{x}_j \neq \mathbf{x}'_j$. Let C be the clause from S' such that $\mathbf{x}_j = \text{left}_{C,t,\mathcal{I}}$.

Define $\mathbf{y}$ as follows. Let p and q be the two indices other than j such that $\mathbf{x}_p$ and $\mathbf{x}_q$ are in the clause C. Define $\mathbf{y}_{p-t}$ and $\mathbf{y}_{q-t}$ each to be the value that does not satisfy the clause C. For each clause $D \neq C$ such that $D \in S'$, let r and s be the two indices greater than t such that $\mathbf{x}_r$ and $\mathbf{x}_s$ are in the clause D. Define $\mathbf{y}_{r-t}$ and $\mathbf{y}_{s-t}$ each to be the value that satisfies D. For any index i such that $t < i \leq n$ and $\mathbf{x}_i = \sigma(E)$ for some clause E other than C, define $\mathbf{y}_{i-t}$ to be the value that satisfies the clause E. Assign all other values of $\mathbf{y}$ arbitrarily. Note that because the clauses in S' are disjoint and for each clause C in S' exactly one of the variables of C is in the range of σ, it is always possible to form the partial assignment $\mathbf{y}$ according to these rules. (Note in particular that if $\sigma(E) = x$ for some clause $E \notin S'$, then $x \notin var(S')$).

Either $\mathbf{x}_j$ satisfies the clause C, or $\mathbf{x}'_j$ does. Assume without loss of generality that $\mathbf{x}_j$ does. Then $\phi(\mathbf{x}', \mathbf{y}) = 0$, since the assignment $(\mathbf{x}', \mathbf{y})$ does not satisfy the clause C. However, $\phi(\mathbf{x}, \mathbf{y}) = 1$, since the assignment $(\mathbf{x}, \mathbf{y})$ satisfies every clause in ϕ. This completes the proof.

4 Proof of the OBDD+ Case

The purpose of this section is to prove Theorem 2, which we now restate.

Theorem 7 (Restatement of Theorem 2). *Let $\Delta > 0.91$. There exists a constant $\epsilon > 0$ such that, with high probability when ϕ is a random 3-XOR formula on n variables with clause density Δ, ϕ is unsatisfiable and $S^+(\phi) \geq 2^{\epsilon n}$.*

The following three lemmas are needed in the proof; for space reasons we are forced to omit their proofs and the proof of Theorem 7 (all of which appear in [10]).

Lemma 9. *For $0 < \epsilon < 1$, let $\Gamma_\epsilon = \{\lceil \epsilon n \rceil, \lceil 2\epsilon n \rceil, \lceil 3\epsilon n \rceil, \ldots, \lceil (1-\epsilon) n \rceil\}$. Let $\Delta > 0.5$. There exists $\epsilon, \delta > 0$ such that, with high probability when ϕ is a random 3-formula on n variables with clause density Δ, the following property holds: For* any *order π, there exists some $t_\pi \in \Gamma_\epsilon$ such that more than δn of the clauses from ϕ are split by t_π according to π.*[2]

[2] In fact, using a slightly more complicated first moment argument, one can prove the stronger statement that this lemma holds even if we fix $t_\pi = n/2$.

Lemma 10. *There exists $\lambda > 0$ such that, with high probability when ϕ is a random 3-XOR formula with clause density $\Delta = 0.6$, ϕ is a $(0.6, 1+\lambda)$ expander.*

Lemma 11. *Let ψ be a 3-formula over n variables with clause density Δ such that ψ is a $(\Delta, 1+\delta)$-expander for some $\delta > 0$. Let $U \subseteq \psi$ be a set of disjoint clauses with $|U| = \lambda n$ for some $\lambda > 0$. Let Ψ be a set of variables and f be a bijection from Ψ to U such that for all $x \in \Psi$, x appears in the clause $f(x)$. Then there exists $\epsilon > 0$ such that there is an SDR σ on ψ for which at least ϵn of the variables from Ψ are not in the range of σ.*

5 Future Work

The obvious open problem is to prove lower bounds for refuting random 3-CNF or 3-XOR formulas in an OBDD-based refutation system where neither the variable order nor the order in which clauses are processed in the refutation is constrained. Although it might seem that one could tweak our techniques to get this result, it may be that this is more difficult than appears at first glance.

For instance, suppose we tried to use the same approach of focusing in on a particular OBDD in the refutation of a random 3-CNF formula such that the OBDD represents the conjunction of about δn clauses, for some appropriately chosen fixed δ, in the hopes of showing that the OBDD must be of exponential size. In the restricted systems from this paper, we were able to choose δ to be an arbitarily small constant. However, in the unrestricted system (still without weakening), we would be forced to choose δ to be greater than about $1/6$, because a random formula with clause density just above the threshold *does* contain sub-formulas with about $n/6$ clauses that have small OBDD representations for some variable order. (For instance, one can look for a large set of disjoint clauses and then choose a variable order where the variables from each clause are adjacent in the order). It is much more difficult to reason about sub-formulas in this regime; for instance, to the best of our knowledge it has not even been proved that a random 3-CNF formula with clause density just above the threshold with high probability does not contain an unsatisfiable sub-formula consisting of $n/6$ clauses (let alone that all sub-formulas slightly larger than this must have large OBDD representations). Certainly one cannot rely on the existence of SDRs when considering sub-formulas with clause density this close to the threshold.

Making progress will probably require using a more sophisticated analysis of the structure of random formulas than we do in this paper. A large amount of research has been done on investigating the structure of random CNF and XOR formulas with densities below the respective satisfiability thresholds, including understanding the solution space structure of such formulas and the occurrence of various phase transitions. (See for example [1] for a survey of this work, along with more general information about SAT solving and random formulas). Finding a way to leverage this type of knowledge in this context is probably a key step towards achieving these more difficult lower bounds.

References

1. Achlioptas, D.: Random satisfiability. In: Handbook of Satisfiability, pp. 245–270 (2009)
2. Alekhnovich, M.: Lower bounds for k-DNF resolution on random 3-CNFs. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 251–256 (2005)
3. Atserias, A., Kolaitis, P.G., Vardi, M.Y.: Constraint propagation as a proof system. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 77–91. Springer, Heidelberg (2004)
4. Bollig, B., Wegener, I.: Improving the variable ordering of OBDDs is NP-complete. IEEE Transactions on Computers 45(9), 993–1002 (1996)
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computing 35, 677–691 (1986)
6. Chvátal, V., Szémeredi, E.: Many hard examples for resolution. Journal of the ACM 35(4), 759–768 (1988)
7. Cook, S., Reckhow, R.: The relative efficiency of propositional proof systems. Journal of Symbolic Logic 44, 36–50 (1979)
8. Dubois, O., Boufkhad, Y., Mandler, J.: Typical random 3-sat formulae and the satisfiability threshold. Tech. Rep (10)003, ECCC (2003)
9. Dubois, O., Mandler, J.: The 3-XORSAT threshold. In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, pp. 769–778 (2002)
10. Friedman, L., Xu, Y.: Exponential lower bounds for refuting random formulas using ordered binary decision diagrams. Tech. Rep. TR13-018, Electronic Colloquium on Computational Complexity (2013)
11. Groote, J.F., Zantema, H.: Resolution and binary decision diagrams cannot simulate each other polynomially. Discrete Applied Mathematics 130, 157–171 (2003)
12. Hall, P.: On representatives of subsets. J. London Math. Soc. 10, 26–30 (1935)
13. Huang, J., Darwiche, A.: Toward good elimination ordering for symbolic SAT solving. In: Proceedings of the Sixteenth IEEE Conference on Tools with Artificial Intelligence, pp. 566–573 (2004)
14. Krajíček, J.: An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. Tech. Rep. (07)007, Electronic Colloquium on Computational Complexity (2007)
15. Pan, G., Vardi, M.Y.: Search vs. symbolic techniques in satisfiability solving. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 235–250. Springer, Heidelberg (2005)
16. Razborov, A.A.: Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution (2003) (manuscript)
17. Segerlind, N.: The complexity of propositional proofs. Bulletin of Symbolic Logic 54, 40–44 (2007)
18. Seiling, D., Wegener, I.: NC-algorithms for operations on binary decision diagrams. Parallel Processing Letters 3(1), 3–12 (1993)
19. Tveretina, O., Sinz, C., Zantema, H.: An exponential lower bound on OBDD refutations for pigeonhole formulas. In: Athens Colloquium on Algorithms and Complexity. Electronic Proceedings in Theoretical Computer Science (2009)
20. Tveretina, O., Sinz, C., Zantema, H.: Ordered binary decision diagrams, pigeonhole formulas and beyond. Journal on Satisfiability, Boolean Modeling and Computation 7, 35–38 (2010)

Parameterized Resolution with Bounded Conjunction

Stefan Dantchev[1] and Barnaby Martin[2,⋆]

[1] School of Engineering and Computing Sciences, Durham University
Science Laboratories, South Road, Durham DH1 3LE, UK
[2] CNRS / LIX UMR 7161, École Polytechnique, Palaiseau, France

Abstract. We provide separations between the parameterized versions of Res(1) (Resolution) and Res(2). Using a different set of parameterized contradictions, we also separate the parameterized versions of Res*(1) (tree-Resolution) and Res*(2).

1 Introduction

In a series of papers [8, 3–5] a program of *parameterized proof complexity* is initiated and various lower bounds and classifications are extracted. The program generally aims to gain evidence that W[i] is different from FPT (usually W[2], though in the journal version [9] of [8] this becomes W[SAT], and in the note [14] W[1] is entertained). Parameterized proof (in fact, refutation) systems aim at refuting *parameterized contradictions* which are pairs $(\mathcal{F}, k)$ in which $\mathcal{F}$ is a propositional CNF with no satisfying assignment of weight $\leq k$. Several parameterized (hereafter often abbreviated as "p-") proof systems are discussed in [8, 3, 5]. The lower bounds in [8], [3] and [5] amount to proving that the systems p-tree-Resolution, p-Resolution and p-bounded-depth Frege, respectively, are not *fpt-bounded*. Indeed, this is witnessed by the *Pigeonhole principle*, and so holds even when one considers parameterized contradictions $(\mathcal{F}, k)$ where $\mathcal{F}$ is itself an actual contradiction. Such parameterized contradictions are termed "*strong*" in [5], in which the authors suggest these might be the only parameterized contradictions worth considering, as general lower bounds – even in p-bounded-depth Frege – are trivial (see [5]). We sympathise with this outlook, but remind that there are alternative parameterized proof systems built from embedding (see [8, 9]) for which no good lower bounds are known even for general parameterized contradictions.

Krajíček introduced the system Res(j) of Resolution-with-bounded-conjunction in [13]. The tree-like variant of this system is normally denoted Res*(j). Res($j+1$) incorporates Res(j) and is ostensibly more powerful. This was demonstrated first for Res(1) and Res(2) in [2], where a quasi-polynomial separation was given. This was improved in [1], until an exponential separation was given in [16], together with similar separations for Res(j) and Res($j+1$), for $j > 1$. Similar separations of Res*(j) and Res*($j+1$) were given in [11]. We are motivated

⋆ The author was supported by ANR Blanc International ALCOCLAN.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 139–149, 2013.

mainly by the simplified and improved bounds of [7], which use relativisations of the *Least number principle*, LNP_n and an ordered variant thereof, the *Induction principle*, IP_n. The contradiction LNP_n asserts that a partial n-order has no minimal element. In the literature it enjoys a myriad of alternative names: the *Graph Ordering Principle* GOP, *Ordering Principle* OP and *Minimal Element Principle* MEP. Where the order is total it is also known as TLNP and GT. The contradiction IP_n uses the built-in order of $\{1, \ldots, n\}$ and asserts that: 1 has property P, n fails to have property P, and any number having property P entails a larger number also having property P. Relativisation of these involves asserting that everything holds only on some non-empty subset of the domain (in the case of IP_n we force 1 and n to be in this relativising subset).

In the world of parameterized proof complexity, we already have lower bounds for p-Res(j) (as we have for p-bounded-depth Frege), but we are still interested in separating levels p-Res(j). We are again able to use the *relativised least number principle*, RLNP_n to separate p-Res(1) and p-Res(2). Specifically, we prove that (RLNP_n, k) admits a polynomial-sized in n refutation in Res(2), but all p-Res(1) refutations of (RLNP_n, k) are of size $\geq n^{\sqrt{(k-3)/16}}$. Although we use the same principle as [7], the proof given there does not adapt to the parameterized world, and instead we look for inspiration to the proof given in [5] for the *Pigeonhole principle*. For tree-Resolution, the situation is more complicated. The relativisation of IP_n, the *Relativised induction principle* RIP_n, admits fpt-bounded proofs in Res*(1), indeed of size $O(k!)$, therefore we are forced to alter this principle. Thus we come up with the *Relativised vectorised induction principle* RVIP_n. We are able to show that (RVIP_n, k) admits $O(n^4)$ refutations in Res*(2), while every refutation in Res*(1) is of size $\geq n^{k/16}$. Note that both of our parameterized contradictions are "strong", in the sense of [5]. We go on to give extended versions of RVIP_n and explain how they separate p-Res*(j) from p-Res*($j+1$), for $j > 1$.

This paper is organised as follows. After the preliminaries, we give our separations of p-Res*(j) from p-Res*($j+1$) in Section 3 and our separation of p-Res(1) from p-Res(2) in Section 4. We then conclude with some remarks and open questions.

2 Preliminaries

A *parameterized language* is a language $L \subseteq \Sigma^* \times \mathbb{N}$; in an instance $(x, k) \in L$, we refer to k as the *parameter*. A parameterized language is *fixed-parameter tractable* (fpt – and in FPT) if membership in L can be decided in time $f(k).|x|^{O(1)}$ for some computable function f. If FPT is the parameterized analog of P, then (at least) an infinite chain of classes vye for the honour to be the analog of NP. The so-called W-hierarchy sit thus: $\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \ldots \subseteq \mathrm{W}[\mathrm{SAT}]$. For more on parameterized complexity and its theory of completeness, we refer the reader to the monographs [10, 12]. Recall that the *weight* of an assignment to a

propositional formula is the number of variables evaluated to true. Of particular importance to us is the parameterized problem Bounded-CNF-Sat whose input is $(\mathcal{F}, k)$ where $\mathcal{F}$ is a formula in CNF and whose yes-instances are those for which there is a satisfying assignment of weight $\leq k$. Bounded-CNF-Sat is complete for the class W[2], and its complement (modulo instances that are well-formed formulae) PCon is complete for the class co-W[2]. Thus, PCon is the language of *parameterized contradictions*, $(\mathcal{F}, k)$ s.t. $\mathcal{F}$ is a CNF which has no satisfying assignment of weight $\leq k$.

A *proof system* for a parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ is a poly-time computable function $P : \Sigma^* \to \Sigma^* \times \mathbb{N}$ s.t. range$(P) = L$. P is *fpt-bounded* if there exists a computable function f so that each $(x, k) \in L$ has a proof of size at most $f(k)|x|^{O(1)}$. These definitions come from [3–5] and are slightly different from those in [8, 9] (they are less unwieldy and have essentially the same properties). The program of *parameterized proof complexity* is an analog of that of Cook-Reckow [6], in which one seeks to prove results of the form W[2] $\neq$co-W[2] by proving that parameterized proof systems are not fpt-bounded. This comes from the observation that there is an fpt-bounded parameterized proof system for a co-W[2]-complete L if W[2] =co-W[2].

Resolution is a refutation system for sets of clauses (formulae in CNF) $\mathcal{F}$. It operates on clauses by the *resolution* rule, in which from $(P \vee x)$ and $(Q \vee \neg x)$ one can derive $(P \vee Q)$ (P and Q are disjunctions of literals), with the goal being to derive the empty clause. The only other permitted rule in weakening – from P to derive $P \vee \ell$ for a literal ℓ. We may consider a Resolution refutation to be a DAG whose sources are labelled by initial clauses, whose unique sink is labelled by the empty clause, and whose internal nodes are labelled by derived clauses. As we are not interested in polynomial factors, we will consider the *size* of a Resolution refutation to be the size of this DAG. Further, we will measure this size of the DAG in terms of the number of variables in the clauses to be resolved – we will never consider CNFs with number of clauses superpolynomial in the number of variables. We define the restriction of Resolution, *tree-Resolution*, in which we insist the DAG be a tree.

The system of *parameterized Resolution* [8] seeks to refute the parameterized contradictions of PCon. Given $(\mathcal{F}, k)$, where $\mathcal{F}$ is a CNF in variables $x_1, \ldots, x_n$, it does this by providing a Resolution refutation of

$$\mathcal{F} \cup \{\neg x_{i_1} \vee \ldots \vee \neg x_{i_{k+1}} : 1 \leq i_1 < \ldots < i_{k+1} \leq n\}. \tag{1}$$

Thus, in parameterized Resolution we have built-in access to these additional clauses of the form $\neg x_{i_1} \vee \ldots \vee \neg x_{i_{k+1}}$, but we only count those that appear in the refutation.

A *j-clause* is an arbitrary disjunction of conjunctions of size at most j. Res(j) is a system to refute a set of j-clauses. There are four derivation rules. The *$\wedge$-introduction rule* allows one to derive from $P \vee \bigwedge_{i \in I_1} \ell_i$ and $Q \vee \bigwedge_{i \in I_2} \ell_i$, $P \vee Q \vee \bigwedge_{i \in I_1 \cup I_2} \ell_i$, provided $|I_1 \cup I_2| \leq j$ (P and Q are j-clauses). The *cut* (or *resolution*) rule allows one to derive from $P \vee \bigvee_{i \in I} \ell_i$ and $Q \vee \bigwedge_{i \in I} \neg \ell_i$, $P \vee Q$. Finally, the two weakening rules allow the derivation of $P \vee \bigwedge_{i \in I} \ell_i$

from P, provided $|I| \leq j$, and $P \vee \bigwedge_{i \in I_1} \ell_i$ from $P \vee \bigwedge_{i \in I_1 \cup I_2} \ell_i$. Res($j$) is more commonly called Res(k), but here k is a variable reserved specifically for the parameter.

If we turn a Res(j) refutation of a given set of j-clauses Σ upside-down, i.e. reverse the edges of the underlying graph and negate the j-clauses on the vertices, we get a special kind of *restricted branching j-program*. The restrictions are as follows. Each vertex is labelled by a j-CNF which partially represents the information that can be obtained along any path from the source to the vertex (this is a *record* in the parlance of [15]). Obviously, the (only) source is labelled with the constant $\top$. There are two kinds of queries, which can be made by a vertex:

1. Querying a new j-disjunction, and branching on the answer: that is, from $\mathcal{C}$ and the question $\bigvee_{i \in I} \ell_i$? we split on $\mathcal{C} \wedge \bigvee_{i \in I} \ell_i$ and $\mathcal{C} \wedge \bigwedge_{i \in I} \neg \ell_i$.
2. Querying a known j-disjunction, and splitting it according to the answer: that is, from $\mathcal{C} \wedge \bigvee_{i \in I_1 \cup I_2} \ell_i$ and the question $\bigvee_{i \in I_1} \ell_i$? we split on $\mathcal{C} \wedge \bigvee_{i \in I_1} \ell_i$ and $\mathcal{C} \wedge \bigvee_{i \in I_2} \ell_i$.

There are two ways of forgetting information. From $\mathcal{C}_1 \wedge \mathcal{C}_2$ we can move to $\mathcal{C}_1$. And from $\mathcal{C} \wedge \bigvee_{i \in I_1} \ell_i$ we can move to $\mathcal{C} \wedge \bigvee_{i \in I_1 \cup I_2} \ell_i$. The point is that forgetting allows us to equate the information obtained along two different branches and thus to merge them into a single new vertex. A sink of the branching j-program must be labelled with the negation of a j-clause from Σ. Thus the branching j-program is supposed by default to solve the *Search problem for Σ*: given an assignment of the variables, find a clause which is falsified under this assignment.

The equivalence between a Res(j) refutation of Σ and a branching j-program of the kind above is obvious. Naturally, if we allow querying single variables only, we get branching 1-programs – decision DAGs – that correspond to Resolution. If we do not allow the forgetting of information, we will not be able to merge distinct branches, so what we get is a class of decision trees that correspond precisely to the tree-like version of these refutation systems. These decision DAGs permit the view of Resolution as a game between a Prover and Adversary (originally due to Pudlak in [15]). Playing from the unique source, Prover questions variables and Adversary answers either that the variable is true or false (different plays of Adversary produce the DAG). Internal nodes are labelled by conjunctions of facts (*records* to Pudlak) and the sinks hold conjunctions that contradict an initial clause. Prover may also choose to forget information at any point – this is the reason we have a DAG and not a tree. Of course, Prover is destined to win any play of the game – but a good Adversary strategy can force that the size of the decision DAG is large, and many Resolution lower bounds have been expounded this way.

We may consider any refutation system as a parameterized refutation system, by the addition of the clauses given in (1). In particular, parameterized Res(j) – p-Res(j) – will play a part in the sequel.

3 Separating p-Res*(j) and p-Res*(j + 1)

The *Induction Principle* IP_n (see [7]) is given by the following clauses:

$$\begin{array}{rl} P_1, \neg P_n & \\ \bigvee_{j>i} S_{i,j} & i \in [n-1] \\ \neg S_{i,j} \vee \neg P_i \vee P_j & i \in [n-1], j \in [n] \end{array}$$

The *Relativised Induction Principle* RIP_n (see [7]) is similar, and is given as follows.

$$\begin{array}{rl} R_1, P_1, R_n, \neg P_n & \\ \bigvee_{j>i} S_{i,j} & i \in [n-1] \\ \neg S_{i,j} \vee \neg R_i \vee \neg P_i \vee R_j & i \in [n-1], j \in [n] \\ \neg S_{i,j} \vee \neg R_i \vee \neg P_i \vee P_j & i \in [n-1], j \in [n] \end{array}$$

The important properties of IP_n and RIP_n, from the perspective of [7], are as follows. IP_n admits refutation in $\mathrm{Res}^*(1)$ in polynomial size, as does RIP_n in $\mathrm{Res}^*(2)$. But all refutations of RIP_n in $\mathrm{Res}^*(1)$ are of exponential size. In the parameterized world things are not quite so well-behaved. Both IP_n and RIP_n admit refutations of size, say, $\leq 4k!$ in p-$\mathrm{Res}^*(1)$; just evaluate variables $S_{i,j}$ from $i := n-1$ downwards. Thus ask in sequence

$$S_{n-1,n}, S_{n-2,n-1}, S_{n-2,n}, \ldots\ldots, S_{n-k,n-k+1}, \ldots, S_{n-k,n},$$

each level $S_{n-i,n-i+1}, \ldots, S_{n-i,n}$ surely yielding a true answer. Clearly this is an fpt-bounded refutation. We are forced to consider something more elaborate, and thus we introduce the *Relativised Vectorised Induction Principle* RVIP_n below. Roughly speaking, we stretch each single level of RIP_n into n copies of itself in RVIP_n, to make things easier for Adversary.

$$\begin{array}{rl} R_1, P_{1,1}, R_n, \neg P_{n,j} & j \in [n] \\ \bigvee_{l>i, m \in [n]} S_{i,j,l,m} & i, j \in [n] \\ \neg S_{i,j,l,m} \vee \neg R_i \vee \neg P_{i,j} \vee R_l & i \in [n-1], j, l, m \in [n] \\ \neg S_{i,j,l,m} \vee \neg R_i \vee \neg P_{i,j} \vee P_{l,m} & i \in [n-1], j, l, m \in [n] \end{array}$$

3.1 Lower Bound: A Strategy for Adversary over RVIP_n

We will give a strategy for Adversary in the game representation of a $\mathrm{Res}^*(1)$ refutation. For convenience, we will assume that Prover never questions the same variable twice (this saves us from having to demand trivial consistencies in future evaluations).

Information conceded by Adversary of the form $R_i, \neg R_i, P_{i,j}$ and $S_{i,j,l,m}$ makes the element i *busy* ($\neg P_{i,j}$ and $\neg S_{i,j,l,m}$ do not). The *source* is the largest element i for which there is a j such that Adversary has conceded $R_i \wedge P_{i,j}$. Initially, the source is 1. Adversary always answers $R_1, P_{1,1}, R_n, \neg P_{n,j}$ (for $j \in [n]$), according to the axioms. Thus $i := 1$ and n are somehow special, and the size of the set inbetween is $n-2$. In the following, i refers to the first index of a variable.

If i is below the source. When Adversary is asked R_i, $P_{i,j}$ or $S_{i,j,l,m}$, then he answers $\bot$.

If i is above the source. When Adversary is asked R_i, or $P_{i,j}$, then he gives Prover a free choice unless: 1.) R_i is asked when some $P_{i,j}$ was previously answered $\top$ (in this case R_i should be answered $\bot$); or 2.) Some $P_{i,j}$ is asked when R_i was previously answered $\top$ (in this case $P_{i,j}$ should be answered $\bot$). When Adversary is asked $S_{i,j,l,m}$, then again he offers Prover a free choice. If Prover chooses $\top$ then Adversary sets $P_{i,j}$ and R_i to $\bot$.

Suppose i is the source. Then Adversary answers $P_{i,j}$ and $S_{i,j,l,m}$ as $\bot$, unless $R_i \wedge P_{i,j}$ witnesses the source. If $R_i \wedge P_{i,j}$ witnesses the source, then, if k is not the next non-busy element above i, answer $S_{i,j,l,m}$ as $\bot$. If l is the next non-busy element above i, then give $S_{i,j,l,m}$ a free choice, unless $\neg P_{l,m}$ is already conceded by Adversary, in which case answer $\bot$. If Prover chooses $\top$ for $S_{i,j,l,m}$ then Adversary sets R_l and $P_{l,m}$ to $\top$.

Using this strategy, Adversary can not be caught lying until either he has conceded that k variables are true, or he has given Prover at least $n-2$ free choices.

Let $T(p,q)$ be some monotone decreasing function that bounds the size of the game tree from the point at which Prover has answered p free choices $\top$ and q free choices $\bot$. We can see that $T(p,q) \geq T(p+1,q) + T(p,q+1) + 1$ and $T(k, n-2-k) \geq 0$. The following solution to this recurrence can be found in [9].

Corollary 1. *There is an $f \in \Omega(n^{k/16})$ s.t. every* p-Res$^*(1)$ *refutation of* RVIP$_n$ *is of size $\geq f(n)$.*

We may increase the number of relativising predicates to define RVIP^r_n (note RVIP^1_n = RVIP$_n$).

$$\begin{array}{rl}
R^1_1, \ldots, R^r_1, P_{1,1}, R^1_n, \ldots, R^r_n \neg P_{n,j} & j \in [n] \\
\bigvee_{l>i, m \in [n]} S_{i,j,l,m} & i,j \in [n] \\
\neg S_{i,j,l,m} \vee \neg R^1_i \vee \ldots \vee \neg R^r_i \vee \neg P_{i,j} \vee R^1_l & i \in [n-1], j,l,m \in [n] \\
\vdots & \\
\neg S_{i,j,l,m} \vee \neg R^1_i \vee \ldots \vee \neg R^r_i \vee \neg P_{i,j} \vee R^r_l & i \in [n-1], j,l,m \in [n] \\
\neg S_{i,j,l,m} \vee \neg R^1_i \vee \ldots \vee \neg R^r_i \vee \neg P_{i,j} \vee P_{l,m} & i \in [n-1], j,l,m \in [n]
\end{array}$$

We sketch how to adapt the previous argument in order to demonstrate the following.

Corollary 2. *There is an $f \in \Omega(n^{k/16j})$ s.t. every* p-Res$^*(j)$ *refutation of* RVIP^j_n *is of size $\geq f(n)$.*

We use essentially the same Adversary strategy in a branching j-program. We answer questions $\ell_1 \vee \ldots \vee \ell_j$ as either forced or free exactly according to the disjunction of how we would have answered the corresponding ℓ_is, $i \in [j]$, before. That is, if one ℓ_i would give Prover a free choice, then the whole disjunction is given as a free choice. The key point is that once some disjunction involving some subset of $R^1_i, \ldots, R^j_i$ or $P_{i,j}$ (never all of these together, of course), is questioned

then, on a positive answer to this, the remaining unquestioned variables of this form should be set to $\bot$. This latter rule introduces the factor of j in the exponent of $n^{k/16j}$.

3.2 Upper Bound: A Res*$(j+1)$ Refutation of RVIP^j_n

We encourage the reader to have a brief look at the simpler, but very similar, refutation of RIP$_n$ in Res*(2), of size $O(n^2)$, as depicted in Figure 1.

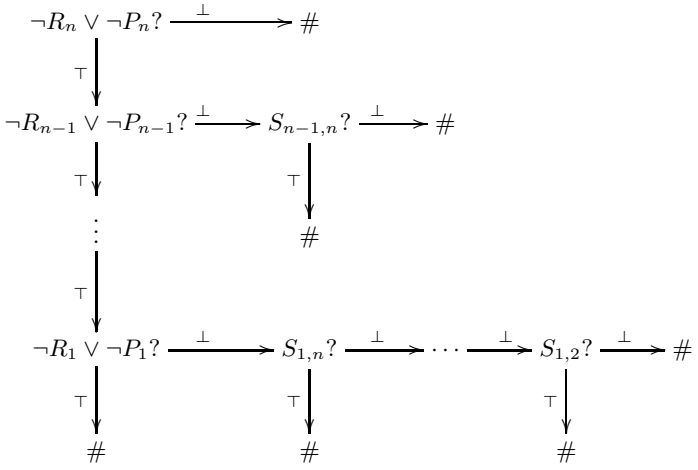


Fig. 1. Refutation of RIP$_n$ in Res*(2)

Proposition 1. *There is a refutation of* RVIP^j_n *in* Res*$(j+1)$*, of size* $O(n^{j+4})$.

Proof. We give the branching program for $j := 1$ in Figure 2. The generalisation to higher j is clear: substitute questions of the form $\neg R_i \vee \neg P_{i,j}$ by questions of the $\neg R^1_i \vee \ldots \vee R^j_i \vee \neg P_{i,j}$.

4 Separating p-Res(1) and p-Res(2)

The *Relativised Least Number Principle* RLNP$_n$ is given by the following clauses:

$$\begin{array}{cc}
\neg R_i \vee \neg L_{i,i} & i \in [n] \\
\neg R_i \vee \neg R_j \vee \neg R_k \vee \neg L_{i,j} \vee \neg L_{j,k} \vee L_{i,k} & i,j,k \in [n] \\
\bigvee_{i \in [n]} S_{i,j} & j \in [n] \\
\neg S_{i,j} \vee \neg R_j \vee R_i & i,j \in [n] \\
\neg S_{i,j} \vee \neg R_j \vee L_{i,j} & i,j \in [n] \\
R_n &
\end{array}$$

The salient properties of RLNP$_n$ are that it is polynomial to refute in Res(2), but exponential in Res(1) (see [7]). Polynomiality clearly transfers to fpt-boundedness in p-Res(2), so we address the lower bound for p-Res(1).

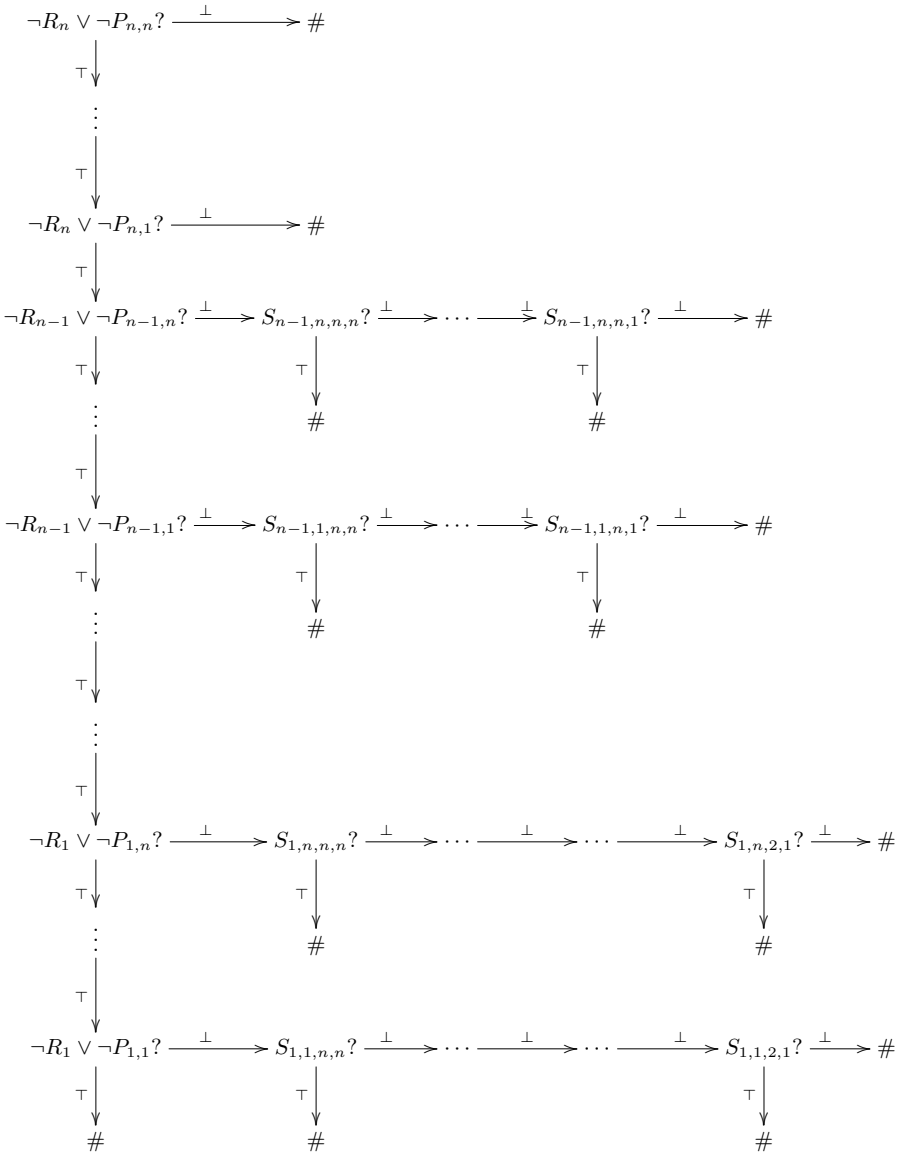


Fig. 2. Refutation of RVIP_n in $\mathrm{Res}^*(2)$

4.1 Lower Bound: A Strategy for Adversary over RLNP$_n$

We will give a strategy for Adversary in the game representation of a p-Res(1) refutation. The argument used in [7] does not adapt to the parameterized case, so we instead use a technique developed specifically for the parameterized Pigeonhole principle in [5].

Recall that a *parameterized clause* is of the form $\neg v_1 \vee \ldots \vee \neg v_{k+1}$ (where each v_i is some R, L or S variable). The i, j appearing in R_i, $L_{i,j}$ and $S_{i,j}$ are termed *co-ordinates*. We define the following *random restrictions*. Set $R_n := \top$. Randomly choose $i_0 \in [n-1]$ and set $R_{i_0} := \top$ and $L_{i_0,n} = S_{i_0,n} := \top$. Randomly choose $n - \sqrt{n}$ elements from $[n-1] \setminus i_0$, and call this set $\mathcal{C}$. Set $R_i := \bot$ for $i \in \mathcal{C}$. Pick a random bijection π on $\mathcal{C}$ and set $L_{i,j}$ and $S_{i,j}$, for $i, j \in \mathcal{C}$, according to whether $\pi(j) = i$. Set $L_{i,j} = L_{j,i} = S_{i,j} = S_{j,i} := \bot$, if $j \in \mathcal{C}$ and $i \in [n] \setminus (\mathcal{C} \cup \{i_0\})$.

What is the probability that a parameterized clause is **not** evaluated to true by the random assignment? We allow that each of $\neg R_n$, $\neg R_{i,0}$, $\neg L_{i_o,n}$ and $\neg S_{i_0,n}$ appear in the clause – leaving $k+1-4 = k-3$ literals, within which must appear $\sqrt{(k-3)/4}$ distinct co-ordinates. The probability that some $\neg R_i$, $i \notin \{i_0, n\}$, fails to be true is bound above by the probability that i is in $[n-1] \setminus (\mathcal{C} \cup \{i_0\})$ – which is $\leq \frac{\sqrt{n}-2}{n-2} \leq \frac{1}{\sqrt{n}}$. The probability that some $\neg L_{i,j}$ fails to be true, where one of the co-ordinates i, j is possibly mentioned before and $(i, j) \neq (i_0, n)$, is bound above by the probability that both i, j are in $[n] \setminus \mathcal{C}$ plus the probability that both i, j are in $\mathcal{C}$ and $i = \pi(j)$. This gives the bound $\leq \frac{\sqrt{n}}{n} \cdot \frac{\sqrt{n}-1}{n-1} + \frac{n-\sqrt{n}}{n} \cdot \frac{n-\sqrt{n}-1}{n-1} \cdot \frac{1}{n-\sqrt{n}-1} \leq \frac{2}{n} \leq \frac{1}{\sqrt{n}}$. Likewise with $\neg S_{i,j}$. Thus we get that the probability that a parameterized clause is **not** evaluated to true by the random assignment is $\leq \frac{1}{\sqrt{n}}^{\sqrt{(k-3)/4}} = n^{-\sqrt{(k-3)/16}}$.

Now we are ready to complete the proof. Suppose fewer than $n^{\sqrt{(k-3)/16}}$ parameterized clauses appear in a p-Res(1) refutation of RLNP$_n$, then there is a random restriction as per the previous paragraph that evaluates all of these clauses to true. What remains is a Res(1) refutation of RLNP$_{\sqrt{n}}$, which must be of size larger than $n^{\sqrt{(k-3)/16}}$ itself, for n sufficiently large (see [7]). Thus we have proved.

Theorem 1. *Every* p-Res(1) *refutation of* RLNP$_n$ *is of size* $\geq n^{\sqrt{(k-3)/16}}$.

5 Concluding Remarks

It is most natural when looking for separators of p-Res*(1) and p-Res*(2) to look for CNFs, like RVIP$_n$ that we have given. p-Res*(2) is naturally able to process 2-clauses and we may consider p-Res*(1) acting on 2-clauses, when we think of it using any of the clauses obtained from those 2-clauses by distributivity. In this manner, we offer the following principle as being fpt-bounded for p-Res*(2) but not fpt-bounded for p-Res*(1). Consider the two axioms $\forall x(\exists y \neg S(x, y) \wedge$

$T(x,y)) \vee P(x)$ and $\forall x, y T(x,y) \to S(x,y)$. This generates the following system Σ_{PST} of 2-clauses.

$$\begin{array}{ll} P_i \vee \bigvee_{j\in[n]}(\neg S_{i,j} \wedge T_{i,j}) & i \in [n] \\ \neg T_{i,j} \vee S_{i,j} & i,j \in [n] \end{array}$$

Note that the expansion of Σ_{PST} to CNF makes it exponentially larger. It is not hard to see that Σ_{PST} has refutations in p-Res$^*(2)$ of size $O(kn)$, while any refutation in p-Res$^*(1)$ will be of size $\geq n^{k/2}$.

All of our upper bounds, i.e. for both RVIP$_n$ and RLNP$_n$, are in fact polynomial, and do not depend on k. That is, they are fpt-bounded in a trivial sense. If we want examples that depend also on k then we may enforce this easily enough, as follows. For a set of clauses Σ, build a set of clauses Σ'_k with new propositional variables A and $B_1, B'_1, \ldots, B_{k+1}, B'_{k+1}$. From each clause $\mathcal{C} \in \Sigma$, generate the clause $A \vee \mathcal{C}$ in Σ'_k. Finally, augment Σ'_k with the following clauses: $\neg A \vee B_1 \vee B'_1$, $\ldots$, $\neg A \vee B_{k+1} \vee B'_{k+1}$. If Σ admits refutation of size $\Theta(n^c)$ in p-Res$^*(j)$ then (Σ'_k, k) admits refutation of size $\Theta(n^c + 2^{k+1})$. The parameterized contradictions so obtained are no longer "strong", but we could even enforce this by augmenting instead a Pigeonhole principle from $k+1$ to k.

It seems hard to prove p-Res(1) lower bounds for parameterized k-clique on a random graph [4], but we now introduce a contradiction that looks similar but for which lower bounds should be easier. It is a variant of the Pigeonhole principle which could give us another very natural separation of p-Res(1) from p-Res(2). Define the contradiction PHP$_{k+1,n,k}$, on variables $p_{i,j}$ ($i \in [k+1]$ and $j \in [n]$) and $q_{i,j}$ ($i \in [n]$ and $j \in [k]$), and with clauses:

$$\begin{array}{ll} \neg p_{i,j} \vee \neg p_{l,j} & i \neq l \in [k+1]; j \in [n] \\ \neg q_{i,j} \vee \neg q_{l,j} & i \neq l \in [n]; j \in [k] \\ \bigvee_{j\in[n]} p_{i,j} & i \in [k] \\ \neg p_{i,j} \vee \bigvee_{l\in[k]} q_{j,l} & j \in [n] \end{array}$$

We conjecture that this principle, which admits fpt-bounded refutation in p-Res(2), does not in p-Res(1).

Finally, we leave open the technical question as to whether suitably defined, further-relativised versions of RLNP$_n$ can separate p-Res(j) from p-Res($j+1$). We conjecture that they can.

Acknowledgements. We thank the reviewers as well as St. Evlogi.

References

1. Atserias, A., Bonet, M.: On the automatizability of resolution and related propositional proof systems. In: 16th Annual Conference of the European Association for Computer Science Logic (2002)
2. Atserias, A., Bonet, M.L., Esteban, J.L.: Lower bounds for the weak pigeonhole principle and random formulas beyond resolution. Inf. Comput. 176(2), 136–152 (2002)

3. Beyersdorff, O., Galesi, N., Lauria, M.: Hardness of parameterized resolution. Technical report, ECCC (2010)
4. Beyersdorff, O., Galesi, N., Lauria, M.: Parameterized complexity of DPLL search procedures. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 5–18. Springer, Heidelberg (2011)
5. Beyersdorff, O., Galesi, N., Lauria, M., Razborov, A.: Parameterized bounded-depth frege is not optimal. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 630–641. Springer, Heidelberg (2011)
6. Cook, S., Reckhow, R.: The relative efficiency of propositional proof systems. Journal of Symbolic Logic 44(1), 36–50 (1979)
7. Dantchev, S.: Relativisation provides natural separations for resolution-based proof systems. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 147–158. Springer, Heidelberg (2006)
8. Dantchev, S., Martin, B., Szeider, S.: Parameterized proof complexity. In: 48th IEEE Symp. on Foundations of Computer Science, pp. 150–160 (2007)
9. Dantchev, S., Martin, B., Szeider, S.: Parameterized proof complexity. Computational Complexity 20 (2011)
10. Downey, R.G., Fellows, M.R.: Parameterized Complexity. In: Monographs in Computer Science. Springer (1999)
11. Esteban, J.L., Galesi, N., Messner, J.: On the complexity of resolution with bounded conjunctions. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, p. 220. Springer, Heidelberg (2002)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science, vol. XIV. An EATCS Series. Springer (2006)
13. Krajíček, J.: On the weak pigeonhole principle. Fundamenta Mathematica, 170, 123–140 (2001)
14. Martin, B.: Parameterized proof complexity and W[1].
CoRR: arxiv.org/abs/1203.5323 (2012)
15. Pudlák, P.: Proofs as games. American Mathematical Monthly, 541–550 (June-July 2000)
16. Segerlind, N., Buss, S.R., Impagliazzo, R.: A switching lemma for small restrictions and lower bounds for k-dnf resolution. SIAM J. Comput. 33(5), 1171–1200 (2004)

Lower and Upper Bounds for the Length of Joins in the Lambek Calculus

Alexey Sorokin

Moscow State University, Faculty of Mechanics and Mathematics,
Moscow Institute of Physics and Technology

Abstract. In 1993 Mati Pentus proved a criterion of conjoinability for the Lambek calculus and multiplicative cyclic linear logic. In 2011 Alexey Sorokin showed that any pair of conjoinable types in the Lambek calculus has the join type of quadratic length with respect to the length of the types in the pair. We prove that the lower bound on the length of joins in the Lambek calculus and multiplicative linear logic is also quadratic.

1 Introduction

The Lambek calculus was introduced by Joachim Lambek in 1958 ([8]). In last the decades this calculus and its relatives, such as the multiplicative cyclic linear logic MCLL ([19]), the pregroup calculus ([7]), the Lambek-Grishin calculus([6]), have found various applications in modelling syntax and semantics of natural language. Also the Lambek calculus has a natural algebraic interpretation ([14]). For a survey of various aspects of the Lambek calculus see [10].

If a type A is derived from a sequence of types Γ in the Lambek calculus, this means that the sequence of grammar categories represented by the types of Γ can play the role of the category A in a sentence of the language. It makes some sense to consider the transitive symmetric closure of the derivability relation: the so-called conjoinability relation. Two types A and B are called conjoinable if there exists a type C such that both sequents $A \to C$ and $B \to C$ are derivable in the Lambek calculus. Intuitively, the categories are conjoinable if there is a context where the expressions of both categories can be used. A meet type for two categories is an analogue of their intersection (which cannot be expressed by means of the Lambek calculus itself).

A characterization of conjoinability in a particular calculus is deeply connected to the generative properties of the categorial grammars based on this calculus: the calculi with the conjoinability relation characterized by the non-abelian free group interpretation, like the Lambek calculus or the pregroup calculus, usually can be modelled by context-free grammars (see [13], [2]). The criteria of conjoinability for different variants of the Lambek calculus were studied in [12], [3] and [9].

A criterion of conjoinability for standard the Lambek calculus was established by Mati Pentus in [12] (the criterion appeared first in the preprint [11]). Although the proof is constructive, this work does not contain any algorithm to construct join types, which are proved to be conjoinable or any bound on the length of

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 150–161, 2013.

the join. Linguistic (sf. [4]) and theoretical ([17]) applications require such an algorithm or at least some bound on the length of the join type. In particular, in Safiullin's algorithm for determinating the Lambek grammars we need to estimate the size of join types to measure the size of the grammar obtained.

In the article [18] it was proved that the upper bound on the length of joins of the Lambek calculus is quadratic. That paper also provided a quadratic algorithm for constructing such a join of types. Since the construction in the paper is rather straightforward (though it contains some technicalities), it is interesting, whether a faster algorithm exists. In the present work we prove that the lower bound on the length of joins is also quadratic which implies that with respect to the multiplicative constant the algorithm from [18] was optimal.

The present paper is mainly of theoretical nature. We give a brief outline of the proof of the quadratic upper bound for the length of joins in the Lambek calculus. Then we prove a new result, which establishes a lower bound on the length of joins (the lower bound is also quadratic). We use the technique of proof nets to prove this result. The author does not know any results concerning lower bounds for the length of join and meet types for the Lambek calculus and related calculi.

2 Preliminaries

In this section we briefly introduce the Lambek calculus and the notion of conjoinability for this calculus. Let Pr be a countable set of primitive types. Then the set of types Tp is the smallest set containing Pr, such that for any types A and B from Tp the types $(A \cdot B), (A/B)$ and $(B\backslash A)$ also belong to Tp (we will omit external brackets). The sequents of the Lambek calculus are of the form $\Gamma \to A$, where A is a type and Γ is a finite sequence of types. We will use small Latin letters $p, q, r, \ldots$ (possibly with subscripts) for primitive types, capital Latin letters $A, B, C, \ldots$ for types and capital Greek letters $\Gamma, \Delta, \Pi, \ldots$ for finite sequences of types. The Lambek calculus has only one axiom scheme $A \to A$, $A \in \mathrm{Tp}$. The rules of the Lambek calculus are the following:

$$\frac{\Pi A \to B}{\Pi \to B/A}(\to /) \qquad \frac{\Gamma B \Delta \to C \quad \Pi \to A}{\Gamma(B/A)\Pi\Delta \to C}(/ \to)$$

$$\frac{A\Pi \to B}{\Pi \to A\backslash B}(\to \backslash) \qquad \frac{\Gamma B \Delta \to C \quad \Pi \to A}{\Gamma \Pi (A\backslash B)\Delta \to C}(/ \to)$$

$$\frac{\Gamma \to A \quad \Delta \to B}{\Gamma\Delta \to A \cdot B}(\to \cdot) \qquad \frac{\Gamma AB\Delta \to C}{\Gamma(A \cdot B)\Delta \to C}(\cdot \to)$$

The Lambek calculus admits cut-elimination, so we can add the cut rule $\dfrac{\Gamma B \Delta \to C \quad \Pi \to B}{\Gamma \Pi \Delta \to C}$ to the calculus. Note that the Lambek calculus is associative, so the types $A \cdot (B \cdot C)$ and $(A \cdot B) \cdot C$ are equivalent (which means each of them is derivable from the other), as well as the types $(A\backslash B)/C$ and $A\backslash(B/C)$. Associativity allows us to omit brackets in such types and write $A \cdot B \cdot C$ or $A\backslash B/C$.

Definition 1

1. *A type* $C \in \mathrm{Tp}$ *is called a join type for types* $A, B \in \mathrm{Tp}$ *if both sequents* $A \to C$ *and* $B \to C$ *are derivable in the Lambek calculus. In this case the types* A *and* B *are called conjoinable.*
2. *A type* $D \in \mathrm{Tp}$ *is called a meet type for types* $A, B \in \mathrm{Tp}$ *types* $A, B \in \mathrm{Tp}$ *if both sequents* $D \to A$ *and* $D \to B$ *are derivable in the Lambek calculus.*

Example 1. The types $(r/p)\backslash r$ and $p/(q/q)$ are conjoinable in the Lambek calculus for any $p, q, r \in \mathrm{Pr}$. They have a join type $(r/p)\backslash((r \cdot q)/q)/(q/q)$ and a meet type $(p/(q/q)) \cdot (q/q)$.

Lemma 1 (Lambek, 1958). *Types* A *and* B *have a join type iff they have a meet type.*

Proof. 1) $\to$ 2) Take $D = (A/C)\cdot C\cdot(C\backslash B)$. 2) $\to$ 1) Take $C = (D/A)\backslash D/(B\backslash D)$.

We need some auxiliary notions to formulate the conjoinability criterion from [12]. Let FG be the free group generated by the elements of Pr. For every type A we define its interpretation $[A]$ in FG. It is done by induction: $[p] = p$ for all $p \in \mathrm{Pr}$, $[A \cdot B] = [A] \odot [B]$, $[A/B] = [A] \odot [B]^{-1}$, $[B\backslash A] = [B]^{-1} \odot [A]$, where $\odot$ is the group operation. For example $[p_3\backslash((p/p)/(r/p_3))] = r^{-1}$.

Theorem 1 (Pentus, 1992). *Types* $A, B \in \mathrm{Tp}$ *are conjoinable iff* $[A]=[B]$.

The length of a type is the number of primitive types occurrences in it, we will denote the length of a type A by $|A|$. For example, $|(p_1/(p_2\backslash p_1)) \cdot p_2| = 4$. It is interesting to estimate the minimal length of a join type C for given conjoinable types A and B. We will show that in this case both lower and upper bounds are quadratic. Namely, for any two conjoinable types A and B there exists a join type of length not greater than $\frac{|A|^2+|B|^2}{2} + \frac{35}{2}(|A| + |B|)$, and for any positive integers k and l of the same parity there exist two conjoinable types A and B such that $|A| = k, |B| = l$ and there is no join for them of length less than $\frac{k^2+l^2}{8}$.

3 Upper Bound

In this section we briefly summarize the proof of the upper bound for the length of join type given in [18]. Let $\mathcal{P}$ be the set $\mathrm{Pr} \cup \{p^{-1} \mid p \in \mathrm{Pr}\}$. For every string $\alpha \in \mathcal{P}^*$ we define its inverse α^{-1} as a string that is obtained from α by reversing it and changing all the p to p^{-1} and vice versa. For example $(pq^{-1}r)^{-1} = r^{-1}qp^{-1}$. For every type $A \in \mathrm{Tp}$ we define its representing string $\langle A\rangle$ in the following way: $\langle p\rangle = p$, $p \in \mathrm{Pr}, \langle A/B\rangle = \langle A\rangle\langle B\rangle^{-1}, \langle A\backslash B\rangle = \langle B\rangle^{-1}\langle A\rangle, \langle A \cdot B\rangle = \langle A\rangle\langle B\rangle$. It it not difficult to see that $[A]$ is the normal form of $\langle A\rangle$ after reducing all the substrings of the form pp^{-1} and $p^{-1}p$ to the empty string. So if $\langle A\rangle = \langle B\rangle$, then $[A] = [B]$ and hence A and B are conjoinable.

We also define the "reverse" mapping ϕ from $\mathcal{P}^*$ to Tp. We set $\phi(\epsilon) = q/q$, where q is a new primitive type, $\phi(p) = p$, $p \in \mathrm{Pr}$ and $\phi(p^{-1}) = p\backslash p/p$.

We extend this mapping to the sequences, defining $\phi(\alpha\beta) = \phi(\alpha)\phi(\beta)$ for α and β from $\mathcal{P}^+$. Note that all the types $A, \phi(\langle A\rangle)$, and $\phi([A])$ are conjoinable. The next lemma follows immediately from the definitions:

Lemma 2. $|\phi([A])| \leq |\phi(\langle A\rangle)| \leq 3|A|$.

The construction of the conjoining type C for given conjoinable types A and B is illustrated in the scheme below. Using this scheme and the proof of Lemma 1 we can give the following estimation of the length of type C: $|C| \leq |A_3| + |B_3| + 3|\phi([A]) \leq |A_1| + |A_2| + 3|\phi(\langle A\rangle)| + |B_1| + |B_2| + 3|\phi(\langle B\rangle)| + 3|\phi([A])|$. The following lemma is proved in [18]. In what follows we just recall some principal moments from the proof. Note that the proofs of all lemmas are constructive, hence the proof also provides an algorithm for constructing the joins.

Lemma 3. *1)* $|A_1| \leq \frac{|A|^2}{2} + |A|$. *2)* $|A_2| \leq |\phi(\langle A\rangle)|$.

We will prove here just the first part since it is this statement that leads to the quadratic growth of the join type length. First we need to give some auxiliary definitions.

Definition 2. *The sets of positive and negative simplified types, denoted* STp^+ *and* STp^-*, respectively, are the smallest sets satisfying the following conditions:*

1. $\mathrm{Pr} \subset \mathrm{STp}^+$, $\mathrm{Pr} \subset \mathrm{STp}^-$,
2. $\forall A, B \in \mathrm{STp}^+ \ (A \cdot B) \in \mathrm{STp}^+$,
3. $\forall A \in \mathrm{STp}^+, B \in \mathrm{STp}^- \ (A/B), (B\backslash A) \in \mathrm{STp}^+$.
4. $\forall p \in \mathrm{Pr}, A, B \in \mathrm{STp}^+ \ (A\backslash p), (p/B), (A\backslash p/B) \in \mathrm{STp}^-$.

For example, the type $(p_1 \cdot p_2)/(p_3\backslash p_4)$ is a positive simplified type, but not a negative one. The next proposition allows one to consider only positive simplified types in the algorithm for constructing join types.

Lemma 4

1. *For every type* $A \in \mathrm{Tp}$ *there exists a positive simplified type* A' *such that the sequent* $A \to A'$ *is derivable,* $\langle A\rangle = \langle A'\rangle$ *and* $|A| = |A'|$.
2. *For every type* $B \in \mathrm{Tp}$ *there exist negative simplified types* $B_1, \ldots, B_r$ *such that the sequent* $B_{(1)} \ldots B_{(r)} \to B$ *is derivable,* $\langle B_{(1)}\rangle \ldots \langle B_{(r)}\rangle = \langle A\rangle$ *and* $\sum_{i=1}^{r} |B_{(i)}| = B$.

Proof. Both statements of the proposition are proved by mutual induction on the structure of the types A and B. For example, let us consider the case $A = C/D$ and prove the first statement. By the induction hypothesis, there exist a type C' and types $D_{(1)}, \ldots, D_{(r)}$ yielding the statement of the lemma for the types C and D respectively. Then we can take $A' = (\ldots(C'/D_{(r)})\ldots)/D_{(1)}$. It is not difficult to verify that this type satisfies the first statement of the lemma.

Let us now prove the second statement for the type $A = C/D$. There exist types $C_{(1)}, \ldots, C_{(s)}$ which yield the second statement of the lemma for the type C and the type D' which yield the statement for the type D. By definition $C_{(s)}$ has the form $E\backslash p/F$ for some positive simplified types E, F. Then we can take $A_{(i)} = C_{(i)}$ for $i < s$ and $A_{(s)} = E\backslash p/(D' \cdot F)$. The other cases are considered in the analogous way.

It is not difficult to see that the types A' and $B_{(1)}, \ldots, B_{(r)}$ from Lemma 4 can be constructed in quadratic time. Indeed, in the recursive procedure from the proof in each step we must find the main connective and then apply the procedure to the obtained subtypes. The division into subtypes can be easily done in linear time, so the whole algorithm is not harder than quadratic. Note also that all positive subtypes of a positive simplified type are positive, as well as all the negative subtypes are negative simplified types. Below we give a proof of the Lemma 3.

Proof. Lemma 4 allows us to prove the bound in the statement of the lemma only in the case when A is a positive simplified type. The proof proceeds by induction on the structure of type A. The basic case, when A is primitive is obvious, so we have to prove only the induction step. The case $A = B \cdot C$ is also trivial since we set $A_1 = B_1 \cdot C_1$. The only difficult case is when the main connective of A is a division connective; without loss of generality we assume that it is right division. So there are four remaining subcases: $A = B/p$, $A = B/(E\backslash p)$, $A = B/(p/D)$, and $A = B/(E\backslash p/D)$. In the first subcase we note that $\phi(\langle A\rangle) = \phi(\langle B\rangle) \cdot p\backslash p/p$ and take $A_1 = (p/B_1)\backslash p/p$. The bound of the lemma is verified by calculation.

Now we consider the main subcase $A = B/(E\backslash p/D)$. The other two subcases are similar. Note that $\phi(\langle A\rangle) = \phi(\langle B\rangle) \cdot \phi(\langle D\rangle) \cdot (p\backslash p/p) \cdot \phi(\langle E\rangle)$. So we set $A_1 = (p/(B_1 \cdot D_1))\backslash p/((E_1\backslash p/D) \cdot D)$. Then $|A_1| = |B_1| + |D_1| + |E_1| + 3 + 2|D| \leq \frac{1}{2}(|B|^2 + |D|^2 + |E|^2) + 2|D| + (|B| + |D| + |E|) + 3 \leq \frac{1}{2}(|B|^2 + |D|^2 + |E|^2 + 2(|B| + |E| + 1)|D|) + |A| + 2 \leq \frac{1}{2}(|B| + |D| + |E| + 1)^2 + |A| = \frac{1}{2}|A|^2 + |A|$. The lemma is proved. □

Let us show that the algorithm constructing the type A_1 in Lemma 3 requires only quadratic time. First assume that the type A is positive simplified. Let us consider the last step in constructing the type A_1 in the proof above. We should first find the main division in the type $A = B/(E\backslash p/D)$. Note that all the types B, E, D are positive simplified, so we do not need additional simplification in the recursion. The types B_1, D_1, E_1 can be recursively constructed "at place", so if we denote the time required to construct the type A_1 by $T(A)$, then we can write $T(A) \leq T(B) + T(D) + T(E) + c|A|$ for some constant c. Since $|A| = |B| + |D| + |E| + 1$ it follows that $T(A) \leq c'|A|^2$ for some constant c'.

If A is not simplified, then we first construct the type A' from Lemma 4 and then apply the algorithm constructing the type A' to it. The first part of this procedure is also quadratic, which yields the overall quadratic complexity. We have proved the following corollary:

Corollary 1. *The type A_1 from Lemma 3 can be constructed in quadratic time in the worst case.*

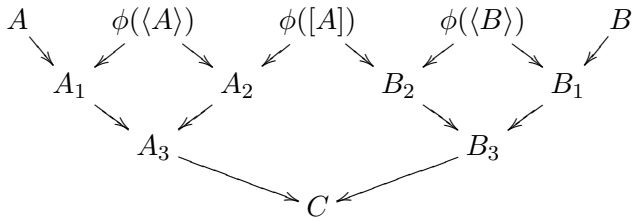


Fig. 1. The scheme of construction of the join type C for conjoinable types A and B

So $|C| \leq \frac{|A|^2}{2} + |A| + 4|\phi(\langle A\rangle)| + \frac{|B|^2}{2} + |B| + 4|\phi(\langle B\rangle)| + 3|\phi([A])|$. Taking into account Lemma 2 and the fact that $[A]$ is shorter than the type A itself (and also shorter than B), we conclude that $|C| \leq \frac{|A|^2+|B|^2}{2} + 12(|A| + |B|) + 9(\min(|A|, |B|)) \leq \frac{|A|^2+|B|^2}{2} + \frac{35}{2}(|A| + |B|)$. The following theorem holds:

Theorem 2. *If types A and B are conjoinable in the Lambek calculus, then they have a join type C of length not greater than $\frac{|A|^2+|B|^2}{2} + \frac{35}{2}(|A| + |B|)$.*

Let us consider the complexity of the algorithm for constructing the joins. Note that all the steps of the algorithm except of constructing types A_1, B_1, $\phi(\langle A\rangle), \phi(\langle B\rangle), \phi([A]), A_2, B_2$ are just the applications of the Rosser lemma. As proved in Lemma 4,constructing A_1, B_1 takes quadratic time in the worst case. If we consider the detailed proof from [18] it is not difficult to see that all other types can be constructed in linear time. The complexity of application of the Rosser lemma is the same as the length of the types in it, so the overall complexity is quadratic. The following corollary holds (we use the fact that the conjoinability criterion can obviously be verified in linear time):

Corollary 2. *There exists an algorithm that takes the types A and B as input and returns either a join type C or "NO" in the case the types are not conjoinable. The algorithms runs in quadratic time and space in the worst case.*

4 Lower Bound

In this section we prove a quadratic lower bound on the length of joins in the Lambek calculus. For this purpose we use the method of proof nets. Because proof nets of the Lambek calculus itself are not convenient for our purposes, we use the proof nets of linear logic. To make their usage possible we need to translate sequents of the Lambek calculus to a variant of multiplicative cyclic linear logic (MCLL, see [19]). The translation itself is an element of mathematical folklore, a proof can be found in the preprint [11].

Let Pr be the same set of primitive types as in the Lambek calculus (in the context of linear logic they are called variables). The set of literals Lit includes the variables and their negations (formally, $\mathrm{Lit} = \mathrm{Pr} \cup \{\overline{p} \mid p \in \mathrm{Pr}\}$). The formulae of linear logic are built from literals with the help of binary operations $\parr$ and $\otimes$. Formally, the set of formulae Fm is the smallest set containing Lit, such that for any elements A and B of Fm the elements $(A \parr B)$ and $(A \otimes B)$ also belong to Fm. The negation can be extended from variables to formulae by defining the external negation operation $(\cdot)^{\perp}$ according to De Morgan laws. We set $p^{\perp} = \overline{p}, \overline{p}^{\perp} = p$, $p \in \mathrm{Pr}$, $(A \parr B)^{\perp} = B^{\perp} \otimes A^{\perp}$, $(A \otimes B)^{\perp} = B^{\perp} \parr A^{\perp}$. The sequents of MCLL have the form $\to \Gamma$, where Γ is a non-empty finite sequence of formulae.

The calculus MCLL has the only axiom scheme $\to \overline{p}p$, $p \in \mathrm{Pr}$. It has the following rules (Λ denotes the empty sequence):

$$\frac{\to \Gamma A B \Delta}{\to \Gamma (A \parr B) \Delta}\ (\to \parr),\ \Gamma\Delta \neq \Lambda \qquad \frac{\to \Gamma A \quad \to B \Delta}{\to \Gamma (A \otimes B) \Delta}\ (\to \otimes)$$

$$\frac{\Gamma\Delta}{\Delta\Gamma}\ (rotate) \qquad \frac{\to \Gamma A \quad \to A^{\perp}\Delta}{\to \Gamma\Delta}\ (cut)$$

Note that the constraint in the rule $\parr \to$ can be reformulated: every derivable sequent must contain at least two formulas. Also note that MCLL admits cut-elimination, so any derivable sequent can be derived without using the cut rule. We can also add two-sided sequents to the calculus. The notation $A_1 \dots A_n \to B$ means $\to A_n^{\perp} \dots A_1^{\perp} B$ (the lefthand formulae are negated and their order is changed to the reverse).

Two formulae A and B from Fm are said to be conjoinable in MCLL if there exists a formula C, called a join formula, such that the sequents $A \to C$ and $B \to C$ are derivable in this calculus. The notion of a meet formula is introduced in a similar way. As well as in the Lambek calculus, the conjoinability relation in multiplicative cyclic linear logic possesses the Rosser property.

Now we want to formulate an algebraic criterion of conjoinability for MCLL. This criterion was proved in [12] for the calculus without "two-formulae"-constraint, but the proof holds also for the calculus with two-formulae constraint with only slight differences. For every formula $A \in \mathrm{Fm}$ we define its algebraic interpretation $[A]$ in the free group FG generated by the variables: $[p] = p, [\overline{p}] = p^{-1}, p \in \mathrm{Pr}$, $[A \parr B] = [A \otimes B] = [A] \odot [B]$. Also we define the function $d\colon \mathrm{Fm} \to \mathbb{Z}$ that equals the difference between the number of $\parr$-s and $\otimes$-s in the formula. For example, $d((p_1 \otimes (p_2 \parr p_3)) \otimes p_4) = -1$.

Theorem 3 (Pentus, 1993). *Formulae A and B are conjoinable in* MCLL *iff* $[A] = [B]$ *and* $d(A) = d(B)$.

We want to define an embedding $\psi\colon \mathrm{Tp} \to \mathrm{Fm}$ of types of the Lambek calculus to formulae of MCLL (this translation is standard and was used, for example, in [11] and [15]). The translation itself is very natural if we consider the division connectives of the Lambek calculus as directed implications. We set $\psi(p) = p, \psi(A \cdot B) = \psi(A) \otimes \psi(B)$, $\psi(A/B) = \psi(A) \parr \psi(B)^{\perp}$, $\psi(B \backslash A) = \psi(B)^{\perp} \parr \psi(A)$.

This mapping is extended to sequents by setting $\psi(\Gamma \to A) = \psi(\Gamma) \to \psi(A)$. The translation is conservative: the sequent $\psi(\Gamma) \to \psi(A)$ is derivable in the calculus MCLL (with the "two-formulae"-constraint) iff it is derivable in the Lambek calculus L. The variant of multiplicative linear logic without this constraint corresponds to the Lambek calculus L^* allowing empty antecedents.

Lemma 5. *Two types A and B are conjoinable in the Lambek calculus iff their images $\psi(A)$ and $\psi(B)$ are conjoinable in* MCLL.

From left to right this lemma follows from the conservativity of MCLL above L, the proof from right to left uses the criteria of conjoinability in these calculi. Note that the shortest join of A and B in the Lambek calculus cannot be shorter than the join of $\psi(A)$ and $\psi(B)$ in MCLL ($\psi(C)$ has the same length as C and it is a join of A and B due to conservativity). Therefore to prove that two types do not have short joins in the Lambek calculus it suffices to show that there are no short joins for their images in the multiplicative cyclic linear logic.

Now we are ready to define the notion of a proof net. Generally, a proof net is a presentation of syntactic derivations in the form that disregards "spurious ambigiuty". For example, if two derivations differ only in the order of rule applications, then they correspond to the same proof net. Usually proof nets are defined as multigraphs that satisfy certain conditions. The notion of a proof net originates in the fundamental work of Girard ([5]). Since that time many other versions of proof nets were proposed for the Lambek calculus and its fragments, as well as for various fragments of linear logic (see [10] for a survey and [15], [1], and [16] for the original formulations). For our purposes we find the variant of [15] the most useful.

The lemma below follows from the derivability criterion in [15] (we use only its soundness part). In fact, we have also omitted one condition in the definition of a proof net. We use only the planarity condition (first used in [1]) that corresponds to the noncommutativity of the calculus and the requirement on the balance of $\parr$-s and $\otimes$-s under the axiom links (perhaps it appeared for the first times in [15]).

Lemma 6. *If a sequent $\to A_1 \dots A_n$ is derivable in* MCLL *then we can divide the occurrences of literals in the sequent $\to A_1 \parr \dots \parr A_n$ in such a way that:*

1. *every pair includes the elements p and $\overline{p}$ for some variable p;*
2. *if we connect the elements in pairs by links in the upper half-plane, then the links do not intersect;*
3. *under each link the number of $\parr$-s equals the number of $\otimes$-s plus 1.*

We call the system of links from Lemma 6 the axiom links. Let α be an occurrence of literal in a formula A. Then $d_A(\alpha)$ denotes the difference between the number of $\parr$-s and $\otimes$-s to the left of α. For, example, if the formula is $((p_1 \parr p_2) \otimes \overline{p}_2 \parr p_4) \otimes (\overline{p}_1 \otimes p_3)$, and α_2 and α_3 are the only occurrences of p_2 and p_3 respectively, then $d_A(\alpha_2) = 1, d_A(\alpha_3) = -1$.

Let A' and B' be two types conjoinable in the Lambek calculus and A, B their translations to MCLL formulae. Let C be their join in the multiplicative cyclic

linear logic. Then the sequents $\rightarrow A^{\perp} \parr C$ and $\rightarrow B^{\perp} \parr C$ are derivable in MCLL. The following lemmas are the main technical tool of the work. From this now on $|A|_q$ denotes the number of occurrences of literal q (not of variable q) in formula A.

Lemma 7

1. *Let p be a variable such that $|A|_p = |B|_p = 1$ and $|A|_{\overline{p}} = |B|_{\overline{p}} = 0$. Let α and β be the occurrences of p in A and B respectively. Then $|B|_p + |B|_{\overline{p}} \geq |d_A(\alpha) - d_A(\beta)| + 1$.*
2. *Let p be a variable such that $|A|_p = |B|_p = 0$ and $|A|_{\overline{p}} = |B|_{\overline{p}} = 1$. Let α and β be the occurrences of $\overline{p}$ in A and B respectively. Then $|B|_p + |B|_{\overline{p}} \geq |d_A(\alpha) - d_A(\beta)| + 1$.*

Proof. It suffices to prove the first statement. Let α_1 be the occurrence of $\overline{p}$ in $A^{\perp}$ that corresponds to the only occurrence of p in A, and let β_1 be the occurrence of $\overline{p}$ corresponding to the occurrence of p in B. It is not difficult to see that $d_{A^{\perp}}(\alpha_1) = -(d(A) - d_A(\alpha))$ and $d_{B^{\perp}}(\beta_1) = -(d(B) - d_A(\beta))$. Since A and B are conjoinable, it holds that $d(B) = d(A)$ so $|d_{A^{\perp}}(\alpha_1) - d_{B^{\perp}}(\beta_1)| = |d_A(\alpha) - d_A(\beta)|$.

Let $d(\alpha_1) = D, d(\beta_1) = D + K$. Let $\mathcal{E}_1$ be the set of axiom links for the sequent $\rightarrow A^{\perp} C$ and $\mathcal{E}_2$ the set of axiom links for the sequent $\rightarrow B^{\perp} C$. We construct the following path in $(\mathcal{E}_1 \times \mathcal{E}_2)^+$: starting from α_1, we follow the link in $\mathcal{E}_1$ to some occurrence α' of $\overline{p}$ in C, then we follow the link in $\mathcal{E}_2$ from α' to some occurrence α'' of p, then we continue the path in $\mathcal{E}_1$, and so on. The trip must at last stop in some occurrence and it is not difficult to see that the final occurrence could be only β_1. Let the last link we traverse be $(\beta', \beta_1) \in \mathcal{E}_2$. Using Lemma 6 and the conjoinability criterion we conclude that $d_{A^{\perp} \parr C}(\beta') = d_{B^{\perp} \parr C}(\beta') = D + K + 1, d_{A^{\perp} \parr C}(\alpha') = d_{B^{\perp} \parr C}(\alpha') = D + 1$. For every link the values of d for its edges differ only by 1, so the path from α' to β' must contain at least $K - 1$ intermediate vertices. Since we never pass any vertex twice and there are no p-s and $\overline{p}$-s in A and B except for α_1 and β_1, there are at least $K + 1$ p-s or $\overline{p}$-s in C. The lemma is proved.

Lemma 8. *Let p be a variable such that $|A|_p = |A|_{\overline{p}} = 1$ and $|B|_p = |B|_{\overline{p}} = 0$. Let α and β be the occurrences of p and $\overline{p}$. Then if $|d(\alpha) - d(\beta)| \neq 1$ then $|C|_p + |C|_{\overline{p}} \geq |d(\alpha) - d(\beta)| + 1$.*

Proof. We use the same method of "links trip" as in the previous lemma. The condition $|d(\alpha) - d(\beta)| \neq 1$ forbids the corresponding occurrences in $A^{\perp}$ to be directly linked. We omit the detailed proof, because it is similar to that of Lemma 7.

We call a formula A simple, if for every variable p it holds that $|A|_p \leq 1$ and $|A|_{\overline{p}} \leq 1$. If $|A|_p = |A|_{\overline{p}} = 1$ then we call p a twin variable for A. If p is a twin variable for A we denote by α_p the occurrence of p and by β_p the occurrence of $\overline{p}$. We denote by $D(A)$ the following sum: $\sum_p (|d_A(\alpha_p) - d_A(\beta_p)| + 1)$, where p ranges over all twin variables for A, such that $|d_A(\alpha_p) - d_A(\beta_p)| \neq 1$. We call

conjoinable formulae A and B a simple conjoinable pair if both A and B are simple formulae that do not have common twin variables. We call a literal q a single literal for the pair (A, B) if $|A|_q = |B|_q = 1$ and $|A|_{\overline{q}} = |B|_{\overline{q}} = 0$. If q is a single literal we denote $\gamma_A(q)$ and $\gamma_B(q)$ its occurrences in A and B respectively. We denote by $E(A, B)$ the sum $\sum_q(|d_A(\gamma_A(q)) - d_B(\gamma_B(q))| + 1)$ where q ranges over all single literals. The next lemma is a consequence of lemmas 7 and 8.

Lemma 9. *Let (A, B) be a simple conjoinable pair. Then for every formula C that is their join, it holds that $|C| \geq D(A) + D(B) + E(A, B)$.*

The following lemma describes the construction of the family of the Lambek calculus types $\{T_k, U_k \mid k \in \mathbb{N}\}$ with large $D(\psi(T_k))$ and $D(\psi(U_k))$ and plays the basic role in establishing the main result of the paper (recall that ψ is a translation from the Lambek calculus types to linear logic formulae).

Lemma 10. *There exists a family of types of the Lambek calculus $\{T_k, U_k \mid k \in \mathbb{N}\}$ with the following properties:*

1. *For every k and l of the same parity T_k and U_l are conjoinable.*
2. *For every k the formulae $\psi(T_k)$ and $\psi(U_k)$ are simple.*
3. *$[T_k] = [\psi(T_k)] = [U_k] = [\psi(U_k)] = \epsilon$ if k is even. $[T_k] = [\psi(T_k)] = [U_k] = [\psi(U_k)] = p_0$ if k is odd.*
4. *For all k, l the formulae $\psi(T_k)$ and $\psi(U_l)$ do not have common twin variables.*
5. *For every even k it holds that $D(T_k) \geq \frac{k^2}{8} + \frac{k}{2} - 4$, $D(U_k) \geq \frac{k^2}{8} + \frac{k}{2} - 4$. For every odd k it holds that $D(T_k) \geq \frac{k^2}{8} + \frac{k}{4} - \frac{9}{2}$, $D(U_k) \geq \frac{k^2}{8} + \frac{k}{4} - \frac{9}{2}$. For every odd k and l it also holds that $E(T_k, U_l) = 1$, otherwise $E(T_k, U_l) = 0$.*

Proof. We describe the construction of the types T_k and U_k and then show that the lemma trivially follows from the construction. We suppose that the set Pr contains types p_i, r_i, s_i, t_i for every natural i and all such types are different. Let $V_m = (q_m \cdot \ldots \cdot q_1)/(p_m \cdot \ldots \cdot p_1)$, $W_1 = q_1/p_1$, $W_{i+1} = q_{i+1}/(p_{i+1}/W_1), i \in \mathbb{N}$. First we consider the case $k = 4m$ for some natural m and then obtain the remaining cases by slight modifications.

In the case $k = 4m$ we define $T_k = V_m \backslash W_m$, then U_k is constructed from T_k by substituting all p_i-s for r_i-s and all the q_i-s for s_i-s. Let us prove the lemma in this case. The third statement can be directly verified and the first statement follows from it by the conjoinability criterion. The second and the fourth statement are obvious. So let us prove the fifth statement.

It is not difficult to prove that after omitting all the brackets $\psi(T_k)$ has the form $p_m \otimes \ldots \otimes p_1 \otimes \overline{q}_1 \parr \ldots \parr \overline{q}_m \parr q_m \parr \ldots \parr q_1 \parr \overline{p}_1 \otimes \ldots \otimes \overline{p}_m$. So if we denote by α_i and $\overline{\alpha}_i$ the occurrences of q_i and $\overline{q}_i$ then $|d_{\psi(T_k)}(\alpha_i) - d_{\psi(T_k)}(\overline{\alpha}_i)| = 2(m-i)-1$ for every natural i. The occurrences of q_i and $\overline{q}_i$ have the same property, so a simple calculation shows that $D(\psi(T_k)) = (4+6+\ldots+2m)\cdot 2 = 2m(m+1)-4 = \frac{k^2}{8} + \frac{k}{2} - 4$. The lemma is proved in the case $k = 4m$.

In the case $k = 4m + 2$ we take as T_k the type T_{k-2} where the primitive type p_1 is substituted with the type p_1/p_{m+1} (if k = 2 then $T_k = p_1/p_1$). It is easy to see that in this case $\psi(T_k)$ after omitting the brackets equals $p_m \otimes$

$\ldots \otimes p_1 \parr \overline{p}_{m+1} \otimes \overline{q}_1 \parr \ldots \parr \overline{q}_m \parr q_m \parr \ldots \parr q_1 \parr p_{m+1} \otimes \overline{p}_1 \otimes \ldots \otimes \overline{p}_m$. Then $D(T_k) = 4 + \ldots + 2m + 2m + 2 + 2m + \ldots + 4 = 2m^2 + 4m - 2 = \frac{k^2}{8} + \frac{k}{2} - \frac{7}{2}$. The type U_k is obtained by the same replacement as in the previous case. Then the lemma also holds.

In the case $k = 4m + 1$ and $k = 4m + 3$ we take $T_k = p_0 \cdot T_{k-1}$. So $D(T_k) \geq \frac{(k-1)^2}{8} + \frac{k-1}{2} - 4 > \frac{k^2}{8} + \frac{k}{4} - \frac{9}{2}$. U_k is obtained by the same procedure as before. Obviously $E(T_k, U_l) = 1$ for all odd k, l. The lemma is proved in all cases.

Below we formulate the main result. Note that the gap between the lower and the upper bounds is fourfold. The author conjectures that in the case of MCLL it is possible to achieve the same factor in leading terms of the lower and upper bounds.

Theorem 4. *For every natural k, l of the same parity there are types T_k, U_l such that $|T_k| = k, |U_l| = l$, the types T_k and U_l are conjoinable and every join of this types has length at least $\frac{k^2+l^2}{8} + \frac{k+l}{4} - 8$.*

Proof. We take the types T_k and U_l from the previous lemma. Then by Lemma 9 for every join C of the formulae $\psi(T_k)$ and $\psi(U_l)$ this join has length at least $D(T_k)+D(U_l)+E(T_k+U_l)$. By Lemma 10 we conclude that $|C| \geq \frac{k^2+l^2}{8} + \frac{k+l}{4} - 8$. The theorem is proved.

A quadratic lower bound on the length of joins implies the same lower bound on the complexity of an algorithm that constructs the join. So the following corollary holds:

Corollary 3. *There is no algorithm that correctly constructs the join type for types A and B given and works less than quadratic time in the worst case.*

The author wants to thank Mati Pentus for helpful discussions during the work. The author also thanks Simone Martini and anonymous referees for thoughtful comments, which helped to improve the article.

5 Conclusion

We have proved that for the Lambek calculus both the upper and lower bounds on the length of a join type are quadratic with respect to the lengths of conjoinable types. In the case of the lower bound we used the method of proof nets, which has never been used before for such purpose. The proof of the upper bound explicitly contains an algorithm constructing a short join and gives a constructive proof of the conjoinability criterion from [12]. This method also allows us to obtain the conjoinability criteria for the calculi where other techniques are difficult to apply (such as the discontinuous Lambek calculus). No bound on the possible length of joins is known in the case of the calculi where the conjoinability criterion uses an interpretation in abelian groups, such as the Lambek-Grishin calculus or the discontinuous Lambek calculus. So it is interesting to study, whether the length of joins (and hence the complexity of algorithms that construct such joins) depends on algebraic characteristics of conjoinability.

References

1. Abrusci, V.M.: Phase semantics and sequent calculus for pure non-commutative classical linear logic. Journal of Symbolic Logic 56(4), 1403–1451 (1991)
2. Béchet, D.: Parsing pregroup grammars and Lambek calculus using partial composition. Studia Logica 87(2), 199–224 (2007)
3. Foret, A.: Conjoinability and unification in Lambek categorial grammars. In: Proceedings of the 5th ROMA Workshop on New Perspectives in Logic and Formal Linguistics, Bulzoni, Roma (2001)
4. Foret, A.: On the computation of joins for non-associative Lambek categorial grammars. In: Proceedings of the 17th International Workshop on Unification, UNIF, vol. 3, pp. 25–38 (2003)
5. Girard, J.-Y.: Linear logic. Theoretical Computer Science 1, 1–101 (1987)
6. Grishin, V.: On a generalization of the Ajdukewicz-Lambek system. In: Studies on Nonclassical Logics and Formal Systems, Nauka, Moscow (1983) (in Russian); English translation in: Proceedings of the 5th ROMA Workshop on New Perspectives in Logic and Formal, Linguistics. Bulzoni Editore, Roma (2001)
7. Lambek, J.: Type grammar revisited. In: Lecomte, A., Perrier, G., Lamarche, F. (eds.) LACL 1997. LNCS (LNAI), vol. 1582, pp. 1–27. Springer, Heidelberg (1999)
8. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
9. Moortgat, M., Pentus, M.: Type similarity for the Lambek-Grishin calculus. In: Proceedings of the 12th Conference on Formal Grammar, Dublin (2007)
10. Moot, R., Retoré, C. (eds.): The Logic of Categorial Grammars. LNCS, vol. 6850. Springer, Heidelberg (2012)
11. Pentus, M.: Equivalent types in Lambek calculus and linear logic. Preprint. Department of mathematical logic, Steklov Mathematical Institute, Moscow 2 (1992)
12. Pentus, M.: The conjoinability relation in Lambek calculus and linear logic. ILLC Prepublication Series ML-93-03. Institute for Logic, Language and Computation, University of Amsterdam (1993)
13. Pentus, M.: Lambek grammars are context-free. In: Proceedings of the Logic in Computer Science, LICS 1993, pp. 429–433 (1993)
14. Pentus, M.: Models of the Lambek calculus. Annals of Pure and Applied Logic 75(1), 179–213 (1995)
15. Pentus, M.: Free monoid completeness of the Lambek calculus allowing empty premises. In: Larrazabal, J.M., Lascar, D., Mints, G. (eds.) Proceedings of Logic Colloquium 1996, pp. 171–209. Springer, Berlin (1998)
16. Roorda, D.: Resource logic: proof theoretical investigations. PhD thesis, FWI, Universiteit van Amsterdam (1991)
17. Safiullin, A.: Derivability of admissible rules with simple premises in the Lambek calculus. Moscow University Mathematics Bulletin 62(4), 168–171 (2007)
18. Sorokin, A.: On the length of joins in Lambek calculus. Moscow University Mathematics Bulletin 66(3), 101–104 (2011)
19. Yetter, D.: Quantales and (noncommutative) linear logic. Journal of Symbolic Logic 55(1), 41–64 (1990)

Graph Expansion, Tseitin Formulas and Resolution Proofs for CSP

Dmitry Itsykson[1,⋆] and Vsevolod Oparin[2,⋆⋆]

[1] Steklov Institute of Mathematics at St.Petersburg
dmitrits@pdmi.ras.ru
[2] St.Petersburg Academic University of Russian Academy of Sciences
oparin.vsevolod@gmail.com

Abstract. We study the resolution complexity of Tseitin formulas over arbitrary rings in terms of combinatorial properties of graphs. We give some evidence that an expansion of a graph is a good characterization of the resolution complexity of Tseitin formulas. We extend the method of Ben-Sasson and Wigderson of proving lower bounds for the size of resolution proofs to constraint satisfaction problems under an arbitrary finite alphabet. For Tseitin formulas under the alphabet of cardinality d we provide a lower bound $d^{e(G)-k}$ for tree-like resolution complexity that is stronger than the one that can be obtained by the Ben-Sasson and Wigderson method. Here k is an upper bound on the degree of the graph and $e(G)$ is the graph expansion that is equal to the minimal cut such that none of its parts is more than twice bigger than the other. We give a formal argument why a large graph expansion is necessary for lower bounds. Let $G = \langle V, E\rangle$ be the dependency graph of the CSP, vertices of G correspond to constraints; two constraints are connected by an edge for every common variable. We prove that the tree-like resolution complexity of the CSP is at most $d^{e(H)\cdot\log_{\frac{3}{2}}|V|}$ for some subgraph H of G.

1 Introduction

Backtracking algorithms are the most popular approach to solving NP-hard problems. The execution of a backtracking algorithm for SAT on unsatisfiable formulas is closely connected to the tree-like resolution proof system. Lower bounds on the complexity of resolution proofs imply the same lower bounds on the running time of backtracking algorithms. The first superpolynomial lower bound for resolutions was proved by Tseitin [13]; Tseitin used formulas that coded the following simple fact: in every graph the number of vertices with odd

⋆ Partially supported by the grants MK-4108.2012.1 from the President of RF, by RFBR grant 12-01-31239-mol-a, by the Programme of Fundamental Research of RAS and by The Ministry of Education and Science of Russian Federation, project 8216.

⋆⋆ Partially supported by RFBR grant 12-01-31239-mol-a, by Réseau STIC franco-russe and ANR NAFIT 008-01.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 162–173, 2013.

degree is even. The first exponential lower bound was proved by Urqhart [14]. The strongest known lower bounds were proved using the methods introduced by Ben-Sasson and Wigderson in [4]. From the practical point of view it is more interesting to have a lower bound for backtracking algorithms on satisfiable formulas; there are several lower bounds on satisfiable formulas [1], [5], [9], [8] under various restrictions on heuristics that choose a variable for splitting and a value that is investigated first. However, all the known lower bounds on satisfiable formulas are proved by reduction to lower bounds on unsatisfiable ones.

Baker [2] introduced a very natural extension of the resolution proof system for constraint satisfaction problems (CSPs) and defined the system NG-RES. Baker studied different backtracking algorithms for CSP; he introduced the notion of the width of a CSP and proved that there exist resolution proofs of size exponential only in the width and polynomial in the other parameters. Baker also gave a hard distribution for backtracking algorithms for CSP and proved a super polynomial lower bound for NG-RES. Mitchell [11] introduced the proof system C-RES that is more powerful than NG-RES and proved an exponential lower bound for the random CSP in C-RES. Mitchell [12] proved a superpolynomial separation between C-RES and NG-RES and Hwang [6] proved an exponential separation.

The paper [7] proves that a linear lower bound on the proof degree in Polynomial Calculus implies an exponential lower bound on the proof size in Polynomial Calculus under fields. The paper [3] presents a linear lower bound on the degree of proofs of Tseitin formulas in Polynomial Calculus under fields and rings. This lower bounds are proved only for alphabets of cardinality p^m for prime p; and this result is not claimed to be optimal.

In this paper we are interested in the precise complexity of backtracking algorithms (or tree-like resolution) on Tseitin formulas under an arbitrary finite alphabet. In the propositional case the strongest lower bound for Tseitin formulas follows from the paper of Ben-Sasson and Wigderson. Namely every tree-like resolution proof of the Tseitin formula based on a graph with maximal degree at most k has the size at least $2^{e(G)-k}$, where $e(G)$ is the expansion of the graph. Recall that the expansion of a graph equals the size of minimal cut such that such that none of its parts is more than twice bigger than the other. Method of Ben-Sasson and Wigderson consists of the following two steps: first, establishing a relationship between the proof size and the width of a proof; second, establishing a relationship between the width of a proof and the expansion of the CSP. Mitchell in [10] generalizes the relationship between the size and width of a proof to a nonbinary case. The trivial case of such a generalization to the alphabet of size d implies the lower bound $2^{e_d(G)-k}$ for the tree-like resolution complexity of Tseitin formulas, where e_d is the size of the minimal cut such that none of its parts is more than d times bigger than the other. Generally speaking $e_d(G)$ can be much smaller than $e(G) = e_2(G)$.

For an arbitrary CSP ϕ the results of [10] implies the following generalization of [4]:

1. $S_T(\phi) \geq 2^{e(\phi)-k-1}$,
2. $S(\phi) \geq \exp\left(\Omega\left(\frac{(e(\phi)-k-1)^2}{n}\right)\right)$,

where $S_T(\phi)$ and $S(\phi)$ are the tree-like and general resolution complexity of ϕ, respectively, and $e(\phi)$ is the expansion of CSP ϕ.

The latter inequality implies the lower bound $2^{e(G)-k-1}$ on the tree-like resolution complexity of Tseitin formulas. By means of a more specific analysis for Tseitin formulas we improve this lower bound to $d^{(e(G)-k)}$.

It is well known that proofs of lower bounds for Tseitin formulas use the graph with high expansion. We study the question whether high expansion is indeed necessary for lower bounds. We give an answer for arbitrary CSP: let $G = \langle V, E\rangle$ be the dependency graph of a CSP; the vertices of G correspond to constraints;and two constraints are connected by an edge for every common variable (thus, the dependency graph is actually a multigraph). We prove that the tree-like resolution complexity of the CSP is at most $d^{e(H)\cdot\log_{\frac{3}{2}}|V|}$ for some subgraph H of G. Thus for the Tseitin formula ϕ based on the graph $G = \langle V, E\rangle$ we have that there is a subgraph H of G such that $d^{e(H)-k} \leq S_T(\phi) \leq d^{e(H)\cdot\log_{\frac{3}{2}}|V|}$.

In Section 2 we give definitions of basic concepts. In Section 3 we explain the relationship between width of the proof and the expansion of the CSP. In Section 4 we prove a stronger lower bound for the tree-like resolution complexity of Tseitin formulas. In Section 5 we prove an upper bound on the tree-like resolution complexity of the CSP in terms of the expansion of the dependency graph.

2 Preliminaries

2.1 The Constraint Satisfaction Problem (CSP)

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of variables that take values from a finite set D, and S be a set of constraints; every constraint defines a subset of variables X' and a set of possible values that variables of X' can take at the same time. A triplet $\langle X, D, S\rangle$ is called a constraint satisfaction problem (CSP). If every constraint restricts at most k variables than we call such problem k-CSP.

A partial substitution is a mapping $\rho : X \to D \cup \{*\}$, where '$*$' means an unspecified value; a support of a substitution is the set $\rho^{-1}(D)$. A substitution is complete if its support equals to X.

A partial substitution ρ satisfies a constraint $R \in S$ if after substituting the variables from the support of ρ with their respective values constraint R is satisfied regardless the values of the other variables. A substitution ρ satisfies CSP $\langle X, D, S\rangle$ if ρ satisfies every constraint $R \in S$. CSP ϕ is satisfiable if there exists at least one substitution that satisfies ϕ.

We call a constraint of the type $\neg(x_1 = a_1 \wedge \cdots \wedge x_k = a_k)$, where $x_1, \ldots, x_k \in X, a_1, \ldots, a_k \in D$, a *nogood*. The notion of a nogood is an extension of the notion of a clause in the propositional case ($D = \{0, 1\}$). For example the nogood $\neg(x_1 = 0 \wedge x_2 = 1)$ is equivalent to the clause $(x_1 \vee \bar{x}_2)$.

In what follows we consider only k-CSP and denote $|D|$ by d. Every restriction in k-CSP may be written as a conjunction of at most d^k nogoods.

2.2 Backtracking Algorithms

Now we define backtracking algorithms for CSP.

A backtracking algorithm is parametrized by two heuristics B and C and a simplification procedure S. A heuristic B takes a CSP ϕ and returns a variable x for splitting. A heuristic C takes a pair (ϕ, x) and returns an order on D (this is an order in which an algorithm substitutes values from D for the variable x).

The simplification procedure $S(\phi, x := a)$ removes from the $\phi[x := a]$ all constraints that have been already satisfied.

A backtracking algorithm $A(\phi)$ is defined as follows

- If ϕ does not contain constraints, return SATISFIABLE.
- If ϕ contains already falsified constraint, return UNSATISFIABLE.
- Pick a variable $x := B(\phi)$. According to the order given by $C(\phi, x)$, for all $a \in D$ make a recursive call $A(S(\phi, x := a))$. If one of recursive call returns SATISFIABLE, immediately return SATISFIABLE, otherwise return UNSATISFIABLE.

The running time of the backtracking algorithm is the size of the recursion tree. We ignore the computational complexity of heuristics B and C.

2.3 Resolution Proof System

We consider only unsatisfiable instances of CSP.

We define a resolution proof system that generalizes the well known system in the propositional case. This definition is due to [2].

The resolution proof system is a way to show that a given CSP is unsatisfiable. We assume that constraints are represented as a set of nogoods.

Let $\{N_a\}_{a \in D}$ be a set of nogoods such that $N_a = \neg(x = a \wedge \alpha_a)$ for every $a \in D$. A nogood $\neg(\bigwedge_{a \in D} \alpha_a)$ is a resolvent of $\{N_a\}_{a \in D}$.

Definition 1. *A sequence of nogoods $\pi = \{N_i\}$ is a resolution proof for CSP ϕ if*

- *every nogood N_i is either a nogood of ϕ or a resolvent of d nogoods that precede N_i: $N_{j_1}, \ldots, N_{j_d}$, where $j_1, \ldots, j_d < i$;*
- *the last nogood in π is the empty nogood $\neg()$ (i.e. contradiction).*

Every resolution proof may be represented as a directed acyclic graph with nogoods as vertices. There is an arc between N_i and N_j if N_i is in the premise of the resolution rule that produced N_j. The proof is called tree-like if its graph is a tree. A tree-like resolution proof system accepts only tree-like proofs.

Similarly to the propositional case, the running of backtracking algorithms on unsatisfiable CSPs and tree-like resolution proofs are equivalent.

Proposition 1 ([2,6]). *The size of the smallest tree-like resolution refutation is exactly the same as the size of the minimal recursion tree of a backtracking algorithm.*

Thus upper and lower bounds on the size of tree-like resolution proofs provide the same upper and lower bounds on the running time of backtracking algorithms.

2.4 Tseitin Formulas and Expansion

The paper [3] generalizes Tseitin formulas [13] to the nonbinary case. Consider a graph $G = \langle V, E\rangle$ and a function $f : V \to \mathbb{Z}_d$. We associate every edge $e \in E$ with a variable x_e. For every vertex u we have a constraint of type

$$\sum_{(u,v)} \gamma_{(u,v)} \cdot x_{(u,v)\in E} = f(u) \bmod d$$

where $\gamma_{(u,v)} \in \{+1, -1\}$. Every edge (u, v) corresponds to a variable $x_{(u,v)}$ and two values $\gamma_{(u,v)}$ and $\gamma_{(v,u)}$ that satisfy $\gamma_{(u,v)} + \gamma_{(v,u)} = 0$. Note that $x_{(u,v)}$ and $x_{(v,u)}$ denote the same variable.

The following lemma is very similar to the propositional case.

Lemma 1. *Tseitin formula $\phi(G, f)$ based on a connected graph G is satisfiable if and only if $\sum_v f(v) = 0$.*

Definition 2. *The expansion of a graph $G = \langle V, E\rangle$ is $e(G) = \min_{A\subseteq V,\, \frac{1}{3}|V|\le|A|\le\frac{2}{3}|V|} |E(A, \bar{A})|$.*

Further we will see a relationship between the expansion of a graph and the sizes of resolution proofs of Tseitin formulas.

3 Resolution Width and Expansion

The paper [4] introduced a technique of proving lower bounds in the propositional resolution proof system, that are quite strong. We generalize that result to CSP.

Let us consider a k-CSP $\phi = \langle X, D, S\rangle$ that is represented by a set of nogoods.

A width of a nogood is the number of variables that appear in it. If π is a resolution proof of ϕ, then a width of π is the maximal width of a nogood in π; we denote it by $W(\pi)$. A width of refutation of CSP ϕ is the minimal width of all resolution proofs of ϕ; we denote it by $W(\phi \vdash 0)$.

Theorem 1 ([10]). *For every k-CSP ϕ the following inequalities are satisfied*

$$S_T(\phi) \ge 2^{W(\phi\vdash 0)-k},$$

$$S(\phi) \ge \exp\left(\Omega\left(\frac{(W(\phi \vdash 0) - k)^2}{n}\right)\right),$$

where $S_T(\phi)$ is the minimal size of a tree-like resolution proof of ϕ and S_ϕ is the minimal size of a resolution proof of ϕ.

Let us consider CSP ϕ; let S be the set of constraints of ϕ (the constraints are no required to be nogoods). Let F be some subset of the set S; we denote by ∂F the set of variables x such that there is exactly one constraint in F that depends on x. The expansion of ϕ is defined as follows

$$e(\phi) = \min_F |\partial F|,$$

where the minimum is over all $F \subseteq S$ such that $\frac{1}{3}|S| \leq |F| \leq \frac{2}{3}|S|$.

Definition 3. *Let ϕ be an unsatisfiable CSP. We say that ϕ is* minimally unsatisfiable *if ϕ becomes satisfiable after removing any of its constraints.*

Theorem 2. *Let ϕ be a minimally unsatisfiable CSP and S be a constraint set of ϕ. Let ϕ satisfy the following property:*

- *for every constraint $f \in S$ every two substitutions that violate f differ in at least two variables.*

Then $W(\phi \vdash 0) \geq e(\phi) - 1$.

Proof. We say that a nogood N is semantically implied by $F \subseteq S$, if every substitution that satisfies F also satisfies N. We denote this implication by $F \models N$. We define Ben-Sasson-Wigderson measure on the set of all nogoods. For a nogood N we define $\mu(N) = \min\{|F| \mid F \subseteq S, F \models N\}$. The following properties are straightforward:

- $\mu(N) \leq 1$ for every nogood N from ϕ;
- $\mu(\neg()) = |S|$;
- If N is the resolvent of $\{N_a\}_{a \in D}$, then $\mu(N) \leq \sum_{a \in D} \mu(N_a)$.

Lemma 2. *Let F be a minimal set of constraints that semantically implies N. Then the size of N is at least $|\partial F|$.*

Proof. Note that for every constraint $f \in F$ there is an assignment ρ_f that falsifies N and f, but ρ_f satisfies every other constraint $g \in F$. Otherwise we can remove such constraint from F and this contradicts to the minimality of F.

For $x \in \partial F$ let $f \in F$ be the constraint that depends on x. Then there exists $a \in D$ such that the assignment obtained by changing the value of variable x in ρ_f to a satisfies f and therefore satisfies N. Thus N depends on x. □

In the propositional case this would finish the proof since the properties of a measure μ imply that every resolution proof contains a nogood N with a measure in $[\frac{1}{3}|S|, \frac{2}{3}|S|]$. Lemma 2 implies that N contains at least $e(\phi)$ variables. However for arbitrary d we can't guarantee that such nogood N exists. We take another way.

Any resolution proof of the formula ϕ contains a nogood N that is the resolvent of nogoods N_a on a variable x, $a \in D$, $\mu(N) > \frac{1}{3}|S|$, and for every premise N_a the inequality $\mu(N_a) \leq \frac{1}{3}|S|$ holds.

Let F_a be a minimal subset of constraints such that $F_a \models N_a$. Since $|F_a| \leq \frac{1}{3} \cdot |S|$, we can choose $D' \subseteq D$ in such a way that for F' defined as $\bigcup_{a \in D'} F_a$ we have $\frac{1}{3} \cdot |S| \leq |F'| \leq \frac{2}{3} \cdot |S|$. Thus $|\partial F'| \geq e(\phi)$, and by Lemma 2 for every variable $y \in \partial F'$ there exists a nogood N_a ($a \in D'$) that depends on y. Therefore $(\partial F' \backslash \{x\}) \subseteq \mathrm{Vars}(N)$, hence $|\mathrm{Vars}(N)| \geq e(\phi) - 1$, where $\mathrm{Vars}(N)$ is the set of variables from the nogood N. □

Corollary 1. *If a Tseitin formula $\phi(G, f)$ is unsatisfiable, then $W(\phi(G, f) \vdash 0) \geq e(G) - 1$.*

Proof. Follows from Theorem 2 and Lemma 1. □

Finally if the degree of all vertices in a graph G is at most k and Tseitin formula $\phi(G, f)$ is unsatisfiable, then Corollary 1 and Theorem 1 imply the following lower bounds:

1. $S_T(\phi) \geq 2^{e(G)-k-1}$,
2. $S(\phi) \geq \exp\left(\Omega\left(\frac{(e(G)-k-1)^2}{n}\right)\right)$.

Note that we have 2 in the base of the exponent in the tree-like case as it was for a binary alphabet. But it is more natural to have number d in the base of the exponent since every node of the tree has d children. In the next section we give more accurate analysis for Tseitin formulas and prove a lower bound $d^{e(G)-k}$ for tree-like resolution.

4 Lower Bound for Tseitin Formulas

In this section we prove a lower bound for the size of tree-like resolution proofs of Tseitin formulas that is stronger than the lower bound from the previous section. Consider a graph $G = \langle V, E \rangle$ and the unsatisfiable Tseitin formula ϕ based on it. Let the maximal degree of G be at most k. We assume that the domain D equals $\mathbb{Z}_d$. We prove that $S_T(\phi) \geq d^{e(G)-k}$, where S_T is the size of a minimal tree-like resolution proof of ϕ.

4.1 Reduced Splitting Tree

Let $G = \langle V, E \rangle$ be a connected graph with the maximal degree of vertices at most k. We consider a protocol of a backtracking algorithm and define the notion of the complexity of graph G. It equals the minimal size of resolution proofs of $\phi(G, f)$. For a connected graph G we define

$$C(G) = \begin{cases} 1, \text{ if } |V| = 1 \\ \min_{e \in E} T(G \setminus e) + 1, \text{ otherwise,} \end{cases}$$

where $T(G)$ is defined for all G with at most two connected components in the following way:

$$T(G) = \begin{cases} d \cdot C(G), \text{ if } G \text{ is connected;} \\ (d-1) \cdot C(G_1) + C(G_2), \text{ otherwise,} \end{cases}$$

where G_1 and G_2 are two connected components of graph G and $C(G_1) \leq C(G_2)$.

Lemma 3. *The minimal running time of a backtracking algorithm on an unsatisfiable Tseitin formula $\phi(G, f)$ based on a connected graph G does not depend on the function f and equals $C(G)$.*

Proof. We prove it by induction on the number of edges. The base case of induction is trivial. Consider now an arbitrary graph $G = \langle V, E\rangle$ and a function $f : V \to \mathbb{Z}_d$.

Let optimal backtracking algorithm start with splitting on a variable x_e. In the first case $G \setminus e$ is connected. Then we have to solve d subproblems of the type $\phi(G \setminus e, f'_a)$, where the function f'_a differs from f on the ends of the edge e. By the induction hypothesis the minimal running time of a backtracking algorithm, on the formula $\phi(G \setminus e, f'_a)$ is equal to $C(G \setminus e)$. Therefore the total number of steps of the optimal backtracking algorithm is $d \cdot C(G \setminus e)$.

In the second case the edge e is a bridge of the graph G. Let G_1 and G_2 be the two connected components of $G \setminus e$. After substitutiing $x_e := a$, the formula $\phi(G, f)$ splits into two independent subformulas ϕ_1 and ϕ_2, that correspond to graphs G_1, G_2 and to functions $f_{1,a}$, $f_{2,a}$, respectively.

We show that there is exactly one value of x_e that makes the formula ϕ_i satisfiable for $i = 1, 2$. The inductive hypothesis implies that the minimal complexity of a backtracking algorithm is $(d-1) \cdot C(G_1) + C(G_2) + 1$.

Let an edge e connect vertices u and v and the vertex v belong to G_1. Note that the values of functions $f_{1,a}$ and f on the vertices of graph G_1 can differ only at vertex v. Lemma 1 implies that if we fix the $f_{1,a}$-values for all vertices in G_1 except v, then there exists exactly one value of $f_{1,a}(v)$ that makes ϕ_1 satisfiable. □

Lemma 3 allows one to present a protocol of a backtracking algorithm in an economical way. We define a rooted tree; nodes of this tree are labeled with connected graphs. For the Tseitin formula $\phi(G, f)$ our tree T looks as follows:

- The root of the tree is labeled with G.
- Every leaf of the tree is labeled with a graph with one vertex.
- Every node of the tree has either one or two children.
- Let node v be labeled with a graph G_v. If v has only one child then it is labeled with $G_v \setminus e$ for some edge e. If v has two children then each of them is labeled with the corresponding connected component of $G_v \setminus e$ for some bridge e in G_v.

We call such a tree a reduced splitting tree.

We define a function f on the nodes of a reduced splitting tree.

$$f(v) = \begin{cases} 1, & \text{if } v \text{ is a leaf;} \\ d \cdot f(u) + 1, & \text{if } u \text{ is a unique child of } v; \\ (d-1) \cdot f(u_1) + f(u_2) + 1, & \text{where } u_1, u_2 \text{ are children of } v \text{ and} \\ & f(u_1) \le f(u_2); \end{cases}$$

For a reduced splitting tree T we define $F(T) = f(r)$, where r is a root of T. It is easy to see that

$$C(G) = \min_T F(T),$$

where the minimum is over all reduced splitting trees for a given graph G.

4.2 Lower Bound

We define the notion of the width of a reduced splitting tree.

Let $G = \langle V, E\rangle$ be a connected graph and ϕ be an unsatisfiable CSP based on G. Let T be a reduced splitting tree for ϕ. We consider a node v labeled with $G_v = \langle V_v, E_v\rangle$. Let $E_{ext} = \{(x, y) \in E | x \in V \vee y \in V_v\}$ be a number of edges that have at least one end in the set V_v. We set $w(v) = |E_{ext} \setminus E_v|$, that is, the number of removed edges that are incident to some vertices from V_v. The width of a tree T is $W(T) = \max_v w(v)$, where the maximum is over all nodes of T.

Lemma 4. *For every connected graph $G = \langle V, E\rangle$ with expansion $e(G)$ and for every reduced splitting tree of an unsatisfiable formula $\phi(G, f)$ the inequality $W(T) \geq e(G)$ holds.*

Proof. Let T be a reduced splitting tree. T contains a node v labeled with $G_v = \langle V_v, E_v\rangle$ such that

- $|V_v| > \frac{2}{3} \cdot |V|$;
- v has two children;
- if u is a child of v, then $|V_u| \leq \frac{2}{3} \cdot |V|$.

There exists a node u, that is a child of v and $|V_u|$ is between $\frac{1}{3}|V|$ and $\frac{2}{3}|V|$. Thus by the definition of the expansion $w(u) \geq e(G)$. □

Lemma 5. *Let T be a reduced splitting tree for Tseitin formula $\phi(G)$. Then there exists a reduced splitting tree T' for $\phi(G)$ such that $W(T') \leq k + \log_d F(T)$.*

Proof. By induction on the number of nodes in the tree T we show that if $F(T) \leq d^b$, then there exists a tree T' for $\phi(G)$ such that $W(T') \leq k + b$. The base case of induction is obvious.

Let the root r of T have only the one child v. Let T_v be a subtree of T with the root v. If $F(T) \leq d^b$, then $F(T_v) \leq d^{b-1}$. By the induction hypothesis we have a tree T'_v such that $W(T') \leq b - 1 + k$. We attach the tree T'_v to r and get a tree T' such that $W(T') \leq (b - 1 + k) + 1 = b + k$.

Let r have two children v_1 and v_2. Let T_1 and T_2 be subtrees with roots in v_1 and v_2 respectively; G_1 and G_2 are labels of v_1 and v_2 respectively. By the definition of F

$$F(T) = (d - 1) \cdot F(T_1) + F(T_2) + 1,$$

We know that $d \cdot F(T_1) < F(T)$. Thus if $F(T) \leq d^b$, then $F(T_1) \leq d^{b-1}$ and $F(T_2) \leq d^b$. Therefore by the induction hypothesis there exist reduced splitting trees T'_1 and T'_2 for G_1 and G_2, respectively, such that $W(T'_1) \leq k + b - 1$ and $W(T'_2) \leq k + b$. We show that T'_1 and T'_2 may be used in the construction of such a reduced splitting tree T' for $\phi(G)$ that $W(T) \leq k + b$.

Let the root r of the tree be labeled with G and children v_1 and v_2 be labeled with the connected components of $G \setminus e$. Let an edge e connect a vertex z from G_1 with a vertex y from G_2. We construct T' as follows.

We modify the tree T'_2: to every label that contains the vertex y we attach a copy of the graph G_1 to y by edge e. The original tree T'_2 contains a leaf that

is labeled with the graph with only one vertex y; after the modification this leaf is labeled with y with G_1 attached by means of the edge e. We make a splitting in this leaf on the variable x_e. We get a leaf that is labeled with z and a leaf w that is marked with G_1. We attach a tree T_1' to w. So we get a reduced splitting tree T' for $\phi(G)$ such that $W(T') \leq \max\{W(T_2'), W(T_1')+1, k\} \leq k+b$. □

Corollary 2. $e(G) \leq k + \log_d C(G)$.

Finally we prove the following theorem:

Theorem 3. *If the degrees of all vertices of a graph G are at most k, then the size of a tree-like resolution proof of unsatisfiable Tseitin formula $\phi(G, f)$ is at least $d^{e(G)-k}$.*

5 Upper Bound for CSP

We consider an arbitrary unsatisfiable CSP $\phi = \langle X, D, S\rangle$. Let $|D| = d$. For every constraint $C \in S$ we denote by $\mathrm{Vars}(C)$ the set of variables x such that C depends on x.

We construct the dependency graph $G = \langle V, E\rangle$ of CSP ϕ. Vertices of this graph correspond to constraints from S. Two constraints C_i and C_j are connected with $|\mathrm{Vars}(C_i) \cap \mathrm{Vars}(C_j)|$ edges, every edge is labeled with a common variable of C_i and C_j.

Note that the dependency graph of a Tseitin formula based on a graph H is isomorphic to H.

Theorem 4. *In the dependency graph $G = \langle V, E\rangle$ of an unsatisfiable CSP ϕ there is a subgraph H with the expansion $e(H) \geq \frac{\log_d S_T(\phi)}{\log_{\frac{3}{2}} |V|}$, where $S_T(\phi)$ is a size of a minimal tree-like resolution refutation of ϕ.*

Proof. We consider the following backtracking algorithm $A(\phi)$

- It constructs a dependency graph $G = \langle V, E\rangle$ of ϕ.
- It finds a minimal cut $U \subseteq V$ such that $\frac{1}{3} \cdot |V| \leq |U| \leq \frac{2}{3} \cdot |V|$.
- For all variables that correspond to edges that connect U with its complement, algorithm A chooses them for splitting one by one.
- Now graph contains several connected components. The algorithm chooses an unsatisfiable component and makes a recursive call on it

Let $Time$ be the running time of the algorithm A; it equals to the size of some tree-like resolution proof of ϕ.

The execution protocol of A can be represented by a tree T with weighted edges (edges correspond to cuts and weights correspond to sizes of cuts). Vertices of the tree T are labelled with CSPs, that are passed in the recursive calls. Let the vertex v contain a formula ϕ and let the algorithm A find a cut U in the dependency graph of ϕ.

Let $X_{\phi,U}$ be the set of variables corresponding to edges in this cut. The weight of the edge that corresponds to this cut is $|X_{\phi,U}|$. A weighted height of T is the maximal weight of the path from the root of T to a leaf. Let us denote the weighted height of T by h. Note that Time $\leq d^h$.

The number of vertices in the dependency graph of CSP in the child of T is at least $\frac{3}{2}$ times smaller than the number of vertices in the parent. Let a vertex u be the parent of a vertex v. Then the number of vertices in the dependency graph of the CSP in the vertex u is at least $\frac{3}{2}$ times the number of vertices in the dependency graph of the CSP in the vertex v

Let us denote the unweighted height of T by h_u; then h_u is at most $\log_{\frac{3}{2}} |V|$. Hence there exists an edge (v, u) with the weight at least $\frac{h}{\log_{\frac{3}{2}} |V|} \geq \frac{\log_d \text{Time}}{\log_{\frac{3}{2}} |V|}$.

Let CSP in v correspond to a dependency graph H. Therefore: $e(H) \geq \frac{\log_d \text{Time}}{\log_{\frac{3}{2}} |V|}$. □

Corollary 3. *Time* $\leq d^{e(H)\cdot\log_{\frac{3}{2}} |V|}$.

Corollary 4. *For an unsatisfiable Tseitin formula ϕ over domain $D = \mathbb{Z}_d$ based on a graph $G = \langle V, E\rangle$ of maximal degree k, there exists a subgraph H of G such that $S_T(\phi) \leq d^{e(H)\cdot\log_{\frac{3}{2}} |V|}$.*

Proof. The dependency graph of ϕ is isomorphic to G. Therefore by the Theorem 4 there is a subgraph H in G such that $S_T(\phi) \leq d^{e(H)\cdot\log_{\frac{3}{2}} |V|}$. □

Thus the minimal running time of a backtracking algorithm on a Tseitin formula based on a graph G satisfies inequalities $d^{e(G)-k} \leq \text{Time} \leq d^{e(H)\cdot\log_{\frac{3}{2}} |V|}$ for some subgraph H of G.

6 Open Questions

- Prove (or disprove) that there exists $c > 1$ such that the size of a dag-like resolution proof of a Tseitin formula based on a graph G is at least $c^{e(G)}$. Such a lower bound exists if $e(G) = \Omega(n)$. This is also true for doubled graphs where every edge has a parallel copy.
- Reduce the gap between the upper and lower bounds.

Acknowledgements. The authors are grateful to Alexander Shen for drawing attention to the case of arbitrary alphabets and to anonymous referees for comments that improved the readability of the paper.

References

1. Alekhnovich, M., Hirsch, E.A., Itsykson, D.: Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. J. Autom. Reason. 35(1-3), 51–72 (2005)
2. Baker, A.B.: Intelligent backtracking on constraint satisfaction problems: Experimental and theoretical results (1995)

3. Buss, S., Grigoriev, D., Impagliazzo, R., Pitassi, T.: Linear gaps between degrees for the polynomial calculus modulo distinct primes. Journal of Computer and System Sciences, 547–556 (1999)
4. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow — resolution made simple. Journal of ACM 48(2), 149–169 (2001)
5. Cook, J., Etesami, O., Miller, R., Trevisan, L.: Goldreich's one-way function candidate and myopic backtracking algorithms. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 521–538. Springer, Heidelberg (2009)
6. Hwang, C.Y.J.: A Theoretical Comparison of Resolution Proof Systems for CSP Algorithms. Master's thesis, Simon Fraser University (2004)
7. Impagliazzo, R., Pudlak, P., Sgall, J.: Lower bounds for the polynomial calculus and the grobner basis algorithm. Computational Complexity 8, 127–144 (1997)
8. Itsykson, D., Sokolov, D.: The complexity of inversion of explicit Goldreichs function by DPLL algorithms. Zapiski nauchnyh seminarov POMI 399, 88–109 (2012)
9. Itsykson, D.: Lower bound on average-case complexity of inversion of goldreich's function by drunken backtracking algorithms. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 204–215. Springer, Heidelberg (2010)
10. Mitchell, D.G.: The resolution complexity of constraint satisfaction (2002)
11. Mitchell, D.G.: Resolution complexity of random constraints. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 295–309. Springer, Heidelberg (2002)
12. Mitchell, D.G.: Resolution and constraint satisfaction. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 555–569. Springer, Heidelberg (2003)
13. Tseitin, G.S.: On the complexity of derivation in the propositional calculus. Zapiski Nauchnykh Seminarov LOMI 8, 234–259 (1968); English translation of this volume: Consultants Bureau, N.Y., pp. 115–125 (1970)
14. Urquhart, A.: Hard examples for resolution. JACM 34(1), 209–219 (1987)

Towards NEXP versus BPP?

Ryan Williams*

Stanford University

Abstract. We outline two plausible approaches to improving the miserable state of affairs regarding lower bounds against probabilistic polynomial time (namely, the class BPP).

1 Introduction

In recent years, researchers have been gradually developing methods for potentially proving non-uniform circuit lower bounds against "large" complexity classes, such as nondeterministic exponential time (NEXP). These methods grew out of thinking about how to generically *derandomize* probabilistic algorithms: given an algorithm A which makes random decisions and solves a problem with high probability, can we construct a deterministic algorithm B with similar running time and equivalent behavior to A? Many non-trivial and surprising connections have been discovered between this basic problem and that of proving circuit lower bounds (e.g., [10,3,8,1,11,7,9]). Most of these papers show how circuit lower bounds can imply interesting derandomizations. Impagliazzo, Kabanets, and Wigderson proved (among many other results) an implication in the opposite direction: some derandomizations can in fact imply circuit lower bounds for NEXP:

Theorem 1 ([7]). *Suppose every problem in* promiseBPP *can be solved (even nondeterministically) in* 2^{n^ε} *time, for every* $\varepsilon > 0$. *Then* NEXP $\not\subset$ P/*poly.*

That is, subexponential-time deterministic simulations of probabilistic polynomial time imply that NEXP cannot be simulated with non-uniform polynomial-size circuits. To be more precise, Theorem 1 boils down to the following claim. In the *Circuit Approximation Probability Problem* (a.k.a. CAPP), a Boolean circuit C is given, and one is asked to approximate the quantity $\Pr_x[C(x) = 1]$ within an additive factor of $1/10$. IKW show that if CAPP can always be solved in $2^{n^{o(1)}}$ time on circuits of size n, then NEXP $\not\subset$ P/poly.

On a circuit with n inputs, the fastest known algorithm for CAPP is simply exhaustive search, taking $\Omega(2^n)$ time. It seems that an improvement from 2^n time to 2^{n^ε} time would be major, and far from something one might hope to establish

* Supported in part by a David Morgenthaler II Faculty Fellowship, and the National Science Foundation under Grant Number CCF-1212372. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 174–182, 2013.

in the near future. Fortunately, the hypothesis of Theorem 1 can be significantly weakened. It has been shown that essentially any nontrivial improvement over exhaustive search for CAPP would already yield the desired lower bounds:

Theorem 2 ([12]). *Suppose for every k, CAPP on circuits of size n^k and n inputs can be solved (even nondeterministically) in $O(2^n/n^k)$ time. Then* $\mathsf{NEXP} \not\subset \mathsf{P}/poly$.

Similarly, the paper [12] also shows how slightly faster circuit satisfiability algorithms would also entail $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$. It is possible that Theorem 2 gives a reasonable approach to proving $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$ – to prove the lower bound, it suffices to show that $\mathsf{NEXP} \subset \mathsf{P}/\mathrm{poly}$ (a strong assumption that is algorithmic in nature) yields a nontrivial algorithm for approximating the acceptance probabilities of circuits. This approach was met with skepticism until it was shown how a variant of Theorem 2 can be applied (along with other ideas) to unconditionally prove that NEXP does not have polynomial-size ACC circuits [13]. The approach has also been recently extended to prove ACC circuit lower bounds for $\mathsf{NEXP} \cap \mathsf{coNEXP}$ as well [14].

This is all fine and good, and we are optimistic that circuit lower bounds will continue to be developed by studying the connections between circuit-analysis algorithms and circuit limitations. However, while $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$ is one of the many longstanding open questions in computational complexity theory, it is not the most embarrassing one. Strictly more embarrassingly open questions arise when we begin to discuss the status of lower bounds against probabilistic polynomial time itself. The above results show that circuit lower bounds already follow from tiny improvements on deterministic exhaustive search for problems that are trivial with randomness. However, it is still consistent with current knowledge that randomness is omnipotent! Complexity theory has not yet proved that $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$ (exponential time with an NP oracle is different from probabilistic polynomial time with two-sided error), nor have we established $\mathsf{EXP} \neq \mathsf{ZPP}$ (exponential time is different from zero-error probabilistic polynomial time), yet we believe that $\mathsf{P} = \mathsf{BPP}$ [8].[1]

Troubled by this, we have recently been thinking about how the above theorems and techniques developed for proving circuit lower bounds could potentially apply to lower bounds against BPP and ZPP.[2] This paper suggests two plausible hypotheses, one of which is significantly weaker than solving the CAPP problem in general. We prove that establishing the truth of either of these hypotheses would yield $\mathsf{NEXP} \neq \mathsf{BPP}$.

In the remainder of the paper, we assume a basic working knowledge of complexity theory, at the level of Arora and Barak's textbook [2]. For example, we

[1] A separation problem like $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$ is strictly more embarrassing than circuit lower bounds, because circuit lower bounds would already imply them, i.e., $\mathsf{EXP}^{\mathsf{NP}} \not\subset \mathsf{P}/\mathrm{poly}$ implies $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$.

[2] It is immediate from the ACC lower bounds work that $\mathsf{NEXP} \neq \mathsf{BPACC}$, where BPACC denotes probabilistic uniform ACC with two-sided error. Moreover, $\mathsf{REXP} \neq \mathsf{BPACC}$ also holds, because $\mathsf{REXP} \subseteq \mathsf{BPP}$ would imply $\mathsf{NP} = \mathsf{RP}$ and hence $\mathsf{NEXP} = \mathsf{REXP}$. We are after bigger fish than these.

expect the reader to understand complexity classes like P/poly, ZPP, BPP, EXP, and NEXP, and possess a high-level familiarity with concepts such as probabilistically checkable proofs and pseudorandomness.

2 Derandomizing CAPP over Simple Distributions of Circuits

In this section, we will show that in order to separate NEXP and BPP, it suffices to give deterministic *heuristics* for the CAPP problem which only barely improve on exhaustive search, and only succeed with very low probability on polynomial-time samplable distributions of circuits.

Given a Boolean circuit C on n inputs, let $tt(C)$ be its *truth table*, the 2^n-bit string whose ith bit equals the value of the circuit on the ith n-bit string. First we show that NEXP = BPP implies the existence of efficient probabilistic algorithms that can print small circuits encoding witnesses to NEXP computations. This basically follows from work of Impagliazzo, Kabanets, and Wigderson [7].

Lemma 1. *Suppose* NEXP = BPP. *Then there is a k such that, for every ℓ and for every* NEXP *verifier V accepting a language L, there is a* BPP *algorithm A_V such that, for all $x \in L$, $A_V(x, r)$ outputs (with probability at least $1 - 1/2^{|x|^\ell}$ over all r) a circuit C_x of size at most $|x|^k$ such that $V(x, tt(C_x))$ accepts.*[3]

Proof. First observe that NEXP = BPP implies NEXP $\subseteq$ P/poly. By Impagliazzo, Kabanets, and Wigderson, NEXP $\subseteq$ P/poly implies that NEXP has *succinct witness circuits*: for every $L \in$ NEXP, for every verifier algorithm V for L, and every $x \in L$, there is a poly($|x|$)-size circuit C_x such that $V(x, tt(C_x))$ accepts. We want to show that these C_x can be constructed in BPP, under the assumptions.

For every NEXP verifier V that accepts a language $L \in$ NEXP, there is a k and an exponential-time algorithm $A(x, i)$ which given a string x and index i, enumerates all possible circuits D of size $|x|^k + k$, checking if $V(x, tt(D))$ accepts. If this ever happens, $A(x, i)$ then outputs the ith bit of the encoding of D (otherwise, let $A(x, i)$ output 0).

Under EXP = BPP, there must exist a BPP algorithm A' equivalent to A: given (x, i), A' outputs (with high probability) the ith bit of such a circuit D. By probability amplification (repeating A' for poly($|x|$) times, for each $i = 1, \ldots, |x|^k + k$, and taking the majority answer for each i), there is a probabilistic polynomial-time algorithm A'' which given $x \in L$ prints a circuit D encoding a witness for x, with $1/2^{|x|^\ell}$ probability of error. Let $A_V = A''$. □

Our next ingredient is the *PCPs of Proximity* of Ben-Sasson *et al.*, which imply succinct PCPs for NEXP.

[3] An algorithm V is a *verifier* for $L \in$ NEXP if there is a k such that for every string x, we have $x \in L$ if and only if there is a y of length $2^{|x|^k}$ such that $V(x, y)$ accepts within $O(2^{|x|^k})$ steps.

Theorem 3 ([4]). *Let $T : \mathbb{Z}^+ \to \mathbb{Z}^+$ be a non-decreasing function. Then for every $s > 0$ and every language $L \in \mathsf{NTIME}[T(n)]$ there exists a PCP verifier $V(x, y)$ with soundness s, perfect completeness, randomness complexity $r = \log_2 T(|x|) + O(\log \log T(|x|))$, query complexity $q = poly(\log T(|x|))$, and verification time $t = poly(|x|, \log T)$. More precisely:*

- *V has random access to x and y, uses at most r random bits in any execution, makes q queries to the candidate proof y. and runs in at most t steps.*
- *If $x \in L$ then there is a string y of length $T(|x|) \log^{O(1)} T(|x|)$ such that $\Pr[V(x, y)$ accepts$] = 1$.*
- *If $x \notin L$ then for all y, $\Pr[V(x, y)$ accepts$] \leq s$.*

A standard perspective to take in viewing PCP results is to think of the polynomial-time verifier V as encoding an exponentially long constraint satisfaction problem, where each setting of the random bits in the verifier yields a new constraint. For our purposes, we will want $T(n)$ to be 2^n, and s to be an arbitrarily small constant (e.g., 1/10). Then Theorem 3 gives a PCP verifier with $n + O(\log n)$ bits of randomness, poly(n) verification time, and poly(n) query complexity. By converting this polynomial-time verifier to a polynomial-size circuit that takes randomness as input, we can produce a reduction from every $L \in \mathsf{NTIME}[2^n]$ to the so-called Succinct-CSP problem, with the properties:

- Every instance x of L is reduced to a poly(n)-size circuit C_x.
- The truth table of C_x, $tt(C_x)$, encodes a constraint satisfaction problem with $2^n n^{O(1)}$ constraints and variables.
- Each constraint in $tt(C_x)$ contains poly(n) variables and can be evaluated in poly(n) time.
- If $x \in L$ then the CSP $tt(C_x)$ is satisfiable.
- If $x \notin L$ then for every variable assignment to the CSP, at most an s-fraction of the constraints are satisfied.

A *polynomial-time samplable distribution* $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots\}$ on strings has the property that there is an n^k-time algorithm $A(1^n, r)$ such that for all n, the probability of drawing a string $x \in \{0,1\}^{\leq n^k}$ from $\mathcal{D}_n$ is exactly

$$\Pr_{r \in \{0,1\}^{n^k}}[A(1^n, r) = x].$$

That is, the polynomial time algorithm A perfectly models the distribution $\mathcal{D}$. (Note that our ensemble of distributions $\{\mathcal{D}_n\}$ is a bit different from standard practice: $\mathcal{D}_n$ does not contain only strings of length n but strings of length up to n^k for some fixed k.) A canonical example of a polynomial-time samplable distribution is the *uniform* distribution on strings. Polynomial-time samplable distributions are central to the study of average-case complexity theory.

We consider *deterministic errorless heuristics* for the CAPP problem, which may print *SAT*, *UNSAT*, or *FAIL*. We pose very weak requirements on the heuristic: if the satisfiable fraction of assignments to the circuit is at least 9/10, the heuristic must output *SAT* or *FAIL*; when the circuit is unsatisfiable, the algorithm always outputs *UNSAT* or *FAIL*.

Finally, we can state the main result of this section:

Theorem 4. *Suppose for every k, and every polynomial-time samplable distribution $\mathcal{D}$, the CAPP problem on circuits of size n^k and n inputs can be solved in $O(2^n/n^k)$ time by a deterministic heuristic H (possibly dependent on $\mathcal{D}$) such that $\Pr_{E\in\mathcal{D}_n}[H(E) \neq FAIL] > 1/2^n$. Then $\mathsf{NEXP} \neq \mathsf{BPP}$.*

That is, to separate NEXP from BPP, it suffices to design for every *polynomial-time samplable distribution of circuits* a heuristic for the CAPP problem which barely improves over exhaustive search, and only succeeds on a negligible measure of circuits from the given distribution. This is a significantly weaker requirement than designing a worst-case CAPP solver: here we get to see the efficient distribution of circuits that the adversary will construct, and we are allowed to fail on the vast majority of circuits output by the adversary.

It is useful to put Theorem 4 in perspective with another result on average-case complexity. Buhrman, Fortnow, and Pavan [5] have shown that if every problem in NP can be solved in polynomial time for every polynomial-time samplable distribution, then $\mathsf{P} = \mathsf{BPP}$. That is, if all NP problems can be efficiently solved in this way, we can separate EXP from BPP. Theorem 4 shows that a significantly weaker assumption suffices to separate NEXP from BPP.

Proof of Theorem 4. Let $L \subseteq \{1^n \mid n \geq 0\}$ be chosen so that $L \in \mathsf{NTIME}[2^n] \setminus \mathsf{NTIME}[2^n/n]$. (Such languages exist, due to the nondeterministic time hierarchy of Žák [15].) By Theorem 3, there is a polynomial-time reduction from L to Succinct-CSP, which outputs a circuit C_{1^n} on $n + O(\log n)$ inputs. If $\mathsf{NEXP} = \mathsf{BPP}$ then Lemma 1 says that for all NEXP languages L and verifiers V for L, there is a probabilistic polynomial-time algorithm A that prints valid witness circuits for V, with probability of error at most $1/3^n$ on inputs of size n. Let V be the verifier which, on input 1^n, tries to check that its certificate is a satisfying assignment for $tt(C_{1^n})$. Then A is then a probabilistic polynomial-time algorithm that (with high probability) on input 1^n prints a circuit D_{1^n} such that $tt(D_{1^n})$ is a satisfying assignment for $tt(C_{1^n})$, for $1^n \in L$. We can think of A as a deterministic algorithm A' which takes 1^n as one input, and a poly($|x|$)-bit random string r as a secondary input, where the overall output of A' is determined by the randomness r.

We design a polynomial-time samplable distribution $\mathcal{D}$ of circuits, as follows. Given 1^n, our polynomial-time algorithm B first runs the reduction of Theorem 3 to produce a circuit C_{1^n} such that $1^n \in L$ if and only if $C_{1^n} \in$ Succinct-CSP. Then B picks a random seed r and runs $A'(1^n, r)$ which (with probability at least $1 - 1/3^n$) prints a circuit D_{1^n} encoding a satisfying assignment for the Succinct-CSP instance C_{1^n}. Using multiple copies of the two circuits C_{1^n} and D_{1^n}, one can construct a polynomially larger circuit E with $n + O(\log n)$ inputs and the following properties:

- If D_{1^n} encodes a satisfying assignment to $tt(C_n)$ then E is unsatisfiable.
- If D_{1^n} does not encode a satisfying assignment to $tt(C_n)$ then E is satisfiable on at least 9/10 of its possible inputs.

(For a proof of this construction, see [12].) Algorithm B then outputs E.

Suppose there is a heuristic H for the CAPP problem which runs in deterministic $O(2^n/n^k)$ time for all desired k, outputs *SAT* or *FAIL* when the fraction of assignments to the circuit which are satisfying is at least 9/10, *UNSAT* or *FAIL* when the fraction is 0, and on circuits randomly drawn from $\mathcal{D}$, H outputs *FAIL* with probability at most $1 - 1/2^n$. We wish to give a nondeterministic algorithm N which recognizes the language L in nondeterministic time $O(2^n/n)$, contradicting the choice of L.

On an input 1^n, N nondeterministically guesses a random seed r for the algorithm A', and runs $B(1^n)$ with this choice of seed r for A'. B outputs a circuit E, which is then fed to H. If H prints *UNSAT* then N accepts, otherwise N rejects.

Note that N can be made to run in time $O(2^n/n)$: although the circuit E has $n+O(\log n)$ inputs, we can choose k to be larger than the constant c in the big-O term, so that the heuristic H runs in time $O(2^{n+c\log n}/(n+c\log n)^k) \leq O(2^n/n)$.

We now argue that N is correct. If $1^n \in L$, then on at least $(1 - 1/3^n)$ of the seeds r, $A'(1^n, r)$ outputs a circuit D_{1^n} encoding a satisfying assignment for $tt(C_{1^n})$. When such an r is guessed, the circuit E drawn from $\mathcal{D}_n$ is unsatisfiable; hence in this case, at least a $(1 - 1/3^n)$ measure of the circuits in $\mathcal{D}_n$ are unsatisfiable. The satisfiability algorithm H outputs *FAIL* on less than $1/2^n$ of the circuits drawn from $\mathcal{D}_n$. Hence there is a random seed $r^\star$ such that the circuit $E^\star$ output by $\mathcal{D}_n$ is unsatisfiable, and $E^\star$ is declared *UNSAT* by H. Note N accepts provided that such an $r^\star$ is guessed by N.

If $1^n \notin L$ then for all seeds r, the circuit D printed by $A'(1^n, r)$ cannot encode a satisfying assignment for $tt(C_{1^n})$, so the resulting circuit E is always satisfied by at least 9/10 of its possible input assignments. The heuristic H on circuit E will always print *SAT* or *FAIL* in this case, and N rejects in either of the two outcomes. □ □

So, $\mathsf{NEXP} \neq \mathsf{BPP}$ would follow from CAPP heuristics that barely beat exhaustive search and output *FAIL* on all but a small fraction of inputs. Should we expect the existence of such heuristics to be easier to establish than worst-case CAPP algorithms? The answer is not clear. However it does seem plausible that one may be able to show $\mathsf{NEXP} = \mathsf{BPP}$ itself implies heuristic algorithms for CAPP, which would be enough to prove the desired separation result.

3 Pseudorandomness for Deterministic Observers

Our second direction for randomized time lower bounds is a simple reflection on Goldreich and Wigderson's work regarding pseudorandomness with efficient deterministic observers [6]. Informally, when one defines pseudorandomness, we have a *pseudorandom generator* (a function that maps short strings to long strings) along with a class of *observers* (efficient algorithms), and the generator is said to be pseudorandom to that class if every observer exhibits extremely similar behavior on the uniform distribution and the distributions of outputs of the generator.

An alternative way to define pseudorandomness is via *unpredictability*: a generator is unpredictable if no observer, given i bits of a random output from the generator, can predict the $(i+1)$st bit with probability significantly better than $1/2$, for all i. A central theorem in the theory of pseudorandomness is that the unpredictability criterion and the pseudorandomness criterion are essentially equivalent, when the class of observers is the set of probabilistic polynomial-time algorithms. That is, a generator which is unpredictable is also pseudorandom, and vice-versa.

What if the class of observers consists of *deterministic* polynomial-time algorithms? Then the connections between pseudorandomness and unpredictability are actually open problems. For every polynomial $p(n)$, Goldreich and Wigderson give an explicit distribution computable in time $\mathrm{poly}(p(n))$ which is unpredictable for all deterministic $p(n)$-time observers, by applying pairwise independent distributions in a clever way. They pose as an open problem whether their distribution is *pseudorandom* for all deterministic $p(n)$-time observers. We show that exhibiting an exponential-time computable distribution that is pseudorandom to linear-time observers implies $\mathsf{EXP} \neq \mathsf{BPP}$. In fact, it suffices to construct an exponential-time generator G that is given the code of a particular linear-time observer A, and G only has to fool A.

Theorem 5. *Suppose for every deterministic linear-time algorithm A, and all $\varepsilon > 0$, there is a $\delta > 0$ and algorithm G that runs in $O(2^m)$ time on inputs of length m, produces outputs of length $m^{1/\varepsilon}$, and*

$$\left|\Pr_{x\in\{0,1\}^n}[A(x)=1] - \Pr_{y\in\{0,1\}^{n^\varepsilon}}[A(G(y))=1]\right| < 1/2 - \delta.$$

Then $\mathsf{EXP} \neq \mathsf{BPP}$.

Proof. Assume $\mathsf{EXP} = \mathsf{BPP}$. Choose a language $L \subseteq \{1^n \mid n \geq 0\}$ such that $L \in \mathsf{TIME}[2^n] \setminus \mathsf{TIME}[2^{n/2}]$ (which can be easily derived by direct diagonalization). By assumption, and by amplification, there is a *deterministic* n^k-time algorithm $B(\cdot,\cdot)$ such that

- If $1^n \in L$, then $\Pr_{x\in\{0,1\}^{n^k}}[B(1^n,x)=1] > 1 - 1/2^n$.
- If $1^n \notin L$, then $\Pr_{x\in\{0,1\}^{n^k}}[B(1^n,x)=1] < 1/2^n$.

Define the algorithm $A(x) = B(1^{|x|^{1/k}}, x)$, which runs in linear time. Let $\varepsilon = 1/(2k)$, and suppose there were a function G satisfying the hypotheses of the theorem for algorithm A and ε. Then we could simulate L in $\mathsf{TIME}[2^{O(n^{1/2})}]$ (a contradiction), as follows: given 1^n, compute the $O(2^{n^{1/2}})$-size set of strings $S = \{G(y) \mid y \in \{0,1\}^{n^{1/2}}\} \subseteq \{0,1\}^{n^k}$ in $O(2^{2n^{1/2}})$ time, then output the majority value of $A(x)$ over all $x \in S$, in $2^{n^{1/2}}\mathrm{poly}(n)$ time. This $O(2^{2n^{1/2}})$ time algorithm decides L, because if $1^n \in L$ then $\Pr_{y\in\{0,1\}^{n^{1/2}}}[A(G(y))=1] > 1/2 + \delta/2$, and if $1^n \notin L$ then $\Pr_{y\in\{0,1\}^{n^{1/2}}}[A(G(y))=1] < 1/2 - \delta/2$. □

The above simple result shows that the ability to (slightly) fool deterministic linear-time algorithms with exponential-time generators is already enough to

separate EXP and BPP. The basic idea can be easily extended in several different ways. For one, we could make the generator G very powerful, and still derive a breakthrough lower bound: if G were also allowed to have free oracle access to the SAT problem (asking exponentially long NP queries about the behavior of A) in the above hypothesis, we could separate $\mathsf{EXP}^{\mathsf{NP}}$ from BPP. For another:

Theorem 6. *Suppose for every deterministic linear-time algorithm A, and all $\varepsilon > 0$, there is an algorithm G that runs in $O(2^m)$ time on inputs of length m, produces outputs of length $m^{1/\varepsilon}$, and for every n, if $\Pr_{x \in \{0,1\}^n}[A(x) = 1] > 1 - 1/n$ then there is a $y \in \{0,1\}^{n^\varepsilon}$ such that $A(G(y)) = 1$. Then $\mathsf{EXP} \neq \mathsf{ZPP}$.*

That is, we only require that, when A accepts the vast majority of n-bit strings, the algorithm G prints *at least one* n-bit string (out of 2^{n^ε}) that is accepted by A. The proof is analogous.

4 Conclusion

In this short paper, we outlined two potential directions for attacking the basic separation problems between exponential time and probabilistic polynomial time. In general we believe that proving separations like $\mathsf{NEXP} \neq \mathsf{BPP}$ are not impossible tasks, but a couple of new ideas will probably be needed to yield the separation. The reader should keep in mind that such separation results will require non-relativizing techniques (there are oracles relative to which $\mathsf{NEXP} = \mathsf{BPP}$ and $\mathsf{NEXP} \neq \mathsf{BPP}$), so no simple black-box arguments are expected to yield new lower bounds against BPP. However, in this day and age, non-relativizing tools are not so hard to come by.

References

1. Andreev, A.E., Clementi, A.E.F., Rolim, J.D.P.: Worst-case hardness suffices for derandomization: A new method for hardness-randomness tradeoffs. TCS: Theoretical Computer Science 221(1-2), 3–18 (1999)
2. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009)
3. Babai, L., Fortnow, L., Nisan, N., Wigderson, A.: BPP has subexponential time simulations unless EXPTIME has publishable proofs. Computational Complexity 3(4), 307–318 (1993)
4. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Short PCPs verifiable in polylogarithmic time. In: IEEE Conference on Computational Complexity, pp. 120–134 (2005)
5. Buhrman, H., Fortnow, L., Pavan, A.: Some results on derandomization. Theory Comput. Syst. 38(2), 211–227 (2005)
6. Goldreich, O., Wigderson, A.: On pseudorandomness with respect to deterministic observers. In: ICALP Satellite Workshops 2000, pp. 77–84 (2000)
7. Impagliazzo, R., Kabanets, V., Wigderson, A.: In search of an easy witness: Exponential time vs. probabilistic polynomial time. JCSS 65(4), 672–694 (2002)

8. Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In: Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, pp. 220–229 (1997)
9. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1-2), 1–46 (2004)
10. Nisan, N., Wigderson, A.: Hardness vs randomness. JCSS 49(2), 149–167 (1994)
11. Sudan, M., Trevisan, L., Vadhan, S.: Pseudorandom generators without the xor lemma. Journal of Computer System Sciences 62(2), 236–266 (2001)
12. Williams, R.: Improving exhaustive search implies superpolynomial lower bounds. In: ACM Symposium on Theory of Computing, pp. 231–240 (2010)
13. Williams, R.: Non-uniform ACC circuit lower bounds. In: IEEE Conference on Computational Complexity, pp. 115–125 (2011)
14. Williams, R.: Natural proofs versus derandomization. In: ACM Symposium on Theory of Computing (to appear, 2013)
15. Žák, S.: A Turing machine time hierarchy. Theoretical Computer Science 26(3), 327–333 (1983)

Information Lower Bounds via Self-reducibility

Mark Braverman[1,⋆], Ankit Garg[1], Denis Pankratov[2], and Omri Weinstein[1]

[1] Princeton University
[2] University of Chicago

Abstract. We use self-reduction methods to prove strong information lower bounds on two of the most studied functions in the communication complexity literature: Gap Hamming Distance (GHD) and Inner Product (IP). In our first result we affirm the conjecture that the information cost of GHD is linear even under the *uniform* distribution, which strengthens the $\Omega(n)$ bound recently shown by [15], and answering an open problem from [10]. In our second result we prove that the information cost of IP_n is arbitrarily close to the trivial upper bound n as the permitted error tends to zero, again strengthening the $\Omega(n)$ lower bound recently proved by [9].

Our proofs demonstrate that self-reducibility makes the connection between information complexity and communication complexity lower bounds a two-way connection. Whereas numerous results in the past [12,2,3] used information complexity techniques to derive new communication complexity lower bounds, we explore a generic way in which communication complexity lower bounds imply information complexity lower bounds *in a black-box manner*.

1 Introduction

The primary objective of this paper is to continue the investigation of the information complexity vs. communication complexity problem. Informally, in a two-party setting, communication complexity (CC) measures the number of bits two parties need to exchange to solve a certain problem. Information complexity (IC) measures the average amount of information the parties need to reveal each other about their inputs in order to solve it. IC is always bounded by CC from above. A key open problem surrounding information complexity is actually understanding the gap between the two:

Problem 1. Is it true that for all functions f it holds that $IC(f) = \Omega(CC(f))$?

The problem, and where it fits more broadly within communication complexity is discussed in [4]. The above question is a natural question in the context of coding theory, where it can be re-interpreted as asking whether an analogue of Huffman coding holds for interactive computation. Shannon's original insight [19] was that the (amortized) number of bits one needs to send in order to transmit a message

⋆ Partially supported by an Alfred P. Sloan Fellowship, an NSF CAREER award, and a Turing Centenary Fellowship.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 183–194, 2013.

X equals to the amount of information it coveys – its entropy $H(X)$. Huffman coding [14] can be viewed as a one-copy version of this result: even when sending one instance of the message, we can guarantee expected cost of $\leq H(X) + 1$ – i.e. messages can be compressed into their information content plus at most one bit. Problem 1 can be viewed as a quest for an interactive analogue of Huffman coding: can a long (interactive) communication protocol that solves f but only conveys $IC(f)$ information be compressed in a way that only requires $O(IC(f))$ communication?

Another direction which motivates Problem 1 are *direct sum* problems in randomized communication complexity [12,2,3,8]. It turns out that the analogue of the Shannon's amortized coding theorem does in fact hold for interactive computation [8], asserting that $\lim_{k\to\infty} CC(f^k)/k = IC(f)$. Thus, understanding the relationship between $IC(f)$ and $CC(f)$ is equivalent to understanding the relationship between computing one copy of f and the amortized cost of computing many copies of f in parallel, which is the essence of the direct sum problem.

Yet another motivation for considering the information complexity of tasks comes from the study private two-party computation [16,18,1]. In this setting Alice and Bob want to compute a function $f(x, y)$ on their private inputs x and y respectively without "leaking" too much information to each other. This can be accomplished using cryptography, assuming Alice and Bob are computationally bounded. Without this assumption, the amount of information that Alice and Bob must reveal to each other is exactly $IC(f)$. In this context, an affirmative answer for Problem 1 would mean that, up to a constant, a protocol minimizing communication (with no special consideration for privacy) will reveal the same amount of information as the most "private" protocol. Moreover, if for a family $\{f_n\}$ of functions we have that $IC(f_n)/CC(f_n) \to 1$ as $n \to \infty$, it means that as n grows, there is *nothing* the parties can do to perform the computation more privately, and the most efficient protocol is also the most private. In this paper we show, for example, that this is the case for the Inner Product function IP_n, whose communication complexity is n, and whose information complexity we show to be $n - o(n)$ (for negligible error).

In this paper we develop a new self-reducibility technique for deriving information complexity lower bounds from communication complexity lower bounds. The technique works for functions that have a "self-reducible structure". Informally speaking f has a self-reducible structure, if for large enough inputs, solving f_{nk} essentially amounts to solving f_n^k (f_{nk} denotes the function f under inputs of length nk, while f_n^k denotes k independent copies of f under inputs of size n). Our departing point is a communication complexity lower bound for f_{nk} (that may be obtained by any means). Assuming self-reducibility, the same bound applies to f_n^k, which through the connection between information complexity and amortized communication complexity [8], implies a lower bound on the information complexity of f_n. In this paper we develop tools to make this reasoning go through.

Ideas of self-reducibility are central in applications of information complexity to communication complexity lower bounds, starting with the work of Bar-Yossef et al. [2]. These argument start with an information complexity lower bound for a (usually very simple) problem, and derive a communication complexity bound on many copies of the problem. The logic of this paper is reversed: we start with a communication complexity lower bound, which we use as a black-box, and use self-reducibility to derive an amortized communication complexity bound, which translates into an information complexity lower bound. An additional conceptual take-away from the present paper, is that to look for a counterexample for Problem 1, one would likely need to consider problems that are highly non-self-reducible.

1.1 Results

We use the self-reducibility technique to prove results about the information complexity of Gap Hamming Distance and Inner Product. We prove that the information complexity of the Gap Hamming Distance problem with respect to the uniform distribution is linear. This was explicitly stated as an open problem by Chakrabarti et al. [10]. Formally, let $\mathsf{IC}_\mu(GHD_{n,t,g}, \varepsilon)$ denote the information cost of the Gap Hamming promise problem, where inputs x, y are n-bit strings distributed according to μ, and the players need to determine whether the Hamming distance between x and y is at least $t+g$, or at most $t-g$, with error at most ε under μ. We prove

Theorem 1. *There exists an absolute constant $\varepsilon > 0$ for which*

$$\mathsf{IC}_{\mathcal{U}}(GHD_{n,n/2,\sqrt{n}}, \varepsilon) = \Omega(n)$$

where $\mathcal{U}$ is the uniform distribution.

For the Inner Product problem, where the players need to compute $\sum_{i=1}^{n} x_i y_i (mod\ 2)$, we prove a stronger bound on its information complexity. Formally,

Theorem 2. *For every constant $\delta > 0$, there exists a constant $\epsilon > 0$, and n_0 such that $\forall\ n \geq n_0$, $IC_{\mathcal{U}_n}(IP_n, \epsilon) \geq (1-\delta)n$. Here $\mathcal{U}_n$ is the uniform distribution over $\{0,1\}^n \times \{0,1\}^n$.*

Note that $IC_{\mathcal{U}_n}(IP_n, \epsilon) \leq (1-2\epsilon)(n+1)$, since the parties can always output a random value $\in \{0,1\}$ with probability 2ϵ, and have one of the parties send its entire input with probability $1-2\epsilon$ (Indeed, this protocol has error $(1/2)\cdot 2\epsilon + (1-2\epsilon)\cdot 1 = 1-\epsilon$, and information cost $(1/2)\cdot 0 + (1-2\epsilon)\cdot(n+1) = (1-2\epsilon)\cdot(n+1)$). Also it is known that $IC_{\mathcal{U}_n}(IP_n, \epsilon) \geq \Omega(n)$, for all $\epsilon \in [0, 1/2)$ [9]. We prove that the information complexity of IP_n can be arbitrarily close to the trivial upper bound n as we keep decreasing the error (though keeping it a constant).

1.2 Discussion and Open Problems

Although in Complexity Theory we often don't care about the constants (and often it is not necessary), proving theorems with the right constants can often lead to deeper insights into the mathematical structure of the problem [5,7]. There are few techniques that allow us to find the right constants and there are fewer problems for which we can. We believe that answering the following problem will lead to development of new techniques and also reveal interesting insights into the problem of computing the XOR of n copies of a function.

Open Problem 1. *Is it true that for small constants ϵ and sufficiently large n, $IC_{\mathcal{U}_n}(IP_n, \epsilon) \geq (1 - 2\epsilon - o(\epsilon))n$? As before $\mathcal{U}_n$ is the uniform distribution. If this is false, is there a different constant $\alpha > 2$ such that as $\epsilon \to 0$ we get $IC_{\mathcal{U}_n}(IP_n, \epsilon) \geq (1 - \alpha \cdot \epsilon)n$?*

Solving this problem may require shedding new light on the rate of convergence of the $IC_\mu(\bullet, \epsilon)$ to $IC_\mu(\bullet, 0)$ as $\epsilon \to 0$, and better understanding the role error plays in information complexity.

It is somewhat difficult to define the exact meaning of the "right" constant for the Gap Hamming Distance problem, since it is a promise problem defined by two parameters (gap and error). Nonetheless, there is a very natural regime in which understanding the exact information complexity of GHD_n is a natural and interesting problem. Namely:

Open Problem 2. *Is it true that for all $\varepsilon > 0$, there is a $\delta > 0$ and a distribution μ such that $\mathsf{IC}_\mu(GHD_{n,n/2,\delta\sqrt{n}}, \delta) > (1 - \varepsilon)n$?*

In other words, does the information complexity of GHD_n tends to the trivial upper bound as we tighten the gap and error parameters? This is related to the same (but weaker) question one can ask about the communication complexity of GHD_n in this regime.

2 Preliminaries

In this section we briefly survey the necessary background for this paper on information theory and communication complexity. For a more thorough treatment of these subjects see [8] and references therein.

Notation. We use capital letters for random variables, calligraphic letters for sets, and small letters for elements of sets. For random variables A and B and an element b we write A_b to denote the random variable A conditioned on the event $B = b$. We write $\Delta(S)$ to denote the space of all distributions over the set S.

2.1 Information Theory

Definition 1. *The* entropy *of a random variable X, denoted by $H(X)$, is defined as $H(X) = \sum_x \Pr[X = x] \log(1/\Pr[X = x])$. The* conditional entropy *of X given Y, denoted by $H(X|Y)$, is* $\mathbf{E}_{\mathrm{y}}[\mathrm{H}(\mathrm{X}|\mathrm{Y} = \mathrm{y})]$.

Definition 2. *The* mutual information *between two random variables A, B, denoted $I(A;B)$, is defined to be the quantity $H(A) - H(A|B) = H(B) - H(B|A)$. The* conditional mutual information *$I(A;B|C)$ is $H(A|C) - H(A|BC)$.*

Fact 1 (Chain Rule). *Let A_1, A_2, B, C be random variables. Then $I(A_1A_2;B|C) = I(A_1;B|C) + I(A_2;B|A_1C)$.*

Definition 3. Kullback-Leibler Divergence *between probability distributions A and B is defined as $\mathbb{D}(A||B) = \sum_x A(x) \log \frac{A(x)}{B(x)}$.*

Fact 2. *For random variables A, B, and C we have $I(A;B|C) = \mathbb{E}_{b,c}(\mathbb{D}(A_{bc}||A_c))$.*

Fact 3. *Let X and Y be random variables. Then for any random variable Z we have $\mathbb{E}_x[\mathbb{D}(Y_x||Y)] \leq \mathbb{E}_x[\mathbb{D}(Y_x||Z)]$.*

Fact 4. *Let A, B, C, D be four random variables such that $I(B;D|AC) = 0$. Then $I(A;B|C) \geq I(A;B|CD)$*

Fact 5. *Let A, B, C, D be four random variables such that $I(A;C|BD) = 0$. Then $I(A;B|D) \geq I(A;C|D)$*

Definition 4. *The* statistical distance (total variation) *between random variables D and F taking values in a set $\mathcal{S}$ is defined as $|D-F| \stackrel{\text{def}}{=} \max_{\mathcal{T}\subseteq\mathcal{S}}(|\Pr[D \in \mathcal{T}] - \Pr[F \in \mathcal{T}]|) = \frac{1}{2}\sum_{s\in\mathcal{S}} |\Pr[D=s] - \Pr[F=s]|$.*

2.2 Communication Complexity

We use standard definitions of the two-party communication model that was introduced by Yao in [20]:

Definition 5. *The* distributional communication complexity *of $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ with respect to a distribution μ on $\mathcal{X} \times \mathcal{Y}$ and* error tolerance *$\epsilon > 0$ is the least cost of a deterministic protocol computing f with error probability at most ϵ when the inputs are sampled according to μ. It is denoted by $\mathcal{D}_\mu(f, \epsilon)$.*

Definition 6. *The* randomized communication complexity *of $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ with error tolerance $\epsilon > 0$, denoted by $R_\epsilon(f)$, is the least cost of a public-coin protocol computing f with error at most ϵ on* every *input.*

For a thorough treatment of pre-1997 results in communication complexity see an excellent monograph by Kushilevitz and Nisan [17].

2.3 Information + Communication: The Information Cost

We consider protocols with *both private and public randomness*. Let $\Pi(X,Y)$ (random variable) denote *the transcript* of the protocol π, i. e., the concatenation of the public randomness with all the messages sent during the execution of π on (X,Y). When $X = x, Y = y$, we write $\Pi(x,y)$. When (X,Y) or (x,y) are clear from the context, we shall omit them and simply write Π for the transcript.

The notion of *internal information cost* was implicit in [2] and was explicitly defined in [3] as follows:

Definition 7. *The* internal information cost *of a protocol π over inputs drawn from a distribution μ on $\mathcal{X} \times \mathcal{Y}$, is given by:*

$$\mathsf{IC}_\mu(\pi) := I(\Pi; X|Y) + I(\Pi; Y|X).$$

Intuitively, the information cost captures the amount of information the two parties learn about each others' inputs during communication. Note that the information cost of a protocol π depends on the prior distribution μ. Naturally, the information cost of a protocol over *any* distribution is a lower bound on the communication cost.

Lemma 1. *[8] For any distribution μ we have $\mathsf{IC}_\mu(\pi) \leq CC(\pi)$.*

Definition 8. *The* information complexity *of f with respect to distribution μ and error tolerance $\epsilon \geq 0$ is defined as*

$$\mathrm{IC}_\mu(f,\epsilon) = \inf_\pi \mathrm{IC}_\mu(\pi),$$

where the infimum ranges over all randomized protocols π solving f with error at most ϵ when inputs are sampled according to μ.

3 Information Complexity of Gap Hamming Distance

Given two strings $x, y \in \{0,1\}^n$, the hamming distance x and y is defined to be $HAM(x,y) = |\{i \mid x_i \neq y_i\}|$. In the Gap Hamming Distance (GHD) problem, Alice gets a string $x \in \{0,1\}^n$ and Bob gets a string $y \in \{0,1\}^n$. They are promised that either $HAM(x,y) \geq n/2 + \sqrt{n}$ or $HAM(x,y) \leq n/2 - \sqrt{n}$, and they have to find which is the case. We can define a general version $GHD_{n,t,g}$, where Alice and Bob have to determine if $HAM(x,y) \geq t + g$ or $HAM(x,y) \leq t - g$, but the parameters $t = n/2$ and $g = \sqrt{n}$ are the most natural as discussed in [11]. In a technical tour-de-force, it was proved in [11] that the randomized communication complexity of the Gap Hamming Distance problem is linear. Formally,

Theorem 3. *For all constants $\gamma > 0$, and $\epsilon \in [0, 1/2)$, $R_\epsilon(GHD_{n,n/2,\gamma\sqrt{n}}) \geq \Omega(n)$.*

One can extend the formulation of GHD beyond the promise-problem setting. This particularly makes sense in a distributional-complexity setting. In this setting, we allow f to take the value $\star$, which means that we don't care about the output. The error in this model is aggregated only over points on which the value of f is not $\star$. Chakrabarti and Regev [11] also prove a distributional version of the linear lower bound over the *uniform* distribution $\mathcal{U}$. Specifically, they prove

Theorem 4. *[11] There exists an absolute constant $\varepsilon > 0$ for which*

$$\mathcal{D}_{\mathcal{U}}(GHD_{n,n/2,\sqrt{n}}, \varepsilon) = \Omega(n).$$

Kerenidis et al. [15] proved that the information complexity of Gap Hamming Distance is also linear, at least with respect to some distribution. The proof of Kerenidis et al. relies on a reduction that shows that a large class of communication complexity lower bound techniques also translate into information complexity lower bounds – including the lower bound for GHD:

Theorem 5. *[15] There exists a distribution μ on $\{0,1\}^n \times \{0,1\}^n$ and an absolute constant $\varepsilon > 0$ such that*

$$\mathsf{IC}_{\mu}(GHD_{n,n/2,\sqrt{n}}, \varepsilon) = \Omega(n).$$

Interestingly, while this approach yields an analogue of Theorem 3 for information complexity, it does not seem to yield an analogue of the stronger Theorem 4.

We give an alternate proof of the linear information complexity lower bound for GHD using the self reducibility technique. Unlike the proof in [15] we do not need to dive into the details of the proof of the communication complexity lower bound for GHD. Rather, our starting point is Theorem 4, which we use as a black-box.

In fact, we will prove a slightly weaker lemma, with Theorem 1 following by a reduction. The reduction is conceptually very simple, though the technical details are somewhat lengthy. We refer the author to the full version of this paper for the complete details [6]. We prove the following:

Lemma 2. *There exists absolute constants $\varepsilon > 0$ and $\gamma > 0$ for which*

$$\mathsf{IC}_{\mathcal{U}}(GHD_{n,n/2,\gamma\sqrt{n}}, \varepsilon) = \Omega(n).$$

4 Proof of Theorem 1

4.1 Proof Idea

We use the self-reducibility argument. Assume that for some $\epsilon > 0$, $IC_{\mathcal{U}}(GHD_n, \epsilon) = o(n)$. Then using information = amortized communication, we can get a protocol τ that solves N copies of GHD_n with $o(nN)$ communication.

The heart of the argument is to use this to solve GHD_{nN} with $o(nN)$ *communication*, which is a contradiction. Say that Alice and Bob are given $x, y \in \{0,1\}^{nN}$ respectively. They sample $c \cdot nN$ random coordinates (for some constant c) and then divide these into cN blocks and run GHD_n on them all in parallel using $o(nN)$ communication. If $HAM(x,y) = nN/2 + \sqrt{nN}$, then the expected hamming distance of each block is $n/2 + \sqrt{n/N}$. Although the gain over $n/2$ is small, the hamming distance is still biased towards being $> n/2$. We will see that on each instance the protocol for GHD_n must gain an advantage of $\Omega(1/\sqrt{N})$ over random guessing. This in turn implies that cN copies suffice to get the correct answer with high probability.

4.2 Formal Proof of Lemma 2

Assume that for some ρ sufficiently small (to be specified later), $\mathsf{IC}_{\mathcal{U}}(GHD_{n,n/2,\sqrt{n}}, \rho) = o(n)$. Thus $\forall\ \alpha > 0$, for n sufficiently large $\mathsf{IC}_{\mathcal{U}}(GHD_{n,n/2,\sqrt{n}}, \rho) \leq \alpha n$. We will need the following theorem from [8]:

Theorem 6. *[8] Let $f : X \times Y \to \{0,1\}$ be a (possibly partial) function, let μ be any distribution on $X \times Y$, and let $I = \mathsf{IC}_{\mu}(f, \rho)$, then for each $\delta_1, \delta_2 > 0$, there is an $N = N(f, \rho, \mu, \delta_1, \delta_2)$ such that for each $n \geq N$, there is a protocol π_n for computing n instances of f over μ^n such that error on each copy is $\leq \rho$. The protocol has communication complexity $< nI(1+\delta_1)$. Moreover, if we let π be any protocol for computing f with information cost $\leq I(1+\delta_1/3)$ w.r.t. μ, then we can design π_n so that for each set of inputs, the statistical distance between the output of π_n and π^n is $< \delta_2$, where π^n denotes n independent executions of π.*

In other words, Theorem 6 allows us to take a low-information protocol for f and turn it into a low-communication protocol for (sufficiently) many copies of f.

Step 1: From GHD to a tiny advantage.
In the first step we show that a protocol for GHD over the uniform distribution has a small but detectable advantage in distinguishing inputs from two distributions that are very close to each other. Denote by μ_η the distribution where $X \in \{0,1\}^n$ is chosen uniformly, and Y is chosen so that $X_i \oplus Y_i \sim B_{1/2+\eta}$ is an i.i.d. Bernoulli random variable with bias η. Note that in this language the GHD problem is essentially about distinguishing $\mu_{-1/\sqrt{n}}$ from $\mu_{1/\sqrt{n}}$.

Lemma 3. *There exists absolute constants $\tau > 0$, $\gamma > 0$ and $\rho > 0$ with the following property. Suppose that for all n large enough there is a protocol π_n such that π_n solves $GHD_{n,n/2,\gamma\sqrt{n}}$ with error ρ w.r.t the uniform distribution. Then for all n large enough for all $\varepsilon < 1/n^2$ we have*

$$Pr_{(x,y)\sim\mu_\varepsilon}[\pi_n(x,y) = 1] - Pr_{(x,y)\sim\mu_0}[\pi_n(x,y) = 1] > \tau \cdot \varepsilon \cdot \sqrt{n}, \qquad (1)$$

and

$$Pr_{(x,y)\sim\mu_{-\varepsilon}}[\pi_n(x,y) = 0] - Pr_{(x,y)\sim\mu_0}[\pi_n(x,y) = 0] > \tau \cdot \varepsilon \cdot \sqrt{n}, \qquad (2)$$

Due to space constrains, we omit the proof of this Lemma. The proof can be found in the full version of this paper [6].

Step 2: From tiny advantage to low-communication GHD.
We can now apply Lemma 3 together with Theorem 6 to show that a low-information solution to $GHD_{n,n/2,\gamma\sqrt{n}}$ with respect to the uniform distribution contradicts the communication complexity lower bound of Theorem 4.

Proof. (of Lemma 2). Assume for the sake of contradiction that for each α there is an n and a protocol π_n with $\mathsf{IC}_{\mathcal{U}}(\pi_n) < \alpha n$ and which solves $GHD_{n,n/2,\gamma\sqrt{n}}$ with error ρ, where the parameters γ and ρ are from Lemma 3. Let $N > \max(n^7, N(GHD_{n,n/2,\gamma\sqrt{n}}, \rho, \mathcal{U}, \delta_1, \delta_2))$, where $\delta_1 = 1$ and $\delta_2 = \varepsilon/2$, where ε is the error parameter in Theorem 4. Then using Theorem 6, for each $c > 1$, cN copies of π_n can be executed with communication $< 2\alpha cn \cdot N$ (as long as the inputs to each π_n are distributed according to $\mathcal{U}$) such that on each copy the error is at most ρ w.r.t $\mathcal{U}$. Also for each set of inputs, the statistical distance between the output of the execution and $\pi_n^{cN} \leq \varepsilon/2$.

Let $t = Pr_{(x,y)\sim\mathcal{U}}[\pi_n(x,y) = 1]$. W.l.o.g. we assume $t = 1/2$. We solve $GHD_{n\cdot N, n\cdot N/2, \sqrt{n\cdot N}}$ over the uniform distribution with a small constant error ε using the protocol depicted in Figure 1.

Input: A pair $x, y \in \{0,1\}^{nN}$.
Output: $GHD_{n\cdot N, n\cdot N/2, \sqrt{n\cdot N}}$.

1. Create cN instances of GHD_n by sampling n random coordinates each time (with replacement): $(x_1, y_1), \ldots, (x_{cN}, y_{cN}) \in \{0,1\}^n \times \{0,1\}^n$.
2. Use compression (Theorem 6) to run $\pi_n(x_1, y_1), \ldots, \pi_n(x_{cN}, y_{cN})$ in communication $2\alpha cNn$.
3. Return $MAJORITY(\pi_n(x_1, y_1), \ldots, \pi_n(x_{cN}, y_{cN}))$.

Protocol 1: The protocol $\Pi_{nN}(x, y)$

The communication cost upper bound follows from the way the protocol $\Pi_{nN}(x, y)$ is constructed. To finish the proof we need to analyze its success probability. Suppose that the hamming distance between x and y is $nN/2 + \ell\sqrt{nN}$, where $\ell > 1$. Note that $\ell < n$ except with probability $e^{-\Omega(n^2)}$. The samples (x_i, y_i) are drawn iid according to the distribution $\mu_{\ell\cdot\sqrt{1/(nN)}}$. Since $N > n^7$ we have $\ell \cdot \sqrt{1/nN} < 1/n^2$. By Lemma 3, the output of π_n on each copy is thus $\tau \cdot \ell/\sqrt{N}$-biased towards 1. An application of the Chernoff bounds along with the fact that, for each set of inputs, the statistical distance between the output of the execution and $\pi_n^{cN} \leq \varepsilon/2$, implies that the probability that the protocol Π_{nN} outputs 1 is at least $1 - e^{-2\tau^2\ell^2 c} - \varepsilon/2$. For constant τ, we can make this expression as close to $1 - \varepsilon/2$ as we like by letting c be a sufficiently large constant. But this means that for an arbitrarily small constant $\alpha > 0$, $\Pi_{nN}(x, y)$ will

solve $GHD_{n\cdot N,n\cdot N/2,\sqrt{n\cdot N}}$ with error $\leq \varepsilon$ (the case when the hamming distance between x and y is $nN/2 - \ell\sqrt{nN}$ is symmetric) in communication $O(\alpha c Nn)$, which can be made arbitrarily small relatively to Nn, leading to a contradiction. Note that we got a randomized protocol for solving $GHD_{n\cdot N,n\cdot N/2,\sqrt{n\cdot N}}$ but we can fix the randomness to get a deterministic algorithm.

5 Information Complexity of Inner Product

The inner product function $IP_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is defined as follows:

$$IP_n(x,y) = \sum_{i=0}^{n} x_i y_i \pmod 2$$

The proof exploits the self-reducible structure of the inner-product function. But since, IP_n is such a sensitive function, we will first prove a statement about the 0-error information cost, and then use continuity of information cost to argue about non-zero errors.

We will need the following lemma from [8]. It is essentially the same as Theorem 6, just that when dealing with 0 error, we cannot ensure that error on each copy is 0. We just have an overall error which is the error introduced if compression fails.

Lemma 4. *Let $f : X \times Y \to \{0,1\}$ be a function, and let μ be a distribution over the inputs. Let π be a protocol computing f with error 0 w.r.t μ, and internal information cost $IC_\mu(\pi) = I$. Then for all $\delta > 0$, $\epsilon > 0$, there is a protocol π_n for computing f^n with error ϵ w.r.t μ^n, with worst case communication cost*

$$= n(I + \delta/4) + O(\sqrt{CC(\pi) \cdot n \cdot (I + \delta/4)}) + O(\log(1/\epsilon)) + O(CC(\pi))$$
$$\leq n(I + \delta) \text{ (for } n \text{ sufficiently large)}$$

The following lemma from [3] relates the information cost of computing XOR of n copies of a function f to the information cost of a single copy.

Lemma 5. *Let f be a function, and let μ be a distribution over the inputs. Then $IC_{\mu^n}(\oplus_n f, \epsilon) \geq n(IC_\mu(f,\epsilon) - 2)$*

The next lemma says that there is no 0-error protocol for IP_n which conveys slightly less information than the trivial protocol.

Lemma 6. *$\forall n, IC_{\mathcal{U}_n}(IP_n, 0) \geq n$, where $\mathcal{U}_n$ is the uniform distribution over $\{0,1\}^n \times \{0,1\}^n$*

Proof. It is known that $D^{\mathcal{U}_n}_\epsilon(IP_n) \geq n - c_\epsilon$, for all constant $\epsilon \in (0, 1/2)$, where c_ϵ is a constant depending just on ϵ [17,13]. Assume that for some n, $IC_{\mathcal{U}_n}(IP_n, 0) \leq n - c$. Then using, Lemma 4 with $\delta = c/2$ and $\epsilon = 1/3$, we can get a protocol π for solving N copies of IP_n with overall error 1/3 w.r.t $\mathcal{U}_n^N$, and $CC(\pi) \leq N(n - c + c/2)$. This gives us a protocol π' for solving IP_{Nn} with error 1/3 w.r.t the uniform distribution, and $CC(\pi') \leq Nn - Nc/2$ (divide the inputs into N chunks, solve the N chunks using π and XOR the answers). But $CC(\pi') \geq Nn - c_{1/3}$, a contradiction.

Proof. (of Theorem 2) We use the continuity of (internal) information cost in the error parameter at $\epsilon = 0$:

Theorem 7. *([5]) For all $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ and $\mu \in \Delta(\mathcal{X} \times \mathcal{Y})$ we have*

$$\lim_{\epsilon \to 0} \mathrm{IC}_\mu(f, \epsilon) = \mathrm{IC}_\mu(f, 0) \tag{3}$$

Given $\delta > 0$, let $l = \lceil \frac{3}{\delta} \rceil$. Then

$$IC_{\mathcal{U}_l}(IP_l, 0) \geq l \geq (1 - \delta)l + 3$$

Since $\lim_{\epsilon \to 0} IC_{\mathcal{U}_l}(IP_l, \epsilon) = IC_{\mathcal{U}_l}(IP_l, 0)$, $\exists\ \epsilon(l, \delta) = \epsilon(\delta)$ s.t.

$$IC_{\mathcal{U}_l}(IP_l, \epsilon) \geq (1 - \delta)l + 2$$

Now using Lemma 5, we get that $IC_{\mathcal{U}_l^N}(\oplus_N IP_l, \epsilon) \geq (1 - \delta)Nl$. Thus $IC_{\mathcal{U}_{Nl}}(IP_{Nl}, \epsilon) \geq (1-\delta)Nl$. Thus for sufficiently large n, $IC_{\mathcal{U}_n}(IP_n, \epsilon) \geq (1-\delta)n$.

References

1. Ada, A., Chattopadhyay, A., Cook, S., Fontes, L., Koucky, M., Pitassi, T.: The hardness of being private. In: 2012 IEEE 27th Annual Conference on Computational Complexity (CCC), pp. 192–202. IEEE (2012)
2. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. Journal of Computer and System Sciences 68(4), 702–732 (2004), http://dx.doi.org/10.1016/j.jcss.2003.11.006
3. Barak, B., Braverman, M., Chen, X., Rao, A.: How to compress interactive communication. In: STOC, pp. 67–76 (2010)
4. Braverman, M.: Interactive information complexity. In: STOC, pp. 505–524 (2012)
5. Braverman, M., Garg, A., Pankratov, D., Weinstein, O.: From information to exact communication. Electronic Colloquium on Computational Complexity (ECCC) 19(171) (2012)
6. Braverman, M., Garg, A., Pankratov, D., Weinstein, O.: Information lower bounds via self-reducibility. Electronic Colloquium on Computational Complexity (ECCC) 19, 177 (2012)
7. Braverman, M., Moitra, A.: An information complexity approach to extended formulations. Electronic Colloquium on Computational Complexity (ECCC) 19, 131 (2012)
8. Braverman, M., Rao, A.: Information equals amortized communication. CoRR abs/1106.3595 (2010)
9. Braverman, M., Weinstein, O.: A discrepancy lower bound for information complexity. Electronic Colloquium on Computational Complexity (ECCC) 18, 164 (2011)
10. Chakrabarti, A., Kondapally, R., Wang, Z.: Information complexity versus corruption and applications to orthogonality and gap-hamming. CoRR abs/1205.0968 (2012)
11. Chakrabarti, A., Regev, O.: An optimal lower bound on the communication complexity of gap-hamming-distance. In: STOC, pp. 51–60 (2011)

12. Chakrabarti, A., Shi, Y., Wirth, A., Yao, A.: Informational complexity and the direct sum problem for simultaneous message complexity. In: Werner, B. (ed.) Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, October 14-17, pp. 270–278. IEEE Computer Society, Los Alamitos (2001)
13. Chor, B., Goldreich, O.: Unbiased bits from sources of weak randomness and probabilistic communication complexity. SIAM Journal on Computing 17(2), 230–261 (1988)
14. Huffman, D.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE 40(9), 1098–1101 (1952)
15. Kerenidis, I., Laplante, S., Lerays, V., Roland, J., Xiao, D.: Lower bounds on information complexity via zero-communication protocols and applications. CoRR abs/1204.1505 (2012)
16. Klauck, H.: Quantum and approximate privacy. Theory Comput. Syst. 37(1), 221–246 (2004)
17. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, Cambridge (1997)
18. McGregor, A., Mironov, I., Pitassi, T., Reingold, O., Talwar, K., Vadhan, S.: The limits of two-party differential privacy. In: 51st Annual Symposium on Foundations of Computer Science (FOCS), pp. 81–90 (2010)
19. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 27 (1948), monograph B-1598
20. Yao, A.C.C.: Some complexity questions related to distributive computing (preliminary report). In: STOC, pp. 209–213 (1979)

On the Encoding Invariance of Polynomial Time Computable Distribution Ensembles

Anton Makhlin*

Lomonosov Moscow State University

Abstract. The notion of polynomial time invertibly samplable distributions (PISamp) was introduced by Vereshchagin in [9] as a tool to prove (NP,PSamp) completeness, which is more convenient than previously used polynomial time computable distributions (PComp). The notion of a PISamp distribution is encoding invariant, while PComp distributions are not, if one-way permutations exist. Here we prove that PComp distributions are not encoding invariant under a weaker assumption that one-way functions exist. This implies that the class of PISamp distributions is strictly larger than that of PComp distributions (under the same assumption).

1 Introduction

One of the key concepts in average-case complexity theory is a distributional decision problem, which consists of a regular decision problem P and an ensemble $\{\mu_0, \mu_1, \mu_2, \dots\}$ of probability distributions on its instances (represented by binary strings). A particular class of such ensembles which is often considered within the theory is the class of polynomial time computable distribution ensembles (PComp). These are the ensembles for which the cumulative probability of distribution μ_n over all strings preceding some given string x lexicographically can be computed by an algorithm running in time polynomial in n. This class was introduced by Levin in [7]. Levin has shown that there is a distributional problem in NP with a simple ("nearly uniform") instance distribution that is *complete* in the class of NP problems with polynomial time computable distributions on instances (i.e. the tractability of that problem implies tractability of all problems within the class).

NP problems with polynomial time computable distributions on instances were extensively studied in [1, 3–5, 7, 8]. Nice properties of PComp distributions were later used in [6] to show the existence of complete problems with a simple instance distribution in the larger and more natural class of polynomial time samplable ensembles (PSamp) introduced in [1]. However, as pointed out in [9] the concept of PComp distributions has the following drawback. Instances of most decision problems are not directly binary strings, but other objects

* The work was in part supported by the RFBR grant 12-01-00864 and the ANR grant ProjetANR-08-EMER-008.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 195–202, 2013.

(graphs, formulae etc.) encoded by binary strings in some way. Clearly the chosen encoding determines which distribution on binary strings will correspond to a fixed distribution on problem instances. The mentioned drawback is that the polynomial time computability of an ensemble of instance distributions may depend on this encoding, even if the considered encodings are equivalent in a natural sense — they may be transformed into each other using an efficient (say, polynomial time) algorithm.

Most definitions of classes in complexity theory are invariant under such encoding changes, for example a set of formulae which lies in P (NP, BPP) will still belong to this class if we chose a different polynomially equivalent encoding. It seems desirable to replace the definition of polynomial time computable ensembles with a class, preserving its nice properties, while achieving encoding invariance in the above sense. Such a class of ensembles, called PISamp was proposed and discussed in [9].

Still it is interesting to determine whether the proposed class PISamp differs from the class of polynomial time computable ensembles and if the latter is indeed not encoding invariant. It is impossible to answer any of these two questions affirmatively without proving that $\mathsf{P} \neq \mathsf{PSPACE}$ since a negative answer to both easily follows from $\mathsf{P} = \mathsf{PSPACE}$. Nevertheless we may settle these questions under some plausible assumption.

The definitions of these classes imply that $\mathsf{PComp} \subset \mathsf{PISamp} \subset \mathsf{PSamp}$. In [1, Theorem 8] it was shown that if one-way functions exist then PComp is different from PSamp in a strong sense: some ensemble in PSamp is not dominated by any ensemble in PComp (see the definition of domination below). Under the same assumption, [9] showed that even PISamp is different from PSamp (in the same strong sense). In [9], it was also shown that PComp differs from PISamp and that PComp is not encoding invariant (both in the strong sense). However, [9] used a stronger assumption of the existence of a one-way function with specific properties (the function is required to map no more than a polynomial number of inputs to any one output). Our contribution consists in proving a positive answer to both of the above questions under the assumption that any one-way function exists. That is, we prove that in this case the class of polynomial time computable ensembles is not encoding invariant (Theorem 3) and thus differs from the encoding invariant class PISamp proposed in [9] (Theorem 2), both statements hold in the strong sense.

2 Average-Case Complexity

In this section we remind some definitions from average-case complexity theory and also mention some related statements. These can be found in the survey paper [2], which contains an extensive overview of the theory. This theory studies the tractability of *distributional decision problems* — algorithmic decision problems paired with a probability distribution over its inputs. The following definition explains a possible way of specifying such a distribution and how the tractability of the corresponding distributional problem is determined.

Definition 1. *Consider a language of binary strings L together with an ensemble $\mu = \{\mu_0, \mu_1, \mu_2, \dots\}$ of distributions over strings. Let us call the distributional problem* (L, μ) tractable on average *if there exists a randomized algorithm A and polynomial p such that for any $k \in \mathbb{N}$ and all n and $x \in Supp(\mu_n)$, $A(n, x, k)$ runs in time $p(n, k)$ and*

$$\Pr_{x \sim \mu_n}[A(n, x, k) = L(x)] > 1 - \frac{1}{k} \ , \tag{1}$$

where the probability is also taken over the internal randomness of A.

In order to define a class of distributional problems one must choose a class of languages C and a class of input distribution ensembles D. The resulting class is denoted (C, D). One class of ensembles, which is often considered to capture the notion of a *natural* input distribution, is the class of polynomial time samplable ensembles.

Definition 2. *An ensemble $\{\mu_0, \mu_1, \mu_2, \dots\}$ is polynomial time samplable if there exists a randomized sampling algorithm G which on input n runs in time polynomial in n and the distribution of its outputs coincides with μ_n. The class of such ensembles is denoted* PSamp.

The class $(\mathsf{NP}, \mathsf{PSamp})$ is well studied. *Complete* problems within this class are of particular interest. To explain what completeness means in this case we define a possible version of reducibility between distributional problems. This is the definition of *strong* reductions, which is similar to Karp reductions in classical complexity theory.

Definition 3. *Ensemble $\{\mu_0, \mu_1, \mu_2, \dots\}$* dominates *ensemble $\{\nu_0, \nu_1, \nu_2, \dots\}$ if there exists a polynomial p for which*

$$\nu_n(x) \leq p(n)\mu_n(x) \tag{2}$$

holds for all n and x.

Definition 4. *A distributional problem (B, ν) reduces to a distributional problem (A, μ) if there exists a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ and a polynomial $q : \mathbb{N} \to \mathbb{N}$, such that:*

- *$x \in B \Leftrightarrow f(x) \in A$;*
- *the* pushforward measure *ν' defined as $\nu'_n(y) = \sum_{x:f(x)=y} \nu_n(x)$ is dominated by $\{\mu_{q(n)}\}$.*

In brief, the intuition behind this definition is that the probability of any instance must not become significantly lower after applying the reduction transformation f. It is not hard to see that any problem which reduces in this way to another problem which is tractable on average is itself tractable on average.

A well known result is the existence of a polynomial time samplable ensemble $\{\mu_0, \mu_1, \mu_2, \dots\}$ such that any other samplable ensemble is dominated

by $\{\mu_{q(0)}, \mu_{q(1)}, \mu_{q(2)}, \dots\}$ for some polynomial q ([7]). It is immediate that any NP complete problem paired with such a distribution will be complete for (NP, PSamp). However one is often more interested in determining the completeness of more practical distributional problems for which the input distribution is relatively simple. This led to the introduction in [7] of the class of ensembles PComp below defined, where it is shown that the class (NP, PComp) indeed contains a complete problem (P, μ) with a simple instance distribution μ. This result was then used in [6] to prove that the problem (P, μ) is complete even in a seemingly larger class (NP, PSamp) by demonstrating that any problem in this class reduces to a problem in (NP, PComp). Thus PComp was used as the main tool for proving this fundamental result of average-case complexity theory.

Definition 5. *For two binary strings x and y let $x \preceq y$ denote that x is not greater than y lexicographically (we consider shorter strings preceding longer ones). The ensemble $\{\mu_0, \mu_1, \mu_2, \dots\}$ is called* polynomial time computable *if there exists an algorithm A such that for all n $A(1^n, x)$ runs in time polynomial in n for any $x \in Supp(\mu_n)$ and*

$$A(1^n, x) = \sum_{y \preceq x} \mu_n(y) \ , \tag{3}$$

where the output of A is interpreted as some number from $[0, 1]$ in binary notation. The class of such ensembles is denoted PComp.

It follows from the definition that all instance probabilities across all distributions of a polynomial time computable ensemble are dyadic rationals. A more relaxed definition may be considered which permits any rational probabilities. However this definition would be equivalent to the one above in the sense of problem reducibility, since for any ensemble μ with rational probabilities an ensemble ν with dyadic rational probabilities exists such that μ and ν dominate one another ([9]).

It is not hard to see that PComp $\subset$ PSamp. Indeed, consider the unit segment divided into parts of length $\mu_n(x)$ ordered with respect to the lexicographical ordering of x. Then choose a dyadic rational $\alpha \in [0, 1]$ with as many digits as any probability $\mu_n(x)$ and select the string x which corresponds to the segment which contains α. The distribution of x chosen this way for uniformly chosen α will be μ_n.

3 Encoding Invariance and Polynomial Time Invertibly Samplable Ensembles

In this section we will present the results discussed in the introduction using the above definitions. First let us define the notion of *encoding invariance*. We will be dealing with classes of arbitrary mappings from $\{0, 1\}^*$. For instance, these may be mappings into $\{0, 1\}$, which define languages, or mappings into $\mathbb{R}^{\mathbb{N}}$, which define distribution ensembles. These mappings from binary strings

often correspond to mappings from some other set F of finite objects (such as graphs or formulae), the elements of which are represented by binary strings in order to formalize algorithmic treatment of such objects. This representation is implemented via an effectively computable (in an informal sense) injective mapping $e_1 : F \to \{0,1\}^*$ which is called an *encoding*. Suppose we now consider another such encoding e_2 and want to know whether these two encodings can be efficiently translated into each other. We formalize the latter by requiring that both $e_1 \circ e_2^{-1}$ and $e_2 \circ e_1^{-1}$ are polynomial time computable and call these encodings *polynomial time equivalent*. (Here we understand a partial function to be computable if there exists an algorithm which returns the corresponding value if the input lies within the functions domain and a special message "`undefined`" otherwise.) Then a class of mappings from binary strings is considered encoding invariant if it contains a mapping from F encoded with e_1 iff it contains this mapping encoded with a polynomial time equivalent e_2. This is stated in the following definition which disregards the initial set F and deals with the encodings via the *recoding* procedure h.

Definition 6. *Consider any class C of (partial) mappings defined on $\{0,1\}^*$ with values in some fixed set. Consider any $f \in C$ and (partial) injective function $h : \{0,1\}^* \to \{0,1\}^*$ such that*

- *h is defined on the domain of f;*
- *both h and h^{-1} are polynomial time computable.*

We will call C encoding invariant *if for any such f, h the function g which is defined on $\{h(x) \mid x \in \operatorname{dom} f\}$ by $g(y) = f(h^{-1}(y))$ is also in C. We will say that g is* obtained from f via the polynomially computable and invertible transformation h.

It is not hard to verify that the complexity classes P, BPP, NP as well the class of ensembles PSamp are all encoding invariant. However, the class PComp may not be encoding invariant since the lexicographical order of codes corresponding to different objects may alter when we change the encoding. Below we show that PComp is indeed not encoding invariant if a one-way function exists. Because of this in [9] a new class of ensembles is proposed as a replacement for PComp, which is encoding invariant by definition.

To define this new class we need to introduce the notion of the preimage of a sampling algorithm A for given n and x. The execution of our randomized algorithm A is determined by an infinite binary string containing the outcomes of coin flips which the algorithm performs in chronological order (since A always terminates, only a finite prefix of this string will be used). Such a string s can be identified with a real number on $[0,1]$ which is given by its binary expansion '$0.s$'. Let us denote the output of $A(n)$ with coin flip outcomes determined by $\alpha \in [0,1]$ via $A(n,\alpha)$. Now we can define

$$A^{-1}(n,x) = \{\alpha \in [0,1] \mid A(n,\alpha) = x\}.$$

Definition 7. *Consider a polynomial time samplable ensemble together with its sampling algorithm G. We call G invertible if for all n and x the set $G^{-1}(n, x)$ is either empty or consists of exactly one subsegment of $[0, 1]$ which can be computed, given n, x, in polynomial time in n (as specified by the values of its end-points in binary notation). We call an ensemble polynomial time invertibly samplable if it can be sampled by an invertible algorithm. The class of such ensembles is denoted* PISamp.

In [9], it is shown that $\mathsf{PComp} \subset \mathsf{PISamp}$. It follows, that $(\mathsf{NP}, \mathsf{PSamp})$ contains complete problems from $(\mathsf{NP}, \mathsf{PISamp})$, and the proof becomes substantially simpler than in the $(\mathsf{NP}, \mathsf{PComp})$ case. In [9], it is also shown that under the assumption that a "nearly injective" one-way function exists the inclusions $\mathsf{PISamp} \subset \mathsf{PSamp}$ and $\mathsf{PComp} \subset \mathsf{PISamp}$ are proper. Let us state this result precisely.

Definition 8. *A polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is called* one-way *if for any polynomial p and randomized algorithm A such that $A(1^n, y)$ runs in time polynomial in n and $|y|$*

$$\Pr[f(A(1^n, f(x))) = f(x)] < \frac{1}{p(n)} \ , \tag{4}$$

where the probability is taken over x uniformly distributed on $\{0,1\}^n$ and the internal randomness of A.

Theorem 1 ([9])

1. *If a one-way function exists, then some ensemble in* PSamp *is not dominated by any ensemble in* PISamp.
2. *If for some polynomial r there exists a one-way function f such that for all n any string has no more than $r(n)$ preimages in $\{0,1\}^n$ under f, then some ensemble in* PISamp *is not dominated by any ensemble in* PComp.

It turns out that the second statement may be improved: the existence of any one-way function is sufficient.

Theorem 2. *If a one-way function exists, then some ensemble in* PISamp *is not dominated by any ensemble in* PComp.

Proof. Let f be one-way. Define ensemble $\{\mu_n\}$ in the following way:

$$\mu_n(z) = \begin{cases} 2^{-n}, & \text{if } z = f(x)x,\ |x| = n, \\ 0, & \text{otherwise.} \end{cases}$$

Here "$f(x)x$" denotes the concatenation of two strings. We will now show that $\{\mu_n\}$ is an ensemble with the required properties.

Invertible samplability of $\{\mu_n\}$ is evident. Indeed, consider a sampling algorithm G which on input n asks for n random bits which form the string x

and outputs $f(x)x$. Then on input n, G only outputs strings of form $f(x)x$, $x \in \{0,1\}^n$, each one after receiving the random bits forming x.

Now suppose that $\{\mu_n\}$ is dominated by some polynomial time computable ensemble $\{\nu_n\}$: there exists a polynomial q such that for all n and z we have

$$\nu_n(z) \geq \mu_n(z)/q(n). \tag{5}$$

Below we will construct an algorithm which uses the computability of $\{\nu_n\}$ to find preimages under f and succeeds with probability $1/q(n)$.

Assume we are given some string y, $y = f(x_0)$, $|x_0| = n$. Our randomized inverting algorithm for f will output a string x with probability proportional to $\nu_n(yx)$ on input n, y. That is, string x is returned with probability $\nu_n^y(yx) = \nu_n(yx)/P_n(y)$, where $P_n(y) = \sum_{|x|=n} \nu_n(yx)$. Since we can compute $\nu_n(yx)$ as well as $P_n(y)$ in time polynomial in n using the computability of $\{\nu_n\}$, the ensemble $\{\nu_n^y\}$ is itself polynomial time computable and thus polynomial time samplable given y. Let us denote the algorithm which samples $\{\nu_n^y\}$ on input n, y via $A(1^n, y)$.

We now have to estimate the success probability $\Pr[f(A(1^n, f(x))) = f(x)]$ for x uniformly distributed on $\{0,1\}^n$. Let $\mathcal{I}_n(y)$ denote the set of all yx such that x is a preimage of length n of y under f, and let $I_n(y)$ denote the cardinality of this set. For a fixed y the probability that $f(A(1^n, y)) = y$ is $S_n(y) = \nu_n(\mathcal{I}_n(y))/P_n(y)$. From (5), it follows that

$$\nu_n(\mathcal{I}_n(y)) \geq \mu_n(\mathcal{I}_n(y))/q(n) \ , \tag{6}$$

where the right-hand side is equal to $I_n(y)/(2^n q(n))$. Now the success probability can be bounded in the following way:

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} \Pr[f(A(1^n, f(x))) = f(x)] = \frac{1}{2^n} \sum_{y \in f(\{0,1\}^n)} I_n(y) S_n(y) \geq$$

$$\geq \frac{1}{2^{2n} q(n)} \sum_{y \in f(\{0,1\}^n)} \frac{I_n^2(y)}{P_n(y)} \ . \tag{7}$$

Note that $\sum_{y \in f(\{0,1\}^n)} I_n(y) = 2^n$ and $\sum_{y \in f(\{0,1\}^n)} P_n(y) \leq 1$. It now follows from the simple technical lemma below that the right-most sum in (7) is at least 2^{2n} and the success probability is thus at least $1/q(n)$ which contradicts f being a one-way function.

Lemma 1. *For any set of positive numbers $a_1, \ldots, a_n, b_1, \ldots, b_n$ such that $\sum_{i=1}^n a_i = A$ and $\sum_{i=1}^n b_i = B$ the following holds:*

$$\sum_{i=1}^n \frac{a_i^2}{b_i} \geq \frac{A^2}{B}.$$

Proof. This can be proven by induction on n. For $n = 2$ we have:

$$\frac{a_1^2}{b_1} + \frac{a_2^2}{b_2} \geq \frac{(a_1 + a_2)^2}{b_1 + b_2} \Leftrightarrow a_1^2 b_2^2 + a_2^2 b_1^2 \geq 2a_1 a_2 b_1 b_2 \Leftrightarrow (a_1 b_2 - a_2 b_1)^2 \geq 0. \tag{8}$$

Now let $\sum_{i=1}^{n+1} a_i = A$, $\sum_{i=1}^{n+1} b_i = B$, and let the statement be proven for sets of smaller size. Then applying the inductive hypothesis twice we get

$$\sum_{i=1}^{n+1} \frac{a_i^2}{b_i} = \sum_{i=1}^{n} \frac{a_i^2}{b_i} + \frac{a_{n+1}^2}{b_{n+1}} \geq \frac{(A - a_{n+1})^2}{B - b_{n+1}} + \frac{a_{n+1}^2}{b_{n+1}} \geq \frac{A^2}{B} \quad . \tag{9}$$

□

From the proof of theorem 2 the following can be concluded:

Theorem 3. *If a one-way function exists, then the class* PComp *is not encoding invariant. Moreover, in this case there is an ensemble of distributions in* PComp *and a polynomial time computable and invertible function that transforms that ensemble into an ensemble which is not dominated by any ensemble in* PComp*.*

Proof. Let f be one-way, and let h map any string x to the string $f(x)x$. Without loss of generality we may assume that the length of $f(x)$ is $|x|^k$ for some constant $k > 0$ and all x. Therefore $f(x)x$ can be parsed into $f(x)$ and x and thus h is both polynomial time computable and invertible. Now consider the ensemble $\{\mu_n\}$ which consists of uniform distributions on strings of length n. If we change the binary encoding using h, the polynomial time computable ensemble $\{\mu_n\}$ will transform into an ensemble which is not dominated by any member of PComp unless f is polynomial time invertible. □

References

1. Ben-David, S., Chor, B., Goldreich, O., Luby, M.: On the Theory of Average Case Complexity. Journal of Computer and System Sciences 44(2), 193–219 (1992)
2. Bogdanov, A., Trevisan, L.: Average-Case Complexity. Foundations and Trends in Theoretical Computer Science 1(2), 1–106 (2006)
3. Gurevich, Y.: Complete and Incomplete Randomized NP Problems. In: Proceedings of 28th Annual Symposium on Foundations of Computer Science, pp. 111–117 (1987)
4. Gurevich, Y.: Average Case Completeness. Journal of Computer and System Sciences 42(3), 346–398 (1991)
5. Gurevich, Y., Shelah, S.: Expected computation time for Hamiltonian path problem. SIAM Journal on Computing 16(3), 486–502 (1987)
6. Impagliazzo, R., Levin, L.A.: No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In: Proceedings of 31st Annual Symposium on Foundations of Computer Science, pp. 812–821 (1990)
7. Levin, L.A.: Average Case Complete Problems. SIAM Journal on Computing 15(1), 285–286 (1986)
8. Venkatesan, R., Levin, L.: Random instances of a graph coloring problem are hard. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 217–222 (1988)
9. Vereshchagin, N.: An Encoding Invariant Version of Polynomial Time Computable Distributions. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 371–383. Springer, Heidelberg (2010)

Improving on Gutfreund, Shaltiel, and Ta-Shma's Paper "If NP Languages Are Hard on the Worst-Case, Then It Is Easy to Find Their Hard Instances"

Nikolay Vereshchagin*

Moscow State University
ver@mccme.ru

Abstract. Assume that NP $\not\subset$ BPP. Gutfreund, Shaltiel, and Ta-Shma in [Computational Complexity 16(4):412-441 (2007)] have proved that for every randomized polynomial time decision algorithm D for SAT there is a polynomial time samplable distribution such that D errs with probability at least $1/6 - \varepsilon$ on a random formula chosen with respect to that distribution. A challenging problem is to increase the error probability to the maximal possible $1/2 - \varepsilon$ (the random guessing has success probability $1/2$). In this paper, we make a small step towards this goal: we show how to increase the error probability to $1/3 - \varepsilon$.

1 Introduction

Suppose that NP is worst-case hard, say, NP $\not\subset$ BPP.[1] This means that every efficient algorithm D fails to solve SATISFIABILITY (SAT) correctly on an infinite sequence of instances. A natural question is the following: given such an algorithm D, how hard is it to generate such instances? I.e. given an input length n, what is the complexity of finding a formula of length at least n on which D errs. Clearly, by exhaustive search one can do that in exponential time (for infinitely many n). Surprisingly, Gutfreund, Shaltiel and Ta-Shma [6] showed that it can actually be done in probabilistic polynomial-time with a constant probability of success.

More specifically the result of [6] is the following. Let D be a probabilistic polynomial-time algorithm trying to decide SAT. We say that a distribution μ over Boolean formulas is δ-hard for D, if with probability at least δ, D fails to decide correctly whether a formula φ drawn from μ is satisfiable or not (where the probability is over the choice of φ and the randomness of D). A *sampler* is a polynomial-time probabilistic algorithm G that given 1^n as input outputs a

* The work was in part supported by the RFBR grant 09-01-00709 and the ANR grant ProjetANR-08-EMER-008.

[1] NP $\not\subset$ BPP means that there is no polynomial time randomized algorithm that given any Boolean formula with probability at least $2/3$ correctly decides whether it is satisfiable.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 203–211, 2013.

Boolean formula of length *at least* n. The result of [6] says that if NP $\not\subset$ BPP, then for every probabilistic polynomial-time algorithm D that tries to decide SAT there exists a sampler G such that for infinitely many n the probability distribution μ_n produced by $G(1^n)$ is δ-hard for SAT. Here $0 < \delta < 1/2$ is some universal constant.

The authors of [6] do not try to optimize δ and do not even carefully compute δ obtained in their proof. Instead they notice that δ derived from their proof is certainly less that $1/3$ and ask whether δ can be arbitrarily close to $1/2$. (Note that $1/2$ is the best one can hope for since an algorithm that decides according to an unbiased coin toss will always give a correct answer on every instance with probability $1/2$.) This question remains open.

In this paper, we will outline the proof of [6] and show that the proof yields the result for every $\delta < 1/6$. Then using an additional trick (Lemma 1) we show how to prove the result for every $\delta < 1/3$ (Theorem 3).

It turns out that the barrier of $1/3$ can be broken for Σ_k^p predicates for every $k > 1$. A result of [4] states that if Σ_k^p is not included in BPP then for every probabilistic polynomial time algorithm D there is a sampler G such that for infinitely many n, algorithm D errs on a random formula produced by $G(1^n)$ with probability close to $1/2$. As we said, for $k = 1$ (that is for NP), this is still open.

For motivation of the study of this question and for its history we refer the reader to an excellent introduction from [4].

2 Generating Hard Instances of Search Version of SAT

We start with presenting the main construction of [6] so that it be clear what our contribution is.

In this paper, we consider Boolean formulas in the basis $\neg, \vee, \wedge, 0, 1$. The length $|\varphi|$ of a formula φ is defined as the number of symbols in it: every variable is counted as one symbol.

Definition 1. *The* search version of SAT *is the following problem: given a Boolean formula φ find an assignment that satisfies it. A (randomized)* SAT solver *is a (randomized) polynomial time algorithm that for every input formula φ either finds its satisfying assignment, or says "don't know". A SAT solver D* errs *on ψ if ψ is satisfiable and $D(\psi) =$ "don't know".*

Theorem 1 ([6]). *Assume that NP $\neq$ P. Given a deterministic SAT solver S one can construct a deterministic polynomial time procedure that given 1^n produces a formula ψ_n of length at least n such that S errs on ψ_n for infinitely many n.*

Proof. Consider the following search problem in NP.

Search Problem P:
Instance: a string 1^n over the unary alphabet.

Solution: a pair (ψ, a) where ψ is a satisfiable formula of length n such that $S(\psi) =$"don't know", and a is its satisfying assignment.

We will call an instance 1^n of P *solvable* if such pair (ψ, a) exists. As SAT is NP complete, the search problem P reduces to the search version of SAT. This means that there is a polynomial time algorithm that given 1^n finds a formula, called φ_n, such that:

(1) if the instance 1^n of P is solvable then φ_n is satisfiable, and
(2) given any satisfying assignment of φ_n we can find (in polynomial time) a solution to the instance 1^n of problem P.

The length of φ_n is bounded by a polynomial n^d and w.l.o.g. we may assume that $|\varphi_n| \geq n$, since SAT is paddable.

The desired procedure works as follows: given 1^n, as input

(a) find the formula φ_n;
(b) run $S(\varphi_n)$;
(c) if $S(\varphi_n)=$"don't know"then output φ_n and halt;
(d) otherwise $S(\varphi_n)$ produces a satisfying assignment for φ_n; given that assignment, find in polynomial time a solution (ψ, a) to the instance 1^n of the problem P; output ψ and halt.

Since we assume that P $\neq$ NP, for infinitely many n the instance 1^n of P is solvable. For such n either $S(\varphi_n)=$"don't know"(and thus S errs on φ_n), or (ψ, a) is a solution to 1^n (and thus S errs on ψ).

The next construction of [6] allows one to generalize this theorem to randomized SAT solvers. This is done as follows. Let S be a randomized SAT solver working in time n^c and let r be string of length at least n^c. We will denote by S_r the algorithm S that uses bits of r as coin flips. Note that S_r is a deterministic algorithm.

Theorem 2 ([6]). *Assume that NP $\not\subset$ BPP. Then for some natural constant d the following holds. Let S be a randomized SAT solver and let n^c denote its running time on formulas of length n. Then there is a deterministic polynomial time procedure that given any binary string r of length $n^{c^2 d}$ produces a formula η_r of length between n and n^{cd}, where for any positive ε for infinitely many n the following holds. For a fraction at least $1 - \varepsilon$ of r's the algorithm S_r errs on η_r.*

Notice that the length of η_r is at most n^{cd}. Therefore the running time of S for input η_r is at most $n^{c^2 d}$. Hence $S_r(\eta_r)$ is well defined.

Proof. The proof is very similar to that of the previous theorem. The only change is that we have to replace the search problem P by the following problem P':
Instance: a binary string r' of length n^c (for some n).
Solution: a satisfiable formula ψ of length n and its satisfying assignment a such that $S_{r'}(\psi) =$"don't know".

Let $r' \mapsto \varphi_{r'}$ be a reduction of P' to the search version of SAT. The length of $\varphi_{r'}$ is bounded by a polynomial n^{cd} of $|r'| = n^c$ and w.l.o.g. we may assume that $|\varphi_{r'}| \geq n$.

The procedure required in the theorem, called **Procedure A**, works as follows: given r of length $n^{c^2 d}$, as input,

(a) let r' stand for the prefix of r of length n^c;
(b) find the formula $\varphi_{r'}$; recall that satisfying assignments of $\varphi_{r'}$ are basically pairs (a formula ψ of length n, its satisfying assignment a) such that $S_{r'}(\psi) =$"don't know";
(c) run $S_r(\varphi_{r'})$;
(d) if $S_r(\varphi_{r'})=$"don't know"then output $\varphi_{r'}$ and halt;
(e) otherwise $S_r(\varphi_{r'})$ produces a satisfying assignment for $\varphi_{r'}$; given that assignment find in polynomial time a solution (ψ, a) to the instance r' of the problem P'; output ψ and halt. (End of Procedure A.)

Let η_r stand for the formula output by the procedure. Since we assume that NP $\not\subset$ BPP, for every positive ε the randomized searching algorithm S errs with probability at least $1-\varepsilon$ for infinitely many input formulas. This implies that for infinitely many n the number of solvable instances r' of the problem P' is at least $(1-\varepsilon)2^{n^c}$. For those r's the formula $\varphi_{r'}$ is satisfiable. Therefore, for all but a fraction ε of r's the algorithm S_r errs on $\varphi_{r'}$ or $S_{r'}$ errs on ψ, which implies that S_r errs on ψ as well.

Remark 1. Theorem 2 holds for $\varepsilon = 1/n^k$ for any constant k. Indeed, the assumption NP $\not\subset$ BPP implies that the randomized searching algorithm S errs with probability at least $1-|\varphi|^{-k}$ for infinitely many input formulas φ.

3 Generating Hard Instances of the Decision Version of SAT

We start with defining samplers and samplable distributions. We will use the framework of Bogdanov and Trevisan [1] rather than the original Levin's one from [2].

Definition 2. *A sampler is a polynomial time probabilistic algorithm G that given 1^n as input outputs a Boolean formula of length* at least *n. If the length of the output formula is always exactly n, we call the sampler* proper. *Sequences $\mu_o, \mu_1, \mu_2, \ldots$ of distributions for which there is a polynomial time sampler are called* polynomial time samplable ensembles of distributions.

We say that a randomized decision algorithm D with randomness r *errs on a formula* φ if $D_r(\varphi) =$YES and φ is not satisfiable or vice verse.

Here is our result.

Theorem 3. *If NP $\not\subset$ BPP then for every probabilistic polynomial time decision algorithm D and every positive ε there is a sampler G such that for infinitely many n with probability at least $1-\varepsilon$ the decision algorithm D errs on the formula produced by $G(1^n)$ with probability at least $1/3-\varepsilon$.*

Remark 2. This result strengthens a result that is implicit in [6], which states the same with $1/6$ in place of $1/3$. In the proof we will explain what is the difference between the construction in [6] and ours. For proper samplers the constant $1/6$ should be reduced to $1/24$ by the following reason. Using padding we may assume that the formula output by the sampler has length either n, or n^{cd} (and not in between). Consider a new sampler $\tilde{G}$ that runs $G(1^n)$ and $G(1^{n^{1/cd}})$ and if either of the runs produces a formula of length n, then we output that formula (if both runs produce a formula of length n then we output each of them with probability $1/2$). This yields the constant $1/24 - \varepsilon$. Indeed, assume that $G(1^m)$ produces a formula φ such that $D(\varphi)$ errs with probability $1/6 - \varepsilon$. Then either the event "$D(\varphi)$ errs and the length of φ is m" or the event "$D(\varphi)$ errs and the length of φ is m^{cd}" has probability at least $1/12 - \varepsilon/2$. In the first case the probability of the event "D errs on the output of $G(1^m)$" is at least $1/24 - \varepsilon/4$. In the second case the probability of the event "D errs on the output of $G(1^{m^{cd}})$" is at least $1/24 - \varepsilon/4$.

Proof (of Theorem 3). Let D and ε be given. First we use the standard amplification, as in [7], to transform the algorithm D into another decision algorithm $\bar{D}$ with a smaller error probability.

Given a formula φ of length n as input the algorithm $\bar{D}$ invokes $D(\varphi)$ polynomial number K of times and outputs the most frequent result among all the results obtained in those runs. If K is large enough (but still polynomial in n) then the probability that the frequency of the result YES in those K runs differs from the probability that $D(\varphi) = \text{YES}$ by more than ε is exponentially small in n. This follows from the Chernoff bound. Note that the number of formulas of length n is also exponential in n. Moreover, we can choose $K = \text{poly}(n)$ so that with probability at least $1 - 2^{-n}$ there is no formula φ of length n for which the frequency of the result YES deviates from the probability that $D(\varphi) = \text{YES}$ by at most ε.

Using the standard binary search techniques we transform the algorithm $\bar{D}$ to a SAT solver S. That is, given a formula φ the algorithm S first runs $\bar{D}(\varphi)$. If the result is YES then it substitutes first $x = 0$ and then $x = 1$ for the first variable x in φ and runs $\bar{D}$ on the resulting formulas $\varphi_{x=0}$, $\varphi_{x=1}$. If at least one of these runs outputs YES, we replace φ by the corresponding formula and recurse. Otherwise we return "don't know"and halt.

If $\bar{D}$ returns NO for the input formula φ, we return "don't know"and halt. Finally, if we have substituted 0s and 1s for all variables and the resulting formula is true, we return the satisfying assignment we have found, and otherwise we return "don't know".

Let n^c be the upper bound of S's running time for input formulas of length n and let r be a string of length n^c used as randomness for S. In its run for input φ the algorithm S_r uses parts of r as coin flips for $\bar{D}$. With some abuse of notation we will denote by $\bar{D}_r$ the algorithm $\bar{D}$ with that randomness. The notation D_r is understood in the same way.

The heart of the construction is a procedure that given any formula ψ and randomness r such that S_r errs on ψ returns at most three formulas such that the algorithm D errs on at least one of those formulas with high probability.

Procedure B. Given a satisfiable input formula ψ and r such that $S_r(\psi)$ ="don't know", run $S_r(\psi)$ to find the place in the binary search tree where S_r is stuck. By the construction of S this may happen in the following three cases:

(1) $\bar{D}_r(\psi)$=NO. In this case output ψ.

(2) $S_r(\psi)$ performs binary search till the very end, it finds a formula η obtained from original formula by substituting all its variables by 0,1 such that η is false while $\bar{D}_r$ claims that η is true. In this case output η.

(3) In the remaining case $S_r(\psi)$ is stuck in the middle of the binary search and thus it has found a formula φ and its variable x such that $\bar{D}_r(\varphi)$ =YES while both $\bar{D}_r(\varphi_{x=0})$ and $\bar{D}_r(\varphi_{x=1})$ are NO. In this case return $\varphi, \varphi_{x=0}, \varphi_{x=1}$. (End of Procedure B.)

We will call formulas returned by this procedure by α, β, γ.[2] They depend on input formula ψ and on randomness r.

By Theorem 2 applied to the search algorithm S there is a polynomial procedure (called Procedure A in the proof) with the following property. Given a string r of length $n^{c^2 d}$ the procedure returns a formula η_r of length between n and n^{cd} such that for infinitely many n, S_r errs on η_r (except for a fraction at most ε of r's).

The sampler G from [6] works as follows. For input 1^n choose a random string r of length $n^{c^2 d}$. Then apply Procedure A to r to obtain η_r. Then apply Procedure B to S, r and η_r to obtain three formulas α, β, γ. Finally choose one of these formulas at random, each with probability 1/3, and output it.

Fix a positive ε. We claim that for infinitely many n with probability at least $1 - 2\varepsilon$ the algorithm D errs on the formula produced by $G(1^n)$ with probability at least $1/6 - \varepsilon$. To prove this claim notice that $S_r(\eta_r)$ calls $\bar{D}_r$ at most $2n^{cd}$ times (two times for each variable). Each time $\bar{D}_r$ is called on an input formula φ of length between n and n^{cd}. Call a string r of length $n^{c^2 d}$ *bad* if in at least one of these runs of $\bar{D}_r$ the frequency of YES answers of D for input φ differs from the probability of the event $D(\varphi)$=YES by more than ε (recall that $\bar{D}_r(\varphi)$ runs $D(\varphi)$ some K times). By construction of $\bar{D}$ a fraction at most

$$\sum_{l=n}^{n^{cd}} 2n^{cd} 2^{-l} < n^{cd} 2^{-n+2} \le \varepsilon$$

r's are bad (for all large enough n). If r is good and S_r errs on η_r then the error probability of D on the formula output by Procedure B is at least $1/3(1/2 - \varepsilon)$. And for infinitely many n the probability that S_r does not err on η_r is at most ε. Thus for infinitely many n with probability at least $1 - 2\varepsilon$ both S_r errs on

[2] Without loss of generality we may assume that Procedure B always outputs three formulas.

η_r and r is good hence D errs on the output formula with probability at least $1/3(1/2-\varepsilon)$.

Up to now we have just recited the arguments from [6]. Now we will present a new trick, which improves the constant $1/6$ to $1/3$. We will change the very last step in the work of this sampler. This time we will output α, β, γ with different probabilities, which are carefully chosen based on the frequencies of YES/NO answers of the algorithm D in the run of $\bar{D}_r$ on inputs α, β, γ.

Recall that Procedure B has run the algorithm $\bar{D}_r$ on inputs α, β, γ, and algorithm $\bar{D}_r$ has done majority vote among some number K of runs of the algorithm D on α, β, γ, respectively (using in each run a part of r as randomness for D). Let a, b, c be the answers obtained during those majority votes and let u_r, v_r, w_r stand for the frequencies of answers a, b, c of the runs of D with randomness from r on α, β, γ, respectively. All numbers u_r, v_r, w_r are thus at least $1/2$. For all good r's u_r is ε-close to the probability of the event $D(\alpha) = a$. Thus with probability at least $1-\varepsilon$ (over the choice of r) we know an ε-approximation to the probability of the event $D(\alpha) = a$. The same applies to b, c and β, γ, respectively. Thus it remains to prove the following lemma, which is essentially our contribution.

Lemma 1. *Assume that we are given bits $a, b, c \in \{YES, NO\}$ and the probabilities $u, v, w \geq 1/2$ of the events $D(\alpha) = a$, $D(\beta) = b$ and $D(\gamma) = c$, respectively. Assume further that at least one of the bits a, b, c is incorrect (we call a incorrect if α is satisfiable and $a = NO$ or the other way around, and similarly for b, c). Based on this information we can find in polynomial time a probability distribution over the set $\{\alpha, \beta, \gamma\}$ such that D errs on a random formula drawn from that distribution with probability at least 1/3.*

Proof. If $u < 2/3$, then consider the probability distribution concentrated on α. In this case the probabilities of both events $D(\alpha) =$NO, $D(\alpha) =$YES are greater than $1/3$. We argue similarly, if v or w is less than $2/3$.

Otherwise let probabilities of α, β, γ be equal to numbers p, q, s such that all the numbers

$$pu+q(1-v)+s(1-w), \quad p(1-u)+qv+s(1-w), \quad p(1-u)+q(1-v)+sw \quad (1)$$

are at least $1/3$. We will argue later that such non-negative rational number p, q, s that sum up to 1 exist. Distinguish now three cases.

Case 1: the bit a is wrong. Then the probability that D errs on α is equal to u. The probability that D errs on β is at least $1-v$ (indeed, if b is correct than the error probability is equal to $1-v$; otherwise it is equal to $v \geq 1-v$, as $v \geq 1/2$). The same holds for c and β. Thus the overall probability that D errs on the formula drawn from the constructed distribution is at least

$$pu + q(1-v) + s(1-w) \geq 1/3.$$

Case 2: the bit b is incorrect. In this case we argue in a similar way and use the assumption that the second number in (1) is at least $1/3$.

Case 3: the bit c is incorrect. In this case the statement follows from the assumption that the third number in (1) is at least $1/3$.

It remains to show that there are non-negative p, q, s such that $p+q+s=1$ and all the numbers in (1) are at least $1/3$. Note that for all non-negative p, q, s with $p+q+s=1$ the arithmetic mean of those numbers is equal to

$$\frac{1+p(1-u)+q(1-v)+s(1-w)}{3} \geq \frac{1}{3}.$$

Thus it suffices to show that there are non-negative p, q, s such that all the three numbers in (1) are equal (and thus the maximum equals to the arithmetical mean):

$$pu+q(1-v)+s(1-w) = p(1-u)+qv+s(1-w) = p(1-u)+q(1-v)+sw.$$

The first equality means that $p(2u-1) = q(2v-1)$ and the second one means that $q(2v-1) = r(2w-1)$. Thus all the three numbers are equal, if p, q, s are proportional to $1/(2u-1), 1/(2v-1), 1/(2w-1)$. As all u, v, w are bounded away from $1/2$ (we are assuming that these numbers are at least $2/3$), all these numbers are bounded by a constant. Thus we are able to find in polynomial time the desired p, q, s.

Actually, we only know ε-approximation to the probabilities of events $D(\alpha) = a$, $D(\beta) = b$ and $D(\gamma) = c$. However, from the proof of the lemma it is clear that, if we use ε-approximation in place of true values, the probability of error of D will decrease by at most ε. The last step of the algorithm $G(1^n)$ is thus the following: we apply Lemma 1 to the ε-approximations we have and sample the output formula with respect to the distribution from Lemma 1. If r is good and S_r errs on η_r then D errs on the the output formula with probability at least $1/3-\varepsilon$.

Take into account a fraction at most ε of bad r's and also a fraction at most ε of r's such that S_r does not err on η_r. We obtain that with probability at least $1-2\varepsilon$ the algorithm D errs on the formula produced by $G(1^n)$ with probability at least $1/3-\varepsilon$.

Remark 3. Theorem 3 remains true for $\varepsilon = 1/n^k$ for any constant k.

Remark 4. Say that NP $\not\subset$ BPP *everywhere* if there is a constant c such that for every randomized SAT solver S and all $n > 1$, S errs on a formula of length between n and n^c. (A randomized SAT solver S errs on a formula φ if $S(\varphi) =$"don't know"with probability more $1/2$.)

If instead of NP $\not\subset$ BPP we assume that NP $\not\subset$ BPP everywhere then all our result holds in a stronger form: the quantifier "for infinitely many n" may be replaced by the universal quantifier.

Theorem 4. *If NP $\not\subset$ BPP everywhere then for every probabilistic polynomial time decision algorithm D and every positive ε there is a sampler G such that* for all *n the decision algorithm D errs on $G(1^n)$ with probability at least $1/3-\varepsilon$.*

The proof of this theorem is entirely similar to that of Theorem 3 and thus we omit it. We only have to replace, in the definition of the search problem P, the requirement "the length of ψ is n" by the requirement "the length of ψ is between n and n^c" (and make a similar change in the definition of problem P'). The constructed sampler will work for almost all n, which is enough, as we can change its behavior for the remaining n's.

References

1. Bogdanov, A., Trevisan, L.: Average-Case Complexity. Foundations and Trends in Theoretical Computer Science 1(2), 1–106 (2006)
2. Leonid, A.: Levin, Average Case Complete Problems. SIAM J. Comput. 15(1), 285–286 (1986)
3. Ben-David, S., Chor, B., Luby, O.G.M.: On the Theory of Average Case Complexity. In: STOC, pp. 204–216 (1989)
4. Gutfreund, D.: Worst-Case Vs. Algorithmic Average-Case Complexity in the Polynomial-Time Hierarchy. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 386–397. Springer, Heidelberg (2006)
5. Bogdanov, A., Talwar, K., Wan, A.: Hard instances for satisfiability and quasi-one-way functions. In: Proceedings of Innovations in Computer Science (ICS 2009), pp. 290–300. Tsinghua University Press (2009)
6. Gutfreund, D., Shaltiel, R., Ta-Shma, A.: If NP Languages are Hard on the Worst-Case, Then it is Easy to Find Their Hard Instances. Computational Complexity (CC) 16(4), 412–441 (2007)
7. Adleman, L.M.: Two Theorems on Random Polynomial Time. In: FOCS 1978, pp. 75–83 (1978)

Amortized Communication Complexity of an Equality Predicate

Vladimir Nikishkin

Moscow Institute of Physics and Technology

Abstract. We study the communication complexity of the direct sum of independent copies of the equality predicate. We prove that the probabilistic communication complexity of this problem is equal to $O(N)$; the computational complexity of the proposed protocol is polynomial in the size of inputs. Our protocol improves the result achieved in 1991 by Feder et al. Our construction is based on two techniques: Nisan's pseudorandom generator (1992, Nisan) and Smith's string synchronization algorithm (2007, Smith).

1 Introduction

In this paper we study the amortized communication complexity of the equality predicate. We deal with the classic model of communication complexity with two participants (Alice and Bob), who want to compute some function of the data distributed between the participants. Alice and Bob can talk to each other via a communication channel. We measure the number of bits that must be transmitted between Alice and Bob to achieve the goal.

More specifically, let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a function of two arguments. We assume that Alice is given the value of x, Bob is given the value of y, and Alice and Bob communicate with each other to compute the value $f(x,y)$.

We use three standard models of communication complexity: deterministic communication protocols, randomized communication protocols with public random bits, and randomized communication protocols with private random bits, see Kushilevitz and Nisan's textbook [1]. We denote communication complexities for these three models by C_{det}, C^{ε}_{pub}, and C^{ε}_{priv}, respectively. In the randomized versions, the superscript denotes the error probability and may be omitted if unnecessary.

Further, let us denote by f^N the direct sum of N independent copies of the initial function f. More precisely, the two arguments of f are an N-tuple of values $(x_1, \ldots, x_N)$ and an N-tuple of values $(y_1, \ldots, y_N)$, and

$$f^N(x_1, \ldots, x_N, y_1, \ldots, y_N) = (f(x_1, y_1), \ldots, f(x_N, y_N)).$$

We assume that Alice is given all values of the x_i, and Bob is given all values of the y_i. Now Alice and Bob need to compute f^N, i.e., to get all the values $f(x_i, y_i)$ for $i = 1, \ldots, N$. It is natural to ask how the communication complexity

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 212–223, 2013.

of the problem f^N grows as N tends to infinity. The asymptotic behavior of this complexity is called the *amortized communication complexity* of f. More formally, the amortized communication complexity of f is defined as

$$\mathrm{AC}(f) = \limsup_{N\to\infty} \frac{C(f^N)}{N}.$$

This definition makes sense for each model of communication, i.e., the value of C in the definition above can be substituted with C_{det}, C^{ε}_{pub}, or C^{ε}_{priv} (or by any other version of communication complexity known in the literature).

The question of communication complexity for the direct sum of several independent copies of the same problem is very natural, and it has been studied extensively for different models of communication complexity. This kind of questions were first raised by Karchmer *et al.* in [14]. Later, Feder *et al.* proved in [11] the first nontrivial lower bound for the deterministic model: $AC(f) = \Omega(\sqrt{C_{det}(f)} - \log n)$. Since then, several interesting results were achieved, mostly for more specific models of communication complexity (see [15–17]). However, the case of the classical randomized communication complexity remains not well understood. We only know that the gap between randomized communication complexity of one single instance of a problem and the corresponding amortized complexity can be rather large. E.g., such a gap was proven in [11] for the *equality predicate*.

The equality predicate $\mathrm{EQ}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is defined as

$$\mathrm{EQ}_n(x,y) = \begin{cases} 1, \text{ if } x = y, \\ 0, \text{ if } x \neq y. \end{cases}$$

The communication complexity of the equality predicate is $C^{\varepsilon}_{priv} = \Theta(\log n)$ for any constant $\varepsilon > 0$, see [1]. On the other hand, the *amortized* randomized complexity of the equality predicate is only $O(1)$ with error probability $O(2^{-\sqrt{N}})$, [11]. In this paper we revisit the amortized communication complexity of the equality predicate. Our protocol achieves the same amortized communication complexity $O(1)$, and has a slightly better error probability $\varepsilon = O(2^{-\frac{N}{\log^2 N}})$. Besides, our protocol has "modular" structure; it consists of several independent gadgets, which makes the construction more flexible. So we hope that a similar technique can be applied to construct communication protocols with small amortized complexity for other functions.

Our construction is based on ideas similar to the protocol from [11]: (a) use random checksums to compare stings on Alice's and Bob's ends, (b) use a communication protocol with a public coin to compare Alice and Bob's checksums, and (c) use a pseudorandom generator to convert a public coin protocol to a private coin model. Still, the implementations of these ideas differ. The last idea (substitute public truly random bits by pseudorandom bits with a privately generated random seed) comes from the following classic theorem:

Theorem 1 ([1]). *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a function of two arguments. For every $\delta > 0$ and every $\varepsilon > 0$, it holds*

$$C^{\varepsilon+\delta}_{priv}(f) < C^{\varepsilon}_{pub}(f) + O(\log n + \log \delta^{-1}).$$

The proof of this theorem is not constructive, i.e., the pseudorandom generator used in the argument requires exponential (in the size of input) computations for Alice and Bob. The generators used in [11] and in our work run in polynomial (in n) time. However, the generator from [11] was created *ad hoc* for the EQ predicate problem, while the generator used in this paper is a well known general purpose generator, first described in [4].

Now let us formulate our main result.

Theorem 2 (the main result). *The randomized communication complexity (for the private coin model) of a direct sum of N equality predicates* EQ_n *is equal to $O(N)$ if $n < N$, with error probability $P_{err} \leq O(2^{-c\frac{N}{\log^2 N}})$. Moreover, we explicitly construct a communication protocol that achieves this communication complexity and requires only polynomial time computations on Alice's and Bob's sides.*

In our construction, we use several classic tools (N. Nisan's pseudorandom generator, BCH codes, deterministic synchronization protocol by A. Orlitsky, [3]) and one relatively new construction (probabilistic synchronization protocol by A. Smith, [3]).

2 Classic Communication Protocols for EQ

2.1 Complexity of EQ_n for Different Types of Communication Protocols

The predicate EQ_n is very well studied. Let us remind the three different communication protocols for this predicate.

Deterministic Model. It is known that $C_{det}(\mathrm{EQ}_n) = n + 1$, see [1]. The bound is achieved by a trivial protocol: Alice transmits her string x to Bob, Bob compares the two strings x and y and sends back one-bit response, 1 if the strings are equal and 0 otherwise. From the standard technique of fooling sets it follows that this bound is tight, i.e., there are no protocols with communication complexity less than $n + 1$.

Private Coin Model. For the randomized communication complexity with private sources of randomness $C^{\varepsilon}_{priv}(\mathrm{EQ}_n) = O(\log \frac{n}{\varepsilon})$. This bound is achieved by several classic communication protocols. Here we describe one of them. Alice and Bob view their inputs x and y as n-digit binary representations of integers (between 0 and $2^n - 1$). Alice chooses a prime number p at random among the first (n/ε) primes. She sends to Bob both p and $x \mod p$. Bob verifies whether $x \mod p = y \mod p$. If x and y are equal modulo p, then Bob returns 1, otherwise he returns 0. If $x = y$, then this protocol always returns the correct result. If $x \neq y$, then the difference $(x-y)$ has at most n prime factors; hence, the protocol returns the wrong answer with probability at most ε.

Public Coin Model. For randomized communication complexity with public source of randomness $C^{\varepsilon}_{pub}(\mathrm{EQ}_n) = O(\log \frac{1}{\varepsilon})$. This bound for communication complexity is achieved by the following protocol. Alice and Bob jointly choose a random n-bit string r. Then Alice computes the inner product $b = \langle x, r\rangle$ and transmits the result (a single bit) to Bob. Bob checks whether $b = \langle y, r\rangle$ and outputs "equal" if so and "not equal" otherwise. Obviously, if $x = y$, then the output is always "equal." On the other hand, if $x \neq y$, then by the properties of the inner product, $Pr[\langle x, r\rangle \neq \langle y, r\rangle] = \frac{1}{2}$. Thus, Bob outputs "not equal" with probability $\frac{1}{2}$. To decrease the probability of a wrong answer, Alice and Bob repeat these procedure several times with several independently chosen random strings r. If Alice and Bob repeat (in parallel or sequentially) l times the described procedure, then the probability that $\langle x, r_i\rangle \neq \langle y, r_i\rangle$ for all $r_1, \ldots, r_l$ is equal to 2^{-l}. So, for $l = \lceil \log 1/\varepsilon \rceil$ we reduce the probability of error to ε, while communication complexity is $O(\log 1/\varepsilon)$.

2.2 Trivial Generalizations for EQ^N_n

The protocols from the previous section can be easily adapted to get some protocols for the direct sum of N copies of EQ_n, i.e., for the function EQ^N_n.

Adaptation of the Protocol from Paragraph 2.1. We run the protocol independently for each pair of blocks(x_i, y_i). The probability of a wrong answer *for at least one pair of blocks* must be bounded by ε. To this end we need to reduce the probability of an error for each of the N pairs of blocks to be less than $\varepsilon' = \varepsilon/N$. This results in communication complexity $O(N(\log(n/\varepsilon'))) = O(N(\log n + \log N + \log 1/\varepsilon)))$. Thus, from the trivial adaptation of the protocol from paragraph 2.1 we get $C^{\varepsilon}_{priv}(\mathrm{EQ}^N) = O(N(\log n + \log N + \log 1/\varepsilon)))$.

Adaptation of the Protocol from Paragraph 2.1. We run the protocol from section 2.1 for each pair of blocks (x_i, y_i) independently. To guarantee that the total probability of error is bounded by ε, we need to reduce the probability of error for each pair of blocks to $\varepsilon' = \varepsilon/N$. Then we get

$$C^{\varepsilon}_{pub}(\mathrm{EQ}^N) = O(N(\log N + \log 1/\varepsilon)).$$

From Public to Private Randomness. The last protocol above can be transformed into a protocol with private source of randomness. Indeed, from Theorem 1 we get immediately

$$C_{priv} = O(N \cdot \log N + N \log \frac{1}{\epsilon} + \log(n \cdot N) + \log \frac{1}{\delta}) = O(N \cdot \log N + \log \frac{1}{\varepsilon}).$$

Note that this communication protocol requires exponential computational complexity (at least for the standard proof of Theorem 1).

How to reduce the obtained (rather trivial) bound $O(N \cdot \log N)$, hopefully to $O(N)$? How to achieve this bound with a communication protocol that requires

only poly-time computations? The construction of such a communication protocol is the main result of this paper. Loosely speaking, we plan to do it in two steps. In the first step, we construct a more effective communication protocol for the communication model with public randomness (this part of our construction is based on ideas of A. Smith). In the second step, we reduce the protocol with public randomness to a protocol with private randomness. In some sense, this idea is similar to the usual proof of Theorem 1: we substitute the sequence of random bits (shared by Alice and Bob) by a sequence of *pseudorandom* bits, which can be obtained as an output of a pseudorandom generator. A random seed of this generator is rather short. So, one of the participants can choose it at random and then send to another participant. E.g., Alice chooses a random seed and sends it to Bob; then Alice and Bob apply the pseudorandom generator to this same seed, and then both participants obtain the same long string of pseudorandom bits. The sharp difference between our construction and the standard general proof of Theorem 1 is that we use an explicit and effectively computable generator (the generator of N. Nisan).

Before explaining the details of our construction, we remind the technical tools used in our proof.

3 Pseudorandomness, Codes and String Synchronization

3.1 Pseudorandom Generator

In our construction we need a pseudorandom generator that fools tests with bounded memory. Technically, we assume that a generator is a mapping $G : \{0,1\}^m \to \{0,1\}^n$, and a test is a randomized Turing machine with working space of some size S. Technically, we define a test (for a pseudorandom generator) as follows.

Definition. A statistical test with memory S is a deterministic Turing machine M with three tapes: a finite working tape of size S, an auxiliary read-only tape with some binary string $a = (a_1, ...a_n, ...)$ (an advice string), and a one-way input tape with an n-bit input x (the reading head on the input tape can move from the left to the right but cannot move back to the left). We always assume that the length of a should not be greater than $exp(S)$. This machine returns 1(true) or 0(false). We denote machine's output by $M_a(x)$. Informally the output means that test accepts/rejects x given an advice string a.

Definition. A function $G : \{0,1\}^m \to \{0,1\}^n$ is called a pseudorandom generator, ε-robust for tests with memory $S(n)$, if for every *statistical test* A with S bits of working memory

$$|Pr_{y \in_r \{0,1\}^n}[A \text{ accepts } y] - Pr_{x \in_r \{0,1\}^m}[A \text{ accepts } G(x)]| < \varepsilon.$$

Here the notation $x \in_r X$ means x *chosen uniformly at random from* X. Note that this definition involves several parameters: n, m, S, ε. In general, these parameters can be chosen independnetly. But typically we use this definition when m, S, ε are some functions of n.

Nisan suggested in [4] an explicit construction of a pseudorandom generator that fools tests with sufficiently small memory:

Theorem 3 ([4]). *There exists a constant $c > 0$ such that for any R and S there exists a pseudorandom generator $G : cS \log R \to R$ (computable in time $poly(R)$) that is 2^{-S}-robust for all statistical tests with S bits of working memory.*

In Section 4.6 we construct some statistical test (with small working memory S) which is roughly equivalent to our communication protocol. Then, we use the standard argument: the protocol with high probability returns the correct answer when running on truly random public bits; futher, the generator of Nisan fools our test; hence, given pseudorandom bits instead of truly random ones, the communication protocol must also return the correct answer with high probability.

3.2 BCH Codes

Our construction involves implicitly the classic BCH-codes, see [12]. We do not employ any specific properties of the construction of the BCH codes. We use only the fact that $\forall m > 3$ and $t < 2^{m-1}$ there exists an explicit construction of a linear code with parameters $[n, k, d]$ such that the codeword length is $n = 2^m - 1$, the number of checksum bits is $n - k \leq mt$, and the minimal distance between codewords of the code is $d \geq 2t + 1$. We also use the fact that BCH codes can be decoded efficiently by the Berlekamp-Messy algorithm, [13].

The BCH codes are not used explicitly in our paper. However, we use a construction by Orlitsky from Section 3.3. This construction involves a linear error correcting code, which is not chosen explicitly in Section 3.3. In our applications, the BCH codes fits perfectly that construction. In what follows we refer to Orlitsky's protocol assuming that the codes used there are the BCH codes.

3.3 Strings Synchronization Protocols

In our communication protocol we will need to solve the following auxiliary problem. Let Alice and Bob each hold an n-bit string, A and B, respectively. We assume that A and B differ from each other in at most e positions. Alice and Bob want to exchange their inputs, i.e., Alice should get string B, and Bob should get string A. We will call this problem the *string synchronization problem* (Alice and Bob want to synchronize their inputs).

Orlitsky suggested in [3] a deterministic communication protocol for the problem of synchronization of a pair of n-bit strings with the Hamming distance from each other at most e. Communication complexity of this protocol is $O(e \log n)$. All computations of Alice and Bob in this protocol run in polynomial time in the length of the strings. More formally, the theorem (see [2]) is as follows:

Theorem 4. *Assume there exists a linear error-correcting code with parameters $(\alpha, R(\alpha))$, with a polynomial time decoding algorithms. Then there exists a one-round communication protocol solving the string synchronization problem with*

communication complexity $C = (1 - R(\alpha)) \cdot n$. *Computational complexity of this protocol is polynomial.*

If the BCH code is used (noted in section 3.2), the communication complexity of this protocol is $O(e \log n)$. The protocol of Orlitsky makes sense if the distance e between strings is very small. In case $e = \Omega(n)$, the communication complexity of Orlitsky's protocol is worse than the trivial upper bound $2n$.

The parameters of the BCH code correspond to the ones of the synchronization protocol code in the following way: $\alpha = d/n, R(\alpha) = n - k$

Adam Smith suggested in [9] a randomized communication protocol for the problem of strings synchronization with an asymptotically optimal bound for communication complexity for the case $e = const \cdot n$. More precisely, Smith proved that for every $\delta(n) = \Omega(\frac{\log \log n}{\sqrt{\log n}})$ there exists an explicit communication protocol (with a private source of randomness) that solves the problem of synchronization of n bit strings that differ in at most e positions, with communication complexity $n(H(\frac{e}{n}) + \delta)$ and error $\varepsilon = 2^{-\Omega(\frac{\delta^3 n}{\log n})}$, where $H(p) = p \log_2 \frac{1}{p} + (1 - p) \log \frac{1}{1-p}$. Algorithms of Alice and Bob in this protocol run in polynomial time.

4 Proof of the Main Theorem

In this section we present a protocol for EQ_n^N and prove Theorem 2.

4.1 Overview of the Protocol

Our protocol runs as follows. First of all, Alice generates a string of truly random bits of length $O(N)$ and sends this string to Bob. They both use Nisan's generator and produce pseudorandom bits from this seed. In what follows, Alice and Bob use this long string of pseudorandom bits.

Then, Alice and Bob iteratively calculate "checksums" (inner products mod 2 with the pseudorandom string) for their n-bit blocks and synchronize strings of the resulting checksums using the probabilistic or deterministic protocol from Section 3.3. As soon as some pair of non-equal blocks X^i, Y^i is revealed (if some checksums for these blocks are different), Alice and Bob remove these blocks from the list of their bit strings and never test them again. Thus, in every consecutive iteration the fraction of non-equal pairs of blocks (that are not discovered yet) becomes smaller and smaller.

in every consecutive iteration, we make the length of the checksums larger and larger, so for each pair of non-equal blocks the probability to be discovered becomes closer and closer to 1. Hence, the fraction of (non-discovered) pairs of non-equal blocks gradually reduces, and only pairs of equal blocks remain untouched at their places. This means that in every consecutive iteration the Hamming distance between arrays of checksums (obtained by Alice and Bob respectively) becomes smaller and smaller.

In each iteration Alice and Bob need to exchange the checksums computed for their blocks of bits (inner products with the same pseudorandom bits). For several initial iterations (technically, for $\log \log N$ iterations) we use the randomized synchronization protocol by Smith. Then we switch to the deterministic protocol by Orlitsky. In what follows we explain this protocol in more detail.

4.2 Generation Stage

Alice generates $r = \log((n \cdot N)^4) \cdot \log(2^{\frac{N}{\log(n \cdot N)}}) = O(N)$ random bits and sends them to Bob. Then Alice and Bob apply Nisan's pseudorandom generator from Section 3.1 and get $R = n^2 N^2$ pseudorandom bits. The length of the seed r is chosen so that the generator is ε-robust against tests with working memory of size $\frac{N}{\log(n \cdot N)}$.

4.3 Probabilistic Synchronization Stage (steps $i = 1, \ldots, \log \log N$)

The input of Alice is a sequence of N blocks $X = (X^1, \ldots, X^N)$, and the input of Bob is a sequence of N blocks $Y = (Y^1, \ldots, Y^N)$. Each block X^j or Y^j is an n-bits string.

We start the discussion with a minor technical difficulty. In our construction we employ the synchronization protocols by Orlitsky and Smith; these protocols need to know in advance the distance between the strings that should be synchronized. As we may not know the initial distance between X and Y, we will add N dummy equal blocks to both X and Y. This simple trick guarantees that in the very first stage of the protocol the fraction of equal pairs of blocks is no less than $1/2$. This trick increases the total number of blocks from N to $2N$, but it will not affect the asymptotic complexity of our protocol.

Now we explain the main part of the protocol. For $i = 1, \ldots, \log \log N$ we repeat the following procedure. We set λ (a constant to be fixed later). Alice and Bob computes for each of their blocks (for all X^j and Y^j) λ random checksums. One checksum for each block X^j or Y^j is the inner products modulo 2 between this block and a new portion of pseudorandom bits generated in the previous stage, e.g., for $X^j = x_1 \ldots x_n$ and $Y^j = y_1 \ldots y_n$ the checksums are the bits

$$x_1 r_1 + \ldots + x_n r_n \pmod 2 \text{ and } y_1 r_1 + \ldots + y_n r_n \pmod 2,$$

where $r_1 \ldots r_n$ is a block from the stream of pseudorandom bits, similar to the protocol in Section 2.1 (Alice and Bob share the same sequence of pseudorandom bits, so they can use the same bits $r_1 \ldots r_n$ in the checksums for both X^j and Y^j). Thus, the resulting string of checksums (for Alice and Bob) consists of $\lambda \cdot$ [number of blocks] bits. E.g., at the very first iteration it makes $\lambda \cdot 2N$ bits since Alice and Bob computes the checksums for all $2N$ blocks (N original blocks and N dummy blocks).

Then Alice and Bob exchange their checksums using the randomized string synchronization protocol by Smith. In the i-th step we run the synchronization protocol assuming that Alice's and Bob's checksums differ from each other in

a fraction at most 2^{-i} of blocks (this threshold for the number of different blocks should be given to the synchronization protocol). When the checksums are exchanged, Alice and Bob remove from their lists all blocks X^j and Y^j whose checksums are not identical. Note that for a pair of equal blocks X^j, Y^j, the random checksums are always equal. If blocks are not equal to each other, then the probability to get λ equal checksums is about $2^{-\lambda}$ (this probability is not *exactly* $2^{-\lambda}$ since Alice and Bob use pseudorandom bits $r_1 \dots r_n$ rather than truly random ones to compute the checksums).

Typically, on each step the number of non-discovered pairs of non-equal blocks X^j, Y^j is reduced by a factor of about $2^{-\lambda}$. We say that the i-th step of the described procedure *fails*, if in this stage Alice and Bob discover less than 50% of the remaining pairs of non-equal blocks X^j, Y^j. If at least one step fails, we cannot guarantee the correctness of the result of the protocol. If no step fails, then on the i-th step the arrays of checksums of Alice and Bob differ from each other in a fraction at most $1/2^i$ of all computed inner products.

The communication complexity of this stage of the protocol is the sum of communication complexities of runs of Smith's protocol at steps $i = 1, \dots, \log\log N$:

$$\sum_{i=1}^{\log\log N} (H(1/2^i) + \delta)\lambda N = O(N),$$

where $\delta = \frac{\log\log nN}{\sqrt{\log nN}}$. The last equation follows from the property of Smith synchronization protocol and from the asymptotic $H(\alpha) = \alpha\log(1-\alpha) + O(1)$ as α tends to 0.

4.4 Deterministic Synchronization Stage ($i = \log\log N + 1, \dots, \log N$)

In this stage we continue essentially the same procedure as in the prevoius stage. There are two important differences: now we use checksums of variable size λ_i, and Alice and Bob apply the deterministic protocol of Orlitsky (instead or the randomized protocol of Smith) to exchange their checksums.

At each step $i = \log\log N, \dots, \log N$ Alice and Bob compute $\lambda_i = \lceil \frac{2^i}{\log^2 N} \rceil$ checksums for each remaining block X^j and Y^j, respectively. Again, each checksum of a block is the inner product (mod 2) with a new portion of pseudorandom bits. Then Alice and Bob exchange the computed lists of checksums. Now they use the deterministic synchronization protocol by Orlitsky, see Section 3.3.

The communication complexity of the deterministic protocol is about $\log N$ times greater than the complexity of the protocol by Smith. But nevertheless we can use it since the Hamming distance between checksums is reasonably small. The communication complexity of this stage is

$$\sum_{i=\log\log N}^{\log N} \lceil \frac{N\lambda_i \log N}{2^i} \rceil \approx \sum_{i=\log\log N}^{\log N} \frac{\log N \cdot 2^i}{\log^2 N \cdot 2^i} \cdot N = O(N).$$

4.5 Summary

When the described stages are completed, we assume that Alice and Bob have discovered all pairs of non-equal blocks. All the remaining pairs X^j, Y^j (all pairs of blocks whose checksums at all steps of the protocol remain equal to each other) are considered equal.

4.6 Probability of Error

We need to estimate the probability of error in our protocol. For simplicity, let us assume first that instead of R pseudorandom bits Alice and Bob share R independent and uniformly distributed random bits (so, we temporarily switch to the model with a public source of randomness). Then stages 4.3 and 4.4 make sense, and we can estimate the probability of error of the protocol.

The protocol may return a wrong answer because of the following reasons: (1) the probabilistic synchronization protocol of Smith fails at some stage; (2) some of the steps $i = 1, \ldots, \log N$ fail since more than 50% of random checksums turn out to be equal for non-equal pairs of blocks X^j, Y^j. Let us estimate the probabilities of each of these two events.

Error in the Probabilistic Synchronization Protocol. Summing up the probabilities of error in Smith's synchronization in every step of our protocol we obtain (for some constant $c > 0$)

$$P(Err) = \sum_{i=1}^{\log\log N} O(2^{-(\frac{N}{\log N})}) \leq O(2^{-\frac{cN}{\log N}}).$$

Failure of Checksum Verification. A step $i = 1, \ldots, \log N$ fails if for more than half of (not discovered yet) pairs of non-equal blocks X^j, Y^j all the checksums turn out to be equal. We estimate the probability of this event using the Chernoff bound. We may assume that after the first $(i-1)$ steps there remain $N/2^i$ pairs of non-equal blocks.

For a pair of blocks X^j, Y^j that are not equal, the probability that their inner products with a random string $r_1 \ldots, r_n$ have the same parity, is equal to $1/2$. When we calculate λ independent checksums, the probability that all pairs of checksums for X^j and Y^j are equal to each other, is $1/2^\lambda$. We say that the whole step of the protocol fails if the event *all checksums are equal* happens for more than 50% of pairs of non equal X^j, Y^j.

We assumed that after $(i-1)$ steps of the protocol the number of remaining non equal pairs of blocks (X^j, Y^j) is $N_i = \frac{N}{2^i}$. For each of these pairs the probability *not to be discovered at step* i is $2^{-\lambda}$. We estimate the probability of failure, i.e., the probability that more than $N_i/2$ pairs remain not revealed. By the Chernoff bound this probability is not greater than $2^{N_i D(q,p)}$, where

$$D(q,p) = q\ln(\frac{q}{p}) + (1-q)\ln(\frac{1-q}{1-p}) \text{ for } q = \frac{1}{2}, p = \frac{1}{2^\lambda}.$$

In the first steps $i = 1, \ldots, \log\log N$ steps of the protocol $\lambda = const$ (a large enough constant), and later for $i = \log\log N + 1, \ldots, \log N$ we have $\lambda_i = \frac{2^i}{\log^2 N}$. Hence, the probability of failure $P(Err_i) = O(2^{-\frac{N}{\log^2 N}})$. Sum up the error probabilities for all steps of stages 2 and 3:

$$\sum_{i=1}^{\log N} O(2^{-\frac{N}{\log^2 N}}) \approx \log N \cdot O(2^{-\frac{N}{\log^2 N}}) = O(2^{-c\frac{N}{\log^2 N}})$$

for some $c > 0$.

Pseudorandom Generator. In this section we construct a statistical test (see the definition in Section 3.1) that simulates one step of our protocol. In a sense, this test verifies that (pseudo)random bits are "suitable" for our communication protocol: they do not cause a failure of the protocol in the i-th iteration. The "advice strings" of this statistical test contain a sequence of pairs of blocks X^j, Y^j from the inputs of Alice and Bob that have passed the checksum tests in the first $i - 1$ rounds. The input x is a string of (pseudo)random bits that should be accepted or rejected. The test rejects x (for a given advice string a), if our communication protocol fails in the i-th round with random bits x while Alice and Bob are given the blocks X^j, Y^j corresponding to the advice a.

The algorithm of the test is straightforward: it computes the checksums for X^j and Y^j as it is done by our communication protocol in the i-th round, with random bits x shared by Alice and Bob, and compares the corresponding checksums for Alice's and Bob's blocks. Note that the test does not simulate the synchronization procedure (the sub-protocols following the construction of Orlitsky and Smith).

The working space of our machine is $O(\frac{N}{\log^2 N})$. This is enough to simulate the computation of the checksums performed by our communication protocol. The test accepts x, if in the simulation at least 50% of non-equal pairs of blocks are successfully revealed, and rejects x otherwise. In other words, a teststring x is rejected if it causes a failure in the i-th round of the protocol.

Theorem 3.1 guarantees that Nisan's pseudorandom generator fools this test. Hence, for our protocol the probability of failure with pseudorandom bits is not much greater than the probability of failure for truly random bits. More precisely, the difference between the probabilities of failure for random and pseudorandom bits is at most

$$2^{-S(N)} = O(2^{-c\frac{N}{\log^2 N}}).$$

We sum up these values for all steps $i = 1, \ldots, \log N$ and get

$$P(Err) \leq \sum_{i=1}^{\log N} O(2^{-\frac{N}{\log^2 N}}) = O(2^{-c_3\frac{N}{\log^2 N}}).$$

Summary. The probability of error of our protocol consists of three parts: (1) the probability of error in Smith's protocol, (2) the probability of failure with

truly random checksums in some step $i = 1, \ldots, \log N$, and (3) the additional probability of failure caused by the difference between truly random and pseudorandom checksums. Therefore,

$$P(Err) = O(2^{-c_1(\frac{N}{\log N})}) + O(2^{-c_2 \frac{N}{\log^2 N}}) + O(2^{-c_3 \frac{N}{\log N}}) = O(2^{-C \frac{N}{\log^2 N}}).$$

This concludes the proof of the correctness of our communication protocol.

References

1. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge Univ. Press (1997)
2. Chuklin, A.: Effective protocols for low-distance file synchronization. arXiv:1102.4712 (2011)
3. Orlitsky, A.: Interactive communication of balanced distributions and of correlated files. SIAM Journal on Discrete Mathematics 6, 548–564 (1993)
4. Nisan, N.: Pseudorandom Generators for Spacebounded Computation. Combinatorica 12(4), 449–461 (1992)
5. Nisan, N., Widgerson, N.: Hardness vs. Randomness. Journal of Computer and System Sciences 49(2), 149–167 (1994)
6. Canetti, R., Goldreich, O.: Bounds on Tradeoffs between Randomness and Communication Complexity. Computational Complexity 3(2), 141–167 (1990)
7. Newman, L.: Private vs. Common Random Bits in Communication Complexity. Information Processing Letters 39(2), 67–71 (1991)
8. Impagliazzo, R., Nisan, N., Widgerson, A.: Pseudorandomness for Network Algorithms. In: Proc. of the 26th ACM Symposium on Theory of Computing, pp. 356–364 (1994)
9. Smith, A.: Scrambling Adversarial Errors Using Few Random Bits, Optimal Information Reconciliation, and Better Private Codes. In: Proc. of the 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 395–404 (2007)
10. Nisan, N., Zukerman, D.: Randomness is Linear in Space. 1993 Journal of Computer and System Sciences 52(1), 43–52 (1996)
11. Feder, T., Kushilevitz, E., Naor, M., Nisan, N.: Amortized Communication Complexity. SIAM J. Comput. 24(4), 736–750 (1991)
12. Bose, R.C.M., Ray-Chaudhuri, D.K.: On A Class of Error Correcting Binary Group Codes. Information and Control 3(1), 68–79 (1960)
13. Berlekamp, E.R.: Nonbinary BCH decoding. IEEE Transactions on in Information Theory 14(2), 242 (1967)
14. Karchmer, M., Raz, R., Wigderson, A.: On Proving Super-Logarithmic Depth Lower Bounds via the Direct Sum in Communication Complexity. In: Proc. of 6th IEEE Structure in Complexity Theory, pp. 299–304 (1991)
15. Parnafes, I., Raz, R., Wigderson, A.: Direct Product Results and the GCD Problem, in Old and New Communication Models. In: STOC 1997 Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, pp. 363–372 (1997)
16. Chakrabarti, A., Shi, Y., Wirth, A., Yao, A.: Informational Complexity and the Direct Sum Problem for Simultaneous Message Complexity. In: Proceedings of 42nd IEEE Symposium on Foundations of Computer Science (2001)
17. Sherstov, A.: Strong Direct Produce Theorems for Quantum Communication and Query Complexity. In: STOC 2011 Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, pp. 41–50 (2011)

On Coloring of Sparse Graphs

Alexandr Kostochka[1,2,⋆] and Matthew Yancey[1]

[1] University of Illinois at Urbana–Champaign, Urbana, IL 61801, USA
[2] Sobolev Institute of Mathematics, Novosibirsk 630090, Russia

Abstract. Graph coloring has numerous applications and is a well-known NP-complete problem. The goal of this paper is to survey recent results of the authors on coloring and improper coloring of sparse graphs and to point out some polynomial-time algorithms for coloring (not necessarily optimal) of graphs with bounded maximum average degree.

Mathematics Subject Classification: 05C15, 05C35

Keywords: graph coloring, k-critical graphs, improper coloring.

1 Introduction

A *proper k-coloring*, or simply *k-coloring*, of a graph $G = (V, E)$ is a function $f : V \to \{1, 2, \ldots, k\}$ such that for each $uv \in E$, $f(u) \neq f(v)$. A graph G is *k-colorable* if there exists a k-coloring of G. The *chromatic number*, $\chi(G)$, of a graph G is the smallest k such that G is k-colorable. The problem of graph coloring with few colors has long been associated with resource assignment. One of the often cited related problems is the problem of assigning radio frequencies to a network of radio towers. The problem is known to be difficult: Determining if a graph is k-colorable when $k \geq 3$ was in Karp's [11] original list of 21 NP-complete problems. Furthermore, it is an NP-complete problem to even color a graph G with $1000\chi(G)$ colors [43]. This situation leads to constructing efficient algorithms for approximate coloring of graphs in special classes and to studying extremal problems on colorings.

What is said, relates also to improper colorings. A *d-improper k-coloring* of a graph G is a vertex coloring of G using k colors such that the graph induced by every color class has maximum degree at most d. By definition, a proper coloring of a graph is exactly a 0-improper coloring. So, a d-improper coloring is a relaxation of a proper coloring. Havet and Sereni [9] describe applications of d-improper k-colorings to frequency assignment problems. But improper colorings are even more complicated than the ordinary coloring. While it is easy to check whether a given graph is 2-colorable, Corrěa, Havet, and Sereni [10] proved that even the problem of existence of a 1-improper 2-coloring in the class of planar graphs is NP-complete.

It is natural that a graph G with a high *average degree*, $2\frac{|E(G)|}{|V(G)|}$, is harder to color with few colors. So, a low average degree of a graph may indicate that it is

⋆ Research of this author is supported in part by NSF grant DMS-0965587 and by grants 12-01-00448 and 12-01-00631 of the Russian Foundation for Basic Research.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 224–234, 2013.

easier to color this graph. However, it could be that although the whole graph has low average degree, some its part is much denser and cannot be colored. A graph parameter controlling such anomalies is the *maximum average degree*, $mad(G) = \max_{H \subseteq G} 2\frac{|E(H)|}{|V(H)|}$. Kurek and Rucin'ski [36] called graphs with low maximum average degree *globally sparse*. In this paper, we describe some recent results of the authors (some of them are joint with O. Borodin and B. Lidický) on coloring and improper coloring of sparse graphs. These results imply polynomial-time algorithms for coloring globally sparse graphs with few colors.

One of the key notions in graph coloring is the one of critical graphs. A graph G is (d-improperly) k-*critical* if G is not (d-improperly) $(k-1)$-colorable, but every proper subgraph of G is (d-improperly) $(k-1)$-colorable. Critical graphs were first defined and used by Dirac [12–14] in 1951-52. A reason to study k-critical graphs is that every k-chromatic graph (i.e. graph with chromatic number k) contains a k-critical subgraph and k-critical graphs have more restricted structure. For example, k-critical graphs are 2-connected and $(k-1)$-edge-connected, which implies that every k-chromatic graph contains a 2-connected and $(k-1)$-edge-connected subgraph. The only 1-critical graph is K_1, and the only 2-critical graph is K_2. The only 3-critical graphs are the odd cycles. There are no k-critical graphs with $k+1$ vertices. For every $k \geq 4$ and every $n \geq k+2$, there exists a k-critical n-vertex graph.

One of the basic questions on k-critical graphs is: What is the minimum number $f_k(n)$ of edges in a k-critical graph with n vertices? This question was first asked by Dirac [16] in 1957 and then was reiterated by Gallai [22] in 1963, Ore [37] in 1967 and others [27, 28, 42]. More generally, we may ask about $f_{k,d}(n)$ — the minimum number of edges in a d-improperly k-critical graph with n vertices.

In this paper, we discuss results towards this problem and some applications of these results.

2 Gallai's Conjecture

Since the minimum degree of any k-critical graph is at least $k-1$,

$$f_k(n) \geq \frac{k-1}{2}n \tag{1}$$

for all $n \geq k$, $n \neq k+1$. Equality is achieved for $n = k$ and for $k = 3$ and n odd. Brooks' Theorem [11] implies that for $k \geq 4$ and $n \geq k+2$, the inequality in (1) is strict. In 1957, in order to to evaluate chromatic number of graphs embedded into fixed surfaces, Dirac [16] proved that for $k \geq 4$ and $n \geq k+2$,

$$f_k(n) \geq \frac{k-1}{2}n + \frac{k-3}{2}. \tag{2}$$

The bound is tight for $n = 2k-1$ and yields $f_k(2k-1) = k^2 - k - 1$. Later, Kostochka and Stiebitz [30] improved (2) to

$$f_k(n) \geq \frac{k-1}{2}n + k - 3 \tag{3}$$

when $n \neq 2k-1, k$. This yields $f_k(2k) = k^2 - 3$ and $f_k(3k-2) = \frac{3k(k-1)}{2} - 2$.

Gallai [22] has found the values of $f_k(n)$ for $n \le 2k-1$ and proved the following general bound for $k \ge 4$ and $n \ge k+2$:

$$f_k(n) \ge \frac{k-1}{2}n + \frac{k-3}{2(k^2-3)}n. \tag{4}$$

For large n, the bound is much stronger than bounds (2) and (3). Based on his description of k-critical graphs with only one vertex of degree greater than $k-1$, Gallai [21] conjectured the following.

Conjecture 1 (Gallai [21]). *If $k \ge 4$ and $n = 1(\mod k-1)$, then*

$$f_k(n) = \frac{(k^2-k-2)n - k(k-3)}{2(k-1)}.$$

Ore [37] observed that Hajós' construction implies

$$f_k(n+k-1) \le f_k(n) + \frac{(k-2)(k+1)}{2} = f_k(n) + (k-1)(k - \frac{2}{k-1})/2, \tag{5}$$

which yields that the limit $\phi_k := \lim_{n\to\infty} \frac{f_k(n)}{n}$ exists and satisfies

$$\phi_k \le \frac{k}{2} - \frac{1}{k-1}. \tag{6}$$

Ore conjectured that his equation (5) is actually an equality:

Conjecture 2 (Ore [37]). *If $k \ge 4$, and $n \ge k+2$, then*

$$f_k(n+k-1) = f_k(n) + (k-1)(k - \frac{2}{k-1})/2.$$

Note that Conjecture 1 is equivalent to Conjecture 2 for $n = 1(\mod k-1)$. Some lower bounds on $f_k(n)$ were obtained in [16, 35, 21, 30, 31, 20]. Recently, the authors [32] proved Conjecture 1 in full.

Theorem 3 ([32]). *If $k \ge 4$ and G is k-critical, then $|E(G)| \ge \left\lceil \frac{(k+1)(k-2)|V(G)|-k(k-3)}{2(k-1)} \right\rceil$. In other words, if $k \ge 4$ and $n \ge k$, $n \ne k+1$, then*

$$f_k(n) \ge F(k,n) := \left\lceil \frac{(k+1)(k-2)n - k(k-3)}{2(k-1)} \right\rceil.$$

The result also confirms Conjecture 2 in the following cases: (a) $k = 4$ and every $n \ge 6$, (b) $k = 5$ and $n \equiv 2(\mod 4)$, and (c) every $k \ge 5$ and $n \equiv 1(\mod k-1)$. By examining known values of $f_k(n)$ when $n \le 2k$, it follows that $f_k(n) - F(k,n) \le k^2/8$.

It is known that there are infinitely many k-*extremal* graphs, i.e. the k-critical graphs G such that $|E(G)| = \frac{(k+1)(k-2)|V(G)|-k(k-3)}{2(k-1)}$. In particular, every graph in the family of so called k-*Ore graphs* is k-extremal. Very recently, the authors managed to describe all k-extremal graphs.

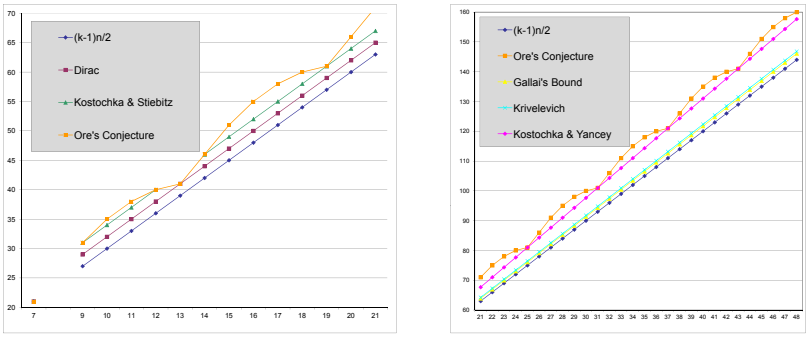


Fig. 1. Comparison of the bounds when $k = 7$

Theorem 4. *Let $k \geq 4$ and G be a k-critical graph. Then G is k-extremal if and only if it is a k-Ore graph. Moreover, if G is not a k-Ore graph, then $|E(G)| \geq \frac{(k^2-k-2)|V(G)|-y_k}{2(k-1)}$, where $y_k = \max\{2k-6, k^2-5k+2\}$.*

The message of Theorem 4 is that although for every $k \geq 4$ there are infinitely many k-extremal graphs, they all have a simple structure. In particular, every k-extremal graph distinct from K_k has a separating set of size 2. The theorem gives a slightly better approximation for $f_k(n)$ and adds new cases where we now know the exact values of $f_k(n)$: They are now known for (a) $k \in \{4, 5\}$ and every $n \geq k+2$, (b) all $k \geq 6$ and $n \equiv 1(\mod k-1)$, (c) $k = 6$ and $n \equiv 0, 2(\mod 5)$, and (d) $k = 7$ and $n \equiv 2(\mod 6)$.

This value of y_k in Theorem 4 is best possible in the sense that for every $k \geq 4$, there exists an infinite family of 3-connected k-critical graphs with $|E(G)| = \frac{(k^2-k-2)|V(G)|-y_k}{2(k-1)}$. By (5), if we construct an n_0-vertex k-critical graph for which our lower bound on $f_k(n_0)$ is exact, then the bound on $f_k(n)$ is exact for every n of the form $n_0 + s(k-1)$. So, we only need to construct

- a 4-critical 6-vertex graph with $\lceil 9\frac{2}{3} \rceil = 10$ edges,
- a 4-critical 8-vertex graph with $\lceil 13 \rceil = 13$ edges,
- a 5-critical 10-vertex graph with $\lceil 22 \rceil = 22$ edges,
- a 5-critical 7-vertex graph with $\lceil 15\frac{1}{4} \rceil = 16$ edges,
- a 5-critical 8-vertex graph with $\lceil 17\frac{3}{4} \rceil = 18$ edges,
- a 6-critical 10-vertex graph with $\lceil 27\frac{1}{5} \rceil = 28$ edges,
- a 6-critical 12-vertex graph with $\lceil 32\frac{4}{5} \rceil = 33$ edges, and
- a 7-critical 14-vertex graph with $\lceil 45\frac{1}{3} \rceil = 46$ edges.

These graphs are presented in Figure 2.

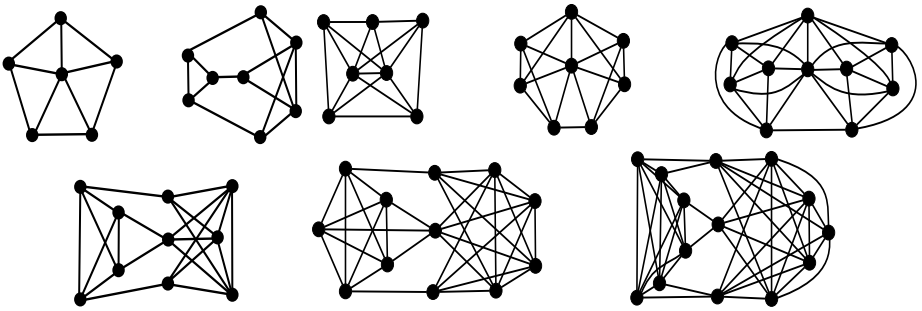


Fig. 2. Minimal k-critical graphs

3 Applications

3.1 Algorithms

Since the proof of Theorem 3 is constructive, it yields the following algorithmic counterpart.

Theorem 5. *If $k \geq 4$, then every n-vertex graph G with $|E(H)| < \frac{(k+1)(k-2)|V(H)|-k(k-3)}{2(k-1)}$ for each $H \subseteq G$ can be $(k-1)$-colored in $O(k^{3.5}n^{6.5}\log(n))$ time.*

The bound on complexity is not best possible. In particular, the algorithm finds maximum flows in auxiliary networks many times, so if one uses the Orlin's bounds on the running time of max-flow problems, the bounds will be improved.

Since every k-Ore graph distinct from the complete graph K_k contains a separating set of size 2 and decomposes into two k-Ore graphs of a smaller order, there is a simple algorithm that in time $O(n^5)$ checks whether a given n-vertex graph is a k-Ore graph. Together with an analog of the algorithm in Theorem 5 that uses the proof of Theorem 4 instead of Theorem 3, this yields the following

Theorem 6. *Let $k \geq 4$ and $y_k = \max\{2k-6, k^2-5k+2\}$. Then there exists a polynomial-time algorithm that for every n-vertex graph G with $|E(H)| < \frac{(k+1)(k-2)|V(H)|-y_k}{2(k-1)}$ for each $H \subseteq G$ either finds a $(k-1)$-coloring of G or finds a subgraph of G that is a k-Ore graph.*

3.2 Local and Global Graph Properties

Krivelevich [35] presented nice applications of his lower bounds on $f_k(n)$ and related graph parameters to finding complex graphs whose small subgraphs are simple. Two his bounds can be improved using Theorem 3 as follows.

Let $f(\sqrt{n}, 3, n)$ denote the maximum chromatic number over n-vertex graphs in which every $\sqrt{n}$-vertex subgraph is 3-colorable. Krivelevich proved that for every fixed $\epsilon > 0$ and sufficiently large n,

$$f(\sqrt{n}, 3, n) \geq n^{6/31-\epsilon}. \tag{7}$$

To show this, he applied his result that every 4-critical t-vertex graph with odd girth at least 7 has at least $31t/19$ edges. If instead of this, we simply use our bound on $f_4(n)$, then repeating almost word by word Krivelevich's proof of his Theorem 4 (choosing $p = n^{-0.8-\epsilon'}$), we get that for every fixed ϵ and sufficiently large n,

$$f(\sqrt{n}, 3, n) \geq n^{1/5-\epsilon}. \tag{8}$$

Another result of Krivelevich is:

Theorem 7 ([35]). *There exists $C > 0$ such that for every $s \geq 5$ there exists a graph G_s with at least $C\left(\frac{s}{\ln s}\right)^{\frac{33}{14}}$ vertices and independence number less than s such that the independence number of each 20-vertex subgraph at least 5.*

He used the fact that for every $m \leq 20$ and every m-vertex 5-critical graph H,

$$\frac{|E(H)| - 1}{m - 2} \geq \frac{\lceil 17m/8 \rceil - 1}{m - 2} \geq \frac{33}{14}.$$

From Theorem 3 we instead get

$$\frac{|E(H)| - 1}{m - 2} \geq \frac{\left\lceil \frac{9m-5}{4} \right\rceil - 1}{m - 2} \geq \frac{43}{18}.$$

Then repeating the argument in [35] we can replace $\frac{33}{14}$ in the statement of Theorem 7 with $\frac{43}{18}$.

3.3 Coloring Planar Graphs

Since planar graphs are sparse, Theorem 3 helps proving results on 3-coloring of planar graphs with restrictions on the structure. The case $k = 4$ of Theorem 3 is as follows:

Theorem 8. *If G is a 4-critical n-vertex graph then $|E(G)| \geq \frac{5n-2}{3}$.*

In [33], the authors gave a 3-page proof of Theorem 8. And this allows to give a half-page proof of the classical Grötzsch's Theorem [24] that *all triangle-free planar graphs are 3-colorable.* The original proof of Grötzsch's Theorem is somewhat sophisticated. The subsequent simpler proofs (see, e.g. [39] and references therein) are still not too simple. The proof below from [33] is the shortest so far:

Proof of Grötzsch's Theorem: Let G be a plane graph with the smallest $|E(G)| + |V(G)|$ for which the theorem does not hold. Then G is 4-critical. Suppose G has n vertices, e edges and f faces.

CASE 1: G has no 4-faces. Then $5f \leq 2e$ and so $f \leq 2e/5$. From this and Euler's Formula $n - e + f = 2$, we obtain $n - 3e/5 \geq 2$, i.e., $e \leq \frac{5n-10}{3}$, a contradiction to Theorem 8.

CASE 2: G has a 4-face (x, y, z, u). Since G has no triangles, $xz, yu \notin E(G)$. If the graph G_{xz} obtained from G by gluing x with z has no triangles, then by the minimality of G, G_{xz} is 3-colorable, and so G also is 3-colorable. Thus G has an x, z-path (x, v, w, z) of length 3. Since G itself has no triangles, $\{y, u\} \cap \{v, w\} = \emptyset$ and there are no edges between $\{y, u\}$ and $\{v, w\}$. But then G has no y, u-path of length 3, since such a path must cross the path (x, v, w, z). Thus the graph G_{yu} obtained from G by gluing y with u has no triangles, and so, by the minimality of G, is 3-colorable. Then G also is 3-colorable, a contradiction. □

Borodin, Lidický and the authors [8] used Theorem 8 to present simple proofs for some 3-coloring results on planar graphs, in particular, for the Grünbaum-Axenov Theorem that every planar graph with at most 3 triangles is 3-colorable. This theorem is sharp in the sense that there are infinitely many plane 4-critical graphs with exactly four triangles. Moreover, Thomas and Walls [38] constructed infinitely many such graphs without 4-faces.

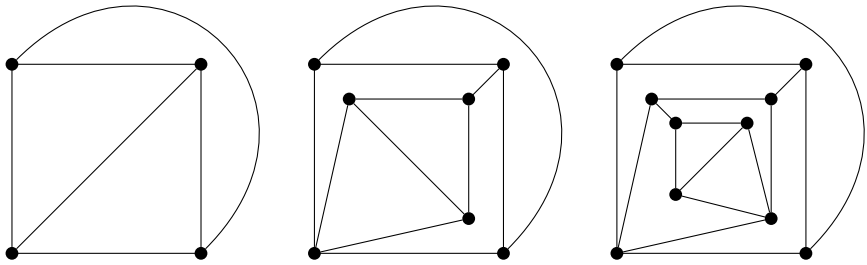


Fig. 3. Some 4-critical graphs from the family described by Thomas and Walls [38]

Very recently, Borodin, Lidický and the authors using Theorem 4 described all plane 4-critical graphs with exactly four triangles and no 4-faces. Using Theorem 8 they also proved:

Theorem 9. *Let G be a triangle-free planar graph and H be a graph such that $G = H - v$ for some vertex v of degree at most 4. Then H is 3-colorable.*

The theorem sharpens the similar result by Jensen and Thomassen [26] where the degree of v was at most 3 and is sharp in the sense that there are infinitely many plane triangle-free graphs that are obtained from a 4-critical graph by deleting a vertex of degree 5.

4 Improper 2-Colorings

As it was mentioned in the introduction, even the problem whether a given planar graph is 1-improperly 2-colorable is NP-complete. This motivates the study of improper 2-colorings for globally sparse graphs. A (j,k)-*coloring* of a graph G is a 2-coloring of $V(G)$ such that every vertex of Color 1 has at most j neighbors of Color 1 and every vertex of Color 2 has at most k neighbors of its color. By definition, a (d,d)-coloring is simply a d-improper 2-coloring. Esperet, Montassier, Ochem and Pinlou [19] proved that for every $j \geq 0$ and $k \geq 1$, the problem of verifying whether a given planar graph has a (j,k)-coloring is NP-complete. It is somewhat nonstandard but convenient to call a graph (j,k)-*critical* if it is not (j,k)-colorable but after deleting any edge or vertex it becomes (j,k)-colorable.

Glebov and Zambalaeva [23] proved that every planar graph G with girth, $g(G)$, at least 16 is $(0,1)$-colorable. Then Borodin and Ivanova [1] proved that every graph G with $mad(G) < \frac{7}{3}$ is $(0,1)$-colorable. Since $mad(G) \leq \frac{2g(G)}{g(G)-2}$ for every planar graph G with girth $g(G)$, this implies that every planar graph G with $g(G) \geq 14$ is $(0,1)$-colorable. Borodin and the first author [6] proved that every graph G with $mad(G) < \frac{12}{5}$ is $(0,1)$-colorable, and this is sharp. This also implies that every planar graph G with $g(G) \geq 12$ is $(0,1)$-colorable. Dorbec, Kaiser, Montassier, and Raspaud [18] have constructed a $(0,1)$-critical graph with girth 9.

Kurek and Ruciński [36] studied improper colorings within the more general framework of *vertex Ramsey probems*. We say that $G \to (H_1, \ldots, H_k)$ if for every coloring $\phi : V(G) \to [k]$, there exists an i such that the subgraph of G induced by the vertices of Color i contains H_i. By definition, a graph G is $(K_{1,j}, K_{1,k})$-vertex Ramsey exactly when G has no (j,k)-coloring. Kurek and Ruciński [36] considered the extremal function $m_{cr}(H_1, \ldots, H_k) = inf\{mad(F) : F \to (H_1, \ldots, H_k)\}$. In particular, Kurek and Ruciński showed that $8/3 \leq m_{cr}(K_{1,2}, K_{1,2}) \leq 14/5$. Ruciński offered 400,000 PLZ cash prize for the exact value of $m_{cr}(K_{1,2}, K_{1,2})$. Recently, Borodin and the authors solved this problem.

Theorem 10 ([9]). *If G is a $(1,1)$-critical graph, then $5|E(G)| > 7|V(G)|$.*

The result is sharp in the sense that there are infinitely many $(1,1)$-critical graphs with $5|E(G)| = 7|V(G)| + 1$. One such graph is present in Fig. 4.

The proof of the result is algorithmic and yields a polynomial-time algorithm that finds a $(1,1)$-coloring for every graph G with $mad(G) \leq \frac{14}{5}$. In a standard manner, the theorem yields that every planar graph with girth at least 7 is $(1,1)$-colorable. It also refines a result by Borodin and Ivanova [2]: They showed that every graph G with girth at least 7 and $mad(G) < \frac{14}{5}$ can be partitioned into two subsets such that every connected monochromatic subgraph has at most two edges. Our result shows that each component contains at most 1 edge.

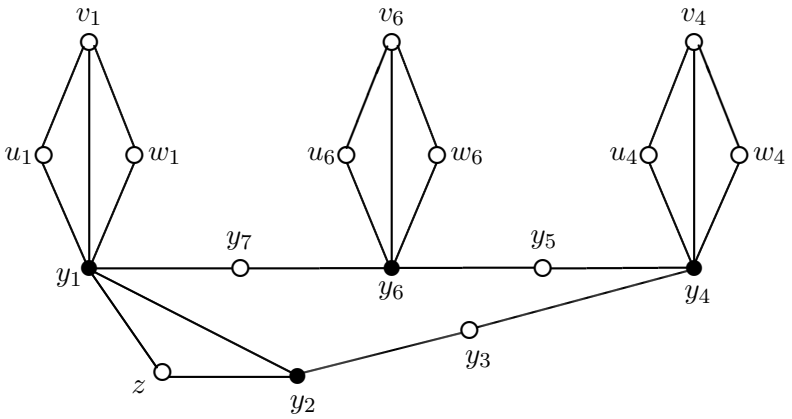


Fig. 4. A $(1,1)$-critical graph G with $5|E(G)| = 7|V(G)| + 1$

For general j and k, the problem of (j,k)-coloring of planar and globally sparse graphs was considered in [3–5]. Borodin and the first author [7] proved that if $k \geq 2j+2$, then every graph with maximum average degree at most $2\left(2 - \frac{k+2}{(j+2)(k+1)}\right)$ is (j,k)-colorable. On the other hand, they constructed graphs with the maximum average degree arbitrarily close to $2\left(2 - \frac{k+2}{(j+2)(k+1)}\right)$ (from above) that are not (j,k)-colorable. Note that most likely if $j \leq k < 2j+2$, then the answer differs from that in the case $k \geq 2j+2$. In particular, the answer for $(1,1)$-colorings in Theorem 10 differs from it.

References

1. Borodin, O.V., Ivanova, A.O.: Near-proper vertex 2-colorings of sparse graphs. Diskretn. Anal. Issled. Oper. 16(2), 16–20 (2009) (in Russian); Translated in: Journal of Applied and Industrial Mathematics 4(1), 21–23 (2010)
2. Borodin, O.V., Ivanova, A.O.: List strong linear 2-arboricity of sparse graphs. J. Graph Theory 67(2), 83–90 (2011)
3. Borodin, O.V., Ivanova, A.O., Montassier, M., Ochem, P., Raspaud, A.: Vertex decompositions of sparse graphs into an edgeless subgraph and a subgraph of maximum degree at most k. J. Graph Theory 65, 83–93 (2010)
4. Borodin, O.V., Ivanova, A.O., Montassier, M., Raspaud, A. (k,j)-Coloring of sparse graphs. Discrete Applied Mathematics 159(17), 1947–1953 (2011)
5. Borodin, O.V., Ivanova, A.O., Montassier, M., Raspaud, A.: (k,1)-Coloring of sparse graphs. Discrete Math 312(6), 1128–1135 (2012)
6. Borodin, O.V., Kostochka, A.V.: Vertex partitions of sparse graphs into an independent vertex set and subgraph of maximum degree at most one (Russian). Sibirsk. Mat. Zh. 52(5), 1004–1010 (2011); Translation in: Siberian Mathematical Journal 52(5), 796–801
7. Borodin, O.V., Kostochka, A.V.: Defective 2-colorings of sparse graphs (submitted)

8. Borodin, O.V., Kostochka, A.V., Lidický, B., Yancey, M.: Short proofs of coloring theorems on planar graphs (submitted)
9. Borodin, O.V., Kostochka, A.V., Yancey, M.: On 1-improper 2-coloring of sparse graphs (submitted)
10. Corrĕa, R., Havet, F., Sereni, J.-S.: About a Brooks-type theorem for improper colouring. Australas. J. Combin. 43, 219–230 (2009)
11. Brooks, R.L.: On colouring the nodes of a network. Math. Proc. Cambridge Philos. Soc. 37, 194–197 (1941)
12. Dirac, G.A.: Note on the colouring of graphs. Math. Z. 54, 347–353 (1951)
13. Dirac, G.A.: A property of 4-chromatic graphs and some remarks on critical graphs. J. London Math. Soc. 27, 85–92 (1952)
14. Dirac, G.A.: Some theorems on abstract graphs. Proc. London Math. 3(2), 69–81 (1952)
15. Dirac, G.A.: Map colour theorems related to the Heawood colour formula. J. London Math. Soc. 31, 460–471 (1956)
16. Dirac, G.A.: A theorem of R. L. Brooks and a conjecture of H. Hadwiger. Proc. London Math. Soc. 7(3), 161–195 (1957)
17. Dirac, G.A.: The number of edges in critical graphs. J. Reine Angew. Math. 268(269), 150–164 (1974)
18. Dorbec, P., Kaiser, T., Montassier, M., Raspaud, A.: Limits of near-coloring of sparse graphs. To Appear in J. Graph Theory
19. Esperet, L., Montassier, M., Ochem, P., Pinlou, A.: A Complexity Dichotomy for the Coloring of Sparse Graphs. To Appear in J. Graph Theory
20. Farzad, B., Molloy, M.: On the edge-density of 4-critical graphs. Combinatorica 29, 665–689 (2009)
21. Gallai, T.: Kritische Graphen I. Publ. Math. Inst. Hungar. Acad. Sci. 8, 165–192 (1963)
22. Gallai, T.: Kritische Graphen II. Publ. Math. Inst. Hungar. Acad. Sci. 8, 373–395 (1963)
23. Glebov, A.N., Zambalaeva, D.Z.: Path partitions of planar graphs. Sib. Elektron. Mat. Izv. 4, 450–459 (2007) (Russian)
24. Grötzsch, H.: Zur Theorie der diskreten Gebilde. VII. Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg. Math.-Nat. Reihe 8, 109–120 (1958/1959) (in Russian)
25. Havet, F., Sereni, J.-S.: Channel assignment and improper choosability of graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 81–90. Springer, Heidelberg (2005)
26. Jensen, T., Thomassen, C.: The color space of a graph. J. Graph Theory 34, 234–245 (2000)
27. Jensen, T.R., Toft, B.: Graph Coloring Problems. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York (1995)
28. Jensen, T.R., Toft, B.: 25 pretty graph colouring problems. Discrete Math 229, 167–169 (2001)
29. Karp, R.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103 (1972)
30. Kostochka, A.V., Stiebitz, M.: Excess in colour-critical graphs. In: Graph Theory and Combinatorial Biology, Balatonlelle, Hungary (1996); Bolyai Society, Mathematical Studies, Budapest, vol. 7, pp. 87–99 (1999)
31. Kostochka, A.V., Stiebitz, M.: A new lower bound on the number of edges in colour-critical graphs and hypergraphs. Journal of Combinatorial Theory, Series B 87, 374–402 (2003)

32. Kostochka, A.V., Yancey, M.: Ore's Conjecture on color-critical graphs is almost true (submitted)
33. Kostochka, A.V., Yancey, M.: Ore's Conjecture for k = 4 and Grötzsch Theorem. Combinatorica (accepted)
34. Kostochka, A.V., Yancey, M.: A Brooks-type result for sparse critical graphs (submitted)
35. Krivelevich, M.: On the minimal number of edges in color-critical graphs. Combinatorica 17, 401–426 (1997)
36. Kurek, A., Rucin'ski, A.: Globally sparse vertex-Ramsey graphs. J. Graph Theory 18, 73–81 (1994)
37. Ore, O.: The Four Color Problem. Academic Press, New York (1967)
38. Thomas, R., Walls, B.: Three-coloring Klein bottle graphs of girth five. J. Combin. Theory Ser. B 92, 115–135 (2004)
39. Thomassen, C.: A short list color proof of Grötzsch's theorem. J. Combin. Theory Ser. B 88, 189–192 (2003)
40. Toft, B.: Color-critical graphs and hypergraphs. J. Combin. Theory 16, 145–161 (1974)
41. Toft, B.: 75 graph-colouring problems. In: Keynes, M. (ed.) Graph Colourings, pp. 9–35 (1988) Pitman Res. Notes Math. Ser., vol. 218. Longman Sci. Tech., Harlow (1990)
42. Tuza, Z.: Graph coloring. In: Gross, J.L., Yellen, J. (eds.) Handbook of Graph Theory, xiv+1167 pp. CRC Press, Boca Raton (2004)
43. Zuckerman, D.: Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. Theory of Computing 3, 103–128 (2007)

On Recognizing Words That Are Squares for the Shuffle Product

Romeo Rizzi[1] and Stéphane Vialette[2]

[1] Dipartimento di Matematica ed Informatica, Università degli Studi di Udine, Italy
rrizzi@dimi.uniud.it
[2] LIGM CNRS UMR 8049, Université Paris-Est, France
vialette@univ-mlv.fr

Abstract. The shuffle of two words u and v of A^* is the language $u \sqcup\!\sqcup v$ consisting of all words $u_1v_1u_2v_2 \dots u_kv_k$, where $k \geq 0$ and the u_i and the v_i are the words of A^* such that $u = u_1u_2 \dots u_k$ and $v = v_1v_2 \dots v_k$. A word $u \in A^*$ is a *square for the shuffle product* if it is the shuffle of two identical words (*i.e.*, $u \in v \sqcup\!\sqcup v$ for some $v \in A^*$). Whereas, it can be tested in polynomial-time whether or not $u \in v_1 \sqcup\!\sqcup v_2$ for given words u, v_1 and v_2 [J.-C. Spehner, *Le Calcul Rapide des Mélanges de Deux Mots*, Theoretical Computer Science, 1986], we show in this paper that it is **NP**-complete to determine whether or not a word u is a square for the shuffle product. The novelty in our approach lies in representing words as *linear graphs*, in which deciding whether or not a given word is a square for the shuffle product reduces to computing some *inclusion-free perfect matching*.

1 Introduction

Let A be an alphabet. The *shuffle* $u \sqcup\!\sqcup v$ of words u and v over A is the finite set of all words obtainable from merging the words u and v from left to right, but choosing the next symbol arbitrarily from u or v [15]. The *iterated shuffle* of u is the language $\epsilon \cup u \cup (u \sqcup\!\sqcup u) \cup (u \sqcup\!\sqcup u \sqcup\!\sqcup u) \cup \dots$ These definitions naturally extend to languages. It is known that the shuffle product is a commutative and associative operation, which is also distributive over union. The operations of shuffle and iterated shuffle have been used by many researchers to describe sequential computation histories of concurrent programs [12]. Interestingly, it was observed in [11] that some aspects of the shuffle product bear strong similarities with genetic recombinations (a non-tree-like event that produces a child sequence by crossing two parent sequences).

For the iterated shuffle, there are basically two kinds of questions that can be addressed depending on whether or not the shuffled element is given as a part of the input. This distinction basically reduces to the two following simpler problems:

- "*Given $u, v \in A^*$, is u in the iterated shuffle of v?*", and
- "*Given $u \in A^*$, is u in the iterated shuffle of some $v \in A^*$?*".

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 235–245, 2013.

If we focus on only one application of the shuffle product, we are left with the two following problems:

- "*Given* $u, v \in A^*$*, is* $u \in v \sqcup\!\sqcup v$*?*", and
- "*Given* $u \in A^*$*, does there exist* $v \in A^*$ *such that* $u \in v \sqcup\!\sqcup v$*?*".

As we shall see soon, these two problems dramatically differ in complexity.

We briefly review the key results that arise in our context. Given words u, v_1 and v_2, it can be tested in $O\left(|u|^2/\log(|u|)\right)$ time whether or not $u \in v_1 \sqcup\!\sqcup v_2$ [13]. (To the best of our knowledge, the first $O(|u|^2)$ time algorithm appeared in [17]. This algorithm can easily be extended to check in polynomial-time whether or not a word is the shuffle of any fixed number of given words.) The shuffle $u \sqcup\!\sqcup v$ of words u and v can be computed in $O\left((|u|+|v|)\binom{|u|+|v|}{|u|}\right)$ time [19]. An improvement and generalization has been proposed in [1], where it is proved that, given words $u_1, u_2, \ldots, u_k$, the shuffle $\sqcup\!\sqcup_{i=1}^{k} u_i$ can be computed in $O\left(\binom{|u_1|+|u_2|+\ldots+|u_k|}{|u_1|,|u_2|,\ldots,|u_k|}\right)$ time.

Given words $u, v_1, v_2, \ldots, v_n \in A^*$, it is however **NP**-complete to decide whether or not $u \in \sqcup\!\sqcup_{i=1}^{k} v_i$. [17,6]. This remains true even if the alphabet has size 3 [6]. Of particular interest, it is shown in [6] that this problem remains **NP**-complete even if all words $v_1, v_2, \ldots, v_n$ are identical, thereby proving that, for two words u and v, it is **NP**-complete to decide whether or not u is in the iterated shuffle of v. Again, this remains true even if the alphabet has size 3.

Strongly related is the problem of shuffling a word with its reverse. Let $u \in A^*$. It is easily seen that if there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$, then u is an *abelian square* (*i.e.*, $u = v\,v'$, where v' is a permutation of v). It is shown in [18] that if u is a binary abelian square, then there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$, thereby proving that it is polynomial-time solvable to decide whether or not a binary word is the shuffle of another word with its reverse. The equivalence is no longer true for larger alphabets. For example, *abcabc* is a ternary abelian square that cannot be written as an element of $v \sqcup\!\sqcup v^R$ for any word $v \in A^*$.

The novelty in our approach lies in the use of *linear graphs* (*i.e*, those graphs with sets of vertices equipped with some total order), in which deciding whether or not a given word is in the shuffle of another word with itself (or its reverse) reduces to computing some constrained perfect matching. In this paper, we show that, given $u \in A^*$, it is **NP**-complete to decide whether or not u is the shuffle of some word $v \in A^*$ with itself (*i.e.*, does there exist some $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$?). It is worth mentioning that this result has been recently proved independently by Buss and Soltys [3] (as an answer to Erickson [10] on the Stack Exchange discussion board). Notice that this hardness result was first claimed by Iwama [9], but it turns out that the proof has a serious flaw [5]. Furthermore, we complete the result of [18] for any alphabet by proving that, given $u \in A^*$, it is polynomial-time solvable to decide whether or not u is the shuffle of some word $v \in A^*$ with its reverse (*i.e.*, does there exist some $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$?).

2 Definitions

We follow standard terminology on words [4]. Let A be an alphabet. The *empty word* is denoted ϵ. A word $v = a_1 a_2 \dots a_n \in A^n$ is a *subsequence* of $u \in A^*$ if there exist $n+1$, not necessarily distinct and possibly empty, words $u_1, u_2, \dots, u_{n+1} \in A^*$ such that $u_1\, a_1\, u_2 \dots u_n\, a_n\, u_{n+1} = u$, and we write $v \preceq u$ to denote this fact. The *reverse* of $u = a_1\, a_2 \dots a_n$ with $a_i \in A$ is the word $u^R = a_n \dots a_2\, a_1$.

The *shuffle* (also sometimes referred to as *ordinary shuffle*) of two words u and v, denoted $u \sqcup\!\sqcup v$, is the language of all words w such that $w = u_1\, v_1\, u_2\, v_2 \dots u_n\, v_n$, where $u_i, v_i \in A^*$, $u_1\, u_2 \dots u_n = u$, and $v_1\, v_2 \dots v_n = v$. It may be defined inductively on words by $u \sqcup\!\sqcup \epsilon = u$, $\epsilon \sqcup\!\sqcup u = u$, and $ua \sqcup\!\sqcup vb = (u \sqcup\!\sqcup vb)a \cup (ua \sqcup\!\sqcup v)b$. The *iterated shuffle* of u is the language $\epsilon \cup u \cup (u \sqcup\!\sqcup u) \cup (u \sqcup\!\sqcup u \sqcup\!\sqcup u) \cup \dots$

For a graph G, we denote $\mathbf{V}(G)$ as the set of vertices and $\mathbf{E}(G)$ as the set of edges. Let $u = u_1\, u_2 \dots u_n \in A^n$ be a word on some alphabet A. The *graph associated to* u, denoted G_u, is defined by $\mathbf{V}(G_u) = \{u_1, u_2, \dots, u_n\}$ and $\mathbf{E}(G_u) = \{\{u_i, u_j\} : i \neq j \ \wedge \ u_i = u_j\}$. (We write (u_i, u_j) for an an edge of $\mathbf{E}(G_u)$ if it is clear from the context that $i < j$, and $\{u_i, u_j\}$ otherwise.) The structure of this underlying graph is linear, *i.e.*, the set of vertices is equipped with a natural total order $<$ defined by $u_i < u_j$ if and only if $i < j$. In other words, the vertices of G_u correspond to the letters of u in the left to right order and there is an edge between any two identical distinct letter of u. Clearly, G_u is the disjoint union of cliques, one clique for each distinct letter. The *length* of an edge $\{u_i, i_j\} \in \mathbf{E}(G_u)$, denoted $\ell(\{u_i, u_j\})$, is defined by $|j - i|$. The *total edge-length* of G_u, denoted $\mathcal{L}(G_u)$, is defined by $\sum_{\{u_i, u_j\} \in \mathbf{E}(G_u)} \ell(\{u_i, u_j\}) = O(|u|^2)$. Recall that two edges of a graph are *independent* if they do not share a common vertex, and that a *matching* $\mathcal{M}$ in G is a set of pairwise independent edges. A matching is *perfect* if it covers all the vertices of the graph. In case the set of vertices is equipped with a total order, a matching $\mathcal{M}$ is said to be *inclusion-free* if there do not exist (independent) edges (u_i, u_j) and (u_k, u_ℓ) in $\mathcal{M}$ such that $u_i < u_k < u_\ell < u_j$ or $u_k < u_i < i_j < u_\ell$. Similarly, a matching $\mathcal{M}$ is said to be *precedence-free* if there do not exist (independent) edges (u_i, u_j) and (u_k, u_ℓ) in $\mathcal{M}$ such that $u_i < u_j < u_k < u_\ell$ or $u_k < u_\ell < u_i < u_j$.

3 Being a Square for the Shuffle Product

This section is devoted to proving hardness of recognizing those words that are squares for the shuffle product. At the heart of our approach is the following property. (See Fig. 1 for an illustration.)

Lemma 1. *Let $u \in A^*$ for some alphabet A, and G_u be the corresponding linear graph. Then, there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$ if and only if there exists an inclusion-free perfect matching in G_u.*

Proof. Indeed, suppose first that there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$. Let $2n = |u|$. Fix the occurrences in u of the two copies of v and write

$I^1 = \{i^1_1, i^1_2, \ldots, i^1_n\}$, $i^1_1 < i^1_2 < \ldots < i^1_n$, for the positions in u of the first copy of v and $I^2 = \{i^2_1, i^2_2, \ldots, i^2_n\}$, $i^2_1 < i^2_2 < \ldots < i^2_n$, for the positions in u of the second copy of v. It is easily seen that $\mathcal{M} = \{\{u_{i^1_j}, u_{i^2_j}\} : 1 \leq j \leq n\}$ is a subset of $\mathbf{E}(G_u)$. Furthermore, $\mathcal{M}$ is a perfect matching since $I^1 \cap I^2 = \emptyset$, and $I^1 \cup I^2 = \{1, 2, \ldots 2n\}$. It is also inclusion-free. Indeed, if it were not the case then there would exist two edges $e = \{u_{i^1_j}, u_{i^2_j}\}$ and $e' = \{u_{i^1_k}, u_{i^2_k}\}$, $j < k$, in $\mathcal{M}$ such that $i^1_j < i^1_k < i^2_k < i^2_j$. This is a contradiction since $i^2_k > i^2_j$ if $k > j$.

Conversely, Let $\mathcal{M} \subseteq \mathbf{E}(G_u)$ be an inclusion-free matching of G_u. Let $I^1 = \{i^1_j : \exists\{u_{i_j}, u_{i_k}\} \in \mathcal{M}$ with $j < k\}$, and $I^2 = \{i^2_k : \exists\{u_{i_j}, u_{i_k}\} \in \mathcal{M}$ with $j < k\}$. Assume $i^1_1 < i^1_2 < \ldots < i^1_n$ and $I^2 = \{i^2_1, i^2_2, \ldots, i^2_n\}$. Let $v = u_{i^1_1} u_{i^1_2} \ldots u_{i^1_n}$ and $v' = u_{i^2_1} u_{i^2_2} \ldots u_{i^2_n}$. We claim that $v = v'$. Suppose, aiming at a contradiction, that $v \neq v'$. Let $j - 1$ be the length of the largest common prefix of v and v'. Then it follows that

1. $v_j \neq v'_j$,
2. $v_j = v'_k$ for some $k > j$,
3. $v_\ell = v'_j$ for $\ell > j$,

and hence $\mathcal{M}$ is not inclusion-free. This is the sought contradiction. □

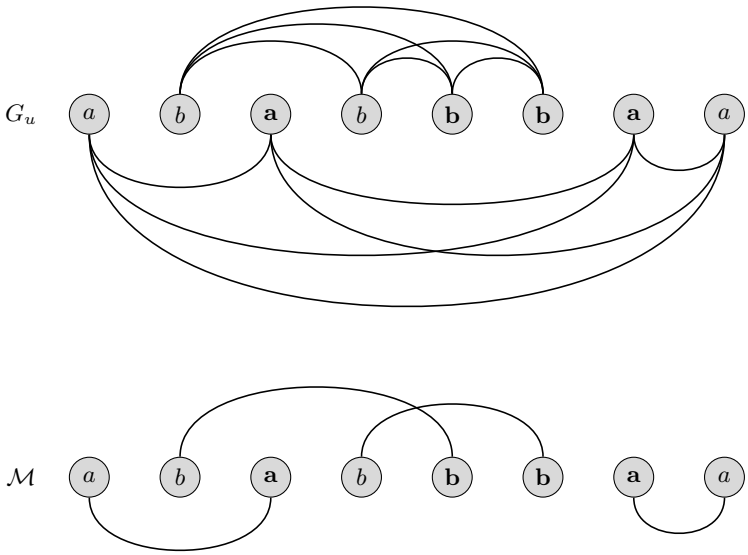


Fig. 1. The linear graph G_u of $u = ababbbaa$ together with an inclusion-free perfect matching $\mathcal{M}$. The perfect matching $\mathcal{M}$ denotes $u \in v \sqcup\!\sqcup v$ for $v = abba$ and reads as $u = \begin{matrix} a\ b & & b & & & a \\ & a & & b\ b & a & \end{matrix}$ with the first copy of v on top and the second on bottom.

We are now in position to prove our main result.

Proposition 1. *It is* **NP***-complete to determine whether or not a word is a square for the shuffle product.*

Proof. The problem is certainly in **NP**. To prove hardness, we propose a polynomial-time reduction from the **NP**-complete Longest Common Subsequence for binary words (01-LCS for short) which is defined as follows: Given a collection of words $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$, and a positive integer k, decide whether there exists a subsequence of size k common to all sequences of U [16]. Without loss of generality, we may assume that $|u_i| = |u_j|$ for $1 \leq i < j \leq m$, and that that we are looking for a common subsequence with p letters 0 and q letters 1, $k = p + q$. We write (U, p, q) for such an instance of 01-LCS.

Let (U, p, q), $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$ and $|u_i| = |u_j| = n$ for $1 \leq i < j \leq m$, be an arbitrary instance of 01-LCS. Let us construct from this instance a word w over the $(3m+6)$-size alphabet A defined as follows:

$$A = \{0,1\} \dot\cup \{s, s'\} \dot\cup \{t, t'\} \dot\cup \{x_i, y_i, z_i : 1 \leq i \leq m\}.$$

The word w is defined by

$$w = W_s\, W_1\, W_2\, \ldots\, W_m\, W_t,$$

where W_s, W_1, W_2, …, W_m and W_t are words in A^* (we refer to these words as our gadget words).

Let us now detail the various gadget words. The source and sink gadget words, denoted W_s and W_t respectively, are defined as follows

$$W_s = s'\, 0^{pq}\, s'\, s\, (0^p\, 1)^q\, 0^p\, s$$
$$W_t = t\, (0^p\, 1)^q\, 0^q\, t\, t'\, 0^{pq}\, t',$$

where s, s', t and t' are four letters that do not occur in any other gadget word. To shorten the exposition, we shall speak about the $(0^p\ 1)^q\ 0^p$*-factor* of W_s (resp. W_t) to designate the factor of W_s (resp. W_t) that occurs between the two occurrences of the letter s (resp. t), and about the 0^{pq}*-factor* of W_s (resp. W_t) to designate the factor of W_s (resp. W_t) that occurs between the two occurrences of the letter s' (resp. t'). For each input word $u_i = u_{i,1}\, u_{i,2}\, \ldots\, u_{i,n}$, the associated gadget word W_i is defined by

$$W_i = x_i\, W_i'\, x_i\, y_i\, W_i'\, y_i,$$

where

$$W_i' = u_{i,1}\, z_i\, u_{i_2}\, z_i\, \ldots\, u_{i,n-1}\, z_i\, u_{i,n}.$$

(Notice that letters x_i, y_i and z_i only occur in the gadget word W_i.)

With the corresponding linear graph G_w in mind, for any letter $a \in A$ occurring only twice in w, we shall write (a, a)-*edge* to designate (without any ambiguity) the unique edge connecting the two occurrences of letter a in G_w.

We now claim that there exists a common subsequence with p letters 0 and q letters 1 common to all sequences of U if and only if w is a square for the shuffle product. It will convenient to see the reduction as a flow-like procedure, where some piece of information (the common subsequence) emitted from gadget W_s (the source) propagates lossless to gadget W_t (the sink) going through all gadgets W_i, $1 \leq i \leq m$ (every such gadget being associated to an input word of our input instance of 01-LCS).

For the forward direction, suppose that there exists a common subsequence v of the words $u_1, u_2, \ldots, u_m$ with p occurrences of the letter 0 and q occurrence of the letter 1. Write $k = p + q$ and $v = v_1 v_2 \ldots v_k$. According to Lemma 1, it is enough to show that G_w has an inclusion-free perfect matching. Now, observe that v is a subsequence of both the $(0^p\ 1)^q\ 0^p$-factor of W_s and the $(0^p\ 1)^q\ 0^p$-factor of W_t. Furthermore, by hypothesis, v also occurs in each gadget word W'_i, $1 \leq i \leq m$. Fix any occurrence of v as a subsequence in the $(0^p\ 1)^q\ 0^p$-factor of W_s, the $(0^p\ 1)^q\ 0^p$-factor of W_t, and in every W'_i gadget word, $1 \leq i \leq m$ (if W'_i contains several occurrences of v as a subsequence, we fix any but the same occurrence in the two W'_i gadget words.) Having disposed of these preliminary steps, we can now turn to defining an inclusion-free perfect matching $\mathcal{M}$ of G_w. This perfect matching contains both *intra-gadget edges* (*i.e.*, edges connecting two identical letters that occur in the same gadget word), and *inter-gadget edges* (*i.e.*, edges connecting two identical letters that occur in distinct - but consecutive - gadget words).

Intra-gadget Edges :

- $\mathcal{M}$ contains (i) the (s, s)-edge, (ii) the (s', s')-edge, and (iii) pq pairwise crossing edges that connect the leftmost pq letters 0 of W_s to the pq letters 0 of the $(0^p\ 1)^q\ 0^p$-factor of W_s that do not correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_s.
- For every $1 \leq i \leq m$, $\mathcal{M}$ contains (i) the (x_i, x_i)-edge,, (ii) the (y_i, y_i)-edge, (iii) $n - 1$ pairwise crossing edges connecting the $n - 1$ occurrences of letter z_i in the leftmost W'_i gadget word to the $n - 1$ occurrences of letter z_i in the rightmost W'_i gadget word, and (iv) $n - p - q$ pairwise crossing edges connecting the $n - p - q$ letters of the leftmost W'_i gadget word that do not correspond to the chosen occurrence of v in W'_i to the $n - p - q$ letters of the rightmost W'_i gadget word that do not correspond to the occurrence of v in W'_i.
- $\mathcal{M}$ contains (i) the (t, t)-edge, (ii) the (t', t')-edge, and (iii) pq pairwise crossing edges connecting the pq letters 0 of the $(0^p\ 1)^q\ 0^p$ factor of W_s that do not correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$ factor of W_t to the last pq letters 0 of W_t.

Inter-gadget Edges :

- $\mathcal{M}$ contains $p+q$ pairwise crossing edges connecting the $p+q$ letters of the $(0^p\ 1)^q\ 0^p$-factor of W_s that correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_s to the $p+q$ letters of the leftmost W'_1 gadget word that correspond to the chosen occurrence of the common subsequence v in W'_1.
- For every $1 \leq i < m$, $\mathcal{M}$ contains $p+q$ pairwise crossing edges connecting the $p+q$ letters of the rightmost W'_i gadget word that correspond to the chosen occurrence of v in W'_i to the $p+q$ letters of the leftmost W'_{i+1} gadget word that correspond to the chosen occurrence of v in W'_{i+1}.
- $\mathcal{M}$ contains pq pairwise crossing edges connecting the $p+q$ letters of the rightmost W'_m gadget word that correspond to the chosen occurrence of the common subsequence v in W'_m to the $p+q$ letters of $(0^p\ 1)^q\ 0^p$-factor of W_t that correspond to the chosen occurrence of the common subsequence v.

It can be easily verified that $\mathcal{M}$ is a perfect inclusion-free matching. Indeed, all inter-gadget edges in $\mathcal{M}$ are pairwise crossing, and no two intra-gadget edges are in inclusion.

For the reverse direction, suppose that w is a square for the shuffle product. Once again, according to Lemma 1, this amount to saying that G_w has an inclusion-free perfect matching $\mathcal{M}$. We begin by a sequence of easy observations. First, observe that the letters s, s', t, t', x_i $(1 \leq i \leq m)$, and y_i $(1 \leq i \leq m)$ occur exactly twice in w, and hence the $2m+4$ edges connecting these vertices two by two have to be in $\mathcal{M}$ since it is perfect. In other words, $\mathcal{M}$ contains the (s,s)-edge, the (s',s')-edge, the (t,t)-edge, the (t',t')-edge, and the (x_i,x_i)-edge and the (y_i,y_i)-edge for $1 \leq i \leq m$. Let us now focus on the source W_s gadget word. Since both the (s,s)-edge and the (s',s')-edge are in $\mathcal{M}$, then it follows that (i) no edge in $\mathcal{M}$ can connect two identical letters occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_s, and (ii) no edge in $\mathcal{M}$ can connect two identical letters occurring in the 0^{pq}-factor of W_s. Moreover, $\mathcal{M}$ contains pq pairwise crossing edges connecting all letters from the 0^{pq}-factor of W_s to pq letters 0 occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_s. Similar considerations apply to W_t yielding (i) no edge in $\mathcal{M}$ can connect two identical letters occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_t, and (ii) no edge in $\mathcal{M}$ can connect two identical letters occurring in the 0^{pq}-factor of W_t. Hence, $\mathcal{M}$ contains pq pairwise crossing edges connecting pq letters 0 occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_t to all letters from the 0^{pq}-factor of W_t. We now turn to the W_i gadget words. For every $1 \leq i \leq m$, $\mathcal{M}$ has to contain both the (x_i,x_i)-edge and (y_i,y_i)-edge, and hence $\mathcal{M}$ contains $n-1$ pairwise crossing edges connecting the $n-1$ letters z_i of the leftmost W'_i gadget word to the $n-1$ letters z_i of the rightmost W'_i gadget word (otherwise one edge connecting two letters z_i would be included in the (x_i,x_i)-edge or in the (y_i,y_i)-edge).

According to the above, $p+q$ letters of W_s have to be involved in some inter-gadget edges of $\mathcal{M}$. But $\mathcal{M}$ contains the (x_1,x_1)-edge, an hence these $p+q$ inter-gadget edges are pairwise crossing and each connect a letter occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_s to a letter in the leftmost W'_1 gadget word. Since the

$2(n-1)$ occurrences of letter z_i are involved in $n-1$ pairwise crossing edges, then it follows that $\mathcal{M}$ contains $n-p-q$ pairwise crossing edges connecting the $n-p-q$ letters $u_{1,j}$ of the leftmost W'_1 gadget word that are not involved in the leftmost $p+q$ inter-gadget edges to $n-p-q$ letters $u_{1,j}$ of the rightmost W'_1 gadget word. Of particular importance, these $n-p-q$ edges have to be position preserving, *i.e.*, each edge connect a letter $u_{1,j}$ of the leftmost W'_1 gadget word to a letter $u_{1,j}$ of the rightmost W'_1 gadget word for a same position j. At this point, $p+q$ letters of the rightmost W'_1 gadget are yet to be involved in $\mathcal{M}$. Since $\mathcal{M}$ contains both the (y_1, y_1)-edge and the (x_2, x_2)-edge, the only solution is that $\mathcal{M}$ contains $p+q$ pairwise crossing edges connecting letters from the rightmost W'_1 gadget word to the leftmost W'_2 gadget word. The same process continues until $p+q$ pairwise crossing edges connecting the rightmost W'_m gadget word to the W_t sink gadget word.

It follows from the examination of $\mathcal{M}$ that the $p+q$ pairwise crossing edges connecting $p+q$ letters of the $(0^p\ 1)^q\ 0^p$-factor of W_s to $p+q$ letters of the leftmost W'_1 gadget word define a word with p letters 0 and q letters 1 that occurs as a subsequence in each input word u_i. □

It is worth noticing that Li and Li [14] proved that computing the largest inclusion-free matching in a linear graph is **NP**-complete. However, their quite complicated proof involves general linear graphs and not linear graphs that are unions of cliques, and hence cannot be used in the context of shuffling words (the above proposition may, however, be seen as a much simpler proof of Li and Li's result).

We also notice that in the proof of Proposition 1, some letters occur exactly two times in the constructed word w. Clearly, in this case, w cannot be the shuffle of $k \geq 3$ identical copies of some word $v \in A^*$. In other words, if w is in the iterated shuffle of some distinct word $v \in A^*$, then w is a square for the shuffle product. We have thus proved the following.

Proposition 2. *It is* **NP***-complete to decide whether or not a word $u \in A^*$ is in the iterated shuffle of some word $v \in A^*$ with $u \neq v$.*

4 Being the Shuffle of a Word with Its Reverse

As we mentioned in Section 1, for a given u over some binary alphabet A, it is polynomial-time solvable to determine whether or not there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$. Indeed, if there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$, then u is an *abelian square* (*i.e.*, $u = v\,v'$, where v' is a permutation of v). Furthermore, if u is a binary abelian square, then there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$ [18]. The equivalence is, however, no longer true for larger alphabets. As a complementary result, we use again linear graphs to show that the result holds for any alphabet. We need the following equivalence which is the analogous of Lemma 1 (proof omitted). (See Fig. 2 for an illustration.)

Lemma 2. *Let $u \in A^*$ for some alphabet A, and G_u be the corresponding linear graph. Then, there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$ if and only if there exists an precedence-free perfect matching in G_u.*

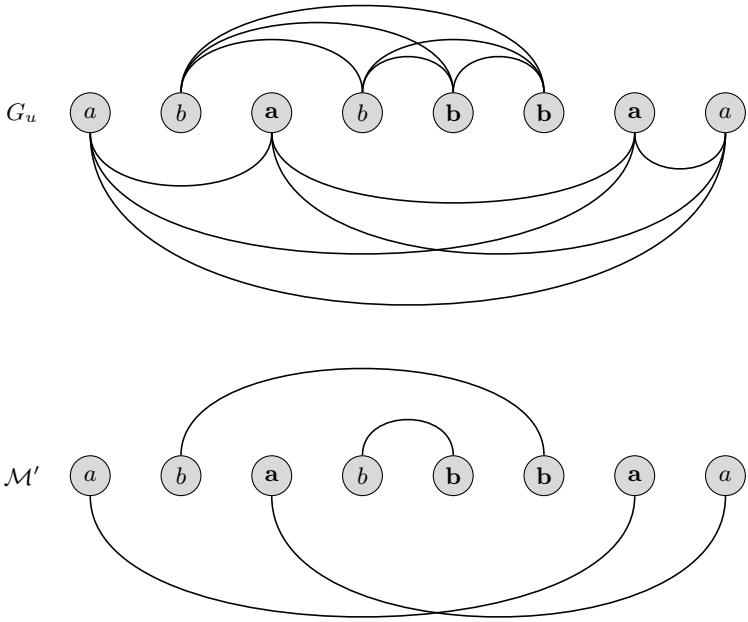


Fig. 2. The linear graph G_u of $u = ababbbaa$ together with and a precedence-free perfect matching $\mathcal{M}'$. The perfect matching $\mathcal{M}'$ denotes $u \in v \sqcup\!\sqcup v^R$ for $v = abba$ and reads as $u = \begin{smallmatrix} a\ b & & b & & & a \\ & a & & b\ b\ a & & \end{smallmatrix}$ with the first copy of v on top and the second on bottom.

Precedence-free matchings in linear graphs arose in the context of 2-intervals [21], and the following proposition makes use of an algorithm for pattern matching in point support 2-intervals [8] that improves on [20].

Proposition 3. *Let $u \in A^*$. It can be checked in $O(|u| \log |u| + \mathcal{L}(G_u)) = O(|u|^2)$ time whether or not there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$.*

Proof. In the context of multiple intervals, Erdong *et al.* [8] have proposed a $O(n \log n + \mathcal{L})$ time algorithm for computing a maximum cardinality precedence-free matching in a linear graph, where $\mathcal{L}$ is the total sum of the edge lengths. The Proposition now follows from Lemma 2. □

5 Conclusion and Open Problems

In this paper we have used a (union of cliques) linear graph framework to show that it is **NP**-complete to recognize those words that squares for the shuffle product. Using the same framework, we have proved that recognizing those words that are the shuffle of another word with its reverse is polynomial-time solvable. There are a number of further directions of investigation in this general subject. We mention several open problems that are, in our opinion, the most interesting.

How hard is the problem of detecting squares for the shuffle product for bounded alphabet words? It is proved in [3] that the problem is **NP**-complete for an alphabet with 9 symbols (it is claimed that this can be improved to 7 letters). Notice that it is claimed without proof in [2] (Fact 2 Subsection 2.2) that detecting squares for the shuffle product is **NP**-complete for binary words. This result – that would be an important improvement over [3] and Proposition 1 – is yet to be confirmed.

It is **NP**-complete to decide whether or not a word $u \in A^*$ is in the iterated shuffle of some word $v \in A^*$ with $u \neq v$ (see Proposition 2). Is this problem equally hard for bounded alphabets?

Given words $u, v_1, v_2, \ldots, v_k \in A^*$, it is **NP**-complete to decide whether or not $u \in \sqcup\!\sqcup_{i=1}^{k} v_i$ [17,6]. Going further, it would make sense to precisely characterize hardness. In particular, under parameterized complexity [7], where does fall this problem in the **W**-hierarchy for parameter k?

Acknowledgment. The authors wish to express their sincere thanks to the referees for their important comments, especially for indicating the relevant reference [3] (as an answer to Erickson [10] on the Stack Exchange discussion board) and for pointing out an error in the original manuscript.

References

1. Allauzen, C.: Calcul efficace du shuffle de k mots. Tech. rep., Institut Gaspard Monge, Université Marne-la-Vallé (2000)
2. Aoki, H., Uehara, R., Yamazaki, K.: Expected length of longest common subsequences of two biased random strings and its application. Tech. Rep. 1185, RIMS Kokyuroku (2001)
3. Buss, S., Soltys, M.: Unshuffling a square is NP-hard (2012) (submitted)
4. Choffrut, C., Karhumäki, J.: Combinatorics of Words. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Springer (1997)
5. Iwama, K.: Personal communication (2012)
6. Warmuth, M.K., Haussler, D.: On the complexity of iterated shuffle. Journal of Computer and System Sciences 28(3), 345–358 (1984)
7. Downey, R., Fellows, M.: Parameterized Complexity. Springer (1999)
8. Erdong, C., Linji, Y., Hao, Y.: Improved algorithms for 2-interval pattern problem. Journal of Combinatorial Optimization 13, 263–275 (1983)
9. Iwama, K.: Unique decomposability of shuffled strings: A formal treatment of asynchronous time-multiplexed communication. In: Proc. 15th Annual ACM Symposium on Theory of Computing (STOC), Boston, Massachusetts, USA, pp. 374–381. ACM (1983)

10. Erickson, J.: How hard is unshuffling a string? Theoretical Computer Science, http://cstheory.stackexchange.com/q/34 (version: 2010-12-01)
11. Kececioglu, J.D., Gusfield, D.: Reconstructing a history of recombinations from a set of sequences. Discrete Applied Mathematics 88(1-3), 239–260 (1998)
12. Kimura, T.: An algebraic system for process structuring and interprocess communication. In: Chandra, A., Wotschke, D., Friedman, E., Harrison, M.A. (eds.) Proc. 8th Annual ACM Symposium on Theory of Computing (STOC), Hershey, Pennsylvania, USA, pp. 92–100. ACM (1976)
13. van Leeuwen, J., Nivat, M.: Efficient recognition of rational relations. Information Processing Letters 14(1), 34–38 (1982)
14. Li, S., Li, M.: On two open problems of 2-interval patterns. Theoretical Computer Science 410(24-25), 2410–2423 (2009)
15. Lothaire, M.: Applied Combinatorics on Words. Encyclopedia of Mathematics and its Applications, vol. 105. Cambridge university press (2005)
16. Maier, D.: The complexity of some problems on subsequences and supersequences. Journal of the ACM 25(2), 322–336 (1978)
17. Mansfield, A.: On the computational complexity of a merge recognition problem. Discrete Applied Mathematics 5, 119–122 (1983)
18. Rampersad, D.H.N., Shallit, J.: Shuffling and unshuffling (2011), http://arxiv.org/abs/1106.5767
19. Spehner, J.C.: Le calcul rapide des melanges de deux mots. In: Theoretical Computer Science pp. 171–203 (1986)
20. Vialette, S.: On the computational complexity of 2-interval pattern matching problems. Theoretical Computer Science 312(2-3), 223–249 (2004)
21. Vialette, S.: Two-interval pattern problems. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, pp. 985–989. Springer (2008)

Cyclic Shift on Prefix-Free Languages

Jozef Jirásek [1,⋆] and Galina Jirásková [2,⋆⋆]

[1] Institute of Computer Science, Faculty of Science, P.J. Šafárik University,
Jesenná 5, 040 01 Košice, Slovakia
jozef.jirasek@upjs.sk
[2] Mathematical Institute, Slovak Academy of Sciences,
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk

Abstract. We prove that the cyclic shift of a prefix-free language represented by a minimal complete n-state deterministic finite automaton is recognized by a deterministic automaton of at most $(2n-3)^{n-2}$ states. We also show that this bound is tight in the quaternary case, and that it cannot be met by using any smaller alphabet. In the ternary and binary cases, we still get exponential lower bounds.

1 Introduction

Cyclic shift is a unary operation on formal languages defined as $\text{SHIFT}(L) = \{w | w = uv \text{ and } vu \in L\}$. The operation preserves regularity since the cyclic shift of a regular language may be expressed as a union of n concatenations [9]. Using such a representation, an upper bound $(n \cdot 2^n - 2^{n-1})^n$ on the state complexity of cyclic shift has been proved already by Maslov [9] in 1970. He also provided a lower bound $(n-2)^{n-2} \cdot 2^{(n-2)(n-2)}$ for incomplete deterministic automata over a growing alphabet of size $2n-2$. It follows that a lower bound for complete deterministic automata over a growing alphabet is $(n-3)^{n-3} \cdot 2^{(n-3)(n-3)}$.

The Maslov's lower bound has been improved by Jirásková and Okhotin [7] by presenting a regular language recognized by a complete n-state deterministic finite automaton, defined over a fixed four-letter input alphabet, that requires at least $(n-1)! \cdot 2^{(n-1)(n-2)}$ deterministic states for its cyclic shift. Nevertheless, the new lower bound does not match the above mentioned upper bound.

In the case of prefix-free regular languages, concatenation is a simple operation. While the state complexity of concatenation is $m \cdot 2^n - 2^{n-1}$ in the general case [9,14], it is only $m+n-2$ if the operands are prefix-free [3,6]. Now a question arises whether such an easy concatenation on prefix-free languages could be used to get the exact value of the state complexity of cyclic shift on this subclass of regular languages. In our paper, we answer this question positively, and prove the tight bound $(2n-3)^{n-2}$ on the state complexity of cyclic shift on prefix-free languages.

⋆ Research supported by grants VEGA 1/0832/12 and APVV-0035-10.
⋆⋆ Research supported by grants VEGA 2/0183/11 and APVV-0035-10.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 246–257, 2013.

Our witness languages are defined over a four-letter alphabet. We also prove the optimality of the size of an input alphabet by showing that the upper bound $(2n-3)^{n-2}$ on the state complexity of cyclic shift on prefix-free languages cannot be met by any language defined over a ternary (or any smaller) alphabet. However, in the ternary and binary cases, we still are able to prove exponential lower bounds $(n-2)! \cdot 2^{n-2}$ and $(n-2) \cdot (3^{n-2}-1)+1$, respectively. Our calculations show that these lower bounds can be exceeded.

The study of cyclic (or circular) shift has applications in coding theory. Cyclic codes are block codes, in which the cyclic shift of a codeword always yields another codeword. Thus $L = \text{SHIFT}(L)$ for a cyclic code L. It is known that the operation of cyclic shift preserves context-freeness [10,11], and that the cyclic shift of a language described by a regular expression of length n can be described by a regular expression of length $O(n^3)$ [2].

In prefix codes, like variable-length Huffman codes or country calling codes, there is no codeword that is a proper prefix of any other codeword. With such a code, a receiver can identify each codeword without any special marker between words. Motivated by prefix codes, the class of prefix-free regular languages have been recently investigated. It is known that every minimal deterministic automaton recognizing a prefix-free regular language must have exactly one final state, from which all transitions go to a dead state. Using this property, tight bounds on the state complexity of basic operations such as union, intersection, concatenation, star, and reversal have been obtained in [3] and strengthened in [6,8]. The nondeterministic state complexity of basic regular operations has been studied in [4,6], while the complexity of combined operations on prefix-free regular languages has been investigated in [5].

2 Preliminaries

We assume that the reader is familiar with basic concepts of regular languages and finite automata and for unexplained notions we refer to [12,13].

For an alphabet Σ, let Σ^* be the set of all strings over Σ, including the empty string ε. A language is any subset of Σ^*. We denote the power-set of a set X by 2^X. For an integer m, let $[m] = \{0, 1, \ldots, m-1\}$.

A *deterministic finite automaton* (DFA) is a quintuple $M = (Q, \Sigma, \cdot, s, F)$, where Q is a finite non-empty set of states, Σ is an input alphabet, $\cdot : Q \times \Sigma \to Q$ is the transition function, $s \in Q$ is the initial (start) state, and $F \subseteq Q$ is the set of final states. In this paper, all DFAs are assumed to be *complete*. The transition function $\cdot$ is extended to the domain $Q \times \Sigma^*$ in a natural way. The *language accepted by the DFA* M is the set of strings $L(M) = \{w \in \Sigma^* \mid s \cdot w \in F\}$. A state q of M is called a *dead state* if no string is accepted by M from q.

A *nondeterministic finite automaton* (NFA) is a quintuple $M = (Q, \Sigma, \cdot, S, F)$, where Q, Σ, and F are defined in the same way as for a DFA, S is the set of initial states, and $\cdot$ is the nondeterministic transition function that maps $Q \times \Sigma$

to 2^Q. The transition function can be naturally extended to the domain $2^Q \times \Sigma^*$. The *language accepted by NFA* M is $L(M) = \{w \in \Sigma^* \mid S \cdot w \cap F \neq \varnothing\}$.

Two automata are *equivalent* if they recognize the same language. A DFA M is *minimal* if every DFA equivalent to M has at least as many states as M. It is well-known that a DFA is minimal if all of its states are reachable and pairwise distinguishable. The *state complexity* of a regular language L, $\mathrm{sc}(L)$, is the number of states in the minimal DFA recognizing the language L.

The *cross-product automaton* [1] for the union of two languages recognized by DFAs $(Q_A, \Sigma, \circ, s_A, F_A)$ and $(Q_B, \Sigma, \bullet, s_B, F_B)$, respectively, is the DFA

$$(Q_A \times Q_B, \Sigma, \cdot, (s_A, s_B), F),$$

where $(p, q) \cdot a = (p \circ a, q \bullet a)$ and $F = (F_A \times Q_B) \cup (Q_A \times F_B)$.

2.1 Prefix-Free Languages

If u, v, w are strings in Σ^* and $w = uv$, then u is a *prefix* of w. If, moreover, $v \neq \varepsilon$, then u is a *proper prefix* of w. A language is *prefix-free* if it does not contain two strings, one of which is a proper prefix of the other.

It is well known that a minimal DFA recognizes a non-empty prefix-free language if and only if it has a dead state and a unique final state, from which all transitions go to the dead state.

3 Cyclic Shift on Prefix-Free Languages

The cyclic shift of a language L is defined as

$$\textsc{shift}(L) = \{uv \mid vu \in L\}.$$

Assume that the language L is recognized by a DFA A. By definition, a string w is in $\textsc{shift}(L)$ if it can be partitioned as $w = uv$ so that the string vu is in L. This means that there is a state q, such that the computation of A on the string v ends in the state q, while the string u is accepted by A from the state q. This gives the following result from [9].

Lemma 1 (Maslov [9]). *Let $A = (Q, \Sigma, \cdot, q_0, F)$ with $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ be an n-state DFA. For $i = 0, 1 \ldots, n-1$, let $B_i = (Q, \Sigma, \cdot, q_i, F)$ and $C_i = (Q, \Sigma, \cdot, q_0, \{q_i\})$ be the DFAs that have the same state set and the same transitions as the DFA A, and differ from A only in their initial and final states. Then*

$$\textsc{shift}(L(A)) = \bigcup_{i=0}^{n-1} L(B_i)\, L(C_i).$$

3.1 Upper Bound for Cyclic Shift on Prefix-Free Languages

Using the above mentioned Maslov's result we now get an upper bound on the number of states of deterministic finite automata recognizing the cyclic shift of prefix-free languages.

Lemma 2 (Upper Bound). *Let $n \geq 3$ and let L be a prefix-free language accepted by a minimal n-state DFA. Then the language* $\text{SHIFT}(L)$ *is accepted by a DFA of at most $(2n-3)^{n-2}$ states.*

Proof. Let $A = (Q, \Sigma, \cdot, q_0, \{q_{n-2}\})$ with $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ be a minimal DFA for a prefix-free language L, in which q_{n-1} is the dead state, and q_{n-2} is the sole final state. Then, by Lemma 1, $\text{SHIFT}(L) = \cup_{i=0}^{n-1} L(B_i)L(C_i)$, where $B_i = (Q, \Sigma, \cdot, q_i, \{q_{n-2}\})$ and $C_i = (Q, \Sigma, \cdot, q_0, \{q_i\})$. Since q_{n-1} is the dead state of A, and all transitions defined in the unique final state q_{n-2} go to the dead state q_{n-1}, the language $L(B_{n-1})$ is empty and $L(B_{n-2}) = \{\varepsilon\}$. Therefore, the language $L(B_{n-1})L(C_{n-1})$ is empty and

$$L(B_{n-2})L(C_{n-2}) = L(C_{n-2}) = L(B_1) \subseteq L(B_1)L(C_1)$$

since $\varepsilon \in L(C_1)$. Hence $\text{SHIFT}(L) = \cup_{i=0}^{n-3} L(B_i)L(C_i)$.

For $i = 0, \ldots, n-3$, the language $L(B_i)L(C_i)$ is accepted by a DFA D_i obtained from the DFAs B_i and C_i as follows. First, since all transitions defined in the unique final state q_{n-2} of B_i go to the dead state, the state q_{n-2} can be merged with the initial state q_0 of C_i. Next, the state q_{n-1} in B_i as well as the states q_{n-1} and q_{n-2} in C_i are all dead, and therefore can be merged into a single dead state. The resulting DFA D_i is deterministic and has $2n-3$ states.

Now the language $\text{SHIFT}(L) = \cup_{i=0}^{n-3} L(B_i)L(C_i)$ is accepted by the cross-product automaton $D_0 \times D_1 \times \cdots \times D_{n-3}$ that has at most $(2n-3)^{n-2}$ states. The construction is illustrated in Fig. 1. □

3.2 Lower Bound in Quaternary Case

Throughout this subsection assume that $n \geq 4$ and $\Sigma = \{a, b, c, d\}$. Recall that $[m] = \{0, 1, \ldots, m-1\}$. Our aim is to prove that the upper bound on the state complexity of cyclic shift of prefix-free languages given in the previous lemma is tight in the case of a four-letter alphabet.

To this aim define a quaternary n-state DFA $A = ([m+2], \Sigma, \cdot, 0, \{m\})$, where $m = n-2$. For each state i in $[m]$,

$$\begin{aligned}
i \cdot a &= i + 1 \bmod m, \\
i \cdot b &= \begin{cases} 1, \text{ if } i = 0, \\ 0, \text{ if } i = 1, \\ i, \text{ otherwise}, \end{cases} \\
i \cdot c &= \begin{cases} 0, \text{ if } i \in \{0, 1\}, \\ i, \text{ otherwise}, \end{cases} \\
i \cdot d &= i + 1,
\end{aligned}$$

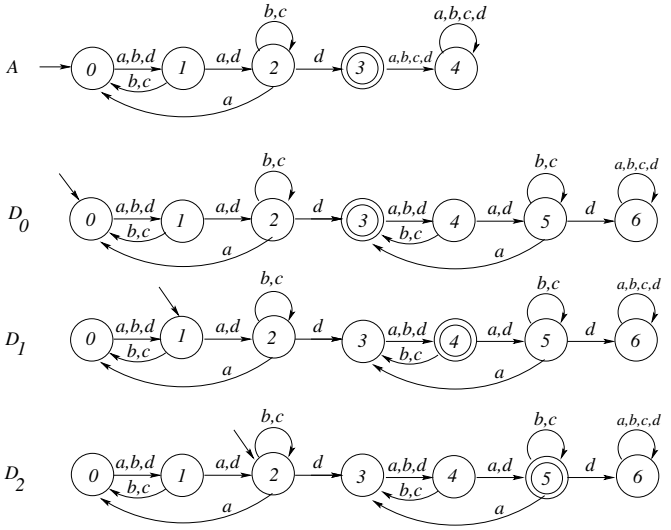


Fig. 1. A five-state DFA A and the resulting DFAs D_i for $L(B_i)L(C_i)$ for $i = 0, 1, 2$

and $m \cdot \sigma = (m+1) \cdot \sigma = m+1$ for each input σ in Σ. The DFA A is depicted in Fig. 2. Since all transitions defined in the unique final state m go to the dead state $m+1$, the language $L(A)$ is prefix-free.

Note that on states in $[m]$, input a causes a great permutation, input b causes a transposition, and input c causes a contraction. Thus, the semigroup of functions of $[m]$ into itself is generated by the inputs a, b, c.

For $i = 0, 1, \ldots, m-1$, construct the DFA $D_i = ([2m+1], \Sigma, \circ, i, \{m+i\})$ accepting the language $L(B_i)L(C_i)$ as described in the proof of the previous lemma. All the automata D_i's have the same transition function $\circ$, defined by $2m \circ \sigma = 2m$ and $i \circ \sigma = (m+i) \circ \sigma = i \cdot \sigma$ for each i in $[m]$ and σ in Σ, and these automata differ only in the initial and final states. Fig. 1 shows the DFAs D_0, D_1, and D_2 corresponding to the DFA A in the case of $m = 3$.

Then the language $\text{SHIFT}(L) = \cup_{i=0}^{m-1} L(D_i)$ is recognized by the cross-product automaton $D_0 \times \cdots \times D_{m-1}$. Our aim is to prove that the cross-product automaton has $(2n-3)^{n-2} = (2m+1)^m$ reachable and pairwise distinguishable states.

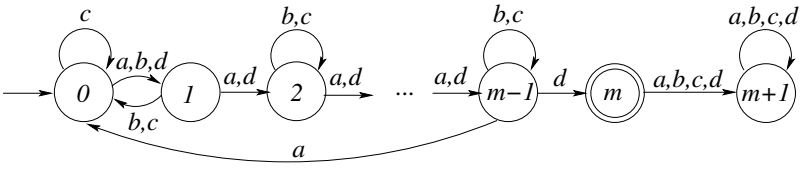


Fig. 2. The quaternary n-state witness DFA A; $m = n-2$

Let us start with reachability. The state set of the cross-product automaton consists of m-tuples in $[2m+1]^m$, and the initial m-tuple is $(0, 1, \ldots, m-1)$. The next lemma shows that every m-tuple in $[2m+1]^m$ is reachable.

Lemma 3 (Reachability). *Every m-tuple in $[2m+1]^m$ is reachable in the cross-product automaton $D_0 \times D_1 \times \cdots \times D_{m-1}$.*

Proof. Let $(k_0, k_1, \ldots, k_{m-1})$ be an arbitrary but fixed m-tuple in $[2m+1]^m$. We will show that there is a string w that moves the cross-product automaton from its initial state $(0, 1, \ldots, m-1)$ to the state $(k_0, k_1, \ldots, k_{m-1})$.

For the m-tuple $(k_0, k_1, \ldots, k_{m-1})$, consider the two disjoint sets of indices I and J defined by

$$I = \{i \in [m] \mid k_i = 2m\},$$
$$J = \{i \in [m] \mid m \leq k_i \leq 2m-1\},$$

that is, the i-th component of the m-tuple is the dead state $2m$ of D_i whenever $i \in I$, it is a state in $\{m, m+1, \ldots, 2m-1\}$ whenever $i \in J$, and it is a state in $[m]$ otherwise. Next, define a function $f : [m] \to [m]$ by

$$f(i) = \begin{cases} 1, & \text{if } i \in I, \\ k_i - m, & \text{if } i \in J, \\ k_i, & \text{otherwise.} \end{cases}$$

Since the symbols a, b, c perform the three basic functions on $[m]$ in the DFA A, there is a string v_f over $\{a, b, c\}$ that moves every state i in $[m]$ to $f(i)$ in A.

Finally, for each ℓ in $[m]$ consider the string $u_\ell = a^{m-1-\ell}\, d\, a^\ell$, and define

$$w = (\prod_{i \in I} u_i u_i \prod_{i \in J} u_i) \cdot v_f, \tag{1}$$

where $\prod$ stands for concatenation (in an arbitrary order).

Our goal is to prove that w is the desired string that moves the cross-product automaton from the initial state $(0, 1, \ldots, m-1)$ to the state $(k_0, k_1, \ldots, k_{m-1})$.

First, notice that each D_ℓ goes from its initial state ℓ to the state $m + \ell$ by the string $u_\ell = a^{m-1-\ell} d a^\ell$ and then to the dead state $2m$ by the next u_ℓ since

$$\ell \xrightarrow{a^{m-1-\ell}} m-1 \xrightarrow{d} m \xrightarrow{a^\ell} m+\ell \xrightarrow{a^{m-1-\ell}} 2m-1 \xrightarrow{d} 2m \xrightarrow{a^\ell} 2m.$$

On the other hand, if $j \neq \ell$, then D_j remains in its initial state j upon reading u_ℓ since D_j moves by $a^{m-1-\ell}$ from j to state $(j + m - 1 - \ell) \bmod m$, in which the transition on d is defined the same way as on a, and therefore reading u_ℓ from j with $j \neq \ell$ results in the same state as reading a^m from j:

$$j \xrightarrow{a^{m-1-\ell}} (j+m-1-\ell) \bmod m \xrightarrow{d} (j+m-\ell) \bmod m \xrightarrow{a^\ell} j.$$

Now consider the string $\prod_{i \in I} u_i u_i \prod_{i \in J} u_i$, that is, the first part of the string w in (1). Recall that the sets of indices I and J are disjoint, and therefore

- every D_ℓ with $\ell \in I$ goes from ℓ to $2m$ by $\prod_{i\in I} u_i u_i$ and remains in $2m$ upon reading $\prod_{i\in J} u_i$;
- every D_ℓ with $\ell \in J$ remains in its initial state ℓ upon reading $\prod_{i\in I} u_i u_i$ and then goes to $m+\ell$ by $\prod_{i\in J} u_i$;
- every D_ℓ with $\ell \notin I \cup J$ remains in ℓ upon reading $\prod_{i\in I} u_i u_i \prod_{i\in J} u_i$.

It follows that the string $\prod_{i\in I} u_i u_i \prod_{i\in J} u_i$ moves the cross-product automaton from its initial state $(0, 1, \ldots, m-1)$ to the state $(k'_0, k'_1, \ldots, k'_{m-1})$, where

$$k'_\ell = \begin{cases} 2m, & \text{if } \ell \in I, \\ m+\ell, & \text{if } \ell \in J, \\ \ell, & \text{otherwise.} \end{cases}$$

Then, after reading the second part of the string w in (1), that is the string v_f, which moves every state i in $[m]$ to state $f(i)$ in the DFA A, each dfa D_ℓ with $\ell \in I$ remains in its dead state $2m$, each dfa D_ℓ with $\ell \in J$ goes from $m+\ell$ to $m + f(\ell) = m + (k_\ell - m) = k_\ell$, while each DFA D_ℓ with $\ell \notin I \cup J$ goes from ℓ to $f(\ell) = k_\ell$.

Hence the string $w = (\prod_{i\in I} u_i u_i \prod_{i\in J} u_i) \cdot v_f$ moves the cross-product automaton from its initial state $(0, 1, \ldots, m-1)$ to the state $(k_0, k_1, \ldots, k_{m-1})$. This proves the lemma. □

The following lemma proves the distinguishability of all the states in the cross-product automaton. Note that only symbols a and d are needed to get this result, which will be used later in the paper when dealing with smaller alphabets.

Lemma 4 (Distinguishability). *Every two distinc states of the cross-product-product automaton $D_0 \times D_1 \times \cdots \times D_{m-1}$ can be distinguished by a string over $\{a, d\}$.*

Proof. Let $(k_0, k_1, \ldots, k_{m-1})$ and $(k'_0, k'_1, \ldots, k'_{m-1})$ be two distinct m-tuples in $[2m+1]^m$. Then there is an i in $[m]$ with $k_i \neq k'_i$, and without loss of generality we may assume that $k_i \neq 2m$. Set

$$w = d^{2m-1-k_i}\, a\, d^{m-1}\, a\, d^{m-1}\, a^{i+1},$$

and let us show that the string w is accepted by the cross-product automaton from $(k_0, k_1, \ldots, k_i, \ldots, k_{m-1})$ and rejected from $(k'_0, k'_1, \ldots, k'_i, \ldots, k'_{m-1})$.

The DFA D_i goes from k_i to the accepting state $m+i$ by the string w since

$$k_i \xrightarrow{d^{2m-1-k_i}} 2m-1 \xrightarrow{a} m \xrightarrow{d^{m-1}} 2m-1 \xrightarrow{a} m \xrightarrow{d^{m-1}} 2m-1 \xrightarrow{a} m \xrightarrow{a^i} m+i.$$

Therefore, the string w is accepted by the cross-product automaton from the state $(k_0, k_1, \ldots, k_i, \ldots, k_{m-1})$.

On the other hand, let us show that the DFA D_i rejects the string w from each state ℓ different from k_i. If $\ell > k_i$, the D_i moves from ℓ to the dead state $2m$

by w since it is already in $2m$ after reading d^{2m-1-k_i}. If $\ell < k_i$, then D_i moves from ℓ to $\ell' = \ell + 2m - 1 - k_i$ by d^{2m-1-k_i}. If $m \leq \ell' < 2m-1$ or $\ell' < m-1$, then D_i moves from ℓ' to the dead state $2m$ by ad^{m-1} or $ad^{m-1}ad^{m-1}$, respectively. If $\ell' = m-1$, the D_i moves from ℓ' to its rejecting state i by $ad^{m-1}ad^{m-1}a^{i+1}$. Hence D_i rejects the string w from each state ℓ with $\ell \neq k_i$.

The transitions in each D_j with $j \neq i$ are the same as in D_i, however, the states $m+i$ and i are rejecting in D_j. Therefore, the DFA D_j rejects the string w from each of its states.

Thus the cross-product automaton rejects w from $(k'_0, k'_1, \ldots, k'_i, \ldots, k'_{m-1})$, which concludes the proof. □

Hence, in the quaternary case, we get a lower bound that matches our upper bound $(2n-3)^{n-2}$ given by Lemma 2. Our next aim is to show that the four-letter alphabet cannot be decreased, that is, to show that the upper bound cannot be met by using any smaller alphabet. On the other hand, we will get still exponential lower bounds in the ternary and binary cases.

3.3 Small Alphabets

Let us start with an upper bound in the ternary case.

Lemma 5. *Let $n \geq 5$. If L is a prefix-free language recognized by a minimal n-state DFA over a ternary input alphabet, then the minimal DFA for* SHIFT(L) *has less than $(2n-3)^{n-2}$ states.*

Proof. Let L be accepted by a minimal n-state DFA A over the alphabet $\{a, b, c\}$. Let $m = n-2$. Let the state set of A be $[m+2]$, with the unique final state m and the dead state $m+1$. Then, since the final state m is reachable in A, there must be a symbol σ in $\{a, b, c\}$ and a state j, from which A goes to m by σ. Without loss of generality, we may assume that $\sigma = c$.

Let $D_0 \times \cdots \times D_{m-1}$ be the cross-product automaton for SHIFT(L) described above. Consider those of its states, in which all the components are less than m, that is, the states in $[m]^m$, and let us show that at least one of them must be unreachable in the cross-product automaton.

For each permutation φ of $[m]$, the state $(\varphi(0), \varphi(1), \ldots, \varphi(m-1))$ may only be reached from the initial state $(0, 1, \ldots, m-1)$ by reading a string w over $\{a, b, c\}$, in which all symbols permute the set $[m]$ in the DFA A. Therefore, no c occurs in w. To reach all such permutation states, the symbols a and b must cause two permutations on $[m]$ generating the group of all permutations on $[m]$ since $m \geq 3$. However, in such a case, no state $(f(0), f(1), \ldots, f(m-1))$ in $[m]^m$, where f is a function from $[m]$ to $[m]$ which is not a permutation, can be reached in the cross-product automaton.

If at least one of the symbols a or b does not cause a permutation on $[m]$, then it is not possible to reach all the states $(\varphi(0), \varphi(1), \ldots, \varphi(m-1))$ where φ is a permutation on $[m]$ and $m \geq 3$. This concludes the proof. □

Now, using a subautomaton of our quaternary witness automaton defined in subsection 3.2 and shown in Fig. 2, we prove an exponential lower bound for the ternary case.

Lemma 6. *For every n with $n \geq 4$, there exists a prefix-free language recognized by an n-state DFA over a ternary alphabet such that every DFA for the language* $\textsc{shift}(L)$ *requires at least $(n-2)!\,2^{n-2}$ states.*

Proof. Consider the DFA B obtained from the DFA A in Fig. 2 by considering only the input symbols a, b, d. Since the symbols a and b cause a great permutation and a transposition on $[m]$, respectively, for each permutation φ on $[m]$, there is a string v_φ over $\{a, b\}$ that moves every state i in $[m]$ to the state $\varphi(i)$.

As shown in the proof of Lemma 3, for each set J of $[m]$ and each permutation φ on $[m]$, the state $(k_0, k_1, \ldots, k_{m-1})$ with

$$k_i = \begin{cases} m + \varphi(i), & \text{if } i \in J, \\ \varphi(i), & \text{otherwise} \end{cases}$$

is reached in the cross-product automaton from the initial state $(0, 1, \ldots, m-1)$ by the string

$$\prod_{i \in J} (a^{m-1-i}\, d\, a^i) \cdot v_\varphi.$$

This gives $(n-2)!\,2^{n-2}$ reachable states. All these states are pairwise distinguishable by Lemma 4. □

Let us continue with the binary case. By using another subautomaton of our quaternary witness, the next lemma shows that the lower bound on the state complexity of cyclic shift of prefix-free languages is exponential even in the case of a two-letter alphabet.

Lemma 7. *For every n with $n \geq 4$, there exists a prefix-free language recognized by an n-state DFA over a binary alphabet such that every DFA for the language* $\textsc{shift}(L)$ *requires at least $(n-2)(3^{n-2}-1)+1$ states.*

Proof. Consider the DFA C obtained from the DFA A in Fig. 2 by considering only the input symbols a and d. Recall that $m = n - 2$.

There are 3^m possibilities of choosing two disjoint subsets I and J of $[m]$. For each of them, as shown in the proof of Lemma 3, the state $(k_0, k_1, \ldots, k_{m-1})$ with

$$k_i = \begin{cases} 2m, & \text{if } i \in I, \\ m + i, & \text{if } i \in J, \\ i, & \text{otherwise} \end{cases}$$

is reached in the cross-product automaton from the initial state $(0, 1, \ldots, m-1)$ by the string

$$\prod_{i \in I} (u_i\, u_i) \prod_{i \in J} u_i$$

with $u_i = a^{m-1-i}\, d\, a^i$. From every such state $(k_0, k_1, \ldots, k_{m-1})$, except for the state with $I = [m]$, the cross-product automaton moves after reading a^j with j in $[m]$ to the state $(k'_0, k'_1, \ldots, k'_{m-1})$ with

$$k_i = \begin{cases} 2m, & \text{if } i \in I, \\ m + (i+j) \bmod m, & \text{if } i \in J, \\ (i+j) \bmod m, & \text{otherwise.} \end{cases}$$

This gives $(n-2)(3^{n-2}-1)+1$ reachable states. The distinguishability again follows from Lemma 4. □

Recall that the state complexity of a regular language L, $\mathrm{sc}(L)$, is defined as the smallest number of states in any DFA recognizing the language L. Denote by $f_k(n)$ the state complexity function of cyclic shift on prefix-free regular languages over a k-letter alphabet defined by

$$f_k(n) = \max\{\mathrm{sc}(\textsc{shift}(L)) \mid L \subseteq \Sigma^*, |\Sigma| = k, L \text{ is prefix-free, and } \mathrm{sc}(L) = n\}.$$

Using this notation, we can summarize our results in the following theorem.

Theorem 1 (State Complexity). *Let $n \geq 5$ and $f_k(n)$ be the state complexity of cyclic shift on prefix-free regular languages over a k-letter alphabet. Then*

(i) $f_1(n) = n$;
(ii) $f_2(n) \geq (n-2)(3^{n-2}-1)+1$;
(iii) $(n-2)! \cdot 2^{n-2} \leq f_3(n) < (2n-3)^{n-2}$;
(iv) $f_4(n) = f_k(n) = (2n-3)^{n-2}$ *for every k with $k \geq 4$.*

Proof. The equality in (i) holds since the cyclic shift of every unary language is the same language. The lower bound on $f_2(n)$ in (ii) is given by Lemma 7, while the bounds on $f_3(n)$ in (iii) follow from Lemmata 5 and 6. The upper bound on $f_k(n)$ in (iv) is given by Lemma 2, and its tightness for $k = 4$ is proved in Lemmata 3 and 4. Since adding new symbols to the quaternary witness automata does not change the proofs of reachability and distinguishability in the quaternary case, the upper bound is tight for every k with $k \geq 4$. □

Hence the tight bound on the state complexity of cyclic shift on prefix-free languages over an alphabet of at least four symbols is $(2n-3)^{n-2}$. Moreover, the alphabet of size at least four is necessary for the tightness. Using any smaller alphabet, the upper bound $(2n-3)^{n-2}$ cannot be met. However, the lower bounds in the binary and ternary cases are still exponential, namely $(n-2)\cdot(3^{n-2}-1)+1$ and $(n-2)! \cdot 2^{n-2}$, respectively. Our calculations given in Table 1 show that the state complexity of cyclic shift on prefix-free languages in the binary and ternary cases is greater than the above mentioned lower bounds. Its exact value in these two cases remains open. The hardest binary and ternary automata for $n = 4, 5, 6, 7$ are shown in Fig. 3 and Fig. 4, respectively.

Table 1. The state complexity of cyclic shift on prefix-free languages

n	$f_2(n)$	$f_3(n)$	$f_4(n) = (2n-3)^{n-2}$
4	17	25	25
5	121	319	343
6	1709	6193	6561
7	36256	154976	161051

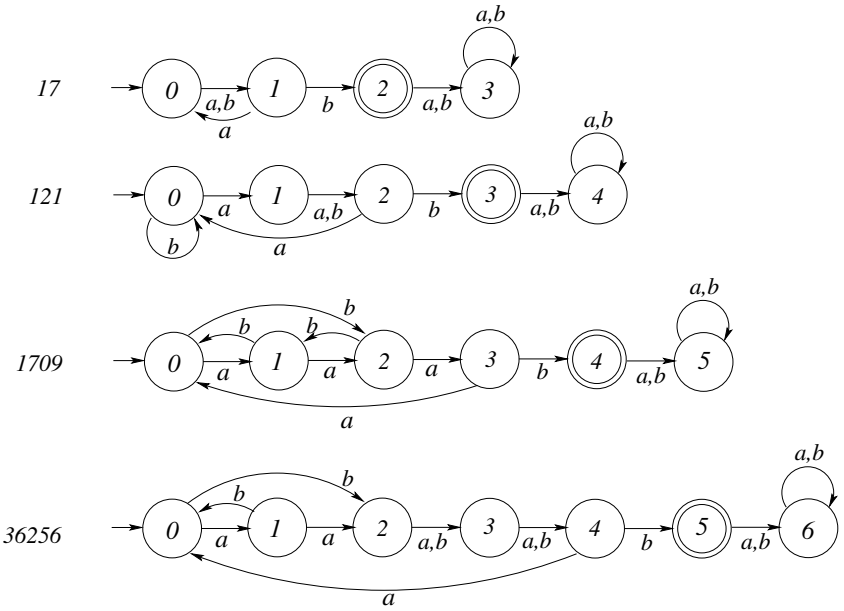


Fig. 3. The hardest binary DFAs; $n = 4, 5, 6, 7$

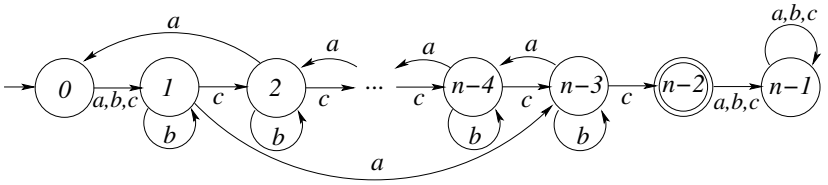


Fig. 4. The hardest ternary DFAs; for $n = 4, 5, 6, 7$

4 Conclusions

We investigated the state complexity of cyclic shift operation in the class of prefix-free regular languages. We obtained the upper bound $(2n-3)^{n-2}$, and we showed that it is tight in the case of a four-letter alphabet. We also proved that this upper bound cannot be met by any prefix-free language defined over a smaller alphabet. In the ternary and binary cases, we were still able to get exponential lower bounds $(n-2)! \cdot 2^{n-2}$ and $(n-2) \cdot (3^{n-2}-1)+1$, respectively. Our calculations showed that these lower bounds can be exceeded.

Notice that for incomplete deterministic finite automata, the tight bound for an alphabet of at least four symbols is $(2n-1)^{n-1}-1$.

The exact values of the state complexity of cyclic shift on binary and ternary prefix-free languages remain open, and are of interest to us. We also conjecture that the state complexity of cyclic shift on prefix-free languages in the binary case is smaller than that in the ternary case.

References

1. Birget, J.-C.: Intersection and union of regular languages and state complexity. Inform. Process. Letters 43, 185–190 (1992)
2. Gruber, H., Holzer, M.: Language operations with regular expressions of polynomial size. Theoret. Comput. Sci. 410, 3281–3289 (2009)
3. Han, Y.-S., Salomaa, K., Wood, D.: Operational state complexity of prefix-free regular languages. In: Automata, Formal Languages, and Related Topics, pp. 99–115. University of Szeged, Hungary (2009)
4. Han, Y.-S., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-free regular languages. Fund. Inform. 90, 93–106 (2009)
5. Han, Y.-S., Salomaa, K., Yu, S.: State complexity of combined operations for prefix-free regular languages. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 398–409. Springer, Heidelberg (2009)
6. Jirásková, G., Krausová, M.: Complexity in prefix-free regular languages. In: McQuillan, I., Pighizzini, G., Trost, B. (eds.) Proc. 12th DCFS, pp. 236–244. University of Saskatchewan, Saskatoon (2010)
7. Jirásková, G., Okhotin, A.: State complexity of cyclic shift. Theor. Inform. Appl. 42, 335–360 (2008)
8. Krausová, M.: Prefix-free regular languages: Closure properties, difference, and left quotient. In: Kotásek, Z., Bouda, J., Černá, I., Sekanina, L., Vojnar, T., Antoš, D. (eds.) MEMICS 2011. LNCS, vol. 7119, pp. 114–122. Springer, Heidelberg (2012)
9. Maslov, A.N.: Estimates of the number of states of finite automata. Soviet Math. Dokl. 11, 1373–1375 (1970)
10. Maslov, A.N.: The cyclic shift of languages. Problemy Peredači Informacii 9, 81–87 (1973) (Russian)
11. Oshiba, T.: Closure property of the family of context-free languages under the cyclic shift operation. Electron. Commun. Japan 55, 119–122 (1972)
12. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company, Boston (1997)
13. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, ch. 2, pp. 41–110. Springer, Heidelberg (1997)
14. Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)

Weak Abelian Periodicity of Infinite Words

Sergey Avgustinovich[1] and Svetlana Puzynina[1,2,⋆]

[1] Sobolev Institute of Mathematics, Russia
avgust@math.nsc.ru
[2] University of Turku, Finland
svepuz@utu.fi

Abstract. We say that an infinite word w is *weakly abelian periodic* if it can be factorized into finite words with the same frequencies of letters. In this paper we study properties of weak abelian periodicity, and its relations with balance and frequency. We establish necessary and sufficient conditions for weak abelian periodicity of fixed points of uniform binary morphisms. Also, we discuss weak abelian periodicity in minimal subshifts.

The study of abelian properties of words dates back to Erdös's question whether there is an infinite word avoiding abelian squares [7]. Abelian powers and their avoidability in infinite words is a natural generalization of analogous questions for ordinary powers. The answer to Erdös's question was given by Keränen, who provided a construction of an abelian square-free word [10]. From that time till nowadays, many problems concerning different abelian properties of words have been studied, including abelian periods, abelian powers, avoidability, complexity (see, e. g., [2], [4], [5], [13]).

Two words are said to be abelian equivalent, if they are permutations of each other. Similarly to usual powers, an abelian k-power is a concatenation of k abelian equivalent words. We define a *weak abelian power* as a concatenation of words with the same frequencies of letters. So, in a weak abelian power we admit words with different lengths; if all words are of the same length, then we have an abelian power. Earlier some questions about avoidability of weak abelian powers have been considered. In [11] for given integer k the author finds an upper bound for length of binary word which does not contain weak abelian k-powers. In [8] the authors build an infinite ternary word having no weak abelian $(5^{11}+1)$-powers.

The notion of abelian period is a generalization of the regular notion of period, and it is closely related to abelian powers. A periodic infinite word can be defined as an infinite power. Similarly, we say that a word is (weakly) abelian periodic, if it is a (weak) abelian ∞-power. In the paper we study the property of weak abelian periodicity for infinite words, in particular, its connections with related notions of balance and frequency. We establish necessary and sufficient conditions for weak abelian periodicity of fixed points of uniform binary morphisms. Also, we discuss weak abelian periodicity in minimal subshifts.

⋆ Supported in part by the Academy of Finland under grant 251371, by Russian Foundation of Basic Research (grant 12-01-00448), and by RF President grant MK-4075.2012.1.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 258–270, 2013.

The paper is organized as follows. In Section 2 we fix our terminology, in Section 3 we discuss some general properties of weak abelian periodicity and its connections with other notions, such as balance and frequencies of letters. In Section 4 we give a criteria for weak abelian periodicity of fixed points of primitive binary uniform morphisms. In Section 5 we study weak abelian periodicity of points in shift orbit closure of uniform recurrent words.

1 Preliminaries

In this section we give some basics on words following terminology from [12] and introduce the concepts used in this paper.

Given a finite non-empty set Σ (called the alphabet), we denote by Σ^* and Σ^ω, respectively, the set of finite words and the set of (right) infinite words over the alphabet Σ. Given a finite word $u = u_1u_2 \dots u_n$ with $n \geq 1$ and $u_i \in \Sigma$, we denote the length n of u by $|u|$. The empty word will be denoted by ε and we set $|\varepsilon| = 0$.

Given words w, x, y, z such that $w = xyz$, x is called a *prefix*, y is a *factor* and z a *suffix* of w. The factor of w starting at position i and ending at position j will be denoted by $w[i, j] = w_iw_{i+1} \dots w_j$. The prefix (resp., suffix) of length n of w is denoted $\mathrm{pref}_n(w)$ (resp., $\mathrm{suff}_n(w)$). The set of all factors of w is denoted by $F(w)$, the set of all factors of length n of w is denoted by $F_n(w)$.

An infinite word w is *ultimately periodic*, if for some finite words u and v it holds $w = uv^\omega$; w is *purely periodic* (or briefly *periodic*) if $u = \varepsilon$. An infinite word is *aperiodic* if it is not ultimately periodic.

An infinite word $w = w_1w_2 \dots$ is *recurrent* if any of its factors occurs infinitely many times in it. The word w is *uniformly recurrent* if for each its factor u there exists C such that whenever $w[i, j] = u$, there exists $0 < k \leq C$ such that $w[i, j] = w[i + k, j + k] = u$. In other words, factors occur in w in a bounded gap.

Given a finite word $u = u_1u_2 \dots u_n$ with $n \geq 1$ and $u_i \in \Sigma$, for each $a \in \Sigma$, we let $|u|_a$ denote the number of occurrences of the letter a in u. Two words u and v in Σ^* are *abelian equivalent*, denoted $u \sim_{ab} v$, if and only if $|u|_a = |v|_a$ for all $a \in \Sigma$. It is easy to see that abelian equivalence is indeed an equivalence relation on Σ^*.

An infinite word w is called *abelian (ultimately) periodic*, if $w = v_0v_1 \dots$, where $v_k \in \Sigma^*$ for $k \geq 0$, and $v_i \sim_{ab} v_j$ for all integers $i, j \geq 1$.

For a finite word $w \in \Sigma^*$, we define *frequency* $\rho_a(w)$ of a letter $a \in \Sigma$ in w as $\rho_a(w) = \frac{|w|_a}{|w|}$.

Definition 1. *An infinite word w is called* weakly abelian (ultimately) periodic, *if $w = v_0v_1 \dots$, where $v_i \in \Sigma^*$, $\rho_a(v_i) = \rho_a(v_j)$ for all $a \in \Sigma$ and all integers $i, j \geq 1$.*

In other words, a word is weakly abelian periodic if it can be factorized into words of possibly different lengths with the same frequencies of letters. In what follows we usually omit the word "ultimately", meaning that there can be a

prefix with different frequencies. Also, we often write WAP instead of weakly abelian periodic for brevity.

Definition 2. *An infinite word w is called* bounded weakly abelian periodic*, if it is weakly abelian periodic with bounded lengths of blocks, i. e., there exists C such that for every i we have $|v_i| \leq C$.*

We mainly focus on binary words, but we also make some observations in the case of general alphabet. One can consider the following geometric interpretation of weak abelian periodicity. Let $w = w_1 w_2 \dots$ be an infinite word over a finite alphabet Σ. We translate w to a graph visiting points of the infinite rectangular grid by interpreting letters of w by drawing instructions. In the binary case, we assign 0 with a move by vector $\mathbf{v}_0 = (1, -1)$, and 1 with a move $\mathbf{v}_1 = (1, 1)$. We start at the origin $(x_0, y_0) = (0, 0)$. At step n, we are at a point (x_{n-1}, y_{n-1}) and we move by a vector corresponding to the letter w_n, so that we come to a point $(x_n, y_n) = (x_{n-1}, y_{n-1}) + v_{w_n}$, and the two points (x_{n-1}, y_{n-1}) and (x_n, y_n) are connected with a line segment. Thus, we translate the word w to a path in $\mathbb{Z}^2$. We denote corresponding graph by g_w. Therefore, for any word w, its graph is a piece-wise linear function with linear segments connecting integer points (see Example 1). It is easy to see that weakly abelian periodic word w has graph with infinitely many integer points on a line with rational slope (we will sometimes write that w is WAP along this line). A bounded weakly abelian periodic word has a graph with bounded differences between letters. Note also that instead of vectors $(1, -1)$ and $(1, 1)$ one can use any other pair of noncollinear vectors $\mathbf{v}_0$ and $\mathbf{v}_1$, and sometimes it will be convenient for us to do so. For a k-letter alphabet one can consider a similar graph in $\mathbb{Z}^k$. Note that the graph can also be defined for finite words in a similar way, and we will sometimes use it.

Definition 3. *We say that a word w is of* bounded width*, if there exist two lines with the same rational slope, so that the path corresponding to w lies between these two lines. Formally, there exist rational numbers a, b_1, b_2, so that $ax + b_1 \leq g_w(x) \leq ax + b_2$.*

Note that we focus on rational a, because words of bounded irrational width cannot be weakly abelian periodic. Equivalently, bounded width means that graph of the word lies on finitely many lines with rational coefficients.

We will also need the notions of frequency and balance, which are closely related to abelian periodicity. Relations between these notions are discussed in the next section. A word w is called *C-balanced* if for each two its factors u and v of equal length $||u|_a - |v|_a| \leq C$ for any $a \in \Sigma$. Actually, the notion of bounded width is equivalent to the notion of balance (see, e.g., [1]). We say that a letter $a \in \Sigma$ has *frequency* $\rho_a(w)$ in an infinite word w if $\rho_a(w) = \lim_{n\to\infty} \rho_a(\mathrm{pref}_n(w))$. Note that for some words the limit does not exist, and we say that such words do not have letter frequencies. Note also that we define here a prefix frequency, though sometimes another version of frequency of letters in words is studied (see Section 5 for definitions). Observe that if a WAP word has a frequency of a letter, then this frequency coincides with the frequency of this letter in factors of corresponding factorization.

A *morphism* is a function $\varphi : \Sigma^* \to B^*$ such that $\varphi(\varepsilon) = \varepsilon$ and $\varphi(uv) = \varphi(u)\varphi(v)$, for all $u, v \in \Sigma^*$. Clearly, a morphism is completely defined by the images of the letters in the domain. For most of morphisms we consider, $\Sigma = B$. A morphism is *primitive*, if there exists k such that for every $a \in \Sigma$ the image $\varphi^k(a)$ contains all letters from B. A morphism is *uniform*, if $|\varphi(a)| = |\varphi(b)|$ for all $a, b \in \Sigma$, and prolongeable on $a \in \Sigma$, if $|\varphi(a)| \geq 2$ and $a = \mathrm{pref}_1(\varphi(a))$. If φ is prolongeable on a, then $\varphi^n(a)$ is a proper prefix of $\varphi^{n+1}(a)$, for all $n \in \mathbb{N}$. Therefore, the sequence $(\varphi^n(a))_{n\geq 0}$ of words defines an infinite word w that is a fixed point of φ.

Remind the definition of Toeplitz words. Let ? be a letter not in Σ . For a word $w \in \Sigma(\Sigma \cup \{?\})^*$, let

$$T_0(w) =?^\omega, T_{i+1}(w) = F_w(T_i(w)),$$

where $F_w(u)$, defined for any $u \in (\Sigma \cup \{?\})^\omega$, is the word obtained from w^ω by replacing the sequence of all occurrences of ? by u; in particular, $F_w(u) = w^\omega$ if w contains no ?.

Clearly,

$$T(w) = \lim_{i\to\infty} T_i(w) \in \Sigma^\omega$$

is well-defined, and it is referred to as the *Toeplitz word* determined by the pattern w. Let $p = |w|$ and $q = |w|_?$ be the length of w and the number of ?'s in w, respectively. Then $T(w)$ is called a (p, q)-Toeplitz word (see, e. g., [3]).

Example 1. Paperfolding word:

$$0010011000110110001001110011011 0 \ldots$$

This word can be defined, e.g., as a Toeplitz word with pattern $w = 0?1?$. The graph corresponding to the paperfolding word with $\mathbf{v}_0 = (1, -1)$, $\mathbf{v}_1 = (1, 1)$ is in Fig. 1. The paperfolding word is not balanced and is WAP along the line $y = -1$ (and actully along any line $y = C$, $C = -1, -2, \ldots$). See Proposition 2 (2) for details.

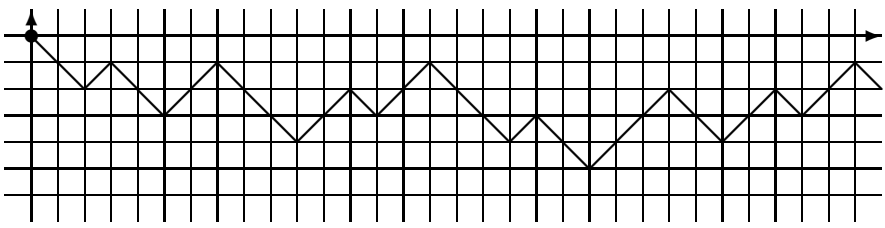


Fig. 1. The graph of the paperfolding word with $\mathbf{v}_0 = (1, -1)$, $\mathbf{v}_1 = (1, 1)$

Example 2. A word obtained as an image of the morphism $0 \mapsto 01$, $1 \mapsto 0011$ of any nonperiodic binary word is bounded WAP.

2 General Properties of Weak Abelian Periodicity

In this section we discuss the relations between notions defined in the previous section and observe some simple properties of weak abelian periodicity. We start with the property of bounded width and its connections to weak abelian periodicity.

Proposition 1. *1. If an infinite word w is of bounded width, then w is WAP.*
2. There exists an infinite word w of bounded width which is not bounded WAP.
3. If an infinite word w is bounded WAP, then w is of bounded width.

Proof. 1. Since w is of bounded width, its graph lies on a finite number of lines with rational coefficients. By the pigeonhole principle it has infinitely many points on one of these lines and hence is WAP.

2. Consider

$$w = 011101000101011101010\cdots = (01)^1 1 (10)^2 0 (01)^3 1 (10)^4 \ldots (01)^{2i-1} 1 (10)^{2i} 0 \ldots$$

Taking its graph with $\mathbf{v}_0 = (-1, 1)$ and $\mathbf{v}_1 = (1, 1)$ we see that it lies on the lines $y = 0, -1, 1, 2$ and hence w is of bounded width. The graph intersects each of these lines infinitely many times, but each of them with growing gaps.

3. Again, take graph of w with $\mathbf{v}_0 = (-1, 1)$ and $\mathbf{v}_1 = (1, 1)$. Bounded WAP means that it intersects some line $y = ax + b$ with a, b rational and gap at most C for some integer C, i. e., the difference between two consecutive points x_i and x_{i+1} is at most C. Therefore, the graph lies between lines $y = ax + b - C/2$ and $y = ax + b + C/2$, and hence w is of bounded width.

In the following proposition we discuss the connections between uniform recurrence and WAP.

Proposition 2. *1. If w is uniformly recurrent and of bounded width, then w is bounded WAP.*
2. There exists a uniformly recurrent WAP word w which is not of bounded width.

Proof. 1. Take graph of w with some vectors, e. g., $\mathbf{v}_0 = (-1, 1)$ and $\mathbf{v}_1 = (1, 1)$. Bounded width means that the graph g_w satisfies $ax + b_1 \le g_w(x) \le ax + b_2$ for some rational numbers a, b_1, b_2. Take the biggest such b_1 and the smallest b_2, i. e., there are integers x_1 and x_2 such that $g_w(x_1) = ax_1 + b_1$, $g_w(x_2) = ax_2 + b_2$. Without loss of generality suppose $x_1 \le x_2$ and consider the factor $w[x_1, x_2]$. Since w is uniformly recurrent, this factor occurs infinitely many times in it with bounded gap. Every position i corresponding to an occurrence of this factor satisfies $g_w(i) = ai + b_1$, otherwise $g_w(i + x_2 - x_1) > a(i + x_2 - x_1) + b_2$, which contradicts the choice of b_2. Hence the word is bounded WAP along the line $y = ax + b_1$ (and moreover along $y = ax + b_2$ and any rational line in between).

2. One of such examples is the paperfolding word w. It can be defined in several equivalent ways, we define it as a Toeplitz word with pattern 0?1? [3]. It is not

difficult to see that $|\mathrm{pref}_{4^k-1}(w)|_0 = 4^k/2$, $|\mathrm{pref}_{4^k-1}(w)|_1 = 4^k/2-1$. Hence, the word is WAP with frequencies $\rho_0 = \rho_1 = \frac{1}{2}$ along the line $y = -1$. On the other hand, taking $n = 2^k + 2^{k-2} + \cdots + 2^{k-2\lfloor \frac{k}{2} \rfloor}$, one gets $|\mathrm{pref}_n(w)|_0 - |\mathrm{pref}_n(w)|_1 = k+1$. Thus, the word is not of bounded width.

Next, we study the relation between WAP property and frequencies of letters.

Proposition 3. *1. There exists an infinite word w with rational frequencies of letters which is not WAP.*
2. If an infinite word w has irrational frequency of some letters, then w is not WAP.
3. If a binary infinite word w does not have frequencies of letters, then w is WAP.
4. There exists a ternary infinite word w which does not have frequencies of letters and which is not WAP.

Proof. 1. Consider

$$w = 01001010(01)^4 \dots 0(01)^{2^n} \dots$$

This word has letter frequencies $\rho_0 = \rho_1 = 1/2$. Suppose it is weakly abelian periodic. If a word has frequencies of letters and is WAP, then these frequencies coincide with frequencies of letters in the corresponding factorization. So, if w is WAP, then there is a sequence $k_1, k_2, \dots$ (the sequence of lengths of factors in the corresponding factorization), such that $|\mathrm{pref}_{k_i} w|_0 = k_i/2 + C$, where C is defined by the first factor of length k_1: $C = k_1/2 - |\mathrm{pref}_{k_1} w|_0/2$. For the word w, the number of 0's in a prefix of length n is $|\mathrm{pref}_n w|_0 = n/2 + \theta(\log n)$. For $n = k_i$ large enough one has $\theta(\log n) > C$, a contradiction. Thus, w is not WAP.

For uniformly recurrent examples see Section 5.

2. Assume that the word w is WAP, then for every letter a there exists a rational partial limit $\lim_{n_k \to \infty} \frac{|\mathrm{pref}_{n_k}(w)|_a}{|\mathrm{pref}_{n_k}(w)|}$. For w having irrational frequency of some letter all such partial limits corresponding to this letter exist and are equal to this irrational frequency. A contradiction.

3. Consider a sequence $(\frac{|\mathrm{pref}_n(w)|_a}{|\mathrm{pref}_n(w)|})_{n \geq 1}$. This sequence is bounded, and has a lower and upper partial limits $r = \underline{\lim}_{n \to \infty} \frac{|\mathrm{pref}_n(w)|_a}{|\mathrm{pref}_n(w)|}$ and $R = \overline{\lim}_{n \to \infty} \frac{|\mathrm{pref}_n(w)|_a}{|\mathrm{pref}_n(w)|}$. Since the sequence does not have a limit, these partial limits do not coincide: $r < R$. Using the graph of w, one gets that the graph intersects every line with slope corresponding to the frequency between r and R. For rational frequencies one gets that the graph intersects the line infinitely many times. Hence there are infinitely many integer points on it (or its shift, depending on the choice of v_0 and v_1). Thus, we proved that w is WAP, and moreover, it is WAP with any rational frequency ρ, $r < \rho < R$ in factors in the corresponding factorization.

4. Consider the word

$$w = 01201^2 2^4 0^6 1^{10} 2^{16} \dots 0^{n_i} 1^{n_{i+1}} 2^{n_{i+2}} \dots,$$

where $n_i = n_{i-1} + n_{i-2}$ for every $i \geq 5$, $n_1 = n_2 = n_3 = n_4 = 1$. The word is organized in a way that after each block a^{n_i} the frequency of the letter a in the prefix ending in this block is equal to $1/2$, i. e., $\rho_a(01201^2 2^4 \dots a^{n_i}) = \frac{1}{2}$ for $a \in \{0, 1, 2\}$. Hence, the frequencies of letters do not exist.

Now we will prove that it is not weakly abelian periodic. Suppose it is, with points $k_1, k_2 \dots$ and rational frequencies ρ_0, ρ_1, ρ_2 in the blocks, i. e. $w = w_1 w_2 \dots$, and $|w_1 \dots w_n| = k_n$ and $\frac{|w_i|_a}{|w_i|} = \rho_a$ for every $a \in \{0, 1, 2\}$ and $i > 1$. By the pigeonhole principle there exists a letter a such that infinitely many k_i are in the blocks of a-s, meaning that at least one of the letters w_{k_i}, w_{k_i+1} is a. Without loss of generality suppose $a = 2$. Using the recurrence relation for n_i, one can find $\lim_{n\to\infty} \frac{|\mathrm{pref}_{k_n} w|_0}{|\mathrm{pref}_{k_n} w|_1} = \frac{1}{\lambda_1}$, where $\lambda_1 = \frac{1+\sqrt{5}}{2}$ is the larger root of the equation $\lambda^2 = \lambda + 1$ corresponding to the recurrence relation. Therefore, the limit is irrational, and hence w cannot be equal to $\frac{\rho_0}{\rho_1}$. Thus, w is not WAP.

Thus, we obtain the following corollary:

Corollary 1. *If a binary word w is not WAP, then it has frequencies of letters.*

This simple corollary, however, is unexpected: from the first glance weak abelian periodicity and frequencies of letters seem to be very close notions. But it turns out that one of them (WAP) does not hold, then the other one should necessarily hold.

We end this section with an observation about WAP of non-binary words. We will show that contrary to ordinary and abelian periodicity, the property WAP cannot be checked from binary words obtained by unifying letters of the original word.

For a word w over an alphabet of cardinality k define $w^{a\cup b}$ to be the word over the alphabet of cardinality $k-1$ obtained from w by unifying letters a and b. In other words, $w^{a\cup b}$ is an image of w under a morphism $b \mapsto a$, $c \mapsto c$ for every $c \neq b$.

Proposition 4. *There exists a ternary word w, such that $w^{0\cup 1}$, $w^{0\cup 2}$, $w^{1\cup 2}$ are WAP, and w itself is not WAP.*

Proof. We use the example we built in the proof of Proposition 3(3), i. e., we take $w = 01201^2 2^4 0^6 1^{10} 2^{16} \dots 0^{n_i} 1^{n_{i+1}} 2^{n_{i+2}} \dots$, where $n_i = n_{i-1} + n_{i-2}$ for every i. Due to space limitations, we omit the calculations.

3 Weak Abelian Periodicity of Fixed Points of Binary Uniform Morphisms

In this section we study the weak abelian periodicity of fixed points of uniform binary morphisms.

Consider a binary uniform morphism φ with matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$. This means that $|\varphi(0)|_0 = a$, $|\varphi(0)|_1 = b$, $|\varphi(1)|_0 = c$, $|\varphi(1)|_1 = d$, and $a + b = c + d = k$, since we

consider a uniform morphism. In a fixed point w of the binary uniform morphism φ the frequencies exist and they are rational. It is easy to see that $\rho_0(w) = \frac{c}{b+c}$, $\rho_1(w) = \frac{b}{b+c}$. It will be convenient for us to consider a geometric interpretation with $\mathbf{v}_0 = (1, -b)$, $\mathbf{v}_1 = (1, c)$. If w is WAP, then the frequency inside the blocks is equal to the frequency in the whole word. Thus, WAP can be reached along a horizontal line $y = C$.

The following theorem gives a characterization of weak abelian periodicity for fixed points of non-primitive binary uniform morphisms. Observe that the theorems in this section are stated for non-primitive morphisms, since for primitive binary uniform morphisms it is easy to check (bounded) WAP directly.

Theorem 1. *Consider a non-primitive binary uniform morphism φ with matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ having a fixed point w starting with letter 0. For any $u \in \{0,1\}^* \cup \{0,1\}^\infty$ let g_u be its graph with vectors $\boldsymbol{v}_0 = (1, -b)$, $\boldsymbol{v}_1 = (1, c)$.*
1. If $g_{\varphi(0)}(x) = 0$ for some x, $0 < x \leq k$, then w is WAP.
2. If $g_{\varphi(0)}(k) \geq -b$, then w is WAP.
3. Otherwise we need the following parameters. Denote $\Delta = g_{\varphi(0)}(k)$, $A = \max\{g_{\varphi(0)}(i) | i = 1, \dots k, w_i = 1\}$, $t = \max\{g_{\varphi(1)}(i) | i = 1, \dots k, w_i = 1\}$.

If φ does not satisfy conditions 1 and 2, then its fixed point w is WAP if and only if $\Delta \frac{A-c}{-b} + t \geq A$.

Proof. 1. If in the condition $g_{\varphi(0)}(x) = 0$, $0 < x \leq k$, the number x is integer, then for every i it holds $g_{\varphi^i(0)}(k^{i-1}x) = 0$, so the word is WAP. If x is not integer, then we have either $g_{\varphi(0)}(\lfloor x \rfloor) < 0$ and $g_{\varphi(0)}(\lceil x \rceil) > 0$ or $g_{\varphi(0)}(\lfloor x \rfloor) > 0$ and $g_{\varphi(0)}(\lceil x \rceil) < 0$. Without loss of generality consider the first case. For any i, one has $g_{\varphi^i(0)}(k^{i-1}\lfloor x \rfloor) < 0$ and $g_{\varphi^i(0)}(k^{i-1}\lceil x \rceil) > 0$, hence there exists x_i, $k^{i-1}\lfloor x \rfloor < x_i < k^{i-1}\lceil x \rceil$, such that $g_{\varphi^i(0)}(x_i) = 0$. Hence, we have an infinite sequence of points $(x_i)_{i=1}^{\infty}$ such that $g_w(x_i) = 0$. By the definition of g_w and the pigeonhole principle we obtain that there is an infinite number of integer points from the set $\lfloor x_i \rfloor, \lceil x_i \rceil$, $i = 1, \dots, \infty$, on one of the lines $x = C$, $C = -\max(b, c) + 1, -\max(b, c) + 2, \dots, \max(b, c) - 1$. So, w is WAP.

2. If $g_{\varphi(0)}(k) \geq 0$, we are in the conditions of the case 1, so the word is WAP. If $0 > g_{\varphi(0)}(k) \geq -b$, then the only possible case is $g_{\varphi(0)}(k) = -b$. This follows from the fact that the condition $0 > g_{\varphi(0)}(k) \geq -b$ means that $a > c$, or, equivalently, $a - c \geq 1$, and therefore $g_{\varphi(0)}(k) = a(-b) + bc = -b(a - c) \geq -b$. Hence $c = a - 1$, and so $g_{\varphi^i(0)}(k^i) = -b$, and thus w is WAP along the line $y = -b$.

3. Suppose that $\Delta \frac{A-c}{-b} + t \geq A$. We need to prove that w is WAP.

Let j be such that $g_{\varphi(1)}(j) = t$. Under these conditions we will prove the following claim: If for some m one has $w_m = 1$ and $g_w(m) \geq A$, then $w_{km+j} = 1$ and $g_w(k(m-1) + j) \geq A$.

Consider the occurrence of 1 at the position m. By the definition of the graph of w, one has that $g_w(m-1) \geq A - c$, and hence $\mathrm{pref}_{m-1}(w)$ contains at least $\frac{c}{b+c}(m-1) - \frac{1}{b+c}(A-c)$ letters 0 and at most $\frac{b}{b+c}(m-1) + \frac{1}{b+c}(A-c)$ letters

1. Therefore, for the image of this prefix one has $g_w(k(m-1)) \geq \Delta\frac{A-c}{-b}$. Since $w_m = 1$, one has $w[k(m-1)+1, km] = \varphi(1)$. Then $g_w(k(m-1)+j) = g_w(k(m-1)) + t \geq \Delta\frac{A-c}{-b} + t$, and we have $\Delta\frac{A-c}{-b} + t \geq A$, and so $g_w(k(m-1)+j) \geq A$. The claim is proved.

Now consider the occurrence of 1 corresponding to the value A defined in the theorem, i. e., we consider $w_i = 1$ such that $g_w(i) = A$. Applying the claim we just proved to $m = i$ we have $w_{k(i-1)+j} = 1$, $g_w(k(i-1)+j) \geq A$. Now we can apply the claim to $m = k(i-1)+j$ and obtain that $w_{k(k(i-1)+j)+j} = 1$, $g_w(k(k(i-1)+j)) \geq A$. Continuing this line of reasoning, one gets infinitely many positions n for which $g_w(n) \geq A$. On the other hand, it is easy to see that $g_w(k^l) < 0$ for all integers l. Hence, w is WAP along one of the lines $y = C$, $A - \max(b,c) + 1 \leq C \leq \max(b,c) - 1$. Additional term $\pm\max(b,c)$ is added to guarantee integer points, since the graph "jumps" by b and c.

Now suppose that $\Delta\frac{A-c}{-b} + t < A$. We need to prove that w is not WAP.

Let j be such that $g_{\varphi(1)}(j) = t$. Under these conditions we prove the following claim: If for all m in the prefix of w of length N such that $w_m = 1$ one has $g_w(m) \leq A$, then for all $N+1 \leq l \leq Nk$ such that $w_l = 1$ we have $g_w(l) < \max_m\{g_w(m) | 1 \leq m \leq N, w_m = 1\}$, or, equivalently, $g_w(l) \leq \max_m\{g_w(m) - 1 | 1 \leq m \leq N, w_m = 1\}$. Roughly speaking, the claim says that maximal values are decreasing. The claim is proved in a similar way as the previous claim, so we omit the proof.

Now consider occurrences of 1 from $\varphi(0)$, i. e., we consider $w_i = 1$ such that $1 \leq i \leq k$. By the conditions of the part 3 of the theorem we have $g_w(i) \leq A$. Applying the latter claim to $m = i$ we have that for all occurrences l of 1 in $w[k+1, k^2]$ it holds $g_w(l) \leq A - 1$. By the definition of the graph g_w, maximal values are attained immediately after the occurrences of 1-s, hence we actually have $g_w(l) \leq A - 1$ for all $k+1 \leq l \leq k^2$. Continuing this line of reasoning, we obtain that for $k^n + 1 \leq i \leq k^{n+1}$ it holds $g_w(l) \leq A - n$. Thus, the word w is not WAP (since w can be WAP only along horizontal lines).

Now we are going to show that a fixed point of a uniform morphism is bounded WAP iff it is abelian periodic. This is probably known or follows from some general characterizations of balance of morphic words (e. g., [1]), but we nevertheless provide a short combinatorial proof to be self-contained.

Theorem 2. *Let w be a fixed point of binary k-uniform morphism φ. The following are equivalent:*
1. w is bounded WAP
2. w is abelian periodic
3. $\varphi(0) \sim_{ab} \varphi(1)$ or k is odd and $\varphi(0) = (01)^{\frac{k-1}{2}}0$, $\varphi(1) = (10)^{\frac{k-1}{2}}1$.

Proof. We prove the theorem in the following way. Starting with a bounded WAP word w, we step by step restrict the form of w and prove that the morphism should satisfy either $\varphi(0) \sim_{ab} \varphi(1)$ or k is odd and $\varphi(0) = (01)^{\frac{k-1}{2}}0$, $\varphi(1) = (10)^{\frac{k-1}{2}}1$. These conditions clearly imply abelian periodicity, and abelian periodicity implies bounded WAP. So, we actually prove $1 \Rightarrow 3 \Rightarrow 2 \Rightarrow 1$, and the only implication to be proved is $1 \Rightarrow 3$.

Suppose that w is bounded WAP and $\varphi(0)$ is not abelian equivalent to $\varphi(1)$, i. e., $a \neq c$. Without loss of generality we may assume that the fixed point starts with 0 and that $a > c$. If $a < c$, we consider a morphism φ^2, so that one has $g_{\varphi^2(0)} \leq 0$. We will prove that either the fixed point is not of bounded width or the morphism is of the form $\varphi(0) = (01)^{\frac{k-1}{2}}0$, $\varphi(1) = (10)^{\frac{k-1}{2}}1$, k odd.

In the proof we will use the following terminology. For a factor u of w such that $\rho_0(u) > \rho_0(w)$, we say that u has m *extra* 0's, if $\frac{|u|_0 - m}{|u| - m} = \rho_0(w)$. In other words, deleting m letters 0 from u gives a word with frequency $\rho_0(w)$. We also admit non-integer values of m. E. g., if $\rho_0(w) = \frac{1}{3}$ and $u = 01$, then u has $\frac{1}{2}$ extra 0's.

Suppose $a > c + 1$. In this case $\varphi^i(0)$ contains $(a - c)^i$ extra zeros. Since $(a - c)^i$ increases as i increases, w is not of bounded width. Hence, the fixed point is not bounded WAP in this case, and hence for bounded WAP one should have $a = c + 1$.

Suppose that $\varphi(0)$ has a prefix x with more than one extra zero. Without loss of generality we assume that x ends with 0, otherwise we may take a smaller prefix. So, $x = x'0$, and x' has $m > 0$ extra 0-s. It is not difficult to show that under the condition $a = c + 1$ the image $\varphi(x')$ also contains m extra 0. An image of x starts with $\varphi(x')x'0$. An image of this word starts in $\varphi^2(x')\varphi(x')x'0$. Continuing taking images, we obtain that for every i the word w has a prefix of the form $\varphi^i(x')\varphi^{i-1}(x') \dots \varphi(x')x'0$. This word contains $(i + 1)m + 1$ extra 0-s, and this amount grows as i grows. Hence word w is not of bounded width, a contradiction. Therefore, we have that every prefix of $\varphi(0)$ has at most one extra 0, in particular, $\varphi(0)$ starts in 01.

In a similar way we show that every suffix of $\varphi(0)$ has at most one extra 0. The only difference is that we obtain a series of factors (not prefixes) of w with growing amount of extra 0-s.

Now consider an occurrence of 0 in $\varphi(0)$, i. e., $w_j = 0$, $1 \leq j \leq k$. By what we just proved, $\rho_0(\mathrm{pref}_{j-1}(\varphi(0)) \geq \rho_0(w)$, and $\rho_0(\mathrm{suff}_{k-j}(\varphi(0)) \geq \rho_0(w)$. Since $\varphi(0)$ has one extra 0, we have $\rho_0(\mathrm{pref}_{j-1}(\varphi(0)) = \rho_0(\mathrm{suff}_{k-j}(\varphi(0)) = \rho_0(w)$. Hence, w_j can be equal to 0 only if in the prefix $\mathrm{pref}_{j-1}(\varphi(0))$ the frequency of 0 is the same as in w.

On the other hand, if the frequencies in the $\mathrm{pref}_{j-1}(\varphi(0))$ are the same as in w, then w_j cannot be equal to 1. Suppose the converse; let $w_j = 1$, then all $w_l = 1$, $l = j, \dots, k - 1$, since by induction in all the prefixes $\mathrm{pref}_l(\varphi(0))$ the frequency of 0 is less than $\rho_0(w)$. Therefore, in $\varphi(0)$ there will be less than one extra 0, a contradiction.

Thus, each time we have $\rho_0(\mathrm{pref}_{j-1}(\varphi(0)) = \rho_0(w)$, we necessarily have $w_j = 0$, otherwise $w_j = 1$. Since $|\varphi(0)|_0 = a$, the frequency $\rho_0(w)$ is reached a times, and $\varphi(0)$ consists of $a - 1$ blocks with one 0 and with frequency $\rho_0(w)$, and one extra block 0. Therefore, $a-1$ divides $a-1+b$, i. e., $b = i(a-1)$ for some integer i. By a similar argument applied to $\varphi(1)$ we get that $d - 1$ divides $c - 1$, which means $i(a - 1)$ divides $a - 1$. Hence $i = 1$, and the matrix of the morphism is

$\begin{pmatrix} a & a-1 \\ a-1 & a \end{pmatrix}$. Combining this with the conditions for positions of 0 in $\varphi(0)$, we obtain $\varphi(0) = (01)^{\frac{k-1}{2}}0$, $\varphi(1) = (10)^{\frac{k-1}{2}}1$.

4 On WAP of Points in a Shift Orbit Closure

In this section we consider the following question: if a uniformly recurrent word w is WAP, what can we say about WAP of other words whose language equals $F(w)$?

As a corollary from Theorem 1 we obtain the following proposition:

Proposition 5. *There exists a binary uniform morphism having two infinite fixed points, such that one of them is WAP, and the other one is not.*

Proof. Consider the morphism $\varphi : 0 \to 0001, 1 \to 1011$. Using Theorem 1 (3), one gets that the fixed point starting from 0 is not WAP. Using Theorem 1 (1), one gets that the fixed point starting from 1 is WAP.

Remark. In particular, this means that there exist two words with same sets of factors such that one of them is WAP while the other one is not.

In this section we need some more definitions.

Let $T : \Sigma^\omega \to \Sigma^\omega$ denote the *shift transformation* defined by $T : (x_n)_{n\in\mathbb{N}} \to (x_{n+1})_{n\in\omega}$. The *shift orbit* of an infinite word $x \in \Sigma^\omega$ is the set $O(x) = \{T^i(x) | i \geq 0\}$ and its *closure* is given by $\overline{O}(x) = \{y \in \Sigma^\omega | \mathrm{Pref}(y) \subseteq \{\mathrm{Pref}(T^i(x)) | i \in \mathbb{N}\}\}$, where $\mathrm{Pref}(w)$ denotes the set of prefixes of a finite or infinite word w. For a uniformly recurrent word w any infinite word x in $\overline{O}(w)$ has the same set of factors as w.

We say that $w \in \Sigma^\omega$ has *uniform frequency* ρ_a of a letter a, if in every word from $O(w)$ the frequency of the letter a exists and is equal to ρ_a. In other words, a letter $a \in \Sigma$ has uniform frequency ρ_a in w if its minimal frequency $\underline{\rho}_a = \lim_{n\to\infty} \inf_{x\in F_n(w)} \frac{|x|_a}{|x|}$ is equal to its maximal frequency $\overline{\rho}_a = \lim_{n\to\infty} \sup_{x\in F_n(w)} \frac{|x|_a}{|x|}$, i. e. $\underline{\rho}_a = \overline{\rho}_a$.

Theorem 3. *Let w be an infinite binary uniformly recurrent word.*
1. If w has irrational frequencies of letters, then every word in its shift orbit closure is not WAP.
2. If w does not have uniform frequencies of letters, then there is a point in a shift orbit closure of w which is WAP.
3. If w has uniform rational frequencies of letters, then there is a point in a shift orbit closure of w which is WAP.
4. There exists a non-balanced word w with uniform rational frequencies of letters, such that every point in a shift orbit closure of w is WAP.

Proof. 1. Follows from Proposition 3 (2).

2. Follows from Proposition 3 (3).

3. In the proof we use the notion of a return word. For $u \in F(w)$, let $n_1 < n_2 < \dots$ be all integers n_i such that $u = w_{n_i} \dots w_{n_i+|u|-1}$. Then the word $w_{n_i} \dots w_{n_{i+1}-1}$ is a *return word* (or briefly *return*) of u in w [6], [9], [13].

We now build a WAP word u from $\overline{O}(w)$. Start with any factor u_1 of w, e. g. with a letter. Without loss of generality assume that $\rho_0(u_1) \geq \rho_0(w)$. Consider factorization of w into first returns to u_1: $w = v_1^1 v_2^1 \dots v_i^1 \dots$, so that v_i^1 is a return to u_1 for $i > 1$. Then there exists $i_1 > 1$ satisfying $\rho_0(v_{i_1}^1) \geq \rho_0$. Suppose the converse, i. e., for all $i > 1$ $\rho_0(v_i^1) < \rho_0$. Due to uniform recurrence, the lengths of the v_i^1's are uniformly bounded, and hence $\rho_0(w) < \rho_0$, a contradiction. Take $u_2 = v_{i_1}^1$, then $u_1 = \mathrm{pref}(u_2)$. Now consider a factorization of w into first returns to u_2: $w = v_1^2 v_2^2 \dots v_i^2 \dots$. Then there exists $i_2 > 1$ satisfying $\rho_0(v_{i_2}^2) \leq \rho_0$, take $u_3 = v_{i_2}^2$. Continuing this line of reasoning to infinity, we build a word $u = \lim_{n\to\infty} u_i$, such that $\rho_0(u_{2i}) \geq \rho_0$, $\rho_0(u_{2i+1}) \leq \rho_0$. So, the graph of w with vectors $\mathbf{v}_0 = (1, -1)$ and $\mathbf{v}_0 = (1, -1)$ intersects the line $y = \rho_0 x$ infinitely many times. Since ρ_0 is rational, by a pigeonhole principle the graph intersects in integer points infinitely many times one of finite number (actually, a denominator of ρ_0) of lines parallel to $y = \rho_0 x$. It follows that u is WAP with frequency ρ_0, and by construction $u \in \overline{O}(w)$.

4. Due to space limitations, we omit the proof of this item.

References

1. Adamczewski, B.: Balances for fixed points of primitive substitutions. Theoret. Comput. Sci. 307, 47–75 (2003)
2. Avgustinovich, S., Karhumäki, J., Puzynina, S.: On abelian versions of critical factorization theorem. RAIRO - Theoretical Informatics and Applications 46, 3–15 (2012)
3. Cassaigne, J., Karhumäki, J.: Toeplitz Words, Generalized Periodicity and Periodically Iterated Morphisms. Eur. J. Comb. 18(5), 497–510 (1997)
4. Cassaigne, J., Richomme, G., Saari, K., Zamboni, L.Q.: Avoiding Abelian powers in binary words with bounded Abelian complexity. Int. J. Found. Comput. Sci. 22(4), 905–920 (2011)
5. Constantinescu, S., Ilie, L.: Fine and Wilf's theorem for abelian periods. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 89, 167–170 (2006)
6. Durand, F.: A characterization of substitutive sequences using return words. Discrete Mathematics 179(1-3), 89–101 (1998)
7. Erdös, P.: Some unsolved problems. Magyar Tud. Akad. Mat. Kutató Int. Közl. 6, 221–254 (1961)
8. Gerver, J.L., Ramsey, L.T.: On certain sequences of lattice points. Pacific J. Math. 83(2), 357–363 (1979)

9. Holton, C., Zamboni, L.Q.: Geometric Realizations Of Substitutions. Bull. Soc. Math. France 126, 149–179 (1998)
10. Keränen, V.: Abelian squares are avoidable on 4 letters. In Automata, languages and programming. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 41–52. Springer, Heidelberg (1992)
11. Krajnev, V.A.: Words that do not contain consecutive factors with equal frequencies of letters. Metody Discretnogo Analiza v Reshenii Kombinatornyh Zadach 34, 27–37 (1980)
12. Lothaire, M.: Algebraic combinatorics on words. Cambridge University Press (2002)
13. Puzynina, S., Zamboni, L.Q.: Abelian returns in Sturmian words. J. Combin. Theory, Ser. A, V. 120(2), 390–408 (2013)

Universality of Regular Realizability Problems

Mikhail N. Vyalyi*

Dorodnitsyn Computing Center of Russian Academy of Science
vyalyi@gmail.com

Abstract. A regular realizability (RR) problem is to test nonemptiness of the intersection of some fixed language (filter) with a given regular language. We show that RR problems are universal in the following sense. For any language L there exists an RR problem equivalent to L under disjunctive reductions on nondeterministic log space.

We deduce from this result the existence of RR problems complete under polynomial reductions for many complexity classes including all classes of the polynomial hierarchy.

It is well-known that the class of languages recognized by deterministic multihead 2-way automata is exactly the class LOG of languages recognized by Turing machines with logarithmically bounded work space.

This remarkable characterization of a resource-bounded complexity class in terms of a computational model without restrictions on resources was generalized in [11] to various classes of automata using auxiliary memory as a source of nondeterminism. Generalized nondeterministic automata models (GNA) introduced in [12] express these computational models in a more convenient way. In particular, each GNA class has a complete *regular realizability* (RR) problem under log space reductions [13]. The RR problems are parameterized by languages. The problem $\mathrm{RR}(L)$ for a language L (called a *filter*) is the question about realizability of regular properties in L. More precisely, the language $\mathrm{RR}(L)$ consists of descriptions of regular languages R such that $R \cap L \neq \varnothing$.

There exist RR problems complete under log space reductions for complexity classes such as LOG, NLOG, P, NP, PSPACE, EXP, Σ_1, see [3,11,12].

In this paper we address the following natural question: what are possible complexities of RR problems?

The motivation behind the question is to find a specific class of algorithmic problems that represents in a unified way all known complexity classes (there are hundreds of them now). We pursue two (somewhat contradictory) goals: a specific class of problems should be wide enough and it should be useful. The latter requirement reflects a hope that analysis of a specific problem might be easier than a general case.

We give a partial answer to this question. It turns out that RR problems are universal: for any problem there exists an equivalent RR problem.

To make precise statements we need to fix a format for descriptions of regular languages and an equivalence relation.

* Partially supported by RFBR grants 11–01–00398 and 12–01–00864.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 271–282, 2013.

We represent a regular language R by a deterministic finite automaton (DFA) A recognizing the language R (and denote this fact as $R = L(A)$). DFAs are described in a natural way by their transition tables. Details of the format used can be found in [13]. Important features for this work are: (i) each binary word w is a description of some DFA $A(w)$, and (ii) testing membership for a regular language $L(w) = L(A(w))$ can be done using deterministic log space. So the formal definition of the language $\mathrm{RR}(L)$ corresponding to the RR problem with the filter L is

$$w \in \mathrm{RR}(L) \;\Leftrightarrow\; L \cap L(A(w)) \neq \varnothing.$$

Equivalence relations considered here are induced by algorithmic reductions. For any language X an RR-representative of X (under reductions of some type) is a language L such that $\mathrm{RR}(L)$ is equivalent to X under reductions of this type.

We can prove the universality result mentioned above for disjunctive reductions using nondeterministic log space (see the definition below in Section 1). The choice of the reduction type needs explanations.

It is obvious that the stronger reductions are used, the easier universality proofs become.

The most natural reductions in the context of regular realizability problems are m-reductions using log space ($\leqslant^{\log}_{\mathrm{m}}$-reductions). But there are some arguments against universality for these reductions (see Remark 1 below).

On the other hand, universality is rather easy to prove for exponential time reductions. But these reductions do not say anything about the most interesting realm of complexities: the class PSPACE and below.

For polynomial time reductions we need the same constructions that are used in our proof below. The algorithmic facts become easier.

FNL reductions cover the class P and below. Note that as shown in [1] basic counting log space classes are closed under $\leqslant^{\mathrm{FNL}}_{\mathrm{dtt}}$-reductions.

For many cases, disjunctive nlog space reductions are weaker than polynomial reductions. In this way we extend a list of classes having complete RR problems under polynomial reductions. In particular, there exist RR problems complete for the classes of the polynomial hierarchy. Note that it is a rather surprising even for the class co-NP: RR problems are formulated by use of the existential quantifier (existence of an accepting path possessing specific properties) and there is no direct way to express them using the universal quantifier.

Thus RR problems are universal. Are they useful in a sense explained above? The proof of universality suggests a negative answer. Reductions used in the proof cut off almost all properties of regular languages and put 'the hard part' of a problem into instances corresponding to finite languages. Of course, nlog space reductions say nothing about languages inside NLOG.

The rest of the paper is organized as follows. In Section 1 we discuss disjunctive nlog space reductions and compare them with polynomial reductions.

In Section 2 we introduce a special class of reductions— *monoreductions*— and present an easy example of a universality result: RR universality in the class of promise problems.

Section 3 contains the proof of the main result. In Section 4 we extend universality to the GNA classes. The idea of the proof is the same. But its implementation involves more technicalities.

All proofs omitted here can be found in the arXiv version of this paper [14].

1 Reductions Used in Universality Results

We recall basic definitions concerning algorithmic reductions. Let $\mathcal{C}$ be a function class. A language A is reduced to a language B under m-reductions by functions from this class (notation $\leqslant_{\mathrm{m}}^{\mathcal{C}}$) if there exists a function $f \in \mathcal{C}$ such that $x \in A$ iff $f(x) \in B$. If the class $\mathcal{C}$ contains the identity map and is closed under compositions, then the m-reduction relation is a preorder, i.e. a transitive and reflexive relation. Languages are equivalent (notation $A \sim_m^{\mathcal{C}} B$) if they are reduced to each other under $\leqslant_{\mathrm{m}}^{\mathcal{C}}$-reductions.

The most known reductions of this type are polynomial reductions (see, e.g. [4,10]).

Weaker $\leqslant_{\mathrm{m}}^{\log}$-reductions are defined by the class of functions that are computable by deterministic Turing machines using space logarithmically bounded w.r.t. the input length. It easy to check that the relation $\leqslant_{\mathrm{m}}^{\log}$ is transitive (see textbooks on complexity theory, say, [4,10]).

The second important type of reductions is Turing reductions. A language A is Turing reducible to a language B if there exists an oracle algorithm recognizing A that uses the oracle B. There are several restricted forms of Turing reductions. In *truth-table reductions* the reducing algorithm generates a list of oracle queries $q_1, \ldots, q_s$ and a Boolean function α depending on s arguments. Then the algorithm asks all queries and outputs $\alpha(\chi_B(q_1), \ldots, \chi_B(q_s))$.

Note that for log space, Turing reductions and truth-table reductions are equivalent [7].

We use in the main result a weaker form of truth-table reductions, namely, disjunctive reductions (notation $\leqslant_{\mathrm{dtt}}$). In this case the function α is disjunction.

Disjunctive reductions can be expressed via m-reductions in the following way. Given a language X, define the language $\mathrm{Seq}(X)$ as the collection of all words of the form $\#x_1\#x_2\#\ldots x_n\#$, where $x_i \in X$ for some i and $\#$ is a delimiter (an additional symbol that does not belong to the alphabet of the language X). The following statement is clear from the definition.

Proposition 1. *$A \leqslant_{\mathrm{dtt}} B$ iff $A \leqslant_{\mathrm{m}} \mathrm{Seq}(B)$.*

Words from $\mathrm{Seq}(X)$ can be identified with finite sequences of words from Σ^* in a natural way. Hereinafter we assume this correspondence.

Definition 1. *A class of languages $\mathcal{C}$ is* normal *if it is closed under the map $X \mapsto \mathrm{Seq}(X)$ and $\leqslant_{\mathrm{m}}^{\log}$-reductions: if $X \in \mathcal{C}$ and $Y \leqslant_{\mathrm{m}}^{\log} X$ then $\mathrm{Seq}(X) \in \mathcal{C}$ and $Y \in \mathcal{C}$.*

Definition 2. *A language X is $\leqslant_{\mathrm{m}}$-*normal *if $X \sim_m \mathrm{Seq}(X)$.*

If a function class is not indicated we assume log space reductions.

We list several simple properties of normal classes and normal languages.

Proposition 2. $\mathrm{Seq}(X)$ *is normal for any* X.

Lemma 1. *Let* $\mathrm{Seq}(X) \leqslant_{\mathrm{m}}^{\log} Y$ *and* $Y \leqslant_{\mathrm{m}}^{\log} \mathrm{Seq}(X)$. *Then* Y *is normal.*

Corollary 1. *If a normal class contains* $\leqslant_{\mathrm{m}}^{\log}$*-complete languages then all* $\leqslant_{\mathrm{m}}^{\log}$*-complete languages in this class are normal.*

For disjunctive reductions we will use functions computable using nondeterministic log space (the class FNL). It means that machines computing FNL functions are nondeterministic Turing machines equipped with the logarithmically bounded work tape and the unbounded oracle tape which is one way and write only. An oracle puts its answer on the oracle tape and overwrites the query. More details on the class FNL and its companions can be found in [2]. In particular, the FNL reductions are transitive and the size of output is polynomially bounded by the input size.

We denote FNL disjunctive reductions by $\leqslant_{\mathrm{dtt}}^{\mathrm{FNL}}$. The FNL m-reduction is denoted by $\leqslant_{\mathrm{m}}^{\mathrm{FNL}}$.

It is clear from the definition that $\leqslant_{\mathrm{dtt}}^{\mathrm{FNL}}$-reductions are stronger than log space reductions and are weaker than polynomial time disjunctive reductions.

We will apply Corollary 1 and $\leqslant_{\mathrm{dtt}}^{\mathrm{FNL}}$-universality results to prove that a complexity class contains complete RR problems. To apply Corollary 1 the class should be normal.

A majority of known complexity classes is normal. We give several examples. In definitions of complexity classes and computational models we follow Arora and Barak's book [4].

A straightforward algorithm for recognizing the language $\mathrm{Seq}(X)$ is to check $x_i \in X$ for all x_i taken from the input

$$\#x_1\# \ldots \#x_m\#$$

and to output the disjunction of the results.

Let $X \in \mathrm{DSPACE}(f(n))$, where $f(n) = \Omega(\log n)$. Then the above algorithm uses $O(f(n))$ space. Thus the class $\mathrm{DSPACE}(f(n))$ is normal (it is closed under $\leqslant_{\mathrm{m}}^{\log}$-reductions by obvious reasons).

With small modifications the same argument is applied to nondeterministic space classes. A nondeterministic algorithm guesses i such that $x_i \in X$ and calls the algorithm recognizing X on the instance x_i. So the classes $\mathrm{NSPACE}(f(n))$ with $f(n) = \Omega(\log n)$ also are normal.

For time complexity classes, closeness under $\leqslant_{\mathrm{m}}^{\log}$-reductions holds for the class P of polynomial time and more powerful classes. The running time of the above algorithm recognizing $\mathrm{Seq}(X)$ is upperbounded by

$$\widetilde{T}(n) = n + \max_{n=n_1+\cdots+n_m} (T(n_1) + \cdots + T(n_m)), \tag{1}$$

where $T(n)$ is the running time of the algorithm recognizing X. It is clear that $T(n) = \mathrm{poly}(n)$ implies $\widetilde{T}(n) = \mathrm{poly}(n)$. So P is normal. For more powerful

classes, normality holds whenever time limitations are closed under the map $T(n) \mapsto \widetilde{T}(n)$. There is a simple sufficient condition for closeness: if $T(n)$ satisfies time limitations then $nT(n)$ is also satisfies time limitations. Applying this observation we get normality for the classes of quasipolynomial time, exponential time, simple exponential time etc.

It is easy to see that under the same conditions nondeterministic time classes are also normal.

The last example consists of classes of the polynomial hierarchy.

Proposition 3. *Each class of the polynomial hierarchy is normal.*

Proof. Closeness under $\leqslant_{\mathrm{m}}^{\log}$-reductions is clear for each class of the polynomial hierarchy. So it remains to show closeness under the map $X \mapsto \mathrm{Seq}(X)$.

Let $\mathrm{ISeq}(X)$ be the language consisting of strings of the form

$$\#i\#x_1 \ldots \#x_m\#$$

such that $x_i \in X$. It is clear that $X \sim_{\mathrm{m}}^{\log} \mathrm{ISeq}(X)$. Indeed, using log space one can extract the ith list element.

By the definition of $\mathrm{ISeq}(X)$ we have

$$(w \in \mathrm{Seq}(X)) \quad \Longleftrightarrow \quad \exists i(\#iw \in \mathrm{ISeq}(X)). \tag{2}$$

Note that if $X \in \Sigma_k^p$ then $\mathrm{ISeq}(X)$ is in Σ_k^p. Thus $\mathrm{Seq}(X)$ is in Σ_k^p due to (2).

Suppose now that $X \in \Pi_k^p$. In this case $\mathrm{ISeq}(X)$ is in Π_k^p and we have for some $V \in \Sigma_{k-1}^p$ and polynomial $p(\cdot)$

$$(\#iw \in \mathrm{ISeq}(X)) \quad \Longleftrightarrow \quad \forall y(|y| \leqslant p(|w|)) \wedge (\#iw\#y \in V).$$

So we need to interchange quantifiers in (2). It is possible because the outer quantifier is polynomially bounded. □

2 Monoreductions

Without loss of generality we consider languages in the binary alphabet.

Let f be an injective map $\{0,1\}^* \to \{0,1\}^*$. A *monoreduction* is a map

$$x \mapsto A_{f(x)}, \tag{3}$$

where $A_{f(x)}$ is the description of the minimal DFA[1] recognizing the 1-element language $\{f(x)\}$. Note that the number of states in the minimal DFA coincides with the number of Myhill–Nerode classes [6]. It is easy to verify that the number of Myhill–Nerode classes for the language $\{w\}$ is just $|w| + 2$ (all prefixes of the word w plus one) and the function $w \mapsto A_w$ is computable on deterministic log space.

[1] We assume that a construction of the minimal automaton is fixed. For minimization algorithms see textbooks on formal languages, e.g. [6].

Informally speaking, the map (3) assigns to a binary word a 'name' in the form of an automaton description. The injectivity condition implies that names of different words are different.

If a map $f(x)$ is log space computable then the corresponding monoreduction is also log space computable. It is easy to see that the map $f(x)$ reduces a nonempty language $X \subseteq \{0,1\}^*$ to an RR problem $\mathrm{RR}(Y)$ iff

$$Y \cap f(\overline{X}) = \varnothing, \quad Y \supseteq f(X). \tag{4}$$

In other words, Y separates images of X and $\overline{X}$ and Y contains the image of X.

Complexity of a reduction in the opposite direction depends heavily on Y.

Take for example $Y = f(X)$. It is not a good choice because complexity of the language $\mathrm{RR}(f(X))$ varies in wide range w.r.t. complexity of the language X. It can be illustrated in the simplest case $f = \mathrm{id}$.

There exists a filter L such that (a) the membership problem for L is in the class of languages recognized by RAM in linear time; (b) $\mathrm{RR}(L)$ is complete for the class Σ_1 of recursively enumerable languages under m-reductions [12].

On the other hand for $X = \{0^n\}$ the language $\mathrm{RR}(X)$ is in LOG. It was shown in [13] that LOG is $\leqslant_{\mathrm{m}}^{\log}$-reducible to any RR problem with an infinite filter.

Nevertheless, the reduction $X \leqslant_{\mathrm{m}}^{\log} \mathrm{RR}(f(X))$ can be inverted if we consider reductions among promise problems. A *promise problem* is a problem of computing a partially defined predicate. In other words, there are two languages L_1 and L_0 such that $L_1 \cap L_0 = \varnothing$. The question is to test membership $w \in L_1$ provided either $w \in L_1$ or $w \in L_0$.

Promise problems have more expressive power than languages (which correspond to total predicates). For many complexity classes, say NP∩co-NP or BPP, the existence of complete languages in a class is an open problem. But there are simple and natural examples of complete promise problems for these classes.

The question about complete RR promise problems is also much easier than the question about complete RR languages.

Theorem 1. *Any promise problem (L_1, L_0) with $L_1 \neq \varnothing$ is equivalent to an RR promise problem under nlog space reductions.*

Proof. Define $\mathrm{RR}(L_1 : |R| = 1)$ as the RR problem with the promise $|L(A)| = 1$, where A is an input DFA. Then the promise problem (L_1, L_0) is $\leqslant_{\mathrm{m}}^{\log}$-reduced to the problem $\mathrm{RR}(L_1 : |R| = 1)$ by the map $w \mapsto A_w$.

In the opposite direction we can prove a weaker reduction

$$\mathrm{RR}(L_1 : |R| = 1) \leqslant_{\mathrm{m}}^{\mathrm{FNL}} (L_1, L_0). \tag{5}$$

To construct the reduction (5) we need a procedure that finds a word accepted by a DFA A provided A recognizes a 1-element language. This procedure is easily implemented in the class FNL: it nondeterministically guesses[2] the word symbol by symbol maintaining the current state of the automaton reading the word. □

[2] Hereinafter *a guess* is a nondeterministic choice.

Remark 1. Is it necessary to use an NLOG-oracle in the reduction (5)? The question is open but the negative answer is more plausible. In the non-uniform settings the class of unambiguous nondeterministic log space coincides with the class of nondeterministic log space [9]. It is quite natural to suggest that simplicity test does not belong to LOG if LOG $\neq$ NLOG.

Remark 2. A unique word accepted by DFA can be easily recovered from special forms of DFA description (say, description of the minimal DFA accepting a 1-element language). But it does not help to improve the reduction (5) because we are interested in regular realizability problems (the answer depends on a language and should be the same for all automata recognizing the language).

3 Universality of RR Problems

In the case of language reductions we are able to prove a weaker version of Theorem 1 using disjunctive reductions.

Theorem 2. *For any nonempty language X there exists a filter L such that*

$$X \leqslant_{\mathrm{m}}^{\log} \mathrm{RR}(L) \leqslant_{\mathrm{dtt}}^{\mathrm{FNL}} X. \tag{6}$$

In the proof of Theorem 2 we use the other extreme case of conditions (4). Namely, we choose $Y = X_f \stackrel{\text{def}}{=} \overline{f(\overline{X})}$.

The idea behind the proof is to approximate the inversion of a monoreduction as close as possible. The difficulty of inversion stems from the fact mentioned in Remark 2: instances of an RR problem are all regular languages and the RR problem might be hard for languages that are not 1-element languages constituting the image of the monoreduction. To overcome this difficulty we choose a filter X_f for a map f such that for most regular languages the problem $\mathrm{RR}(X_f)$ is trivial.

The first step toward implementation of this idea is to make an RR problem trivial for all infinite languages.

Definition 3. *An infinite language is* regularly immune *if it does not contain any infinite regular language.*

Suppose that $f(\{0,1\}^*)$ is contained in a regularly immune language. Then any infinite regular language intersects $\overline{f(\{0,1\}^*)} \subset \overline{f(\overline{X})} = X_f$. Thus for any infinite instance of $\mathrm{RR}(X_f)$ the answer is positive.

For a finite instance of $\mathrm{RR}(X_f)$ an m-reducing algorithm should indicate a word from X. It seems too hard for arbitrary X.

By this reason the second step is to use disjunctive reductions. In this case a reducing algorithm should just produce the list of all words accepted by an automaton and this task is much easier.

Note that the cardinality of a finite language can be exponentially larger than the number of states in a DFA recognizing the language.

Thus the next step is to choose a regularly immune set D possessing a special property: the cardinality of any regular language in D is polynomially upper-bounded by the number of states in a DFA recognizing the language.

Last but not least, all actions mentioned above should be efficiently implemented. We are going to construct a $\leqslant^{\mathrm{FNL}}_{\mathrm{dtt}}$-reduction. So we need FNL implementations.

To implement the plan outlined above we start from specification of D. Let β be the Thue–Morse morphism $\beta(0) = 01$, $\beta(1) = 10$ and let $\mathrm{sq}(\cdot)$ be the map

$$\mathrm{sq}\colon x \mapsto \beta(x)1^2 0^{|x|^2+3}\beta(x)1^2 0^{|x|^2+3}. \tag{7}$$

We choose $D = \mathrm{Im}(\mathrm{sq})$, i.e. the image of all binary words under the map sq.

To prove that D is regularly immune one can use the Parikh theorem [6,8]. It says that the lengths of words from a regular (or even context-free) language form a *semilinear set*, i.e. a finite union of arithmetic progressions.

Lemma 2. *D is regularly immune.*

Proof. The lengths of words from D form the set $\{2n^2 + 4n + 10 : n \in \mathbb{N}\}$. It is clear that its intersection with any arithmetic progression is finite. □

To give an upper bound on the cardinality of a regular language contained in D we make a couple of observations. By definition words from D are squares. Moreover, they are incomparable in the following sense.

Proposition 4. *Let $pq_1, pq_2 \in D$. Then $p \in \beta(\{0,1\}^*)(\varepsilon \cup \{0,1\})$.*

Proof. Note that $w = \mathrm{sq}(x)$ can be recovered from the prefix $\beta(x)11$: the first occurrence of 11 starting at an even position[3] signals that the prefix $\beta(x)$ is completed and x is uniquely determined by this prefix. □

Proposition 4 plays an important role in our arguments. Typically we will use the fact that a common prefix of words from D does not contain 0^3 (easily follows from Proposition 4). Just now we indicate another simple corollary.

Corollary 2. *No word from D is a prefix of another word from D.*

Now we are ready to prove an upper bound required in the outlined above plan of the proof of Theorem 2.

Lemma 3. *Let A be a DFA such that $L(A) \subset D$. Then $|L(A)| \leqslant |Q|$, where Q is the state set of A.*

Proof. It is sufficient to consider the minimal DFA B recognizing $L(A)$. The states of B are in one-to-one correspondence with Myhill–Nerode classes. Consider two words $u_1u_1 \neq u_2u_2$ in $L(A)$. We prove that $u_1u_2 \notin L(A)$. It implies that u_1 and u_2 are not equivalent and the number of Myhill–Nerode classes is not less than $|L(A)|$.

Suppose that $u_1u_2 = vv \in L(A)$. Either u_1 is a prefix of v or v is a prefix of u_1. In both cases we come to a contradiction with Proposition 4: both u_1 and v contain 0^3 as a subword. □

To prove Theorem 2 we also need several algorithmic facts.

[3] We enumerate positions in a word starting with 0.

At first, note that the binary representation of the length of a word x has size $O(\log |x|)$. So arithmetic operations with numbers of this magnitude can be performed using log space. This fact is widely used below.

The following statements are proved easily.

Proposition 5. *Infiniteness of a regular language is in* NLOG *provided a language is represented by a DFA recognizing it.*

Proposition 6. *Halves of words from D, i.e. words in the form $\beta(u)1^2 0^{|u|^2+3}$, can be recognized using log space. Moreover, a recognizing algorithm can read input in one way.*

Proposition 7. *The membership problem for the language D is in* LOG. *The map* sq *and the inverse map* sq^{-1} *are log space computable.*

Our next goal is an algorithm that extracts words accepted by an automaton recognizing a finite language and outputs their images under the inverse map sq^{-1} provided the regular language in question is contained in the set D.

Let A be a DFA such that $L(A) \subset D$, and let Q be the state set, s the initial state, and t an accepting state of A. We denote by δ_A the transition function $\delta\colon Q \times \{0,1\}^* \to Q$ of A extended to the set of words in the input alphabet of A in a natural way.

We define the set $Q_k(t) \subseteq Q$ of states as follows: $q \in Q_k(t)$ iff $\delta_A(s,u) = q$ and $\delta_A(q,u) = t$ for u such that $uu \in D$ and $|u| = k$.

Note that for any $q \in Q_k(t)$ the word uu is unique. Indeed, if two words uu, vv with $u \neq v$ satisfy the conditions listed above with the same q, then $uv \in L(A)$ and the word is not a square. But this contradicts the assumption $L(A) \subset D$.

We will call this unique word uu an (A,t,q,k)-word.

Proposition 8. *The set of quadruples $\langle A,t,q,k\rangle$ such that A is a DFA description, t,q are states of A, $k \leqslant |Q|$, $q \in Q_k(t)$, is in* NLOG.

Moreover, there exists an FNL *algorithm that decodes (A,t,q,k)-words, i.e. running on an input $\langle A,t,q,k\rangle$ the algorithm outputs x such that* sq(x) *is a (A,t,q,k)-word.*

Proof. The recognizing algorithm guesses symbols of uu one by one (uu is the word from the definition of the set $Q_k(t)$). While reading a symbol the following actions are performed:

- simulating actions of the automaton A that reads u starting from two states: s and q (the whole computation paths are not stored, the algorithm maintains current states only);
- counting the length of u;
- simulating an operation of the algorithm from Proposition 6.

After guessing k symbols the algorithm checks conditions $\delta_A(s,u) = q$ and $\delta_A(q,u) = t$. It also checks that the algorithm from Proposition 6 gives the positive answer. If all checks are successful, then the algorithm accepts the quadruple $\langle A,t,q,k\rangle$.

Correctness of the algorithm is clear as well as the required log space bound.

To construct a decoding algorithm we modify the recognizing algorithm.

Let $u = \beta(x)1^20^{|x|^2+3}$ be the half of an (A, t, q, k)-word.

Guessing a symbol is replaced in the decoding algorithm by two trials. For each possible variant α of the next symbol (there are two of them) the modified algorithm simulates reading the pair $\alpha\bar{\alpha}$ and asks an NLOG oracle about possibility of successful completion of the recognizing algorithm with current data (this can be done using nlog space). The oracle answers positively for exactly one value of α. This value is the next symbol of x and the decoding algorithm outputs it. □

Finally, the reducing algorithm needs to detect trivial cases.

Lemma 4. *Testing conjunction* $|L(A)| < \infty$ *and* $L(A) \subset D$ *is in* NLOG.

We use the well-known equality NLOG = co-NLOG [5,10]. So it is sufficient to test disjunction $|L(A)| = \infty$ or $L(A) \setminus D \neq \varnothing$.

The first check is Proposition 5.

The second is based on the following observation. If $L(A) \setminus D \neq \varnothing$, then

- either there exists $w \in L(A)$ of odd length;
- or there exists $w \in L(A)$ of even length that is not a square;
- or there exists $ww \in L(A) \setminus D$.

All three conditions are in NLOG. The last can be verified by proper modifications of the algorithms from Propositions 6 and 8. So the algorithm for Lemma 4 should nondeterministically choose one of them and test the chosen condition.

Tying up loose ends we get the proof of the main result.

Proof (of Theorem 2). Let us prove that

$$X \leqslant_{\mathrm{m}}^{\log} \mathrm{RR}(X_{\mathrm{sq}}) \leqslant_{\mathrm{dtt}}^{\mathrm{FNL}} X. \tag{8}$$

The first reduction is the monoreduction by the map sq. By Proposition 7 this monoreduction is log space computable.

Now we construct the second reduction. Let A be an instance of $\mathrm{RR}(X_{\mathrm{sq}})$. The reducing algorithm checks infiniteness of $L(A)$ and $L(A) \setminus D \neq \varnothing$ using Lemma 4. If $L(A)$ is infinite or it contains a word from $\overline{D}$, then the reducing algorithm forms a query list of length 1 containing a fixed element $x_0 \in X$ and outputs the list.

Otherwise the algorithm tries all possible values of k from 0 to $|Q| - 1$, all accepting states t and all states $q \in Q_k(t)$ (to check this condition the recognizing algorithm from Proposition 8 is used).

For each triple k, t, q, where $q \in Q_k(t)$, there exists a unique word $uu = \mathrm{sq}(x) \in D$. The algorithm outputs x using the decoding algorithm from Proposition 8 and places x in the query list.

To prove correctness of the reduction recall that D is regularly immune. So every infinite regular language has a common word with X_{sq}. If $L(A)$ is finite and

contains a word from $\bar{D}$, then it has nonempty intersection with X_{sq}. Finally, if a finite language $L(A)$ is contained in D then $L(A) \cap X_{\mathrm{sq}} \neq \varnothing$ iff the output list contains a word from X. □

Corollary 3. *Each class Σ_k^p, Π_k^p of the polynomial hierarchy contains an RR problem that is complete for the class under $\leqslant_{\mathrm{m}}^{\mathrm{FNL}}$-reductions (and under polynomial reductions).*

4 Universality of Generalized Nondeterministic Models

As it mentioned above, RR problems are closely related to the models of generalized nondeterminism (GNA) introduced in [12]. GNA classes are parametrized by languages of infinite words (certificates). It was shown in [13] that each GNA class contains an RR problem complete for the class under $\leqslant_{\mathrm{m}}^{\log}$-reductions. The filter of the RR problem consists of prefixes of certificates for the GNA class.

Note that filters in the proof of Theorem 2 are not prefix closed. So they do not correspond to any GNA class. What RR problems correspond to GNA classes? To be prefix closed is a necessary condition only. The second condition is the following: each filter word is a proper prefix of a filter word. These two conditions guarantee that the filter is the prefix set for some set of certificates.

To satisfy the second condition a very simple modification of a filter is needed.

Proposition 9. *If $L \subseteq \Sigma^*$ and $\# \notin \Sigma$, then*

$$\mathrm{RR}(L) \leqslant_{\mathrm{m}}^{\log} \mathrm{RR}(L\#^*) \leqslant_{\mathrm{m}}^{\log} \mathrm{RR}(L).$$

Thus for any RR problem with a prefix closed filter there exists an equivalent RR problem with a filter that is the prefix set of a language of infinite words.

So universality of GNA classes follows from the following generalization of Theorem 2.

Theorem 3. *For any nonempty language X there exists a prefix closed filter P such that*

$$X \leqslant_{\mathrm{m}}^{\log} \mathrm{RR}(P) \leqslant_{\mathrm{dtt}}^{\mathrm{FNL}} X. \tag{9}$$

We follow in the proof of Theorem 3 the same plan as for Theorem 2. Again we use monoreductions. We choose the filter P in the form X'_f for a map f and a language X' which is $\leqslant_{\mathrm{m}}^{\log}$-equivalent to X.

The map f is chosen to satisfy the following conditions: the image of f is the set $D^{\uparrow}$ of all extensions of words of D (i.e. words uv, $u \in D$); $f(X') \subset D$; $f(\overline{X'})$ contains all extensions with nonempty suffixes (thus $f(\overline{X'})$ is suffix closed and $\overline{f(\overline{X'})}$ is prefix closed).

The set $D^{\uparrow}$ is not regularly immune. But in the crucial case $L(A) \subset D^{\uparrow}$ the set of D-prefixes $R_D = \{w : w \in D \text{ and } wv \in L(A)\}$ is finite. In the proof of Theorem 3 this set plays a role of a finite regular language. In particular, the cardinality of R_D is polynomially upperbounded by the number of states of

the automaton A and the list of elements of R_D can be produced by an FNL algorithm.

Technical details of the proof are omitted in this extended abstract.

Taking into account normality of classes in the polynomial hierarchy we get the following corollary.

Corollary 4. *For each k, the classes Σ_k^p, Π_k^p contain RR problems with prefix-closed filters that are complete for the corresponding classes under polynomial reductions.*

Acknowledgments. The author is grateful to the anonymous referees for their valuable and helpful comments.

References

1. Allender, E., Ogihara, M.: Relationships among PL, #L, and the determinant. Informatique Théorique et Applications 30, 1–21 (1996)
2. Àlvarez, C., Balcázar, J.L., Jenner, B.: Adaptive logspace reducibility and parallel time. Mathematical Systems Theory 28(2), 117–140 (1995)
3. Anderson, T., Loftus, J., Rampersad, N., Santean, N., Shallit, J.: Detecting palindromes, patterns and borders in regular languages. Information and Computation 207, 1096–1118 (2009)
4. Arora, S., Barak, B.: Computational complexity: a modern approach. Cambridge Univ. Press, Cambridge (2009)
5. Immerman, N.: Nondeterministic space is closed under complement. SIAM Journal on Computing 17(5), 935–938 (1988)
6. Kozen, D.: Automata and Computability. Springer, New York (1997)
7. Ladner, R.E., Lynch, N.A.: Relativization of questions about log space computability. Mathematical Systems Theory 10(1), 19–32 (1976)
8. Parikh, R.J.: On context-free languages. Journal of the ACM 13(4), 570–581 (1966)
9. Reinhardt, K., Allender, E.: Making nondeterminism unambiguous. SIAM Journal on Computing 29(4), 1118–1131 (2000)
10. Sipser, M.: Introduction to the theory of computation, 3rd edn. Cengage Learning, Boston (2012)
11. Vyalyi, M.N.: On models of a nondeterministic computation. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 334–345. Springer, Heidelberg (2009)
12. Vyalyi, M.N.: On nondeterminism models for two-way automata. In: Proc. VIII Int. Conf. on Discrete Models in Control System Theory, pp. 54–60. MAKS Press, Moscow (2009) (in Russian)
13. Vyalyi, M.N.: On regular realizability problems. Problems of Information Transmission 47(4), 342–352 (2011)
14. Vyalyi, M.N.: On complexity of regular realizability problems. arXiv:1211.0606v2

Potential Functions in Strategic Games*

Paul G. Spirakis[1,2] and Panagiota N. Panagopoulou[2]

[1] Computer Engineering and Informatics Department, University of Patras
[2] Computer Technology Institute & Press "Diophantus"
{spirakis,panagopp}@cti.gr

Abstract. We investigate here several categories of strategic games and antagonistic situations that are known to admit potential functions, and are thus guaranteed to either possess pure Nash equilibria or to stabilize in some form of equilibrium in cases of stochastic potentials. Our goal is to indicate the generality of this method and to address its limits.

1 Introduction

A *strategic game* is a model of interactive decision making, helping us in analyzing situations in which two or more individuals, called *players*, make decisions (or choose *actions*) that will influence one another's welfare. The most important solution concept of a strategic game is the well-known *Nash equilibrium* [14], which captures a steady state of the game, in the sense that no player has an incentive to change her action if all the other players preserve theirs. The classical theorem of Nash [14] proves that every finite game has a *randomized* Nash equilibrium, i.e., there exists a combination of *mixed strategies*, one for each player, such that no player can increase her expected payoff by unilaterally deviating. A mixed strategy for a player is actually a probability distribution over the set of her available actions. However, Nash's proof of existence is non-constructive, and the problem of computing a Nash equilibrium has been identified among the "inefficient proofs of existence" [15], and it was eventually shown to be complete for the complexity class PPAD [2].

The apparent difficulty of computing a randomized Nash equilibrium, together with the fact that the concept of non-deterministic strategies has received much criticism in game theory, raises the natural question of what kind of games possess a *pure* Nash equilibrium, i.e., a Nash equilibrium where each player chooses deterministically one of her available actions. In this paper we

* This work was supported by the project Algorithmic Game Theory, co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales, Investing in knowledge society through the European Social Fund, and by the EU FP7/2007-2013 (DG CONNECT – Communications Networks, Content and Technology Directorate General, Unit H5 – Smart Cities & Sustainability), under grant agreement no. 288094 (project eCOMPASS).

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 283–297, 2013.

discuss strategic games that are guaranteed to have pure Nash equilibria via the existence of *potential functions*, which enable the application of optimization theory to the study of pure Nash equilibria. More precisely, a potential function for a game is a real-valued function, defined on the set of possible action combinations (or *outcomes*) of the game, such that the pure equilibria of the game are precisely the local optima of the potential function. If a game admits a potential function, there are nice consequences for the existence and tractability of pure Nash equilibria. In particular, if the game is finite, i.e., the player set and actions sets are finite, then the potential function achieves a global optimum, which is also a local optimum, and hence the game has at least one equilibrium in pure strategies. Going one step further, in any such game, *the Nash dynamics converges*. This means that the directed graph with outcomes as nodes and payoff-improving defections by individual players as edges has no cycles, implying that it has a sink corresponding to a pure Nash equilibrium, and that if we start with an arbitrary action combination and let one player at a time perform an improvement step, i.e., change her action to increase her payoff, then such a sink will be eventually reached.

We survey here several categories of strategic games that are known to admit potential functions (or some kind of generalization of a potential). Our goal is to give some insight on both the characterization of games guaranteed to possess pure Nash equilibria and the complexity of reaching such an equilibrium via a sequence of selfish payoff-improving steps. We note that potential games are not the only class of games known to possess pure Nash equilibria; another well-studied one is the class of *games of strategic complementarities*, introduced in [21]. Roughly speaking, a game has strategic complementarities if there is an order on the set of the players' pure strategies such that an increase in one player's strategy makes the other players want to increase their strategies as well. Based on a fixpoint theorem due to Tarski [20], games of strategic complementarities are known to possess at least one equilibrium in pure strategies, and furthermore, the set of pure Nash equilibria has a certain order structure. Uno [22] revealed an important relation between potential games and games with strategic complementarities: any finite game with strategic complementarities admits a *nested* pseudo-potential (which are extensions of potentials). A couple of natural questions are raised here: (1) How exactly are these two classes of games related, i.e., does the existence of (even a generalized notion of) a potential function is equivalent to the existence of strategic complementarities? (2) Are these two notions necessary in order to characterize the existence of pure Nash equilibria, or do there exist classes of games neither admitting any kind of potential nor having complementarities that are nevertheless guaranteed to have pure equilibria? The answer to these questions could give us more insight in order to better understand the nature of games that have pure Nash equilibria.

2 Games and Potential Functions

Strategic Games and Nash Equilibria. A *game* refers to any situation in which two or more decision-makers interact. We focus on *finite games in*

strategic form: A **finite strategic form game** is any Γ of the form $\Gamma = \langle N, (C_i)_{i\in N}, (u_i)_{i\in N}\rangle$ where N is a finite nonempty set, and, for each $i \in N$, C_i is a finite nonempty set and u_i is a function from $C = \times_{j\in N} C_j$ into the set of real numbers $\mathbb{R}$. In the above definition, N is the set of players in the game Γ. For each player $i \in N$, C_i is the set of *actions* available to player i. When the strategic form game Γ is played, each player i must choose one of the actions in the set C_i. For each combination of actions, or *pure strategy profile* $c = (c_j)_{j\in N} \in C$ (specifying one action for each player), the number $u_i(c)$ represents the *payoff* that player i would get in this game if c were the combination of actions implemented by the players. When we study a strategic form game, we assume that all the players choose their actions simultaneously.

A *pure Nash equilibrium* of a strategic game is a combination of actions, one for each player, from which no player has an incentive to unilaterally deviate.

Definition 1. *A **pure Nash equilibrium** of game $\Gamma = \langle N, (C_i)_{i\in N}, (u_i)_{i\in N}\rangle$ is a pure strategy profile $c = (c_j)_{j\in N} \in C$ such that, for all players $i \in N$,*

$$u_i(c) \geq u_i(c'_i, c_{-i}) \quad \forall c'_i \in C_i \ .$$

Not all games are guaranteed to possess a pure Nash equilibrium; however, if we extend the strategy set of each player to include any probability distribution on her set of actions, then a (*mixed strategy*) Nash equilibrium is guaranteed to exist [14].

Potential Games. Potential games, defined in [13], are games with the property that the incentive of all players to unilaterally deviate from a pure strategy profile can be expressed in one global function, the potential function.

Fix an arbitrary game in strategic form $\Gamma = \langle N, (C_i)_{i\in N}, (u_i)_{i\in N}\rangle$ and some vector $\mathbf{b} = (b_1, \ldots, b_{|N|}) \in \mathbb{R}^{|N|}_{>0}$. A function $P : C \to \mathbb{R}$ is called

– An *ordinal potential* for Γ if, $\forall c \in C$, $\forall i \in N$, $\forall a \in C_i$,

$$P(a, c_{-i}) - P(c) > 0 \iff u_i(a, c_{-i}) - u_i(c) > 0 \ . \tag{1}$$

– A $\mathbf{b}$-*potential* for Γ if, $\forall c \in C$, $\forall i \in N$, $\forall a \in C_i$,

$$P(a, c_{-i}) - P(c) = b_i \cdot (u_i(a, c_{-i}) - u_i(c)) \ . \tag{2}$$

– An *exact potential* for Γ if it is a $\mathbf{b}$-potential for Γ where $b_i = 1$ for all $i \in N$.

It is straightforward to see that the existence of an ordinal, exact, or $\mathbf{b}$-potential function P for a finite game Γ guarantees the existence of at least one pure Nash equilibrium in Γ: each local optimum of P corresponds to a pure Nash equilibrium of Γ and vice versa. Thus the problem of finding pure Nash equilibria of a potential game Γ is equivalent to finding local optima for the optimization problem with state space the pure strategy space C of the game and objective the potential function of the game.

Furthermore, the existence of a potential function P for a game $\Gamma = \langle N, (C_i)_{i\in N}, (u_i)_{i\in N}\rangle$ implies a straightforward algorithm for constructing a pure

Nash equilibrium of Γ: The algorithm starts from an arbitrary strategy profile $c \in C$ and, at each step, one single player performs a *selfish step*, i.e., switches to a pure strategy that strictly improves her payoff. Since the payoff of the player increases, P increases as well. When no move is possible, i.e., when a pure strategy profile $\hat{c}$ is reached from which no player has an incentive to unilaterally deviate, then $\hat{c}$ is a pure Nash equilibrium and a local optimum of P. This procedure however does not imply that the computation of a pure Nash equilibrium can be done in polynomial time, since the improvements in the potential can be very small and too many.

3 Congestion Games and Selfish Routing

Congestion games, introduced in [19], are games in which each player chooses a particular subset of resources out of a family of allowable subsets for her (her strategy set), constructed from a basic set of primary resources for all the players. The *delay* associated with each primary resource is a non-decreasing function of the number of players who choose it, and the total delay received by each player is the sum of the delays associated with the primary resources she chooses. In [19] it was shown that any game in this class possesses at least one Nash equilibrium in pure strategies, and this result follows from the existence of a potential function. Later, Monderer and Shapley [13] showed that every finite potential game is *isomorphic* to a congestion game.

3.1 Congestion Games

A *congestion model* $\langle N, E, (\Pi_i)_{i\in N}, (d_e)_{e\in E}\rangle$ is defined as follows. N denotes the set of players $\{1, \ldots, n\}$. E denotes a finite set of resources. For $i \in N$ let Π_i be the set of strategies of player i, where each $\varpi_i \in \Pi_i$ is a nonempty subset of resources. For $e \in E$ let $d_e : \{1, \ldots, n\} \to \mathbb{R}$ denote the delay function, where $d_e(k)$ denotes the cost (e.g. delay) to each user of resource e, if there are exactly k players using e. The *congestion game* associated with this congestion model is the game in strategic form $\langle N, (\Pi_i)_{i\in N}, (u_i)_{i\in N}\rangle$, where the payoff functions u_i are defined as follows: Let $\Pi \equiv \times_{i\in N}\Pi_i$. For all $\varpi = (\varpi_1, \ldots, \varpi_n) \in \Pi$ and for every $e \in E$ let $\sigma_e(\varpi)$ be the number of users of resource e according to the configuration ϖ: $\sigma_e(\varpi) = |\{i \in N : e \in \varpi_i\}|$. Define $u_i : \Pi \to \mathbb{R}$ by $u_i(\varpi) = -\sum_{e\in\varpi_i} d_e(\sigma_e(\varpi))$. In a *network congestion game* the families of subsets Π_i are represented implicitly as paths in a network. We are given a directed network $G = (V, E)$ with the edges playing the role of resources, a pair of nodes $(s_i, t_i) \in V \times V$ for each player i and the delay function d_e for each $e \in E$. The strategy set of player i is the set of all paths from s_i to t_i. If all origin-destination pairs (s_i, t_i) of the players coincide with a unique pair (s, t) we have a *single-commodity network congestion game* and then all users share the same strategy set, hence the game is symmetric.

In a *weighted congestion model* we allow the users to have different demands, and thus affect the resource delay functions in a different way, depending on their own weights. The *weighted congestion game* associated with a weighted congestion model is the game in strategic form $\langle (w_i)_{i\in N}, (\Pi_i)_{i\in N}, (u_i)_{i\in N}\rangle$, where the payoff functions u_i are defined as follows. For any configuration $\varpi \in \Pi$ and for all $e \in E$, let $\Lambda_e(\varpi) = \{i \in N : e \in \varpi_i\}$ be the set of players using resource e according to ϖ. The cost $\lambda^i(\varpi)$ of user i for adopting strategy $\varpi_i \in \Pi_i$ in a given configuration ϖ is equal to the cumulative delay $\lambda_{\varpi_i}(\varpi)$ on the resources that belong to ϖ_i: $\lambda^i(\varpi) = \lambda_{\varpi_i}(\varpi) = \sum_{e\in\varpi_i} d_e(\theta_e(\varpi))$, where, for all $e \in E$, $\theta_e(\varpi) \equiv \sum_{i\in\Lambda_e(\varpi)} w_i$ is the load on resource e with respect to the configuration ϖ. The payoff function for player i is then $u_i(\varpi) = -\lambda^i(\varpi)$. A configuration $\varpi \in \Pi$ is a pure Nash equilibrium if and only if, for all $i \in N$, $\lambda_{\varpi_i}(\varpi) \leq \lambda_{\pi_i}(\pi_i, \varpi_{-i}) \quad \forall \pi_i \in \Pi_i$, where (π_i, ϖ_{-i}) is the same configuration as ϖ except for user i that has now been assigned to path π_i. In a *weighted network congestion game* the strategy sets Π_i are represented implicitly as $s_i - t_i$ paths in a directed network $G = (V, E)$.

Since the payoff functions u_i of a congestion game can be implicitly computed by the resource delay functions d_e, in the following we will denote a general (weighted or unweighted) congestion game by $\langle N, E, (\Pi_i)_{i\in N}, (w_i)_{i\in N}, (d_e)_{e\in E}\rangle$.

The following theorem [19], [13] proves the strong connection of unweighted congestion games with the exact potential games.

Theorem 1 ([19], [13]). *Every (unweighted) congestion game is an exact potential game.*

Proof. Fix an arbitrary (unweighted) congestion game $\Gamma = \langle N, E, (\Pi_i)_{i\in N}, (d_e)_{e\in E}\rangle$. For any pure strategy profile $\varpi \in \Pi$, the function

$$\Phi(\varpi) = \sum_{e\in\bigcup_{i\in N}\varpi_i} \sum_{k=1}^{\sigma_e(\varpi)} d_e(k) \tag{3}$$

(introduced in [19]) is an exact potential function for Γ. □

The converse of Theorem 1 does not hold in general, however in [13] it was proven that every (finite) exact potential game Γ is *isomorphic* to an unweighted congestion game. In [9] it was shown that this does not hold for the case of weighted congestion games. In particular, it was shown that there exist weighted single commodity network congestion games with resource delays being either linear or 2-wise linear functions of the loads, for which pure Nash equilibria cannot exist. Furthermore, there exist weighted single-commodity network congestion games which admit no exact potential function, even when the resource delays are identical to their loads. On the other hand, it was shown that any weighted (multi-commodity) network congestion game with linear resource delays admits a weighted potential function, yielding the existence of a pure Nash equilibrium which can be constructed in pseudo-polynomial time:

Theorem 2 ([9]). *For any weighted multi-commodity network congestion game* $\langle N, E, (\Pi_i)_{i\in N}, (w_i)_{i\in N}, (d_e)_{e\in E}\rangle$ *with linear resource delays, i.e.,* $d_e(x) =$

$a_e x + b_e, e \in E, a_e, b_e \geq 0$, *function* $\Phi : \Pi \to \mathbb{R}$ *defined for the configuration* $\varpi \in \Pi$ *as*

$$\Phi(\varpi) = \sum_{e \in E} d_e(\theta_e(\varpi))\theta_e(\varpi) + \sum_{i=1}^{n} \sum_{e \in \varpi_i} d_e(w_i)w_i$$

is a **b***-potential for* $b_i = 1/2w_i, i \in N$.

In [16], it was shown that weighted (multi-commodity) network congestion games with resource delays *exponential* to their loads also admit a weighted potential function:

Theorem 3 ([16]). *For any weighted multi-commodity network congestion game* $\langle N, E, (\Pi_i)_{i \in N}, (w_i)_{i \in N}, (d_e)_{e \in E} \rangle$ *with exponential resource delays, i.e.,* $d_e(x) = \exp(x)$, *function* $\Phi : \Pi \to \mathbb{R}$ *defined for the configuration* $\varpi \in \Pi$ *as*

$$\Phi(\varpi) = \sum_{e \in E} \exp(\theta_e(\varpi))$$

is a **b***-potential for* $b_i = \frac{\exp(w_i)}{\exp(w_i)-1}, i \in N$.

3.2 Concurrent Congestion Games and Coalitions

In this section we study the effect of concurrent greedy moves of players in atomic congestion games where n selfish agents (players) wish to select a resource each (out of m resources) so that their selfish delay there is not much. This problem of "maintaining" global progress while allowing concurrent play is examined and answered in [7] with the use of potential functions. Two orthogonal settings are examined: (i) A game where the players decide their moves without global information, each acting "freely" by sampling resources randomly and locally deciding to migrate (if the new resource is better) via a random experiment. Here, the resources can have quite arbitrary latency that is load dependent. (ii) An "organised" setting where the players are pre-partitioned into selfish groups (coalitions) and where each coalition does an improving coalitional move.

Concurrent Congestion Games. A setting where selfish players perform best improvement moves in a sequential fashion and eventually reach a Nash equilibrium is not appealing to modern networking, where simple decentralized distributed protocols better reflect the essence of the networks liberal nature. In fact, it is unrealistic to assume that global coordination between the players can be enforced and that the players are capable of monitoring the configuration of the entire network. Furthermore, even if a player can grasp the whole picture, it is computationally demanding to decide her best move.

In [7], the advantages and the limitations of such a distributed protocol for congestion games on parallel edges under very general assumptions on the latency functions are investigated. A restricted model of distributed computation that allows a limited amount of global knowledge is adopted. In each round, every

player can only select a resource uniformly at random and check its current latency. Migration decisions are made concurrently on the basis only of the current latency of the resource to which a player is assigned and the current latency of the resource to which the player is about to move. Migration decisions take advantage of local coordination between the players currently assigned to the same resource, in the sense that at most one player is allowed to depart from each resource. The only global information available to the players is an upper bound α on the slope of the latency functions.

In this setting, an (ε, α)-approximate equilibrium ((ε, α)-EQ), which is dictated by the very limited information available to the players, is a state where at most εm resources have latency either considerably larger or considerably smaller than the current average latency. This definition relaxes the notion of exact pure Nash equilibria and introduces a meaningful notion of approximate (bicriteria) equilibria for the myopic model of migrations described above. In particular, an (ε, α)-EQ guarantees that unless a player uses an overloaded resource (i.e., a resource with latency considerably larger than the average latency), the probability that she finds (by uniform sampling) a resource to migrate and significantly improve her latency is at most ε. Furthermore, it is unlikely that any (ε, α)-EQ reached by the protocol assigns a large number of players to overloaded resources (even though this possibility is allowed by the definition of an (ε, α)-EQ).

A simple oblivious protocol for this restricted model of distributed computation is presented in [7]. According to this myopic protocol, in parallel each player selects a resource uniformly at random in each round and checks whether she can significantly decrease her latency by moving to the chosen resource. If this is the case, the player becomes a potential migrant. The protocol uses a simple local probabilistic rule that selects at most one (this is a local decision between players on the same resource) potential migrant to defect from each resource.

If the number of players is $\Theta(m)$ and they start from a random initial allocation, the protocol reaches an (ε, α)-EQ in $O(\log(E[\Phi(0)]/\Phi_{\min}))$ time, where $E[\Phi(0)]$ is Rosental's expected potential value as the game starts and $\Phi_{\min}$ is the corresponding value at a Nash equilibrium. The proof of convergence given in [7] is technically involved and is omitted here.

Congestion Games with Coalitions. In many practical situations, the competition for resources takes place among coalitions of players instead of individuals. In such settings, it is important to know how the competition among coalitions affects the rate of convergence to an (approximate) pure Nash equilibrium. A *congestion game with coalitions* is a natural model for investigating the effects of non-cooperative resource allocation among static coalitions. In congestion games with coalitions, the coalitions are static and the selfish cost of each coalition is the total delay of its players.

In this setting, [7] present an upper bound on the rate of (sequential) convergence to approximate Nash equilibrium in single-commodity linear congestion games with static coalitions. The restriction to linear latencies is necessary

because this is the only class of latency functions for which congestion games with static coalitions is known to admit a potential function and a pure Nash equilibrium. Sequences of ε-moves are considered, i.e., selfish deviations that improve the coalitions' total delay by a factor greater than ε. Combining the approach of [3] with the potential function of [8], it is shown that if the coalition with the largest improvement in its total delay moves in every round, an approximate Nash equilibrium is reached in a small number of steps.

More precisely, for any initial configuration s_0, every sequence of largest improvement ε-moves reaches an approximate Nash equilibrium in at most $\frac{kr(r+1)}{\varepsilon(1-\varepsilon)} \log \Phi(s_0)$ steps, where k is the number of coalitions, $r = \lceil \max_{j\in[k]}\{n_j\} / \min_{j\in[k]}\{n_j\} \rceil$ denotes the ratio between the size of the largest coalition and the size of the smallest coalition, and $\Phi(s_0)$ is the initial potential. This bound holds even for coalitions of different size, in which case the game is not symmetric. This bound implies that in network congestion games, where a coalition's best response can be computed in polynomial time, an approximate Nash equilibrium can be computed in polynomial time. Moreover, in the special case that the number of coalitions is constant and the coalitions are almost equisized, i.e. when, $k = \Theta(1)$ and $r = \Theta(1)$, the number of ε-moves to reach an approximate Nash equilibrium is logarithmic in the potential of the initial state.

3.3 Social Ignorance in Congestion Games

Most of the work on congestion games focuses on the full information setting, where each player knows the precise weights and the actual strategies of all players, and her strategy selection takes all this information into account. In many typical applications of congestion games however, the players have incomplete information not only about the weights and the strategies, but also about the mere existence of (some of) the players with whom they compete for resources. In fact, in many applications, it is both natural and convenient to assume that there is a *social context* associated with the game, which essentially determines the information available to the players. In particular, one may assume that each player has complete information about the set of players in her *social neighborhood*, and limited (if any) information about the remaining players.

In this section we investigate how such social-context-related information considerations affect the inefficiency of pure Nash equilibria and the convergence rate to approximate pure Nash equilibria. To come up with a manageable setting that allows for some concrete answers, we make the simplifying assumption that each player has complete information about the players in her social neighborhood, and no information whatsoever about the remaining players. Therefore, since each player is not aware of the players outside her social neighborhood, her individual cost and her strategy selection are not affected by them. In fact, this is the model of *graphical congestion games*, introduced by Bilò et al. [1]. The new ingredient in the definition of graphical congestion games is the social graph, which represents the players social context. The social graph is defined on the set of players and contains an edge between each pair of players that know

each other. The basic idea (and assumption) behind graphical congestion games is that the individual presumed cost of each player only depends on the players in her social neighborhood, and thus her strategy selection is only affected by them.

The social graph $G(V, E)$ is defined on the set of players $V = N$ and contains an edge $\{i, j\} \in E$ between each pair of players i, j that know each other. We consider graphical games with weighted players and simple undirected social graphs.

Given a graphical congestion game with a social graph $G(V, E)$, a configuration s and a resource e, let $V_e(s) = \{i \in V : e \in s_i\}$ be the set of players using e in s, let $G_e(s)(V_e(s), E_e(s))$ be the social subgraph of G induced by $V_e(s)$, and let $n_e(s) = |V_e(s)|$ and $m_e(s) = |E_e(s)|$. For each player i (not necessarily belonging to $V_e(s)$), let $\Gamma_e^i(s) = \{j \in V_e(s) : \{i, j\} \in E\}$ be i's social neighborhood among the players using e in s. In any configuration s, a player i is aware of a *presumed congestion* $s_e^i = w_i + \sum_{j \in \Gamma_e^i(s)} w_j$ on each resource e, and of her *presumed cost* $p_i(s) = \sum_{e \in s_i} w_i(a_e s_e^i + b_e)$. We note that the presumed cost coincides with the actual cost if the social graph is complete. For graphical congestion games, a configuration s is a pure Nash equilibrium if no player can improve her *presumed cost* by unilaterally changing her strategy.

In [1] it is shown that graphical linear congestion games with unweighted players are potential games:

Theorem 4. *Every graphical linear congestion game defined over an undirected social graph is an exact potential game, and thus always converges to a Nash equilibrium.*

Proof. The potential function establishing the result is

$$\Phi(s) = \sum_{e \in E} \left[a_e(m_e(s) + n_e(s)) + b_e n_e(s) \right] \quad .$$

□

In [6] it is shown that graphical linear congestion games with weighted players also admit a potential function:

Theorem 5. *Every graphical linear congestion game with weighted players admits a potential function, and thus a pure Nash equilibrium.*

Proof. The potential function establishing the result is

$$\Phi(s) = \sum_{e \in R} \left[a_e \left(\sum_{i \in V_e(s)} w_i^2 + \sum_{\{i,j\} \in E_e(s)} w_i w_j \right) + b_e \sum_{i \in V_e(s)} w_i \right] \quad .$$

□

4 Potential Functions in Population Dynamics: Generalized Moran Process

In this section we consider the Moran process, as generalized by Lieberman et al. [12]. A population resides on the vertices of a finite, connected, undirected graph and, at each time step, an individual is chosen at random with probability proportional to its assigned "fitness" value. It reproduces, placing a copy of itself on a neighboring vertex chosen uniformly at random, replacing the individual that was there. The initial population consists of a single mutant of fitness $r > 0$ placed uniformly at random, with every other vertex occupied by an individual of fitness 1. The main quantities of interest are the probabilities that the descendants of the initial mutant come to occupy the whole graph (fixation) and that they die out (extinction); almost surely, these are the only possibilities. In general, exact computation of these quantities by standard Markov chain techniques requires solving a system of linear equations of size exponential in the order of the graph so is not feasible. In [4] a potential function is used to show that, with high probability, the number of steps needed to reach fixation or extinction is bounded by a polynomial in the number of vertices in the graph. This bound allows to construct fully polynomial randomized approximation schemes (FPRAS) for the probability of fixation (when $r \geq 1$) and of extinction (for all $r > 0$).

In the following, we consider only finite, connected, undirected graphs $G = (V, E)$ of order $n = |V|$, and r denotes the fitness of the initially introduced mutant in the graph. Given a set $X \subseteq V$, we denote by $W(X) = r|X| + |V \setminus X|$ the *total fitness* of the population when exactly the vertices of X are occupied by mutants.

We first show that the Moran process on a connected graph G of order n is expected to reach absorption in a polynomial number of steps. To do this, we use the potential function given by $\Phi(X) = \sum_{x \in X} \frac{1}{\deg(x)}$ for any state $X \subseteq V$. Note that $1 < \Phi(V) \leq n$ and that, if $(X_i)_{i \geq 0}$ is a Moran process on G then $\Phi(X_0) = \frac{1}{\deg(x)} \leq 1$ for some vertex $x \in V$ (the initial mutant). The following lemma shows that the potential strictly increases in expectation when $r > 1$ and strictly decreases in expectation when $r < 1$.

Lemma 1. *Let $(X_i)_{i \geq 0}$ be a Moran process on a graph $G = (V, E)$ and let $\emptyset \subset S \subset V$. If $r \geq 1$, then $E[\Phi(X_{i+1}) - \Phi(X_i) | X_i = S] > \left(1 - \frac{1}{r}\right) \cdot \frac{1}{n^3}$. Otherwise, $E[\Phi(X_{i+1}) - \Phi(X_i) | X_i = S] < \frac{r-1}{n^3}$.*

Proof. Write $W(S) = n + (r-1)|S|$ for the total fitness of the population. For $\emptyset \subset S \subset V$, and any value of r, we have

$$E[\Phi(X_{i+1}) - \Phi(X_i) | X_i = S] = \frac{r-1}{W(S)} \sum_{xy \in E, x \in S, y \in \overline{S}} \frac{1}{\deg(x) \deg(y)}.$$

The sum is minimized by noting that there must be at least one edge between S and $\overline{S}$ and that its endpoints have degree at most $(n-1) < n$. The greatest

weight configuration is the one with all mutants if $r \geq 1$ and the one with no mutants if $r < 1$. Therefore, if $r \geq 1$, we have $E\left[\Phi(X_{i+1}) - \Phi(X_i) | X_i = S\right] > \frac{r-1}{rn} \cdot \frac{1}{n^2} = \left(1 - \frac{1}{r}\right) \frac{1}{n^3}$ and, if $r < 1$, $E\left[\Phi(X_{i+1}) - \Phi(X_i) | X_i = S\right] < (r-1)\frac{1}{n^3}$. □

Martingale techniques are used to bound the expected absorption time. It is well known how to bound the expected absorption time using a potential function that decreases in expectation until absorption. This has been made explicit by Hajek [10] and [4] uses the following formulation based on that of He and Yao [11].

Theorem 6. *Let $(Y_i)_{i \geq 0}$ be a Markov chain with state space Ω, where Y_0 is chosen from some set $I \subseteq \Omega$. If there are constants $k_1, k_2 > 0$ and a non-negative function $\psi : \Omega \to \mathbb{R}$ such that*

- *$\Psi(S) = 0$ for some $S \in \Omega$,*
- *$\Psi(S) \leq k_1$ for all $S \in I$ and*
- *$E[\Psi(Y_i) - \Psi(Y_{i+1}) | Y_i = S] \geq k_2$ for all $i \geq 0$ and all S with $\Psi(S) > 0$,*

then $E[\tau] \leq k_1/k_2$, where $\tau = \min\{i : \Psi(Y_i) = 0\}$.

The above theorem is useful in bounding the absorption time of the Moran process:

Theorem 7. *Let $G = (V, E)$ be a graph of order n. For $r < 1$, the absorption time τ of the Moran process on G satisfies $E[\tau] \leq \frac{1}{1-r} n^3$.*

Proof. Let $(Y_i)_{i \geq 0}$ be the process on G that behaves identically to the Moran process except that, if the mutants reach fixation, we introduce a new non-mutant on a vertex chosen uniformly at random. That is, from the state V, we move to $V - x$, where x is chosen u.a.r., instead of staying in V. Writing $\tau' = \min\{i : Y_i = \emptyset\}$ for the absorption time of this new process, it is clear that $E[\tau] \leq E[\tau']$. The function Φ meets the criteria for Ψ in the statement of Theorem 6 with $k_1 = 1$ and $k_2 = (1-r)n^{-3}$. The first two conditions of the theorem are obviously satisfied. For $S \subset V$, the third condition is satisfied by Lemma 1 and we have

$$E[\Phi(Y_i) - \Phi(Y_{i+1}) | Y_i = V] = \frac{1}{n} \sum_{x \in V} \frac{1}{\deg(x)} > \frac{1}{n} > k_2 \ .$$

Therefore, $E[\tau] \leq E[\tau'] \leq \frac{1}{1-r} n^3$. □

The following corollary is immediate from Markov's inequality.

Corollary 1. *The Moran process on G with fitness $r < 1$ reaches absorption within t steps with probability at least $1 - \varepsilon$, for any $\varepsilon \in (0, 1)$ and any $t \geq \frac{1}{1-r} n^3/\varepsilon$.*

For $r > 1$, the proof needs slight adjustment because, in this case, Φ increases in expectation, and we obtain:

Corollary 2. *The Moran process on G with fitness $r > 1$ reaches absorption within t steps with probability at least $1 - \varepsilon$, for any $\varepsilon \in (0,1)$ and any $t \geq \frac{r}{r-1} n^3 \Phi(G)/\varepsilon$.*

We refer to [4] for detailed proofs.

5 Bounding the Chromatic Number of Graphs

One of the central optimization problems in Graph Theory and Computer Science is the problem of *vertex coloring* of graphs: Given a graph $G = (V, E)$ with n vertices, assign a color to each vertex of G so that no pair of adjacent vertices gets the same color (i.e., so that the coloring produced is *proper*) and so that the total number of distinct colors used is minimized. The global optimum of vertex coloring is the *chromatic number* $\chi(G)$, defined as the minimum number of colors required to properly color the vertices of graph G.

The problem of coloring a graph using the minimum number of colors is NP-hard, and the chromatic number cannot be approximated to within $\Omega(n^{1-\epsilon})$ for any constant $\epsilon > 0$, unless NP $\subseteq$ co-RP [5]. Despite these negative approximation results, several upper bounds on the chromatic number have been proven in the literature; these bounds are related to various graph-theoretic parameters. In particular, given a graph $G = (V, E)$, let n and m denote the number of vertices and number of edges of G. Let $\Delta(G)$ denote the maximum degree of a vertex in G, and let $\Delta_2(G)$ be the maximum degree that a vertex $v \in V$ can have subject to the condition that v is adjacent to at least one vertex of degree no less than the degree of v (note that $\Delta_2(G) \leq \Delta(G)$). Denote by $\omega(G)$ the *clique number* of G, i.e., the maximum size of a clique in G, and by $\alpha(G)$ the *independence number* of G, i.e., the maximum size of an independent set in G. Then, it is known that

$$\chi(G) \leq \min\left\{\Delta_2(G) + 1,\ \frac{n + \omega(G)}{2},\ n - \alpha(G) + 1,\ \frac{1 + \sqrt{1 + 8m}}{2}\right\} . \quad (4)$$

A different, potential-based method for proving all the above bounds on the chromatic number in a unified manner is given in [17]. The method is *constructive*, in the sense that it actually computes *in polynomial time* a proper coloring using a number of colors that satisfies these bounds. In particular, the vertices of a graph $G = (V, E)$ are viewed as players in a strategic game. Given a finite, simple, undirected graph $G = (V, E)$ with $|V| = n$ vertices, we define the *graph coloring game* $\Gamma(G)$ as the game in strategic form where the set of players is the set of vertices V, and the action set of each vertex is a set of n colors $X = \{x_1, \ldots, x_n\}$. A *configuration* or *pure strategy profile* $\mathbf{c} = (c_v)_{v \in V} \in X^n$ is a vector representing a combination of actions, one for each vertex. That is, c_v is the color chosen by vertex v. For a configuration $\mathbf{c} \in X^n$ and a color $x \in X$, we denote by $n_x(\mathbf{c})$ the number of vertices that are colored x in $\mathbf{c}$,

i.e., $n_x(\mathbf{c}) = |\{v \in V : c_v = x\}|$. The *payoff* that vertex $v \in V$ receives in the configuration $\mathbf{c} \in X^n$ is

$$\lambda_v(\mathbf{c}) = \begin{cases} 0 & \text{if } \exists u \in N(v) : c_u = c_v \\ n_{c_v}(\mathbf{c}) & \text{else} \end{cases} .$$

A configuration $\mathbf{c} \in X^n$ of the graph coloring game $\Gamma(G)$ is a pure Nash equilibrium, or an *equilibrium coloring*, if, for all vertices $v \in V$ and for all colors $x \in X$, $\lambda_v(x, \mathbf{c}_{-v}) \leq \lambda_v(\mathbf{c})$. A vertex $v \in V$ is *unsatisfied* in the configuration $\mathbf{c} \in X^n$ if there exists a color $x \neq c_v$ such that $\lambda_v(x, \mathbf{c}_{-v}) > \lambda_v(\mathbf{c})$; else we say that v is *satisfied*. For an unsatisfied vertex $v \in V$ in the configuration $\mathbf{c}$, we say that v performs a *selfish step* if v unilaterally deviates to some color $x \neq c_v$ such that $\lambda_v(x, \mathbf{c}_{-v}) > \lambda_v(\mathbf{c})$.

For any graph coloring game $\Gamma(G)$, define the function $\Phi : P \to \mathbb{R}$, where $P \subseteq X^n$ is the set of all configurations that correspond to proper colorings of the vertices of G, as $\Phi(\mathbf{c}) = \frac{1}{2} \sum_{x \in X} n_x^2(\mathbf{c})$, for all proper colorings $\mathbf{c}$.

Theorem 8. *Φ is an exact potential function for $\Gamma(G)$.*

Proof. Fix a *proper* coloring $\mathbf{c}$. Assume that vertex $v \in V$ can improve its payoff by deviating and selecting color $x \neq c_v$. This implies that the number of vertices colored c_v in $\mathbf{c}$ is at most the number of vertices colored x in $\mathbf{c}$, i.e., $n_{c_v}(\mathbf{c}) \leq n_x(\mathbf{c})$. If v indeed deviates to x, then the resulting configuration $\mathbf{c}' = (x, \mathbf{c}_{-v})$ is again a proper coloring (vertex v can only decrease its payoff by choosing a color that is already used by one of its neighbors, and v is the only vertex that changes its color). The improvement on v's payoff will be $\lambda_v(\mathbf{c}') - \lambda_v(\mathbf{c}) = n_x(\mathbf{c}') - n_{c_v}(\mathbf{c}) = n_x(\mathbf{c}) + 1 - n_{c_v}(\mathbf{c})$. Moreover, $\Phi(\mathbf{c}') - \Phi(\mathbf{c}) = \frac{1}{2}\left(n_x^2(\mathbf{c}') + n_{c_v}^2(\mathbf{c}') - n_x^2(\mathbf{c}) - n_{c_v}^2(\mathbf{c})\right) = \lambda_v(\mathbf{c}') - \lambda_v(\mathbf{c})$. □

Therefore, if any vertex v performs a selfish step (i.e., changes its color so that its payoff is increased), then the value of Φ is increased as much as the payoff of v is increased. Now, the payoff of v is increased by at least 1. So after any selfish step the value of Φ increases by at least 1. Now observe that, for all proper colorings $\mathbf{c} \in P$ and for all colors $x \in X$, $n_x(\mathbf{c}) \leq \alpha(G)$. Therefore

$$\Phi(\mathbf{c}) = \frac{1}{2} \sum_{x \in X} n_x^2(\mathbf{c}) \leq \frac{1}{2} \sum_{x \in X} (n_x(\mathbf{c}) \cdot \alpha(G)) = \frac{1}{2}\alpha(G) \sum_{x \in X} n_x(\mathbf{c}) = \frac{n \cdot \alpha(G)}{2} .$$

Moreover, the minimum value of Φ is $\frac{1}{2}n$. Therefore, if we allow any unsatisfied vertex (but only one each time) to perform a selfish step, then after at most $\frac{n \cdot \alpha(G) - n}{2}$ steps there will be no vertex that can improve its payoff. This implies that a pure Nash equilibrium will have been reached. Of course, we have to start from an initial configuration that is a proper coloring so as to ensure that the procedure will terminate in $O(n \cdot \alpha(G))$ selfish steps; this can be found easily since there is always the trivial proper coloring that assigns a different color to each vertex of G.

In [17] it was also shown that *any* equilibrium coloring of $\Gamma(G)$ uses a number of colors that satisfies all the bounds given in 4. This, combined to the existence of the potential function, implies that, for any graph G, a proper coloring that uses at most $k \leq \min\left\{\Delta_2(G)+1,\ \frac{n+\omega(G)}{2},\ \frac{1+\sqrt{1+8m}}{2},\ n-\alpha(G)+1\right\}$ colors can be computed in polynomial time.

References

1. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: Graphical Congestion Games. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 70–81. Springer, Heidelberg (2008)
2. Chen, X., Xiaotie, D.: Settling the complexity of two-player Nash equilibrium. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 261–272 (2006)
3. Chien, S., Sinclair, A.: Convergence to approximate Nash equilibria in congestion games. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 169–178 (2007)
4. Díaz, J., Goldberg, L.A., Mertzios, G.B., Richerby, D., Serna, M.J., Spirakis, P.G.: Approximating fixation probabilities in the generalized Moran process. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), pp. 954–960 (2012)
5. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. J. Comput. System Sci. 57(2), 187–199 (1998)
6. Fotakis, D., Gkatzelis, V., Kaporis, A.C., Spirakis, P.G.: The Impact of Social Ignorance on Weighted Congestion Games. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 316–327. Springer, Heidelberg (2009)
7. Fotakis, D., Kaporis, A.C., Spirakis, P.G.: Atomic congestion games: fast, myopic and concurrent. Theory Comput. Syst. 47(1), 38–59 (2010)
8. Fotakis, D., Kontogiannis, S., Spirakis, P.G.: Atomic congestion games among coalitions. ACM Transactions on Algorithms (TALG) 4(4), article No. 52 (2008)
9. Fotakis, D., Kontogiannis, S., Spirakis, P.G.: Selfish unsplittable flows. Theoretical Computer Science 348(2), 226–239 (2005)
10. Hajek, B.: Hitting-time and occupation-time bounds implied by drift analysis with applications. Advances in Applied Probability 14(3), 502–525 (1982)
11. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artificial Intelligence 127, 57–85 (2001)
12. Lieberman, E., Hauert, C., Nowak, M.A.: Evolutionary dynamics on graphs. Nature 433, 312–316 (2005)
13. Monderer, D., Shapley, L.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
14. Nash, J.F.: Non-cooperative games. Annals of Mathematics 54(2), 286–295 (1951)
15. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. Journal of Computer and System Sciences 48(3), 498–532 (1994)
16. Panagopoulou, P.N., Spirakis, P.G.: Algorithms for pure Nash equilibria in weighted congestion games. ACM Journal of Experimental Algorithmics 11 (2006)
17. Panagopoulou, P.N., Spirakis, P.G.: A game theoretic approach for efficient graph coloring. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 183–195. Springer, Heidelberg (2008)

18. Panagopoulou, P.N., Spirakis, P.G.: Playing a game to bound the chromatic number. The American Mathematical Monthly 119(9), 771–778 (2012)
19. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
20. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journals of Mathematics 5, 285–308 (1955)
21. Topkis, D.: Equilibrium points in nonzero-sum n-person submodular games. SIAM Journal of Control and Optimization 17, 773–787 (1979)
22. Uno, H.: Strategic complementarities and nested potential games. Journal of Mathematical Economics 47(6), 728–732 (2011)

The Probabilistic Min Dominating Set Problem*

Nicolas Boria[1], Cécile Murat[1], and Vangelis Th. Paschos[1,2]

[1] Paris Sciences et Lettres Research University, Université Paris-Dauphine, LAMSADE CNRS, UMR 7243
[2] Institut Universitaire de France

Abstract. We present a natural wireless sensor network problem, which we model as a probabilistic version of the MIN DOMINATING SET problem. We show that this problem, being a generalization of the classical MIN DOMINATING SET, is **NP**-hard, even in bipartite graphs. We first study the complexity of PROBABILISTIC MIN DOMINATING SET in graphs where MIN DOMINATING SET is polynomial, mainly in trees and paths and then we give some approximation results for it.

1 Wireless Sensor Networks and Probabilistic Dominating Set

Very frequently, in wireless sensor networks [1], one wishes to identify a subset of sensors, called "master" sensors, that will have a particular role in messages transmission, namely, to centralize and process messages sent by the rest of the sensors, called "slave" sensors, in the network. These latter sensors will be only nodes of intermediate messages transmission, while the former ones will be authorized to make several operations on messages received and will be, for this reason, better or fully equipped and preprogrammed.

Hence, in order to design such a network, one must identify a subset of sensors (the master sensors) such that, every other sensor is linked to some sensor of this set. In other words, one wishes to find a dominating set in the graph of sensors. If we suppose that equipment of master sensors induces some additional cost with respect to that of the slave ones, if this cost is the same for all master sensors, we have a minimum cardinality dominating set problem (MIN DOMINATING SET), while if any master sensor has its own cost, we have a minimum weight dominating set problem.

In this paper, we consider a more general setting where any sensor is allowed to break down with a given probability, and one must be able to recompute a new set of master sensors very quickly (and in any case as quickly as solution from scratch is not allowed). The framework of *probabilistic combinatorial optimization* that we adopt in this paper was introduced by [2, 3]. In [2–10], restricted versions of routing and network-design probabilistic minimization problems (in complete graphs) have been studied under the robustness model dealt here (called *a priori optimization*). In [11–14], the analysis

* Research supported by the French Agency for Research under the program TODO, ANR-09-EMER-010.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 298–309, 2013.

of the probabilistic minimum travelling salesman problem, originally performed in [2, 3], has been revisited. Several other combinatorial problems have been also handled in the probabilistic combinatorial optimization framework, including minimum coloring ([15, 16]), maximum independent set and minimum vertex cover ([17, 18]), longest path ([19]), Steiner tree problems ([20, 21]), minimum spanning tree [6, 22].

For simplicity, we deal with master sensors of uniform equipment cost (hopefully, it will be clear later that this assumption is not restrictive for the model) and we suppose that any sensor is actually present in the network, with probability p_i (so that its probability to be broken down is $1 - p_i$) depending on its construction, proper equipment, age, etc. Informally, the approach we propose, in order to maintain the network operational at any time, is the following:

- design an algorithm M that, given a set D of master sensors of the network, if some sensors of D fail, it adapts D to the surviving network (in other words, the new D becomes the new master set of sensors for the surviving part of the network); this algorithm must be as efficient as possible, so that long idle periods for the network are avoided;
- given the network, its sensors' surviving probabilities p_i and M, compute a solution D^*, called *"a priori" solution* that, informally, will "resist" in the best possible way to node failures.

It is clear that given a network of sensors, identifying a master set of them is equivalent to determining a dominating set in the associated graph where vertices are the sensors of the network and, for any linked pair of them, an edge links the corresponding vertices. The MIN DOMINATING SET problem is formally defined as follows. Let $G(V, E)$ be a connected undirected graph defined on a set V of vertices with a set $E \subseteq V \times V$ of edges. A vertex-set D is said to be a dominating set of G if, for any $v \in V \setminus D$, v has at least one neighbor in D. In the MIN DOMINATING SET problem, the objective is to determine a minimum-size dominating set in G. The decision version of MIN DOMINATING SET problem is one of the first 21 **NP**-complete problems [23] and remains **NP**-complete even in bipartite graphs, while it is polynomial in trees.

Here, we associate a probability p_i to every vertex $v_i \in V$ (the probability that sensor i remains operational). We specify also a strategy M, called *modification strategy*, that when given a dominating set D of G and a subgraph $G' = G[V']$ induced by a set $V' \subseteq V$ (the surviving sensors), it transforms D into a set D' that is a dominating set of G'. Let us note that the simplest modification strategy, consisting of just returning $D' = D \cap V'$ is not feasible since it does not always produces feasible dominating sets for G'. For example, consider the graph of Figure 1 for which the set $D = \{v_3, v_6\}$ is dominating. If we consider $V' = \{v_1, v_2, v_4, v_6, v_7\}$, then $D' = D \cap V' = \{v_6\}$ is no longer a dominating set of G'. Hence, we will consider the following somewhat more complicated modification strategy M associated with our problem: *given a graph $G(V, E)$, a dominating set D of G and a subgraph G', set $D' := \emptyset$; for any $v_i \in V'$: if $v_i \in D$, set $D' := D' \cup \{v_i\}$; otherwise, if $\Gamma(v_i) \cap (D \cap V') = \emptyset$, set $D' := D' \cup \{v_i\}$, where,*

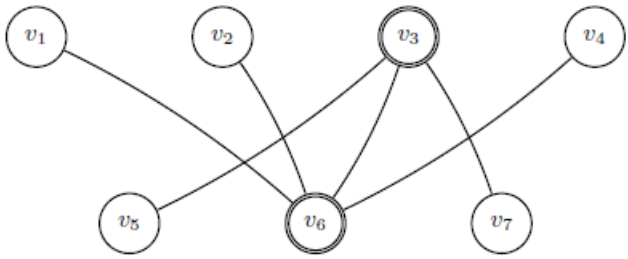


Fig. 1. A graph together with a dominating set (bold-circled vertices)

for any vertex v, $\Gamma(v)$ denotes the set of its neigbors. In other words, M first takes $D \cap V'$ in D' and then completes it with all the non-dominated vertices of V'. It is easy to see that the complexity of M is polynomial, since it is bounded by the sum of the degrees of the vertices of $V' \setminus D$ that is at most $O(|E|)$.

Consider now a graph $G(V, E)$, a probability p_i for every vertex $v_i \in V$, a dominating set D of G and the modification strategy M just specified. The *functional* of M can be expressed by:

$$\mathbb{E}(G, D, \mathtt{M}) = \sum_{V' \subseteq V} \Pr[V'] \, |D'| \tag{1}$$

where $\Pr[V']$ is the probability of V' (i.e., the probability that only the vertices of V' will not be broken down) and D' is the dominating set returned by M for $G[V']$. Following M, if a vertex $v_i \in D$, then it will be always in D' if it belongs to V'; hence, its contribution to $\mathbb{E}(G, D, \mathtt{M})$ in (1) will be equal to p_i. On the other hand, if $v_i \notin D$ but $v_i \in V'$, it will be included in D' only if all its neigbors in G that belonged to D are not in V'; in this case, its contribution to $\mathbb{E}(G, D, \mathtt{M})$ in (1) will be equal to $p_i \prod_{v_j \in \Gamma(v_i) \cap D}(1 - p_j)$. Based upon these remarks, starting from (1) we get:

$$\begin{aligned}
\mathbb{E}(G, D, \mathtt{M}) &= \sum_{V' \subseteq V} \Pr[V'] \sum_{v_i \in V} 1_{v_i \in D'} \;=\; \sum_{v_i \in V} \sum_{V' \subseteq V} \Pr[V'] \, 1_{v_i \in D'} \\
&= \sum_{v_i \in D} \sum_{V' \subseteq V} \Pr[V'] \, 1_{v_i \in V'} \\
&\quad + \sum_{v_i \notin D} \sum_{V' \subseteq V} \Pr[V'] \, 1_{(v_i \in V') \cap (\Gamma(v_i) \cap (D \cap V') = \emptyset)} \\
&= \sum_{v_i \in D} p_i + \sum_{v_i \notin D} \sum_{V' \subseteq V} \Pr[V'] \, 1_{(v_i \cap V' = v_i)} 1_{\Gamma(v_i) \cap (D \cap V') = \emptyset} \\
&= \sum_{v_i \in D} p_i + \sum_{v_i \notin D} p_i \prod_{v_j \in \Gamma(v_i) \cap D} (1 - p_j)
\end{aligned} \tag{2}$$

It is easy to see that, from (2), $\mathbb{E}(G, D, \mathtt{M})$ can be computed in polynomial time. Also, setting $p_i = p$, i.e., considering that vertices of G have identical probabilities (this is quite natural if we assume identical sensors), one gets:

$$\mathbb{E}(G, D, \mathtt{M}) = p|D| + \sum_{v_i \notin D} p(1-p)^{|\Gamma(v_i) \cap D|} \tag{3}$$

We consider $\mathbb{E}(G, D, \mathtt{M})$ *as the objective function of the problem handled here which, by symmetry, we call* PROBABILISTIC MIN DOMINATING SET. *Its goal is to determine a dominating set* D^* *of* G*, called a priori dominating set, that minimizes* $\mathbb{E}$*.*

In other words, for our wireless sensor network design problem, the a priori solution D^* has the property that, under the modification strategy M described above, the solution constructed on the surviving network minimizes in average the additional cost needed so that this network remains operational.

Proposition 1. PROBABILISTIC MIN DOMINATING SET *is* ***NP****-hard, even in bipartite graphs and inapproximable in polynomial time within ratio* $O(\log n)$*.*

As we have already mentioned, MIN DOMINATING SET is polynomial in trees. In Section 2, we explore complexity of PROBABILISTIC MIN DOMINATING SET in those graphs, while in Section 3, we give approximation results in general graphs. The main results in Section 2 imply that PROBABILISTIC MIN DOMINATING SET is polynomial in trees with degrees bounded by $O(\log n)$ and in general trees assuming identical probabilities. Remains however open if it is polynomial in general trees with distinct vertex-probabilities, which seems to be a difficult problem. Results of Section 3 are quite pessimistic since, for instance, although the classical MIN DOMINATING SET problem is approximable within ratio $O(\log n)$, PROBABILISTIC MIN DOMINATING SET is approximable within ratio $\Delta - \ln \Delta$, in the case of identical sensors, and within ratio $\Delta^2 / \ln \Delta$ when heterogeneous sensors are assumed, where Δ denotes the maximum degree of the input graph. But this is due to the fact that PROBABILISTIC MIN DOMINATING SET is much harder than its deterministic counterpart.

For reasons of paper's size, some of the results presented in what follows are given without detailed proofs, or without proofs at all. All these proofs can be found in [24].

2 Probabilistic Dominating Set on Paths, Cycles and Trees

We handle in this section PROBABILISTIC DOMINATING SET on paths, cycles and trees. Let us recall that MIN DOMINATING SET in these graphs is polynomial. We prove that PROBABILISTIC DOMINATING SET on paths and cycles remains polynomial for any vertex-probability, while PROBABILISTIC MIN DOMINATING SET in trees is polynomial either when the maximum degree of the input tree is bounded, or when the vertex-probabilities are all equal.

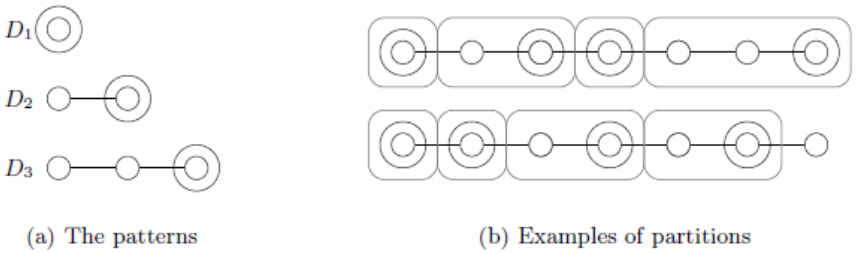
Recall that the *contribution* of a node refers to the probability for this node to be present in D' for a given a priori solution D and for the modification strategy M adopted in Section 1. Let us recall that the contribution of a node v_i

that belongs to D is p_i, and for a node that does not belong to D, its contribution amounts to $p_i \prod_{v_j \in \Gamma(v_i) \cap D}(1 - p_j)$. This notion will be extended to a set: the *contribution* $C(V')$ of a node-set V' is the expected number of nodes of V' in D'. In other words, $C(V')$ is the sum of the contributions of all the nodes in V'.

We first handle paths and cycles. Given a path we consider its nodes labeled in the following way: the leftmost endpoint of the path is labelled by v_1 (which might be referred as the first, or left end node), while the rightmost endpoint will be labelled by v_n (last or right end node). Of course, all nodes in between will be labelled in increasing order from left to right.

Proposition 2. PROBABILISTIC DOMINATING SET *on paths can be solved in polynomial time.*

Proof. We show how PROBABILISTIC DOMINATING SET on paths can be solved by dynamic programming. First, let us make some preliminary remarks that will help us to build the final algorithm. First, for a given node v_i in a dominating set, the "next" dominating node (if any) will be v_{i+1}, v_{i+2} or v_{i+3}. Indeed, if none of them is in the dominating set, then v_{i+2} wouldn't be dominated. Then, The last dominating node is either v_n or v_{n-1}. Considering these two facts, one can see that, except the very last node of the path, any dominating set D on a path can be partitioned in elementary patterns of shapes D_1, D_2 or D_3, which are illustrated in Figure 2.



(a) The patterns (b) Examples of partitions

Fig. 2. Partitionning nodes with elementary patterns

At this point, it is interesting to notice that the contribution of a given pattern can be determined, regardless the rest of the sequence, but depending on its position in the path. To be more precise, denoting by $D_j(i)$ the pattern D_j with its dominating node being v_i:

$$C(D_1(i)) = p_i \qquad \forall i$$

$$C(D_2(i)) = p_i + p_{i-1}(1 - p_i) \qquad i = 2 \tag{4a}$$

$$C(D_2(i)) = p_i + p_{i-1}(1 - p_i)(1 - p_{i-2}) \qquad i \geqslant 3 \tag{4b}$$

$$C(D_3(i)) = p_i + p_{i-1}(1 - p_i) + p_{i-2}(1 - p_{i-3}) \qquad i \geqslant 4 \tag{4c}$$

One notices that the contribution of each pattern is defined regardless of the rest of the pattern sequence. This will be very useful for the method. Indeed, if a pattern has no impact in terms of contribution neither on the preceding sequence, nor on the following one, then it is quite easy to give a recursive definition of W_i which is the sequence up to node v_i of minimum contribution, where v_i is a dominating node:

$$C(W_i) = \begin{cases} 0 & i = 0 \\ p_1 & i = 1 \\ \min\limits_{j=1,2} \left(C(W_{i-j}) + C(D_j(i))\right) & i = 2, 3 \\ \min\limits_{j=1,2,3} \left(C(W_{i-j}) + C(D_j(i))\right) & 4 \leqslant i \leqslant n \end{cases} \tag{5}$$

Indeed, a sequence W_i always ends with a dominating node. So, leaving aside particular cases when $i \leqslant 3$, it ends necessarily with one of the three patterns defined earlier. If it ends with D_1 (resp., D_2 or D_3) then it should be completed with a minimal sequence up to node v_{i-1} (resp., v_{i-2} or v_{i-3}), namely W_{i-1} (resp., W_{i-2} or W_{i-3}). That among these three possibilities that returns the minimal contribution, is that chosen for W_i.

At this point, we only have a set of optimal sequences, but these are not necessarily optimal dominating sets. Indeed, a dominating set can end with a dominated (i.e., non-dominating) node, whereas a sequence cannot. To take this distinction into account, and allow our final solution to end with a dominated node, we define the contribution of an optimal dominating set D^* as follows:

$$C(D^*) = \min\left(C(W_n), C(W_{n-1}) + p_n(1 - p_{n-1})\right)$$

In all, $3n$ contributions $C(D_j(i))$ have to be computed and, in order to compute a W_i, one basically compares 3 possible values, which amounts to $6n$ operations to get an optimal anticipatory solution for PROBABILISTIC DOMINATING SET on paths. This concludes the proof. □

By adapting the previous method, one can extend the result of Proposition 2 to cycles.

Proposition 3. PROBABILISTIC DOMINATING SET *on cycles can be solved in polynomial time.*

With a similar, yet more complex method, one can generalize the result of Proposition 2 from paths (trees of maximum degree bounded by 2) to trees with bounded maximum degrees. In what follows, v_1 will denote the root of the input tree T, T_i will denote the subtree rooted at node v_i (so, $T_1 = T$), F_i will denote the set of children of given node v_i, and v_{f_i} the father of v_i. The depth $D(T_i)$ of a subtree T_i will refer to the minimum number of edges on a path between the root and a leaf of the subtree.

Proposition 4. PROBABILISTIC MIN DOMINATING SET *in trees of maximum degree bounded by k can be solved in $O^*(2^k)$, where $O^*(\cdot)$ notation ignores the polynomial terms.*

Proof. Consider a tree T and a dominating set D on T. For any subtree T_i of T, there are only three possible configurations regarding its root v_i:

1. $v_i \in D$;
2. $v_i \notin D$, and $v_{f_i} \notin D$; in this case, the root has to be dominated in the subtree T_i (i.e., dominated by at least one of its children);
3. $v_i \notin D$ and $v_{fi} \in D$; in this case, the root might be non-dominated in the subtree T_i (i.e., not dominated by any of its children).

Considering Cases 1 to 3, for a given dominating set D in a tree T, it is always possible to partition nodes of T into three sets: D, S and N, where D is the dominating set itself, S is the set of nodes that are dominated, but not by their fathers, and N is the set of nodes dominated by their fathers (and also possibly by some of their children). Figure 2 gives an example of such a partition.

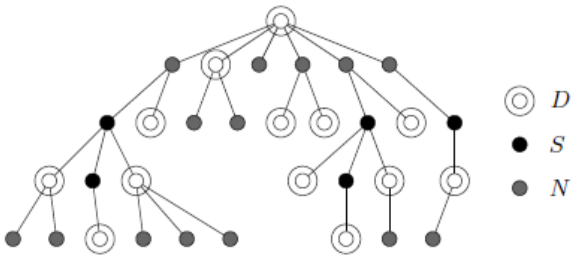


Fig. 3. Partitioning nodes in three subsets

Now, let us analyze what possible cases can occur for nodes of F_i with respect to the status of v_i. The following cases can occur:

- $v_i \in D$; then children of v_i (if any) might be in D, or in N;
- $v_i \in S$; then children of v_i might be in D or in S, and at least one of them should be in D;
- $v_i \in N$; in this case, children of v_i might be in D or in S.

It is interesting to notice that the situation of a given node impacts only its own contribution if the status of its children is known, so that, once more, it is possible to run a dynamic programming method. It will be based upon three partial solutions for each subtree T_i, namely, W_i, W_i' and W_i'', where W_i (resp., W_i', W_i'') is the partial solution of minimum contribution for subtree T_i rooted at v_i when $v_i \in D$ (resp., $v_i \in S$, $v_i \in N$).

Partial solution W_i can be computed on a tree of any depth. Children of v_i can belong either to D, or to N, so that there are $2^{|F_i|}$ possible combinations to evaluate (bounded by 2^k). The combination minimizing the overall contribution leads to the structure we are trying to define, namely W_i. Setting $D_i = D \cap F_i$ (the subset of children of v_i that are dominating nodes), W_i is defined by:

$$C(W_i) = \min_{D_i \subseteq F_i} \left(p_i + \sum_{v_j \in D_i} C(W_j) + \sum_{v_j \in F_i \setminus D_i} C\left(W_j''\right) \right) \tag{6}$$

This value is quite easy to initialize with leaves where $W_i = p_i$.

Let us now consider W_i'', which can also be initialized on leaves (unlike W_i'). Indeed, if a leaf is dominated, then its father has to be dominating, or else it would not be dominated. So that a leaf can only be in D or in N. As we said earlier, children of v_i in a partial solution W_i'' can belong either to D or to S. Thus, the following holds:

$$C(W_i'') = \min_{D_i \subseteq F_i} \left(p_i (1 - p_{f_i}) \prod_{v_j \in D_i} (1 - p_j) + \sum_{v_j \in D_i} C(W_j) + \sum_{v_j \in F_i \setminus D_i} C\left(W_j'\right) \right) \tag{7}$$

Note there are also at most 2^k combinations to examine in this case.

Finally, let us specify W_i'. In this case, children of v_i should be in S or in D and at least one of them should be in D. Of course, by definition, none of them can be in N. Once more, each combination of sons in S or in D (at most 2^k combinations) leads to a specific contribution for the subtree T_i. The partial solution W_i' is the one minimizing this contribution. Thus, the following holds:

$$C(W_i') = \min_{D_i \subseteq F_i, |D_i| \geqslant 1} \left(p_i \prod_{v_j \in D_i} (1 - p_j) + \sum_{v_j \in D_i} C(W_j) + \sum_{v_j \in F_i \setminus D_i} C\left(W_j'\right) \right) \tag{8}$$

Note that such a value can be computed for any subtree T_i of depth at least 1 (not on leaves). This might be a problem when computing a value W_i' or W_i'' on a tree T_i of depth 1 since, according to (7) and (8), in order to compute W_i' and W_i'', one needs values W_j' for all the children v_j of v_i but these values do not exist for leaves. To keep all formulæ valid and still ensure that a leaf will never be in S, we will initialize W_j' to an arbitrarily large value M for any v_j that is a leaf. Thus, when applying (7) and (8), all leaf-children of v_i will be forced to be in D_i.

Note also, that definitions of $C(W_i')$ and $C(W_i'')$ differ only by a factor $(1-p_{f_i})$ in the contribution of v_i.

The dynamic programming method runs in a "bottom up" way. It is initialized with leaves where values W_i, W_i' and W_i'' are equal to p_i , M and $p_i(1-p_{f_i})$, respectively. Then, each subtree T_i is associated with structures W_i, W_i' and W_i'' (apart from leaves that are associated only with W_i and W_i'', and the overall tree which will be associated only with W_i and W_i' for obvious reasons), whose computation relies on the same structures on subtrees induced by children of the root v_i. In other words, in order to compute, for instance, a value W_i, one

needs to have already computed values W_j and W''_j for all children v_j of v_i. Since the method is easily initialized with leaves, all values can be computed for all subtrees, starting from leaves and ending with the whole tree.

Finally, the optimum D^* is given by $D^* = \text{argmin}_{W=W_i,W'_i}(C(W))$. In all, $O(2^k n) = O^*(2^k)$ operations are necessary to compute D^*. □

Corollary 1. *ie* PROBABILISTIC MIN DOMINATING SET *in trees with maximum degree bounded by* $O(\log n)$ *can be solved in polynomial time.*

Based on the idea that the complexity of the dynamic programmig algorithm is actually exponential in the number of distinct probabilities among the neighbors of a given vertex, we also show the following result, whose proof has been excluded from the paper due to lenght constraints.

Proposition 5. PROBABILISTIC MIN DOMINATING SET *is polynomial in general trees with equiprobable nodes.*

3 Polynomial Approximation of Probabilistic Min Dominating Set

3.1 Networks with Identical Sensors

We consider in this section that the probability that a sensor fails is the same for all of them.

As we have already mentioned, MIN DOMINATING SET is approximate equivalent to MIN SET COVER, in the sense that an approximation algorithm for one of them can be transformed in polynomial time into an approximation algorithm for the other one achieving the same approximation ratio. Recall also that the natural greedy algorithm for MIN SET COVER achieves approximation ratio either $1 + \ln |S_{\max}|$ where $|S_{\max}|$ is the cardinality of the largest set $S_{\max}$ in the set-system describing the instance of MIN SET COVER [25, 26], or $O(\log n)$ where n is the cardinality of the ground set describing this instance [27]. In the transformation of MIN DOMINATING SET into MIN SET COVER, any set S of the set-system becomes a vertex with degree $|S| + 1$. So, in the derived instance of MIN DOMINATING SET, $\Delta = |S_{\max}| + 1$, where Δ denotes the maximum degree of the derived graph. For facility and because of the form of the functional given in (3), we will use in what follows the former of the above ratios.

The following two easy lemmata that will be used later hold. The proof of the second one (Lemma 2) is immediate.

Lemma 1. *For any instance of* MIN DOMINATING SET *of size* n *and maximum degree* Δ*, any minimal (for inclusion) solution (hence, the minimum-size one also) has size bounded below by* $n/(\Delta + 1)$.

Lemma 2. *Let* D *be a minimal dominating set in a connected graph* $G(V, E)$*. Then,* $V \setminus D$ *is also a dominating set and, moreover, the smallest of them is smaller than* $n/2$.

Proposition 6. *The set D selected to be the smallest between the $(1+\ln\Delta)$-approximate solution computed by the greedy algorithm in G and the complement of it with respect to V achieves for* PROBABILISTIC MIN DOMINATING SET *approximation ratio bounded above by $\Delta - \ln\Delta$, where Δ is the maximum degree of G, when sensors have identical failure probabilities.*

Proof (Sketch). Revisit (3) and observe that the following expressions can be easily derived for $\mathbb{E}(G,D,\mathtt{M})$: $\mathbb{E}(G,D,\mathtt{M}) \leqslant p^2|D| + pn(1-p)$ and $\mathbb{E}(G,D,\mathtt{M}) \geqslant p(1-(1-p)^{\Delta})|D| + pn(1-p)^{\Delta}$.

Denote by D^* and $\hat{D}$ an optimal solution for PROBABILISTIC MIN DOMINATING SET and for MIN DOMINATING SET in G, respectively. Remark that since D^* is a feasible solution for MIN DOMINATING SET, we have $|D^*| \geqslant |\hat{D}|$.

Remark that the size of D (as it is selected) guarantees the ratio $1+\ln\Delta$ and, simultaneously, according to Lemma 2, it is smaller than $n/2$. Then, it holds that $\mathbb{E}(G,D,\mathtt{M}) \leqslant p^2(1+\ln\Delta)|\hat{D}| + p(1-p)n$ and $\mathbb{E}(G,D^*,\mathtt{M}) \geqslant p(1-(1-p)^{\Delta})|D^*| + pn(1-p)^{\Delta}$.

Putting all the above together, we get:

$$\frac{\mathbb{E}(G,D,\mathtt{M})}{\mathbb{E}\,(G,D^*,\mathtt{M})} \leqslant \min\left(\Delta + 1 - p(\Delta - \ln\Delta), \frac{\Delta+1}{1+\Delta(1-p)^{\Delta}}\right) \leqslant \Delta - \ln\Delta$$

as claimed. □

3.2 Networks with Heterogeneous Sensors

Let us now suppose that the sensors of the network are heterogeneous so that each of them has its own failure probability that can be different from that of another sensor in the network. We will show that, in this case, any PROBABILISTIC MIN DOMINATING SET-solution achieves approximation ratio bounded above by $O(\Delta^2/\log\Delta)$.

Proposition 7. *Any* PROBABILISTIC MIN DOMINATING SET*-solution achieves approximation ratio bounded above by $O(\Delta^2/\log\Delta)$, in networks with distinct sensor failure probabilities.*

Proof (Sketch). Revisit (2) and observe that any dominating set D satisfies the trivial inequality $\mathbb{E}(G,D,\mathtt{M}) \leqslant \sum_{v_i \in V} p_i$.

Fix an optimal a priori dominating set D^* and, for a probability p' that will be fixed later, partition the vertices of G into four subsets:

- D_1^*: the set of vertices of D^* whose probabilities are at least p'; furthermore, set $|D_1^*| = \kappa$;
- D_2^*: the rest of vertices of D^*, i.e., $D_2^* = D^* \setminus D_1^*$;
- $\bar{D}_1^*$: the set $\Gamma_{\bar{D}^*}(D_1^*)$ of neighbours of D_1^* in $\bar{D}^* = V \setminus D^*$, i.e., $\bar{D}_1^* = \Gamma(D_1^*) \cap \bar{D}^*$;
- $\bar{D}_2^*$: the set $\Gamma_{\bar{D}^*}(D_2^*) \setminus \bar{D}_1^*$, i.e., the vertices of $\bar{D}^*$ that have no neighbours in D_1^*, i.e., $\bar{D}_2^* = \bar{D}^* \setminus \Gamma_{\bar{D}^*}(D_1^*)$.

Denote, for simplicity, by p the largest vertex-probability. With respect to the partition above, using some algebraic and combinatorial arguments, one can prove that:

$$\frac{\mathbb{E}(G, D, \mathtt{M})}{\mathbb{E}\,(G, D^*, \mathtt{M})} \leqslant \min\left(\frac{x}{(1-p')^{\Delta}}, \frac{x(\Delta+1)}{(x-1)p'}\right)$$

for some properly chosen x. If one fixes $p' = \ln\Delta/\Delta$ and $x = \Delta/\ln\Delta$, then both ratios become $\approx \Delta^2/\ln\Delta$. □

4 Conclusion

In this paper we have studied a generalization of a wireless sensor network problem modelled as a probabilistic version of MIN DOMINATING SET. This has led to a problem that is quite more difficult to handle than its original deterministic version, in particular when trying to approximately solve it. We have proposed algorithms for paths, cycles and trees (cases where MIN DOMINATING SET is polynomial), as well as we have tried a first study of approximability of PROBABILISTIC MIN DOMINATING SET in general graphs.

References

1. Sandos, A.C., Bendali, F., Mailfert, J., Duhamel, C., Hou, K.M.: Heuristics for designing energy-efficient wireless sensor network topologies. J. Networks 4, 436–444 (2009)
2. Jaillet, P.: Probabilistic traveling salesman problem. Technical Report 185, Operations Research Center. MIT, Cambridge Mass., USA (1985)
3. Bertsimas, D.J.: Probabilistic combinatorial optimization problems. Phd thesis, Operations Research Center. MIT, Cambridge Mass., USA (1988)
4. Averbakh, I., Berman, O., Simchi-Levi, D.: Probabilistic a priori routing-location problems. Naval Res. Logistics 41, 973–989 (1994)
5. Bertsimas, D.J.: On probabilistic traveling salesman facility location problems. Transportation Sci. 3, 184–191 (1989)
6. Bertsimas, D.J.: The probabilistic minimum spanning tree problem. Networks 20, 245–275 (1990)
7. Bertsimas, D.J., Jaillet, P., Odoni, A.: A priori optimization. Oper. Res. 38, 1019–1033 (1990)
8. Jaillet, P.: A priori solution of a traveling salesman problem in which a random subset of the customers are visited. Oper. Res. 36, 929–936 (1988)
9. Jaillet, P.: Shortest path problems with node failures. Networks 22, 589–605 (1992)
10. Jaillet, P., Odoni, A.: The probabilistic vehicle routing problem. In: Golden, B.L., Assad, A.A. (eds.) Vehicle Routing: Methods and Studies. North-Holland, Amsterdam (1988)
11. Balaprakash, P., Birattari, M., Stützle, T., Dorigo, M.: Estimation-based metaheuristics for the probabilistic traveling salesman problem. Computers and Operations Research 37, 1939–1951 (2010)
12. Bianchi, L., Knowles, J., Bowler, N.: Local search for the probabilistic traveling salesman problem: correlation to the 2-p-opt and 1-shift algorithms. European J. Oper. Res. 161, 206–219 (2005)

13. Birattari, M., Balaprakash, P., Stützle, T., Dorigo, M.: Estimation-based local search for stochastic combinatorial optimization using delta evaluations: a case study on the probabilistic traveling salesman problem. INFORMS J. Computing 20, 644–658 (2008)
14. Campbell, A.M., Thomas, B.W.: Probabilistic traveling salesman problem with deadlines. Transportation Sci. 42, 1–21 (2008)
15. Murat, C., Paschos, V.T.: On the probabilistic minimum coloring and minimum k-coloring. Discrete Appl. Math. 154, 564–586 (2006)
16. Bourgeois, N., Della Croce, F., Escoffier, B., Murat, C., Paschos, V.T.: Probabilistic coloring of bipartite and split graphs. J. Comb. Optimization 17, 274–311 (2009)
17. Murat, C., Paschos, V.T.: A priori optimization for the probabilistic maximum independent set problem. Theoret. Comput. Sci. 270, 561–590 (2002)
18. Murat, C., Paschos, V.T.: The probabilistic minimum vertex-covering problem. Int. Trans. Opl. Res. 9, 19–32 (2002)
19. Murat, C., Paschos, V.T.: The probabilistic longest path problem. Networks 33, 207–219 (1999)
20. Paschos, V.T., Telelis, O.A., Zissimopoulos, V.: Steiner forests on stochastic metric graphs. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA. LNCS, vol. 4616, pp. 112–123. Springer, Heidelberg (2007)
21. Paschos, V.T., Telelis, O.A., Zissimopoulos, V.: Probabilistic models for the STEINER TREE problem. Networks 56, 39–49 (2010)
22. Boria, N., Murat, C., Paschos, V.T.: On the PROBABILISTIC MIN SPANNING TREE problem. J. Mathematical Modelling and Algorithms (to appear)
23. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of computer computations, pp. 85–103. Plenum Press, New York (1972)
24. Boria, N., Murat, C., Paschos, V.T.: An emergency management model for a wireless sensor network problem. In: Cahier du LAMSADE 325, LAMSADE, Université Paris-Dauphine (2012)
25. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. 9, 256–278 (1974)
26. Lovász, L.: On the ratio of optimal integral and fractional covers. Discrete Math. 13, 383–390 (1975)
27. Slavík, P.: A tight analysis of the greedy algorithm for set cover. In: Proc. STOC 1996, pp. 435–441 (1996)

Dichotomy of the H-Quasi-Cover Problem*

Jiří Fiala** and Marek Tesař***

Department of Applied Mathematics and DIMATIA, Charles University,
Malostranské nám. 25, 118 00 Prague, Czech Republic
{fiala,tesar}@kam.mff.cuni.cz

Abstract. We show that the problem whether a given simple graph G admits a quasi-covering to a fixed connected graph H is solvable in polynomial time if H has at most two vertices and that it is NP-complete otherwise.

As a byproduct we show constructions of regular quasi-covers and of multi-quasi-covers that might be of independent interest.

Keywords: Computational complexity, dichotomy, graph cover.

1 Introduction

A *homomorphism* between two graphs G and H is an edge-preserving mapping $f : V(G) \to V(H)$. We focus on homomorphisms f that satisfy local constraints. For instance it might be required for each vertex u of G that all neighbors of its image $f(u)$, are used when the mapping f is restricted on the neighborhood of u, formally $|f^{-1}(v) \cap N_G(u)| \geq 1$ for each $v \in N_H(f(u))$. In other words f should act surjectively between $N_G(u)$ and $N_H(f(u))$ for each $u \in V(G)$. In such a situation we say that f is a *locally surjective* homomorphism.

We focus in a particular case of locally surjective homomorphisms, called *quasi-coverings*. These satisfy that for every vertex u of G there exists a positive integer c such that $|f^{-1}(v) \cap N_G(u)| = c$ for every $v \in N_H(f(u))$ — in such a case we say that $f|_{N_G(u)}$ is c-fold between $N_G(u)$ and $N_H(f(u))$. Note that the constant c may vary for different vertices of G. If such a quasi-covering projection from G to H exists, we say that G quasi-covers H or that G is a quasi-cover of H.

Locally surjective homomorphisms and quasi-covers are closely related to homomorphisms that are *locally injective (bijective, resp.)*, i.e. those edge-preserving mappings satisfying that for every vertex u it holds that $N_G(u)$ is mapped to $N_H(f(u))$ injectively (bijectively, resp.). Locally bijective homomorphisms are also known as *covering projections*. Similarly, locally injective homomorphisms are sometimes called *partial covering projections*, while locally surjective homomorphisms are also known as *role assignments*.

Covers and quasi-covers are discrete variants of the corresponding notions in algebraic topology. To obtain a quasi-cover consider a 2-cell embedding of

* Supported by Charles University as GAUK 95710.
** Supported by MŠMT ČR grant LH12095 and GAČR grant P202/12/G061.
*** Supported by the grant SVV-2012-265313.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 310–321, 2013.

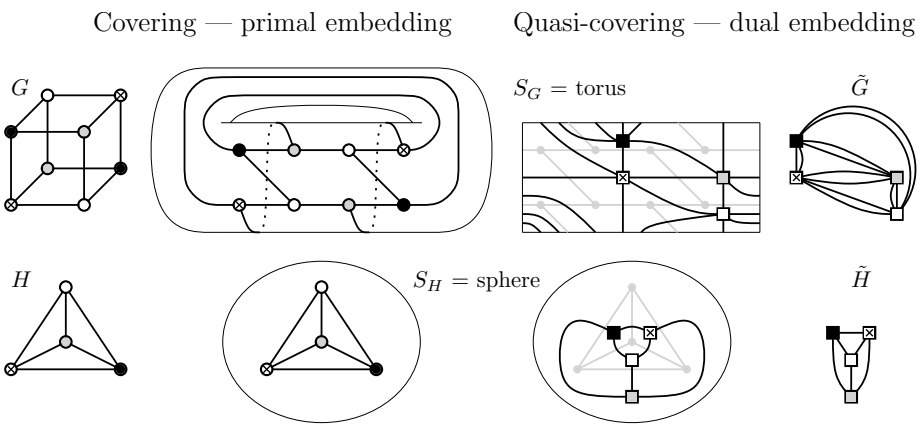


Fig. 1. Example of obtaining a quasi-covering from $G = Q_3$ that covers $H = K_4$. The mappings are indicated by vertex colors.

a graph H in an orientable surface S_H and a graph G covering H via f (see Figure 1 for an example). By using an 2-cell embedding of G where every vertex u uses the same neighbor ordering as $f(u)$, we obtain a surface S_G with the following property: the covering f extends to a mapping between S_G and S_H which respects edges and faces of both embeddings. In addition this mapping is a local homeomorphism except of those faces whose length is a multiple of the length of its image (the length is measured in the number of vertices on the face). The mapping on these faces contains singularity of degree being equal to the ratio of the two face lengths.

We construct duals $\tilde{G}$ and $\tilde{H}$ from the two 2-cell embeddings of G and H in S_G and S_H, resp. and factor the mapping between S_G and S_H to a homomorphism between $\tilde{G}$ and $\tilde{H}$. Moreover, as the boundary of each face of G (in S_G) must be mapped homeomorphically onto the boundary of the appropriate face of H (in S_H), we get that the resulting mapping between duals $\tilde{G}$ and $\tilde{H}$ is c-fold on the neighborhood of any vertex of $\tilde{G}$.

We follow the usual scenario for the question whether a graph G admits a possibly specific homomorphism to H. Since such tests allow no simple criterion, we define several classes of decision problems: H-Hom, H-QCover, H-LIHom, H-LSHom, and H-LBHom, resp. In all of them H is a fixed target graph and the query is whether a graph G on the input admits a homomorphism to H of the appropriate constraint: being a homomorphism, a quasi-covering, locally injective, locally surjective, and locally bijective, resp.

The computational complexity of H-Hom was fully determined by Hell and Nešetřil [11]. They show that the problem is solvable in polynomial time only for bipartite H and that it is NP-complete otherwise.

The study of H-LSHom was initiated by Kristiansen and Telle [16] and a full dichotomy was completed by Fiala and Paulusma [9]. For connected H they

showed that H-LSHom is NP-complete whenever H has at least three vertices; for disconnected H the condition is more elaborate.

The complexity of locally bijective homomorphisms was first studied by Bodlaender [3] and by Abello et al. [1]. Despite the subsequent effort of several authors (see e.g. papers by Kratochvíl et al. [13–15] or a survey by Fiala and Kratochvíl [8]) the complete characterization has not been settled yet.

The dichotomy for the computational complexity of the H-LIHom problem is also not known. Some partial results can be found in [4–6, 17, 2]. It might be of independent interest that locally injective homomorphisms generalize the notion of $L(2,1)$-labelings, which are motivated by the frequency assignment problem. Fiala and Kratochvíl [7] also considered the list version of the H-LIHom problem and provided a dichotomy.

In our paper we show that the H-QCover problem yields for connected graphs H the same dichotomy as the H-LSHom problem:

Theorem 1. *Let H be a connected graph. If H has at least three vertices, then the H-QCover problem is NP-complete. Otherwise, it is solvable in linear time.*

This is in contrast with the well known fact that testing the existence of a covering between two embedded graphs that locally extends to a homeomorphism of the embedding admits a straightforward quadratic-time algorithm: if the mapping is determined for any edge, it has a unique extension to adjacent edges given by the ordering of the edges around a vertex in the embedding. Therefore also the corresponding problem for the quasi-coverings between the associated duals is polynomially solvable with the same time complexity.

2 Preliminaries

Unless said otherwise, we consider simple and connected graphs. We denote the set of vertices of a graph G by $V(G)$ and its edge set by $E(G)$. We denote the degree of a vertex v in G by $\deg_G(v)$ and the set of all neighbors of v — the *neighborhood* of v — by $N_G(v)$. In a *d-regular* graph all vertices are of the same degree d.

For the definition of other standard graph theoretic terms (like paths, complete bipartite graphs), see e.g. a monograph by Nešetřil and Matoušek [18].

We call a mapping $f : X \to Y$ between two sets *c-fold* if for all $y \in Y$ it holds that $|f^{-1}(y)| = c$.

Recall that a homomorphism $f : G \to H$ is a *quasi-covering* if for each vertex $v \in V(G)$ there exists an integer c such that $f|_{N_G(v)}$ is c-fold between $N_G(v)$ and $N_H(f(v))$. Note that quasi-covering which is 1-fold on every vertex of G is indeed a covering projection.

Observe that the composition of a c-fold and a d-fold mapping is a cd-fold mapping. Hence a composition of two quasi-coverings is also a quasi-covering. We use this fact also in the case when one of these two mappings is a covering projection or an automorphism.

By a *boundary* δH of an induced subgraph H of a graph G we mean the set of vertices of H that are adjacent to a vertex outside H.

The symbol $\mathrm{lcmd}(G)$ stands for the least common multiple of degrees of all non-isolated vertices in G.

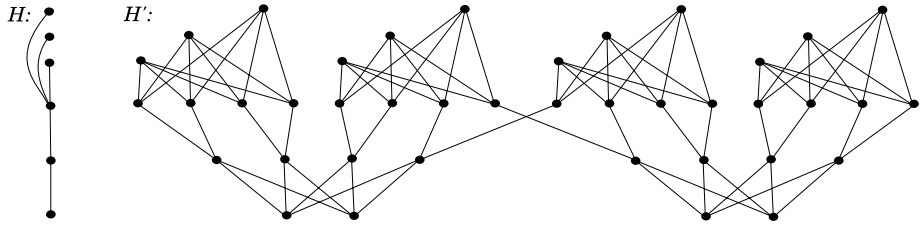


Fig. 2. Construction of the graph H' for a graph H with $d = 4$. Copies of vertices of graph H are in horizontal lines.

Proposition 1. *For every graph H there exists a regular connected graph H' such that H' quasi-covers H.*

Proof. Without loss of generality assume that $E(H)$ is not empty, as otherwise H itself is 0-regular and H' could be chosen to consist of a single isolated vertex. Let $d = \mathrm{lcmd}(H)$. We construct a d-regular graph H' and quasi-covering $h\colon H' \to H$ as follows.

For every vertex $x \in V(H)$ we insert into $V(H')$ vertices $x_1, x_2, \ldots, x_{d\deg_H(x)}$. All these $d\deg_H(x)$ vertices are mapped onto x by h (see Figure 2). For every edge $xy \in E(H)$ we add d^2 edges between sets $h^{-1}(x)$ and $h^{-1}(y)$ in such a way that every $x_i \in h^{-1}(x)$ is incident with $\frac{d}{\deg_H(x)}$ of these d^2 edges. Analogously every $y_i \in h^{-1}(y)$ is incident with $\frac{d}{\deg_H(y)}$ of them. This can be done e.g. by using $\deg_H(x)\deg_H(y)$ copies of the complete bipartite graph $K_{\frac{d}{\deg_H(y)},\frac{d}{\deg_H(x)}}$.

If H' is not connected, we restrict H' to any of its connected component containing at least one edge. The obtained graph H' is d-regular since for every $x_i \in V(H')$ it holds that $\deg_{H'}(x_i) = \deg_H(x)\frac{d}{\deg_H(x)} = d$.

By the construction, for every neighbor v of $h(x_i) = x$ in H we have that $|h^{-1}(v) \cap N_{H'}(x_i)| = \frac{d}{\deg_H(x)}$. Therefore, h is $\frac{d}{\deg_H(x)}$-fold between $N_{H'}(x_i)$ and $N_H(x)$, i.e. a quasi-covering as required. □

Kratochvíl, Proskurowski, and Telle [13] proved existence of a cover with a special property, which we also use in our paper.

Proposition 2 ([13]). *For every d-regular connected graph H', there exists a d-regular graph A with a specified vertex a, such that any bijective mapping between $N_A(a)$ and $N_{H'}(x_i)$ for arbitrary $x_i \in V(H')$ can be extended to a covering projection $g : A \to H'$ satisfying $g(a) = x_i$.*

We use the Proposition 2 to construct an analogous graph, called *multi-quasi-cover* of H as follows:

Lemma 1. *Let H be a connected graph with at least two vertices and let $d = \mathrm{lcmd}(H)$. There exists a d-regular graph A with specified vertex a, such that for any vertex $x \in V(H)$ it holds that any $\frac{d}{\deg_H(x)}$-fold mapping $\varphi : N_A(a) \to N_H(x)$ can be extended to a quasi-covering $f : A \to H$, such that $f(a) = x$.*

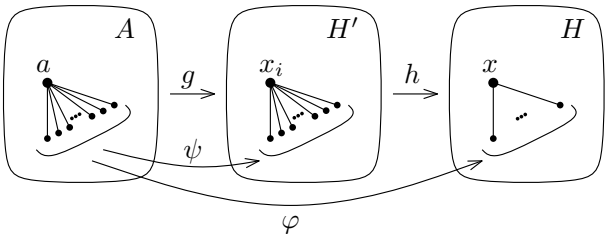


Fig. 3. Construction of *multi-quasi-cover* A of H

Proof. According to Proposition 1 we first construct a d-regular connected graph H' and a quasi-covering $h : H' \to H$. Then we use Proposition 2 for H' and obtain the desired d-regular graph A with a specified vertex a (see Figure 3).

For the given $x \in V(H)$ and $\frac{d}{\deg_H(x)}$-fold mapping $\varphi : N_A(a) \to N_H(x)$ we choose arbitrarily $x_i \in h^{-1}(x)$ and determine a bijective mapping $\psi : N_A(a) \to N_{H'}(x_i)$ such that $h \circ \psi = \varphi$. Such ψ exists since both φ and $h|_{N_{H'}(x_i)}$ are $\frac{d}{\deg_H(x)}$-fold, hence it suffices to match arbitrarily vertices of $\varphi^{-1}(y)$ and $h^{-1}(y) \cap N_{H'}(x_i)$ for each neighbor y of x.

Let g be the extension of ψ according to Proposition 2. Then $f = h \circ g$ is a composition of two quasi-coverings, i.e. a quasi-covering as well. Since $g|_{N_A(a)} = \psi$, we get that $f|_{N_A(a)} = h \circ g|_{N_A(a)} = h \circ \psi = \varphi$, i.e. f extends φ. Finally, $f(a) = h(g(a)) = h(x_i) = x$ as required. □

We involve arguments already used by Fiala and Paulusma [9].

Let $N_G^d(u)$ be the set of vertices at distance at most d from u in the graph G. By induction on d one gets:

Observation 1. *If $f : G \to H$ is locally surjective homomorphism then f is also a surjective mapping between sets $N_G^d(u)$ and $N_H^d(f(u))$ for any $u \in V(G)$ and any d.*

Definition 1. *We say that x is a* maximal distance vertex *in a connected graph H, if there exists a vertex $z \in V(H)$ such that the distance between x and z attains the maximum among distances between all possible pairs of vertices in H. This maximum distance is called the* diameter *of H, and is denoted by* $\mathrm{diam}(H)$.

Observation 1 provides the following corollaries:

Corollary 1 ([9]). *Let H be a graph and let x be a maximal distance vertex in H. If G contains H as an induced subgraph such that $\delta H = \{x\}$, then any locally surjective homomorphism $f : G \to H$ has the property that f restricted to H is an automorphism of H.*

Corollary 2. *Let H be a graph, x be its maximal distance vertex, and let M be the set of vertices at distance* $\mathrm{diam}(H)$ *from x. If G contains H as an induced subgraph such that $\delta H \subseteq M$ then any locally surjective homomorphism $f : G \to H$ satisfying that $f(x)$ is a maximal distance vertex, has the property that f restricted to H is an automorphism of H.*

Proof. By the choice of x we get that $|N_G^{\mathrm{diam}(H)}(x)| = |N_H^{\mathrm{diam}(H)}(f(x))| = |V_H|$. A surjective mapping between sets of the same size is a bijection.

3 Coloring Gadgets

For the purpose of our NP-hardness reductions we build a specific gadget according to the following needs:

Definition 2. *Let H be a connected graph and let x be its vertex of degree $k \geq 1$. We say that the graph $F = \mathrm{CG}_H(x, m)$ with m specified vertices $u_1, \ldots, u_m$ is a* coloring gadget *for H of size m and for k colors if it has the following properties:*

- *F allows at least one quasi-covering $f : F \to H$ that maps all specified vertices u_i to x,*
- *whenever a graph G contains F as an induced subgraph with $\delta F \subseteq \{u_1, \ldots, u_m\}$ and whenever $f : G \to H$ is a quasi-covering, then*
  - *i) f restricted to F is a quasi-covering projection as well,*
  - *ii) $\deg_H(f(u_1)) = k$,*
  - *iii) $N_H(f(u_i)) = N_H(f(u_1))$ for each specified vertex u_i*

In this section we show that a coloring gadget exists for every connected graph H on at least three vertices.

Lemma 2. *Let H be a connected graph on at least three vertices whose all maximal distance vertices are of degree one. Then, for any neighbor x of a maximal distance vertex and any positive integer m the $\mathrm{CG}_H(x, m)$ exists.*

In particular, the above lemma applies on every path or a tree on at least three vertices.

Proof. Let z_1 be a maximal distance vertex in H, let x be its neighbor, and let y be a vertex at the maximal distance from z_1. Let $z_2, \ldots, z_t$ be the neighbors of x other than z_1 that are also at the maximal distance from y (see Figure 4).

We take $m+2$ copies $H_1, \ldots, H_{m+2}$ of the graph H and merge all copies of y into a new vertex w. Then, we merge the first $m+1$ copies of each z_i into a new vertex v_i, and obtain the coloring gadget F. For specified vertices $u_1, \ldots, u_m$ we choose the first m copies of x.

A quasi-covering $F \to H$ can be obtained if we project each H_i onto H. It means that to show that F is a coloring gadget we only need to prove the conditions $i), ii)$, and $iii)$ from the Definition 2. Assume that F is an induced

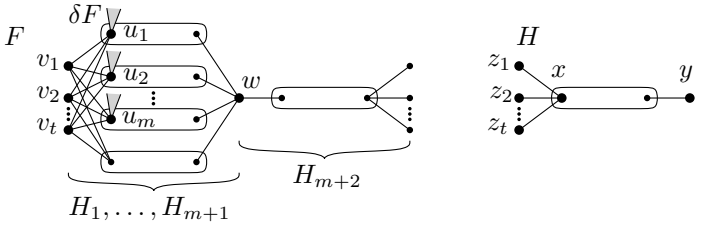


Fig. 4. The coloring gadget F for H with all maximal distance vertices of degree one

subgraph of G, such that $\delta F \subseteq \{u_1, \ldots, u_m\}$ and that $f : G \to H$ is a quasi-covering.

Since H_{m+2} is an induced subgraph of G with boundary $\delta H_{m+2} = \{w\}$, we apply Corollary 1 and get that $f(w)$ is a maximal distance vertex in H and also that $f|_{H_{m+2}}$ is an isomorphism to H.

We split w back into $m+2$ vertices $w_1, \ldots, w_{m+2}$. Denote the resulting graph by G'. We also alter f on the new vertices $w_1, \ldots, w_{m+2}$, which we map onto $f(w)$. The resulting mapping is denoted by f'. Since $f(w)$ is a maximal distance vertex in H, it has a unique neighbor. Hence f' is a 1-fold on each $N_{G'}(w_i)$, i.e. a quasi-covering $G' \to H$.

We focus on the copy H_{m+1} in G' and apply Corollary 2 with respect to f' and obtain that $v_1, \ldots, v_t$ are mapped on maximum distance vertices of H. Since maximum distance vertices have unique neighbor and f coincides with f' on $N_G(v_1)$, we get that $f(u_1) = \cdots = f(u_m)$ and moreover $\deg_H(f(u_1)) = \deg_H(x)$. This shows that conditions *ii*) and *iii*) from the definition of coloring gadget hold.

By the construction of F and by the fact that $f(w)$ and y can be exchanged by an automorphism of H we get that for each $i \in \{1, \ldots, m\}$ it holds that $|N_{G'}^{\mathrm{diam}(H)-1}(w_i)| = |N_H^{\mathrm{diam}(H)-1}(f(w))|$. By Observation 1 we get that both $f'|_{H_i}$ and $f|_{H_i}$ are bijections between $V(H_i)$ and $V(H)$. This means that neighbors of u_i inside the copy H_i must be mapped to $\deg_H(x)$ distinct neighbors of $f(u_1)$ in H. Hence $f|_{H_i}$ is an isomorphism between H_i and H. Therefore, $f|_F$ is a quasi-covering and the condition *i*) holds. □

Lemma 3. *Let H be a connected graph with a maximal distance vertex x of degree at least two. For every positive integer m a coloring gadget $\mathrm{CG}_H(x, m)$ exists.*

Proof. Let $k = \deg_H(x)$ and $d = \frac{\mathrm{lcmd}(H)}{k}$.

To construct $F = \mathrm{CG}_H(x, m)$ we first take $(m+1)d$ mutually disjoint copies of H and denote them H_i^t with $i \in \{1, \ldots, m+1\}$ and $t \in \{1, \ldots, d\}$. Intuitively, the symbol x_i^t will denote the vertex of H_i^t corresponding to x in H.

Separately we construct a dk-regular multi-quasi-cover A of H with a specified vertex a according to Lemma 1. Denote the dk neighbors of a in A by w_j^t where $j \in \{1, \ldots, k\}$ and $t \in \{1, \ldots, d\}$. We now remove the vertex a from A to obtain the graph B.

In the next step we insert into F the disjoint union of $m+1$ copies $B_1, \ldots, B_{m+1}$ of the graph B (see Figure 5). For every $j \in \{1, \ldots, k\}$ and $t \in \{1, \ldots, d\}$ we merge all $m+1$ copies of the vertex w_j^t in $B_1, \ldots, B_{m+1}$ into a single vertex v_j^t.

We finalize the construction of the graph F by adding edges $x_i^t v_j^t$ for all $i \in \{1, \ldots, m+1\}$, $j \in \{1, \ldots, k\}$, and $t \in \{1, \ldots, d\}$.

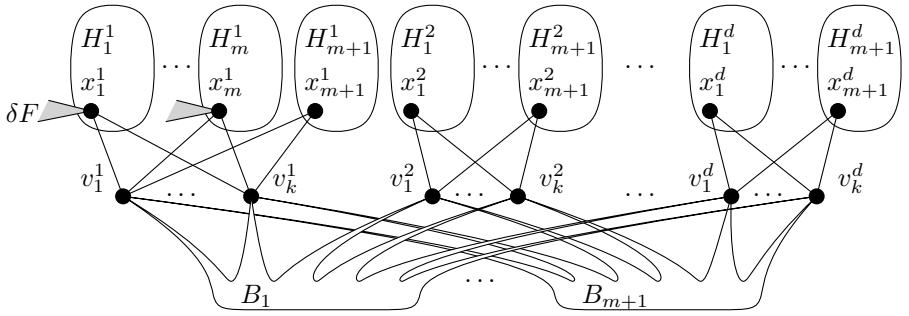


Fig. 5. Example of the construction of $CG_H(x, m)$ for maximal vertex x of degree $k \geq 2$

For the m specified vertices $u_1, \ldots, u_m$ of the coloring gadget we use the vertices $x_1^1, \ldots, x_m^1$.

To show that F quasi-covers H we define a quasi-covering $f : F \to H$ as follows:

- on every H_i^t let f act as an isomorphism to H, such that $f(x_i^t) = x$,
- let f act as a bijection between $v_1^t, v_2^t, \ldots, v_k^t$ and $N_H(x)$ for each t,
- since the so far defined mapping f is d-fold between each δB_i and $N_H(x)$ (and all the neighbors of vertices in δB_i out of B_i are mapped to x) we may extend it to a quasi-covering inside each subgraph B_i according to Lemma 1.

Note that the quasi-covering $f_A : A \to H$ obtained by Lemma 1 is $\frac{dk}{\deg_H(y)}$-fold between $N_A(w_j^t)$ and $N_H(y)$, where $y = f_A(w_j^t)$. Hence, the mapping f is $\frac{(m+1)dk}{\deg_H(y)}$-fold between $N_A(v_j^t)$ and $N_H(y)$, i.e. a quasi-covering.

Assume now that F is an induced subgraph of G that allows a quasi-covering $f : G \to H$ and such that $\delta F \subseteq \{x_1^1, \ldots, x_m^1\}$. We show that conditions *i*), *ii*) and *iii*) from the Definition 2 hold. Since x is a maximal distance vertex, Corollary 1 yields that f restricted to each H_i^1 is an isomorphism of H_i^1 and H. Hence $\deg_H(f(x_i^1)) = \deg_H(x) = k$ for each $i \in \{1, \ldots, m\}$, i.e. *ii*) holds.

Let $x' = f(x_{m+1}^1)$. Observe that the vertex x_{m+1}^1 has also *exactly* k neighbors outside H_{m+1}^1 (in contrast with vertices $x_1^1, \ldots, x_m^1$ that might have further neighbors outside F), the vertices $v_1^1, \ldots, v_k^1$ must be mapped bijectively onto the k neighbors of x'. Hence $N_H(f(x_i^1)) = N_H(x')$ for each i, i.e. *iii*) holds.

Consequently, the restriction of f to F is 2-fold on the vertices $x_1^1, \ldots, x_m^1$, i.e. a quasi-covering and *i*) holds as well. This argument concludes the proof that F with specified vertices $u_1, \ldots, u_m$ is a coloring gadget for H. □

4 The NP-Hardness Reduction

Recall that for a fixed graph H the problem H-QCover is defined as follows:

Problem: H-QCover
Input: A graph G
Query: Does G allow a quasi-covering to H?

Note that for all graphs H the problem H-QCover belongs to the class NP, since the properties of a quasi-covering can be verified in polynomial time.

In order to prove Theorem 1 we distinguish several cases according to the structure of the graph H. We first show an NP-hardness reduction from the following well-known NP-complete problem [10, problem LO6]:

Problem: 2-in-4 SAT
Input: A formula Φ in CNF where every clause contains exactly four literals
Query: Could Φ be satisfied such that every clause contains exactly two positively valued literals?

Since 2-in-4 SAT is the only version of SAT problem we use, we reserve the word *satisfiable* for formulas which are 2-in-4 satisfiable.

Lemma 4. *Let H be a connected graph on at least three vertices. If H has a maximal distance vertex $x \in V(H)$ of degree two or if all maximal vertices of H are of degree one and some maximal vertex has neighbor x of degree two, then the H-QCover problem is NP-complete.*

Proof. Let Φ be an instance of 2-in-4 SAT. Denote the clauses of Φ by $C_1, \ldots, C_m$ and its variables by $v_1, \ldots, v_n$. We construct a graph $G_{\Phi,H}$ as follows:

We start with a disjoint union of a copy of the coloring gadget $\mathrm{CG}_H(x, n)$ with specified vertices $u_1, u_2, \ldots, u_n$ and a copy of $\mathrm{CG}_H(x, 2m)$ with specified $w_1, w'_1, w_2, w'_2, \ldots, w_m, w'_m$. The existence of these gadgets is guaranteed by Lemmata 2 and 3. Then we include extra $2n$ new vertices $p_1, q_1, p_2, q_2, \ldots, p_n, q_n$ and connect each vertex u_i with vertices p_i and q_i.

If any variable v_i is one of the positive literals of C_j, then we join w_j with p_i and also w'_j with q_i . As a counterpart, if $\neg v_i \in C_j$ then we insert edges $w_j q_i$ and $w'_j p_i$. This step concludes the construction of $G_{\Phi,H}$ (see Figure 6).

Claim. If $G_{\Phi,H}$ is an induced subgraph of a quasi-cover G of H such that $\delta(G_{\Phi,H}) \subseteq \{p_1, q_1, \ldots, p_n, q_n\}$, then Φ is satisfiable.

Suppose that $f : G \to H$ is a quasi-covering. The properties of both coloring gadgets yield that the images of both sets of specified vertices are in H of degree 2 — the same as $\deg(x)$. Denote the two neighbors of $f(u_1)$ by y and z.

Then we know that for all $i \in \{1, \ldots, n\}$ it holds that $\{f(p_i), f(q_i)\} = \{y, z\}$. Consequently, the vertices y and z are the two neighbors of each $f(w_j)$. We assign $v_i = \mathsf{true}$ if and only if $f(p_i) = y$ (thus $v_i = \mathsf{false} \iff f(p_i) = z$).

As f restricted to each coloring gadget is a quasi-covering, it must be a quasi-covering also on the subgraph remaining after the removal of both gadgets except

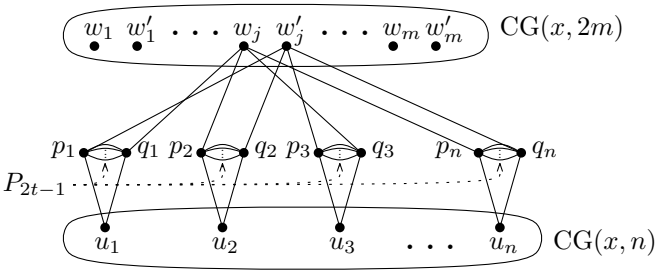


Fig. 6. An example of the construction of the graph G for $H = P_t$. The edges depicted are related with the clause $C_j = \neg v_1 \vee v_2 \vee \neg v_3 \vee v_n$. The graph G_{Φ,P_t} differs from G only by the presence of the paths P_{2t-1}.

their boundaries. Therefore, two neighbors of each w_j are mapped on y and two of them on z. Since these neighbors correspond to literals in C_j, we know that there are exactly two positive literals in every clause C_j. Therefore, we have obtained the desired satisfying assignment and proved the claim.

Now we resume the proof of Lemma 4 and extend $G_{\Phi,H}$ into a graph G such that G quasi-covers H if Φ is satisfiable.

According to Lemma 1 we construct a $2d$-regular multi-quasi-cover of H with a specified vertex a and $d = \frac{\text{lcmd}(H)}{2}$. Let B be the graph resulting by the deletion of a from the multi-quasi-cover.

We start the construction of G with d copies of $G_{\Phi,H}$. To obtain G, we then perform the following steps for each $i \in \{1, \ldots, n\}$:

- First we determine o_i to be the number of occurrences of v_i in Φ.
- Then we insert in the so far constructed graph exactly $o_i + 1$ copies of B.
- Now we identify $o_i + 2$ sets, each of size $2d$: the first set consists of the copies of vertices p_i and q_i while the others are formed by neighbors of the deleted vertex a in the $o_i + 1$ copies of B.
- On this set system we build $2d$ disjoint transversals[1] and merge vertices of each transversal into a single vertex. (See Figure 7) In other words, we merge $2d$ $(o_i + 2)$-tuples of distinct vertices into $2d$ single vertices, such that the boundary of each $G_{\Phi,H}$ and of each B is preserved.

Suppose now that Φ is satisfiable. Let y and z be the neighbors of x. We define $f : V(G) \to V(H)$ as follows:

- $f(u_i) = f(w_j) = f(w'_j) = x$ for all i and j in all d copies.
- if $v_i = \mathsf{true}$ then $f(p_i) = y$ and $f(n_i) = z$, otherwise $f(p_i) = z$ and $f(n_i) = y$;
- extend the so far defined f to all copies of B by Lemma 1;
- extend f to a quasi-covering of all coloring gadgets of all $G_{\Phi,H}$ by Lemma 2 or 3.

[1] By a *transversal* of a set system $\mathcal{S}$ we mean the range of an injective map $\varphi : \mathcal{S} \to \bigcup \mathcal{S}$ such that $\forall S \in \mathcal{S} : \varphi(S) \in S$.

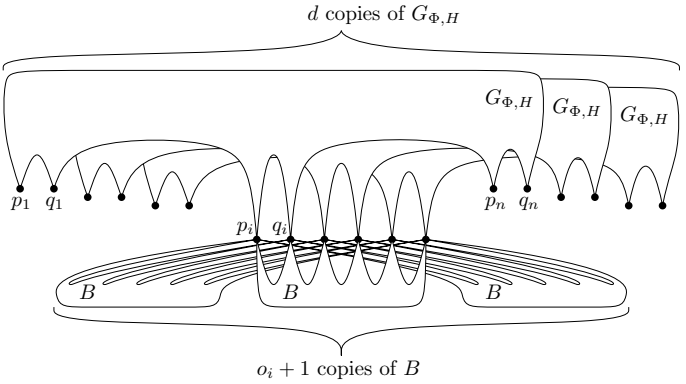


Fig. 7. The result of the i-th iteration, when $d = 3$ and when variable v_i has two occurrences

The obtained mapping is $(o_i + 1)$-fold on each $N(p_i)$ and $N(q_i)$, hence a quasi-covering. □

Note that for $H = P_t$ the above construction yields $d = 1$ and $B = P_{2t-1}$. Hence the graph G consists from a single copy of $G_{\Phi,H}$, where each pair of vertices p_i and q_i is joined by $o_i + 1$ paths of length $2t - 1$, as depicted in Figure 6.

Lemma 5. *Let H be a connected graph on at least three vertices. If H has a maximal distance vertex $x \in V(H)$ of degree $k > 2$ or if all maximal vertices of H are of degree one and some maximal vertex has neighbor x of degree $k > 2$, then the H-*QCover *problem is* NP*-complete.*

We skip the proof of Lemma 5 due to the space limitation.

Lemmas 4 and 5 constitute the NP-hardness part of the proof of Theorem 1. The polynomial part is straightforward: only edgeless graphs quasi-cover P_1; while a graph quasi-covers P_2 if and only if it is bipartite without isolated vertices. Both these classes could be recognized in linear time.

5 Conclusion

We have proved the dichotomy for the computational complexity of H-QCover problem when the graph H is connected. This can be combined with a construction of Fiala and Paulusma [9, Proposition 5] to get a classification also for disconnected simple graphs:

Corollary 3. *The H-*QCover *problem is polynomially solvable if either H is edgeless or if H is bipartite and at least one of its components is isomorphic to K_2. Otherwise, it is* NP*-complete.*

The construction provides a quasi-cover of a chosen component, while it forbids all locally surjective homomorphisms to other components. Note that the other constructions presented in that paper do not provide quasi-covers, so our classification of connected graphs was a key ingredient for Corollary 3.

The classification is open for multigraphs with possible semi-edges; these appear naturally in the topological models.

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. Bílka, O., Lidický, B., Tesař, M.: Locally injective homomorphism to the simple Weight graphs. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 471–482. Springer, Heidelberg (2011)
3. Bodlaender, H.L.: The classification of coverings of processor networks. Journal of Parallel Distributed Computing 6, 166–182 (1989)
4. 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)
5. Fiala, J., Kratochvíl, J.: Partial covers of graphs. Discussiones Mathematicae Graph Theory 22, 89–99 (2002)
6. Fiala, J., Kratochvíl, J., Pór, A.: On the computational complexity of partial covers of Theta graphs. Discrete Applied Mathematics 156, 1143–1149 (2008)
7. Fiala, J., Kratochvíl, J.: Locally injective graph homomorphism: Lists guarantee dichotomy. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 15–26. Springer, Heidelberg (2006)
8. Fiala, J., Kratochvíl, J.: Locally constrained graph homomorphisms — structure, complexity, and applications. Computer Science Review 2(2), 97–111 (2008)
9. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignment problem. Theoretical Computer Science 349(1), 67–81 (2005)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. Ltd. (1979)
11. Hell, P., Nešetřil, J.: On the complexity of H-colouring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
12. Holyer, I.: The NP-completeness of edge-coloring. SIAM Journal on Computing 10(4), 718–720 (1981)
13. Kratochvíl, J., Proskurowski, A., Telle, J.A.: Covering regular graphs. Journal of Combinatorial Theory B 71, 1–16 (1997)
14. Kratochvíl, J., Proskurowski, A., Telle, J.A.: Covering directed multigraphs I. colored directed multigraphs. In: Möhring, R.H. (ed.) WG 1997. LNCS, vol. 1335, pp. 242–257. Springer, Heidelberg (1997)
15. Kratochvíl, J., Proskurowski, A., Telle, J.A.: Complexity of graph covering problems. Nordic Journal of Computing 5, 173–195 (1998)
16. 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)
17. Lidický, B., Tesař, M.: Complexity of Locally Injective Homomorphism to the Theta Graphs. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 326–336. Springer, Heidelberg (2011)
18. Matoušek, J., Nešetřil, J.: Invitation to Discrete Mathematics. Oxford University Press (2008)

QCSP on Partially Reflexive Cycles – The Wavy Line of Tractability

Florent Madelaine[1] and Barnaby Martin[2,⋆]

[1] Clermont Université, Université d'Auvergne, LIMOS, BP 10448, F-63000 Clermont-Ferrand, France
[2] CNRS / LIX UMR 7161, École Polytechnique, Palaiseau, France

Abstract. We study the (non-uniform) quantified constraint satisfaction problem QCSP($\mathcal{H}$) as $\mathcal{H}$ ranges over partially reflexive cycles. We obtain a complexity-theoretic dichotomy: QCSP($\mathcal{H}$) is either in NL or is NP-hard. The separating conditions are somewhat esoteric hence the epithet "wavy line of tractability" (see Figure 5 at end).

1 Introduction

The *quantified constraint satisfaction problem* QCSP($\mathcal{B}$), for a fixed *template* (structure) $\mathcal{B}$, is a popular generalisation of the *constraint satisfaction problem* CSP($\mathcal{B}$). In the latter, one asks if a primitive positive sentence (the existential quantification of a conjunction of atoms) Φ is true on $\mathcal{B}$, while in the former this sentence may be positive Horn (where universal quantification is also permitted). Much of the theoretical research into CSPs is in respect of a large complexity classification project – it is conjectured that CSP($\mathcal{B}$) is always either in P or NP-complete [11]. This *dichotomy* conjecture remains unsettled, although dichotomy is now known on substantial classes (e.g. structures of size ≤ 3 [20,3] and smooth digraphs [12,1]). Various methods, combinatorial (graph-theoretic), logical and universal-algebraic have been brought to bear on this classification project, with many remarkable consequences. A conjectured delineation for the dichotomy was given in the algebraic language in [4].

Complexity classifications for QCSPs appear to be harder than for CSPs. Indeed, a classification for QCSPs will give a fortiori a classification for CSPs (if $\mathcal{B} \uplus \mathcal{K}_1$ is the disjoint union of $\mathcal{B}$ with an isolated element, then QCSP($\mathcal{B} \uplus \mathcal{K}_1$) and CSP($\mathcal{B}$) are polynomially equivalent). Just as CSP($\mathcal{B}$) is always in NP, so QCSP($\mathcal{B}$) is always in Pspace. However, no overarching polychotomy has been conjectured for the complexities of QCSP($\mathcal{B}$), as $\mathcal{B}$ ranges over finite structures, but the only known complexities are P, NP-complete and Pspace-complete (see [2,17] for some trichotomies). It seems plausible that these complexities are the only ones that can be so obtained (for more in this see [6]).

In this paper we study the complexity of QCSP($\mathcal{H}$), where $\mathcal{H}$ is a partially reflexive cycle. In this respect, our paper is a companion to the similar classification for partially reflexive forests in [16]. We derive a classification between those

⋆ The author is supported by ANR Blanc International ALCOCLAN.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 322–333, 2013.

cases that are in NL and those that are NP-hard. For some of the NP-hard cases we are able to demonstrate Pspace-completeness. The dichotomy, as depicted in Figure 5 at the end, is quite esoteric and deviates somewhat from similar classifications (e.g. for retraction as given in [10]). To our minds, this makes it interesting in its own right. Some of our hardness proofs come from judicious amendments to the techniques used in [16]. Several others use different elaborate encodings of retraction problems, known to be hard from [10]. All but one of our NL-membership results follow from a majority polymorphism in an equivalent template (indeed – the so-called *Q-core* of [14]), as they did in [10]. However, $\mathcal{C}_{0111}$ is special. It has no QCSP-equivalent that admits a majority (indeed, it omits majority and is a Q-core), so we have to give a specialised algorithm, based on ideas from [13], to demonstrate that QCSP($\mathcal{C}_{0111}$) is in L. Indeed, and in light of the observations in [14], this is the principal news from the partially reflexive cycles classification that removes it from being simply a sequel to partially reflexive forests: for a partially reflexive forest $\mathcal{H}$, either the Q-core of $\mathcal{H}$ admits a majority polymorphism and QCSP($\mathcal{H}$) is in NL, or QCSP($\mathcal{C}$) is NP-hard. The same can not be said for partially reflexive cycles, due to the odd case of $\mathcal{C}_{0111}$.

This paper is organised as follows. After the preliminaries, we address small cycles in Section 3. Then we deal with reflexive cycles, cycles whose loops induce a path and cycles with disconnected loops in Sections 4, 5 and 6, respectively. Finally we give our classification in Section 7 and our conclusions in Section 8. For reasons of space, many proofs are omitted.

2 Definitions and Preliminaries

Let $[n] := \{1, \ldots, n\}$. A graph $\mathcal{G}$ has vertex set G, of cardinality $|G|$, and edge set $E(\mathcal{G})$. For a sequence $\alpha \in \{0,1\}^*$, of length $|\alpha|$, let $\mathcal{P}_\alpha$ be the undirected path on $|\alpha|$ vertices such that the ith vertex has a loop iff the ith entry of α is 1 (we may say that the path $\mathcal{P}$ is *of the form* α). We will usually envisage the domain of a path with n vertices to be $[n]$, where the vertices appear in the natural order. Similarly, for $\alpha \in \{0,1\}^*$, let $\mathcal{C}_\alpha$ be the $|\alpha|$ cycle with domain $[n]$ and edge set $\{(i,j) : |j-i| = 1 \bmod n\} \cup \{(i,i) : \alpha[i] = 1\}$ (note $|n-1| = |1-n| = 1 \bmod n$). If α and α' are sequences in $\{0,1\}^n$ such that $\alpha[i] = \alpha'[i+1 \bmod n]$ then $\mathcal{C}_\alpha$ and $\mathcal{C}_{\alpha'}$ are isomorphic.

A partially reflexive cycle is one that may include some self-loops. For such an m-cycle $\mathcal{C}$, whose vertices we will imagine to be $v_1, \ldots, v_m$ in their natural mod m adjacencies, let $[v_i \Rightarrow v_j]$ be shorthand for a conjunction specifying a path, whichever is the fastest way mod m, from v_i to v_j. For example, if $m = 5$, then 1.) $[v_1 \Rightarrow v_3]$ is $E(v_1, v_2) \wedge E(v_2, v_3)$, 2.) $[v_3 \Rightarrow v_1]$ is $E(v_3, v_2) \wedge E(v_2, v_1)$, and 3.) $[v_4 \Rightarrow v_1]$ is $E(v_4, v_5) \wedge E(v_5, v_1)$. We ask the reader to endure the following relaxation of this notation; $[v_i, v_{i+1} \Rightarrow v_j]$ indicates an edge from v_i to v_{i+1} then a path to v_j (which may not be the same as $[v_i \Rightarrow v_j]$ as the latter may take the other path around the cycle). Finally, let $\mathrm{Ref}(v_i, \ldots, v_j)$ indicate $E(v_i, v_i) \wedge \ldots \wedge E(v_j, v_j)$, whichever is the quickest way around the cycle mod m. All graphs in this paper are undirected, so edge statements of the form $E(x, y)$ should be read as asserting $E(x, y) \wedge E(y, x)$.

The problems CSP($\mathcal{T}$) and QCSP($\mathcal{T}$) each take as input a sentence Φ, and ask whether this sentence is true on $\mathcal{T}$. For the former, the sentence involves the existential quantification of a conjunction of atoms – *primitive positive* logic. For the latter, the sentence involves the arbitrary quantification of a conjunction of atoms – *positive Horn* logic. By convention equalities are permitted in both of these, but these may be propagated out by substitution in all but trivial degenerate cases. The *retraction problem* RET($\mathcal{B}$) takes as input some $\mathcal{G}$, with $\mathcal{H}$ an induced substructure of $\mathcal{G}$, and asks whether there is a homomorphism $h : \mathcal{G} \to \mathcal{H}$ such that h is the identity on $\mathcal{H}$. It is important that the copy of $\mathcal{H}$ is specified in $\mathcal{G}$; it can be that $\mathcal{H}$ appears twice as an induced substructure and there is a retraction from one of these instances but not to the other. The problem Ret($\mathcal{H}$) is easily seen to be logspace equivalent with the problem CSP($\mathcal{H}^c$), where $\mathcal{H}^c$ is $\mathcal{H}$ expanded with all constants (one identifies all elements assigned to the same constant and enforces the structure $\mathcal{H}$ on those constants).

The *direct product* $\mathcal{G} \times \mathcal{H}$ of two graphs $\mathcal{G}$ and $\mathcal{H}$ has vertex set $\{(x, y) : x \in G, y \in H\}$ and edge set $\{((x, u), (y, v)) : x, y \in G, u, v \in H, (x, y) \in E(\mathcal{G}), (u, v) \in E(\mathcal{H})\}$. Direct products are (up to isomorphism) associative and commutative. The kth power $\mathcal{G}^k$ of a graph $\mathcal{G}$ is $\mathcal{G} \times \ldots \times \mathcal{G}$ (k times). A homomorphism from a graph $\mathcal{G}$ to a graph $\mathcal{H}$ is a function $h : G \to H$ such that, if $(x, y) \in E(\mathcal{G})$, then $(h(x), h(y)) \in E(\mathcal{G})$. A *$k$-ary polymorphism* of a graph is a homomorphism from $\mathcal{G}^k$ to $\mathcal{G}$. A ternary function $f : G^3 \to G$ is designated a *majority* operation if $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$, for all $x, y \in G$.

A positive Horn sentence Φ in the language of graphs induces naturally a graph $\mathcal{G}_\Phi$ whose vertices are the variables of Φ and whose edges are the atoms of Φ. In the case of primitive positive Φ one would call $\mathcal{G}_\Phi$ the *canonical database* and Φ its *canonical query*. With positive Horn Φ there is additional extra structure and one may talk of a vertex-variable as being existential/ universal and as coming before/ after (earlier/ later), in line with the quantifier **block** and its order in Φ. Variables in the same quantifier block will not need their orders considered (there is commutativity within a block anyway). A typical reduction from a retraction problem Ret($\mathcal{C}$), where $|C| = m$, builds a positive Horn Φ sentence involving variables $v_1, \ldots, v_m$ where we want $\mathcal{G}_\Phi$ restricted to $\{v_1, \ldots, v_m\}$ (itself a copy of $\mathcal{C}$) to map automorphically to $\mathcal{C}$. Typically, we can force this with some evaluation of the variables (some of which might be universally quantified). The other valuations are *degenerate* and we must ensure at least that they map $\mathcal{G}_\Phi$ restricted to $\{v_1, \ldots, v_m\}$ homomorphically to $\mathcal{C}$.

3 Small Cycles

The classification for QCSP for cycles of length ≤ 4 is slightly esoteric, although it does match the analogous classification for Retraction (the former is a dichotomy between NL and Pspace-complete; the latter is a dichotomy between P and NP-complete). The following has appeared in [15], where it was given as an application of counting quantifiers in QCSPs. We review it now because we will need to generalise it later in Section 4.

Proposition 1 ([15]). *QCSP($\mathcal{C}_{1111}$) is Pspace-complete.*

Proof. We will reduce from the problem QCSP($\mathcal{K}_4$) (known to be Pspace-complete from, e.g., [2]). We will borrow heavily from the reduction of CSP($\mathcal{K}_4$) to Ret($\mathcal{C}_{1111}$) in [9]. We introduce the following shorthands (x', x'' must appear nowhere else in ϕ, which may contain more free variables than just x).

$$\exists^{\geq 1} x\ \phi(x) := \exists x\ \phi(x)$$
$$\exists^{\geq 2} x\ \phi(x) := \forall x' \exists x\ E(x', x) \wedge \phi(x)$$
$$\exists^{\geq 3} x\ \phi(x) := \forall x'' \forall x' \exists x\ E(x'', x) \wedge E(x', x) \wedge \phi(x)$$

On $\mathcal{C}_{1111}$, it is easy to verify that, for each $i \in [4]$, $\exists^{\geq i} x\ \phi(x)$ holds iff there exist at least i elements x satisfying ϕ. Thus our borrowing the notation of counting quantifiers is justified.

We now reduce an instance Φ of QCSP($\mathcal{K}_4$) to an instance Ψ of QCSP($\mathcal{C}_{1111}$).

We begin with a cycle $\mathcal{C}_{1111}$ on vertices 1, 2, 3 and 4, which we realise through their canonical query (without quantification) as $\theta(v_1, v_2, v_3, v_4) := E(v_1, v_2) \wedge E(v_2, v_3) \wedge E(v_3, v_4) \wedge E(v_4, v_1)$ (recall that the canonical query is in fact the reflexive closure of this, but this will not be important in this case or many future cases – when it is important it will be stated explicitly). If Φ contains an atom $E(x, y)$, then this gives rise to a series of atoms in Ψ as dictated by the gadget in Figure 1 (for each atom we add many new vertex-variables, corresponding to the vertices in the gadget that are not $x, y, 1, 2, 3, 4$). These atoms can be seen to join up with the atoms of θ as in the right end of the figure. Build Ψ''' from Φ by this process and conjunction with θ. Then make Ψ'' from Ψ''' by existentially quantifying all of the variables other than those associated to atoms of Φ (x, y in the figure) and v_1, v_2, v_3, v_4 ($1, \ldots, 4$ in the figure). Now, we build Ψ' from Ψ'', by copying the quantifier order of Φ on the outside of the existential quantifiers that we already have. Thus $\Psi'(v_1, v_2, v_3, v_4)$ is a positive Horn formula with precisely four free variables.

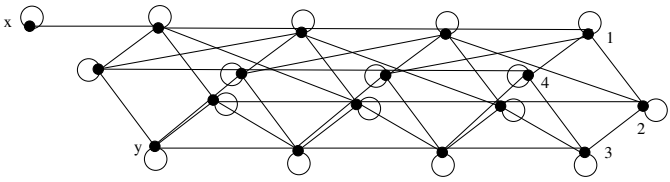


Fig. 1. Edge gadget in reduction from QCSP($\mathcal{K}_4$) to QCSP($\mathcal{C}_{1111}$)

It is not hard to see that when v_1, v_2, v_3, v_4 are evaluated as (an automorphism of) $1, 2, 3, 4$, then we have a faithful simulation of QCSP($\mathcal{K}_4$). This is because x and y, as in the gadget drawn, may evaluate precisely to distinct vertices on $\mathcal{C}_{1111}$. Finally, we build $\Psi := \exists v_1 \exists^{\geq 2} v_2 \exists^{\geq 3} v_3 \exists^{\geq 2} v_4\ \Psi'(v_1, v_2, v_3, v_4)$. It is not hard to see that Ψ forces on some evaluation of v_1, v_2, v_3, v_4 that these map isomorphically to $1, 2, 3, 4$ in $\mathcal{C}_{1111}$. Further, a rudimentary case analysis shows

us that when they do not, we can still evaluate the remainder of Ψ', if we could have done when they did. In fact, if v_1, v_2, v_3, v_4 are not mapped isomorphically (but still homomorphically, of course) to $1, 2, 3, 4$ then we can extend each of the edge gadgets to homomorphism under **all maps** of vertices x and y to $1, 2, 3, 4$ (not just ones in which x and y are evaluated differently).

Proposition 2. *QCSP($\mathcal{C}_{0111}$) is in L.*

Proof. Recall $\mathcal{C}_{0111}$ has vertices $\{1, 2, 3, 4\}$ in cyclic order and 1 is the only non-loop. Consider an input Φ for QCSP($\mathcal{C}_{0111}$), w.l.o.g. without any equalities, and its evaluation on $\mathcal{C}_{0111}$ as a game on Φ between *Prover*, playing (evaluating on $\mathcal{C}_{0111}$) existential variables, and *Adversary*, playing universal variables. Adversary never gains by playing 3, as any existential edge witness to anything from $\{4, 1, 2\}$ is already an edge-witness to 3. That is, if $E(x, 3)$ then already each of $E(x, 4), E(x, 1)$ and $E(x, 2)$. Similarly, Prover never gains by playing 1. Thus, Φ is true on $\mathcal{C}_{0111}$ iff it is true with all universal variables relativised to $\{4, 1, 2\}$ and all existential variables relativised to $\{2, 3, 4\}$. (This intuition is formalised in the notion of U-X-surjective hyper-endomorphism in [13]. What we are saying is that $\begin{smallmatrix}1&13\\2&2\\3&3\\4&4\end{smallmatrix}$ is a surjective hyper-endomorphism of $\mathcal{C}_{0111}$.) Henceforth, we will make this assumption of relativisation in our inputs.

Given an input Φ we will describe a procedure to establish whether it is true on $\mathcal{C}_{0111}$ based around a list of forbidden subinstances.

- (*i.*) An edge $E(x, y)$ in $\mathcal{G}_\Phi$ with the later of x and y being universal.
- (*ii.*) A 3-star $E(x_1, y), E(x_2, y), E(x_3, y)$ where x_1, x_2, x_3 are universal variables coming before y existential.
- (*iii.*) A path $y_1, \ldots, y_m$ of existential variables, where: both y_1 and y_m have edges to **two** earlier universal variables, and $y_2, \ldots, y_{m-1}$ have edges each to **one** earlier universal variable.
- (*iv.*) A path $y_1, \ldots, y_m$ of existential variables, where y_1 comes before y_m, and y_1 has an edge to an earlier universal variable. y_m has edges to two earlier universal variables at least one of which comes after y_1. Finally, $y_2, \ldots, y_{m-1}$ each have edges to an earlier universal variable.

These cases are illustrated in Figure 2. Using the celebrated result of [19] it can be seen that one may recognise in logspace whether or not Φ contains any of these forbidden subinstances. It is not hard to see that if Φ contains such a subinstance then Φ is false on $\mathcal{C}_{0111}$ (the universal variables adjacent and before y_m can be played as either $1, 2$ or $1, 4$ to force y_m to be either 2 or 4). We now claim all other instances Φ are true on $\mathcal{C}_{0111}$ and we demonstrate this by giving a winning strategy for Prover on such an instance. Owing to Case (*i*) being omitted, Adversary has no trivial win. Prover will now *always play* 3 *if she can.* Owing to the omission of Case (*ii*), Prover never has to answer a variable adjacent to more than two elements. It can be seen that there are few circumstances in which she can not play 3. Indeed, the only one is if she is forced at some point to play 2 or 4 as a neighbour to Adversary's having played 1. In

this case, Adversary can force this response to be propagated as in the chain of cases (iii) and (iv), but because these cases are forbidden, Adversary will never succeed here in winning the game.

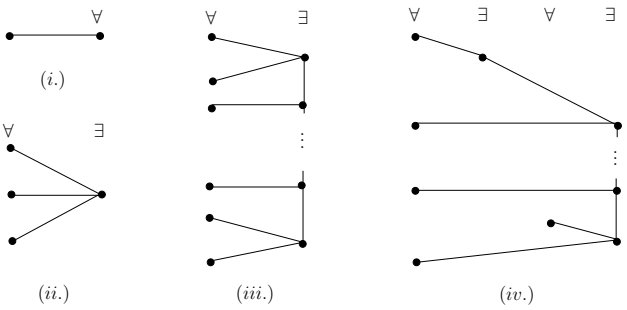


Fig. 2. Cases from proof of Proposition 2

4 The Reflexive Cycles

We will use similar edge gadgets to those of Figure 1 to prove NP-hardness of QCSP($\mathcal{C}_{1^m}$), for $m \geq 4$. If $m \geq 4$ is even, then the edge gadget $\mathcal{E}_m$ consists of m copies of $\mathcal{C}_{1^m}$ where each copy – with vertices $1, \ldots, m$, is connected to its successor by edges joining vertex k with vertices k and $k+1 \pmod m$. In the first of the copies, the vertex $\frac{m}{2}+1$ is labelled y and a reflexive path of length $\frac{m}{2}-1$ is added to the vertex labelled 1, which culminates in a vertex labelled x. The last of the copies of $\mathcal{C}_{1^m}$ retains the vertex labelling $1, \ldots, m$ – we consider the other vertices (except for x and y) to become unlabelled. Of course, $\mathcal{E}_4$ is already drawn in Figure 1. The left end of $\mathcal{E}_6$ is drawn in Figure 3 (to the right). If $m \geq 4$ is odd, then the edge gadget $\mathcal{E}_m$ is drawn in a similar manner, except vertex $\frac{m+1}{2}+1$ becomes y and the reflexive path of length $\frac{m-3}{2}$ that culminates in x at one end and at the other end a vertex that makes a triangle with vertices 1 and 2, respectively. The left end of $\mathcal{E}_5$ is drawn in Figure 3 (to the left). These gadgets are borrowed from [9] and have the property that when the right-hand cycle $\mathcal{C}_{1^m}$ is evaluated automorphically to itself then the rest of the cycles are

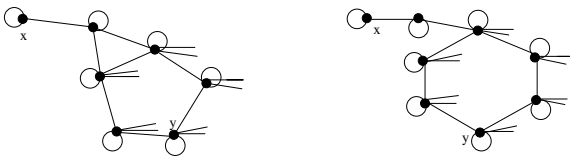


Fig. 3. Left ends of the edge gadgets in reductions from QCSP($\mathcal{K}_5$) to QCSP($\mathcal{C}_{1^5}$) and QCSP($\mathcal{K}_6$) to QCSP($\mathcal{C}_{1^6}$). In the former case, the full gadget contains a chain of five copies of $\mathcal{C}_{1^5}$; in the latter case it is a chain of six copies of $\mathcal{C}_{1^6}$.

also evaluated automorphically (but may twist $\frac{1}{m}$th each turn – this is why we have m copies; m is a minimum number, more copies would still work). Finally, in the left-hand cycle it can be seen that x can be evaluated anywhere except y.

Just as in Proposition 1, we want to try to force vertex-variables $v_1, \ldots, v_m$, corresponding to $1, \ldots, m$, to be evaluated (up to isomorphism) around $\mathcal{C}_{1^m}$.

Proposition 3. *QCSP($\mathcal{C}_{1^m}$), for any $m \geq 4$ is NP-hard.*

5 Cycles Whose Loops Induce a Path

We begin by recalling the result for QCSP($\mathcal{P}_{101}$) from [16], on which our proof for Propositions 5 and Corollary 1 will be based. In this proof we introduce the notions of *pattern* and $\forall$-*selector* that will recur in the sequel.

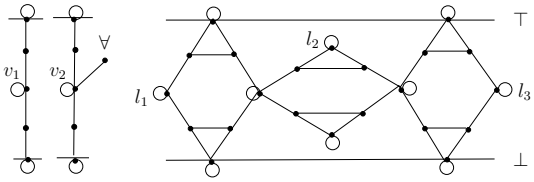


Fig. 4. Variable and clause gadgets in reduction to QCSP($\mathcal{P}_{101}$)

Proposition 4. *QCSP($\mathcal{P}_{101}$) is Pspace-complete.*

Proof. For hardness, we reduce from *quantified not-all-equal 3-satisfiability*, whose Pspace-completeness is well known [18], where we will ask for the extra condition that no clause has three universal variables (of course, any such instance would be trivially false). From an instance Φ of QNAESAT we will build an instance Ψ of QCSP($\mathcal{P}_{101}$) such that Φ is in QNAE3SAT iff Ψ in QCSP($\mathcal{P}_{101}$). We will consider the quantifier-free part of Ψ, itself a conjunction of atoms, as a graph, and use the language of homomorphisms.

Our reduction involves a graph $\mathcal{G}_\Phi$, whose vertices will give rise to the variables of Ψ, and whose edges will give rise to the atoms listed in the quantifier-free part of Ψ. Most of these variables will be existentially quantified, but a small handful will be universally quantified. $\mathcal{G}_\Phi$ consists of two reflexive paths, labelled $\top$ and $\bot$ which contain inbetween them gadgets for the clauses and variables of Φ. We will be able to assume that the paths $\top$ and $\bot$ are evaluated to vertices 1 and 3 in P_{101}, respectively (the two ends of P_{101}). The gadgets are drawn in Figure 4. The *pattern* is the path $\mathcal{P}_{101}$, that forms the edges of the diamonds in the clause gadgets as well as the tops and bottoms of the variable gadgets. The $\forall$-*selector* is the path $\mathcal{P}_{10}$, which travels between the universal variable node v_2 and the labelled vertex $\forall$. For this proof in full, please see the long version of this paper or the arxiv version of [16].

Proposition 5. *Let $\mathcal{C}_{0^d1^e}$ be a cycle in which $e > d+3$ (d odd) or $e > d+2$ (d even). Then QCSP($\mathcal{C}_{0^d1^e}$) is Pspace-complete.*

Proof. The reduction is similar to that employed for Proposition 4. We use the pattern $\mathcal{P}_{10^d1}$ and $\forall$-selector $\mathcal{P}_{0^{\lfloor \frac{e}{2} \rfloor}1}$. The key part to the reduction is how we get $v_\top$ and $v_\bot$ to evaluate suitably. Let $x_1, \ldots, y_1, \ldots, z_1, \ldots$ be variables not appearing in $\Psi'(v_\top, v_\bot)$ (cf. proof of Proposition 4). In the following, interpret $\lfloor \frac{e-d-5}{2} \rfloor$ to be 0, if $\frac{e-d-5}{2} < 0$. For d odd, set $\Psi := \forall x_1 \exists x_2, \ldots, x_{\frac{d+1}{2}}$

$$\exists x_{\frac{d+3}{2}}, \ldots, x_{\frac{d+3}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, \exists v_\top \forall y_1 \exists y_2, \ldots, y_{\frac{d+1}{2}} \exists y_{\frac{d+3}{2}}, \ldots, y_{\frac{d+3}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, \exists v_\bot$$
$$\exists z_1, \ldots, z_d$$
$$[x_1 \Rightarrow x_{\frac{d+3}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, v_\top] \wedge [y_1 \Rightarrow y_{\frac{d+3}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, v_\bot] \wedge [v_\top, z_1, \ldots, z_d, v_\bot] \wedge$$
$$\mathrm{Ref}(x_{\frac{d+3}{2}}, \ldots, x_{\frac{d+3}{2}+\lfloor \frac{e-d-5}{2} \rfloor}) \wedge \mathrm{Ref}(y_{\frac{d+3}{2}}, \ldots, y_{\frac{d+3}{2}+\lfloor \frac{e-d-5}{2} \rfloor}) \wedge \Psi'(v_\top, v_\bot)$$

For d even, set $\Psi := \forall x_1 \exists x_2, \ldots, x_{\frac{d}{2}}$

$$\exists x_{\frac{d+2}{2}}, \ldots, x_{\frac{d}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, \exists v_\top \forall y_1 \exists y_2, \ldots, y_{\frac{d}{2}} \exists y_{\frac{d+2}{2}}, \ldots, y_{\frac{d}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, \exists v_\bot$$
$$\exists z_1, \ldots, z_d$$
$$[x_1 \Rightarrow x_{\frac{d}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, v_\top] \wedge [y_1 \Rightarrow y_{\frac{d}{2}+\lfloor \frac{e-d-5}{2} \rfloor}, v_\bot] \wedge [v_\top, z_1, \ldots, z_d, v_\bot] \wedge$$
$$\mathrm{Ref}(x_{\frac{d+2}{2}}, \ldots, x_{\frac{d}{2}+\lfloor \frac{e-d-5}{2} \rfloor}) \wedge \mathrm{Ref}(y_{\frac{d+2}{2}}, \ldots, y_{\frac{d}{2}+\lfloor \frac{e-d-5}{2} \rfloor}) \wedge \Psi'(v_\top, v_\bot)$$

Note how the previous proof breaks down in boundary cases, for example on the cycle $\mathcal{C}_{0^2 1^3}$.

The following proofs make use of reductions from $\mathrm{Ret}(\mathcal{C})$, where $|C| = m$. It is ultimately intended that the variables $v_1, \ldots, v_m$ in the created instance map automorphically to the cycle. The cycle will be found when the universally quantified v_1 is mapped to a non-looped vertex at maximal distance from the looped vertices (sometimes this is unique, other times there are two). We then require that the universally quantified x_1 be mapped to a neighbour of v_1 at maximal distance from the loops (given v_1's evaluation, this will either be unique or there will be two). All other maps of v_1 and x_1 lead to degenerate cases.

Proposition 6. *Let $\mathcal{C}$ be an odd m-cycle which contains an induced $\mathcal{P}_{11100}$ (or is $\mathcal{C}_{0^2 1^3}$). Then QCSP($\mathcal{C}$) is NP-hard.*

Proposition 7. *Let $\mathcal{C}$ be an even m-cycle which contains an induced $\mathcal{P}_{11100}$. Then QCSP($\mathcal{C}$) is NP-hard.*

We note that the previous two propositions do not quite use the same techniques as one another. All cases of Proposition 5 involving more than one non-loop are weakly subsumed by Propositions 6 and 7 in the sense that Pspace-completeness only becomes NP-hardness.

It is interesting to note that the Proposition 7 breaks down on even cycles with two consecutive loops only. It is no longer possible to ensure to encircle the cycle. For these cases we will need yet another specialised construction.

Proposition 8. *For $m \geq 6$, let $\mathcal{C}$ be an even m-cycle which contains only two consecutive loops. Then QCSP($\mathcal{C}$) is NP-hard.*

6 Cycles in Which the Loops Induce a Disconnected Graph

Let $D_{\mathcal{C}} := \{\{\lceil \frac{d_1}{2} \rceil, \ldots, \lceil \frac{d_m}{2} \rceil\}\}$ be the multiset (of two or more elements), where $d_1, \ldots, d_m$ are the maximal non-looped induced sections (paths) of a cycle $\mathcal{C}$ in which the loops induce a disconnected graph. E.g. a single non-loop between two loops contributes a value $\lceil 1/2 \rceil = 1$ to $D_{\mathcal{C}}$. We need to split into three cases.

Proposition 9. *Let $\mathcal{C}$ be a partially reflexive m-cycle in which the loops induce a disconnected graph. If $D_{\mathcal{C}}$ contains a unique maximal element $\lceil \frac{d}{2} \rceil$, then QCSP($\mathcal{C}$) is Pspace-complete.*

Proposition 10. *Let $\mathcal{C}$ be a partially reflexive m-cycle in which the loops induce a disconnected graph. If $D_{\mathcal{C}}$ contains only one value, then QCSP($\mathcal{C}$) is Pspace-complete.*

Corollary 1. *Let $\mathcal{C}$ be a partially reflexive m-cycle in which the loops induce a disconnected graph. Then QCSP($\mathcal{C}$) is Pspace-complete.*

7 Classification

Theorem 1. *Let $m = d + e \geq 5$. Then QCSP($\mathcal{C}_{0^d1^e}$) is in NL if I.) m is odd and $e = 1$ or 2, or II.) m is even and $e = 0$ or 1. Otherwise, QCSP($\mathcal{C}_{0^d1^e}$) is NP-hard.*

Proof. Pspace-hardness for irreflexive odd cycles is well-known (see [17]). Hardness for cycles with disconnected loops follows from Corollary 1. Otherwise, for most cycles hardness follows from Propositions 6 and 7. For reflexive cycles see Proposition 3 and for cycles with a single non-loop see Proposition 5. Finally, the outstanding cases of even cycles with two loops are taken care of in Proposition 8.

Now we address the NL cases. For even cycles with no loops, we are equivalent to QCSP($\mathcal{K}_2$) (in NL – see [17]). For even cycles $\mathcal{C}_{0^{2i+1}1}$, QCSP($\mathcal{C}_{0^{2i+1}1}$) is equivalent to QCSP($\mathcal{P}_{0^{i+1}1}$) (in NL – see [16]). This is because there are surjective homomorphisms from both $(\mathcal{P}_{0^{i+1}1})^2$ to $\mathcal{C}_{0^{2i+1}1}$ and $\mathcal{C}_{0^{2i+1}1}$ to $\mathcal{P}_{0^{i+1}1}$ (see [7]). For odd cycles, there are surjective homomorphisms from $(\mathcal{P}_{0^i1})^2$ to $\mathcal{C}_{0^{2i}1}$ and from $(\mathcal{P}_{0^i1})^2$ to $\mathcal{C}_{0^{2i-1}11}$. Thus, QCSP($\mathcal{C}_{0^{2i}1}$) is equivalent to both QCSP($\mathcal{C}_{0^{2i-1}11}$) and QCSP($\mathcal{P}_{0^i1}$) and the result follows from [16].

Theorem 2. *For $\mathcal{C}$ a partially reflexive cycle, either QCSP($\mathcal{C}$) is in NL or it is NP-hard.*

Proof. Owing to Theorem 1, it remains only to consider partially reflexive cycles on four or fewer vertices.

Firstly, we consider NL-membership. Each of $\mathcal{C} := \mathcal{C}_{001}, \mathcal{C}_{011}, \mathcal{C}_{111}, \mathcal{C}_{0000}, \mathcal{C}_{0001}$ and $\mathcal{C}_{0011}$ admits a majority polymorphism. It follows form [5] that QCSP($\mathcal{C}$) reduces to the verification of a polynomial number of instances of CSP($\mathcal{C}^c$), each of which is in NL by [8]. Finally, the case $\mathcal{C}_{0111}$ is taken care of in Proposition 2.

For hardness, it is well-known that *quantified* 3*-colouring* QCSP($\mathcal{K}_3$) is Pspace-complete [18]. And the like result for $\mathcal{C}_{1111}$ appears as Proposition 1.

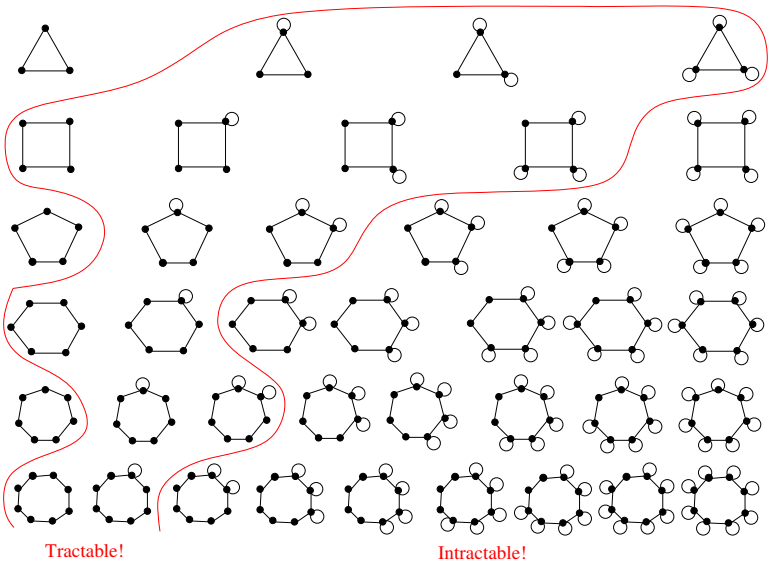


Fig. 5. The wavy line of tractability

8 Conclusion

We have given a systematic classification for the QCSP of partially reflexive cycles. Many of the tractable cases can be explained by the notion of Q-core – a minimal equivalent QCSP template (see [14]) – and this is done implicitly in Theorem 1. All NP-hard cases we have seen in this paper have templates that are already Q-cores, with the sole exception of $\mathcal{C}_{0101}$, whose Q-core is $\mathcal{P}_{101}$. By contrast, all of the tractable cases are not Q-cores, except $\mathcal{C}_{0011}$ and $\mathcal{C}_{0111}$.

Since finding an algorithm for QCSP($\mathcal{C}_{0111}$) we became aware of a polymorphism enjoyed by this structure. $\mathcal{C}_{0111}$ has a ternary polymorphism f so that there is $c \in C_{0111}$ such that each of the three binary functions $f(c, u, v)$, $f(u, c, v)$ and $f(u, v, c)$ is surjective. The code from Figure 6 will verify such a polymorphism and is intended to run on the excellent program of Miklós Maróti[1]. The naming of the vertices has been altered according to the bijection $\begin{smallmatrix}1&3\\2&0\\3&1\\4&4\end{smallmatrix}$ (vertices are numbered from 0 for the computer). If follows from [5] that $\mathcal{C}_{0111}$ is 2-collapsible, and hence QCSP($\mathcal{C}_{0111}$) may be placed in NL by other means.

```
arity 3; size 4; idempotent; preserves 01 10 12 21 23 32 30 03 00 11 22;
value 302=0; value 320=2; value 311=1; value 123=1; value 223=2;
value 003=0; value 032=0; value 030=1; value 230=2
```

Fig. 6. Code for Maróti's program (semicolons indicate new line)

[1] See: `http://www.math.u-szeged.hu~maroti/applets/GraphPoly.html`

We would like to improve the lower bound from NP-hardness to Pspace-hardness in the cases of Propositions 3, 6, 7 and 8. This might be quite messy in the last three cases, involving careful consideration of the hardness proof for the retraction problem. For the reflexive cycles, though, it just requires more careful analysis of the degenerate cases. This is because we may only have a homomorphic image under f of the cycle for $v_1, \ldots, v_m$, but the universal variables may be evaluated outside of the image of $f(\{v_1, \ldots, v_m\})$. We will require something of the following form ($\mathcal{E}_m$ is defined at the beginning of Section 4).

Conjecture 1. Let f be a function from $\{1, \ldots, m\}$ in $\mathcal{E}_m$ to $\mathcal{C}_{1^m}$ that is a non-surjective homomorphism. Let f_{ij}, for some $i, j \in \{1, \ldots, m\}$, be the partial function that extends f from $\{1, \ldots, m, x, y\}$ in $\mathcal{E}_m$ to $\mathcal{C}_{1^m}$, by mapping $x \mapsto i$ and $y \mapsto j$. Then f_{ij} can be extended to a homomorphism from $\mathcal{E}_m$ to $\mathcal{C}_{1^m}$.

The proof of this seems to be rather technical. For those who doubt it, let us remind ourselves that the length of the chain in $\mathcal{E}_m$ may be any fixed function of m, and the reduction of Propositions 1 and 3 will still work. The conjecture is surely easier to prove if we make the chains much longer (say exponential in m).

Finally, we conjecture that none of the NL cases are NL-hard, and that most likely our dichotomy can be perfected to L/ Pspace-complete.

Acknowledgements. The authors thank the referees and are grateful to St. Catherine.

References

1. Barto, L., Kozik, M., Niven, T.: The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). SIAM Journal on Computing 38(5), 1782–1802 (2009)
2. Börner, F., Bulatov, A.A., Chen, H., Jeavons, P., Krokhin, A.A.: The complexity of constraint satisfaction games and qcsp. Inf. Comput. 207(9), 923–944 (2009)
3. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. J. ACM 53(1), 66–120 (2006)
4. Bulatov, A., Krokhin, A., Jeavons, P.G.: Classifying the complexity of constraints using finite algebras. SIAM Journal on Computing 34, 720–742 (2005)
5. Chen, H.: The complexity of quantified constraint satisfaction: Collapsibility, sink algebras, and the three-element case. SIAM J. Comput. 37(5), 1674–1701 (2008)
6. Chen, H. Meditations on quantified constraint satisfaction. CoRR abs/1201.6306 (2012), Appeared in Festschrift for Dexter Kozen 60th.
7. Chen, H., Madelaine, F., Martin, B.: Quantified constraints and containment problems. In: 23rd Annual IEEE Symposium on Logic in Computer Science, pp. 317–328 (2008)
8. Dalmau, V., Krokhin, A.A.: Majority constraints have bounded pathwidth duality. Eur. J. Comb. 29(4), 821–837 (2008)
9. Feder, T., Hell, P.: List homomorphisms to reflexive graphs. J. Comb. Theory, Ser. B 72(2), 236–250 (1998)

10. Feder, T., Hell, P., Jonsson, P., Krokhin, A.A., Nordh, G.: Retractions to pseudoforests. SIAM J. Discrete Math. 24(1), 101–112 (2010)
11. Feder, T., Vardi, M.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. SIAM Journal on Computing 28, 57–104 (1999)
12. Hell, P., Nešetřil, J.: On the complexity of H-coloring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
13. Madelaine, F., Martin, B.: A tetrachotomy for positive equality-free logic. In: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011 (2011)
14. Madelaine, F., Martin, B.: Containment, equivalence and coreness from CSP to QCSP and beyond. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 480–495. Springer, Heidelberg (2012)
15. Madelaine, F., Martin, B., Stacho, J.: Constraint satisfaction with counting quantifiers. In: Hirsch, E.A., Karhumäki, J., Lepistö, A., Prilutskii, M. (eds.) CSR 2012. LNCS, vol. 7353, pp. 253–265. Springer, Heidelberg (2012)
16. Martin, B.: QCSP on partially reflexive forests. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 546–560. Springer, Heidelberg (2011)
17. Martin, B., Madelaine, F.: Towards a trichotomy for quantified H-coloring. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 342–352. Springer, Heidelberg (2006)
18. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
19. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4), 1–24 (2008)
20. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of STOC 1978, pp. 216–226 (1978)

Quantum Alternation*

Abuzer Yakaryılmaz

University of Latvia, Faculty of Computing, Raina bulv. 19, Rīga, LV-1586, Latvia
abuzer@lu.lv

Abstract. We introduce the concept of quantum alternation as a generalization of quantum nondeterminism. We define the first quantum alternating Turing machine (qATM) by augmenting alternating Turing machine (ATM) with a fixed-size quantum register. We focus on space-bounded computation, and obtain the following surprising result: One-way qATMs with constant-space (one-way alternating quantum finite automata (1AQFAs)) are Turing-equivalent. Then, we introduce strong version of qATM: The qATM that must halt in every computation path. We show that strong qATMs (similar to private ATMs) can simulate deterministic space with exponentially less space. This leads to shifting the deterministic space hierarchy exactly by one level. We also focus on realtime versions of 1AQFAs (rtAQFAs) and obtain many interesting results: (i) any language recognized by a rtAQFA is in quadratic deterministic space, (ii) two-alternation is better than one-alternation, (iii) two-alternation is sufficient to recognize a NP-complete language and so any language in NP can be recognized by a poly-time log-space qATM with two alternations, (iv) three-alternation is sufficient to recognize a language that is complete for the second level of the polynomial hierarchy and so any language in the second level of the polynomial hierarchy can be recognized by a poly-time log-space qATM with three alternations.

1 Introduction

Anne Condon, in her famous PhD thesis [6], introduced a general computational model, i.e. *probabilistic game automaton*, that unifies many important computational models and concepts: *Alternation* of Chandra, Kozen, and Stockmeyer [4], *private alternation* of Reif [15], *Arthur-Merlin games* of Babai [2], *interactive proof systems* of Goldwasser, Micali, and Rackoff [8], *game against nature* of Papadimitriou [12], etc. In this framework, Arthur-Merlin (AM) proof systems and alternation are the "weakest" ones since both are the games with *complete information*. Recently, Yakaryılmaz [18] showed that if the game automaton is augmented with a fixed-size quantum register, then the obtained AM proof systems, called qAM, becomes Turing equivalent by using constant space and, in the case of strong soundness, i.e. the nonmembers are always rejected with high probability, they seem more powerful than private-coin interactive proof systems. Parallel to this work, we examine alternation counterpart of qAM in this paper.

* This work was partially supported by FP7 FET-Open project QCS. A preliminary report on some contents of this paper was [19].

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 334–346, 2013.

Alternation was introduced by Chandra and Stockmeyer [5] and Kozen [9] as a generalization of nondeterminism. It was shown that alternation shifts the deterministic hierarchy $\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{EXPSPACE}$ by exactly one level [4]. On the other hand, the class of languages recognized by alternating finite automata is still the regular languages [4]. Reif [15] introduced private alternation by assuming that universal player can hide some information from the existential player, and showed that private alternation shifts the deterministic space hierarchy $\mathsf{L} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE}$ by exactly one level.

We introduce quantum alternation for the first time, namely q-alternation. The main result is that one-way q-alternating finite automata can recognize any Turing-recognizable language. In the classical case, the class of languages recognized by any space-bounded (private) ATMs is a proper subset of decidable languages [15]. Since q-alternating machines may not halt the computation in every path, we also introduce the strong version of q-alternation by forbidding infinite computations. Then, we show that strong q-alternation, similar to private alternation, shifts the deterministic space hierarchy by exactly one level.

Since 1AQFAs are too powerful, we also focus on realtime versions of one-way AQFAs (rtAQFA) and then obtain many interesting results: (i) any language recognized by a rtAQFA is in quadratic deterministic space, (ii) two-alternation is better than one-alternation, (iii) two-alternation is sufficient to recognize a NP-complete language and so any language in NP can be recognized by a poly-time log-space qATM with two alternations, (iv) there-alternation is sufficient to recognize a language which is complete for second level polynomial hierarchy and so any language in the second level of polynomial hierarchy can be recognized by a poly-time log-space qATM with three alternations.

We refer the reader to [1] and [11] for some references on computational complexity and quantum computation, respectively. We refer the reader to [18] for a related work. Some of our proofs are based on the ones given in [18]. We will define our model based on Turing machine and we assume the reader familiar with the basics of alternating Turing machines (ATM). After giving the formal definitions of models in Section 2, we present all the results in Section 3.

2 Definition of q-Alternation

As mentioned earlier, alternation and AM systems are games with *complete information*. Moreover, they could be obviously related to each other, e.g. alternation can be "inherited" from AM systems as follows: The verifier is replaced by a universal player and all provers are represented by an existential player. (We refer the reader to Condon [6] for the technical details.)

In this section, we introduce the notion of *quantum alternation for the first time*. We define quantum alternation similar to qAM system [18]: Its quantum part is only a fixed-size quantum register. But, different from qAM systems, this register can be accessible by both players (the universal and existential states). We call the model *q-alternation* due to its "very" limited quantum part, and give the definition based on ATM: *q-alternating Turing Machine* (qATM).

The most general quantum operator is a superoperator, which generalizes stochastic and unitary operators and also includes measurement. Formally, a superoperator $\mathcal{E}$ is composed by a finite number of operation elements, $\mathcal{E} = \{E_1, \ldots, E_k\}$, satisfying

$$\sum_{i=1}^{k} E_i^\dagger E_i = I, \tag{1}$$

where $k \in \mathbb{Z}^+$ and the indices are the measurement outcomes. When a superoperator, say $\mathcal{E}$, is applied to the quantum register in state $|\psi\rangle$, i.e. $\mathcal{E}(|\psi\rangle)$, we obtain the measurement outcome i with probability $p_i = \langle\widetilde{\psi}_i|\widetilde{\psi}_i\rangle$, where $|\widetilde{\psi}_i\rangle$, *the unconditional state vector*, is calculated as $|\widetilde{\psi}_i\rangle = E_i|\psi\rangle$ and $1 \leq i \leq k$. (Note that using unconditional state vector simplifies calculations in many cases.) If the outcome i is observed ($p_i > 0$), the new state of the system is obtained by normalizing $|\widetilde{\psi}_i\rangle$, which is $|\psi_i\rangle = \frac{|\widetilde{\psi}_i\rangle}{\sqrt{p_i}}$. Moreover, the quantum register can be initialized to a predefined quantum state by a special operator called initialize operator. In this paper, the entries of quantum operators are defined by rational numbers.

Fig. 1. The details of superoperators

A qATM is an ATM augmented with a fixed-size quantum register, based on which existential and universal branches are determined. Any configuration of a qATM can be represented by a pair $(c, |\psi\rangle)$, where c represent the classical configuration of the machine and $|\psi\rangle$ is the state of quantum register. Let $\{c_1, \ldots, c_{k_c}\}$ be the transitions determined by the classical transition function of the machine with respect to the classical state and the symbol(s) under the tape head(s) in configuration c, where k_c is the total number branches. In each step, the machine implements one quantum and one classical transitions as described below. The machine applies a superoperator (see Figure 1) determined by the classical state and the symbol(s) under the tape head(s) in configuration c, $\mathcal{E}_c = \{E_{c,1}, \ldots, E_{c,k_c}\}$, to the register, i.e.

$$|\psi_i\rangle = \frac{|\widetilde{\psi}_i\rangle}{\sqrt{p_i}} \text{ if } p_i \neq 0, \text{ where } p_i = \langle\widetilde{\psi}_i|\widetilde{\psi}_i\rangle,\ |\widetilde{\psi}_i\rangle = E_{c,i}|\psi\rangle, \text{ and } 1 \leq i \leq k_c,$$

and then the following transition(s) is (are) implemented:

$$(c, |\psi\rangle) \rightarrow \{(c_i, |\psi_i\rangle) \mid p_i > 0, 1 \leq i \leq k_c\}.$$

Note that, the transitions having zero probability ($p_i = 0$) are not implemented. (In terms of two-person games [6], we can say that *a player who makes the universal or existential choices uses a quantum register to make its choices, therefore any choice with zero probability can never be a part of this player's strategy. We can also formalize this issue by assuming the players having capability of postselection on the outcomes of the computation. The details will be given in the full version this paper.*) The computation starts when the machine is in the initial classical configuration and the initial quantum state. The computation is terminated with the decision of "accepting" ("rejection") if the machine enters an

accepting (rejecting) configuration. The accepting criteria of qATM is the same as ATM. For any given input, we have a computation tree representing all moves of the machine. The input is accepted if and only if there exists a *finite accepting subtree*[1] for a nondeterministic strategy in this computation tree. If we remove the work tape of a qATM, and restrict the input head to one-way, we obtain a one-way q-alternating finite automaton (1QAFA). As a further restriction, if the tape head is allowed to be stationary on the same tape square at most a fixed number of steps, then we obtain a realtime q-alternating finite automaton (rtQAFA). We can also limit the number of alternation between universal and existential states. More specifically, Σ_k (resp., Π_k) represents that there can be at most $k-1$ alternation starting with existential (resp., universal) one, where $k \geq 1$. In any model (and also any complexity class), we can replace the "A" that stands for alternation with Σ_k or Π_k to represent such a restriction. rtΣ_1QFA is also called realtime nondeterministic quantum finite automaton (rtNQFA). Note that rtNQFAs were originally defined as realtime QFAs with one-sided unbounded error setting [20]. But, both models can simulate each other.

3 Main Results

We present our results in five subsections. We begin with a basic algorithm in order to give some intuitions to the reader. As can be seen from this algorithm, due to negative transitions, some predefined branches can disappear during the computation. This phenomena was shown to increase (or seemingly increase) in the computational power of nondeterministic quantum models [7,17,21,20]. In this paper, we show that this phenomena actually can lead to significantly increase in the computational power of quantum models if universal branching is allowed as well (even the quantum resource is very limited).

3.1 A rtΣ_2QFA Example

We give a rtΣ_2QFA for nonstochastic language $\texttt{NH} = \{a^x b a^{y_1} b a^{y_2} b \cdots a^{y_t} b \mid x, t, y_1, \ldots, y_t \in \mathbb{Z}^+ \text{ and } \exists k \ (1 \leq k \leq t), x = \sum_{i=1}^{k} y_i\}$ [10], based on which we will also show a separation result soon.

Theorem 1. NH *can be recognized by a rtΣ_2QFA.*

Proof. Let $\mathcal{A}$ be the rtΣ_2QFA. We assume that the input is of the form

$$a^x b a^{y_1} b a^{y_2} b \cdots a^{y_t} b \mid x, t, y_1, \ldots, y_t \in \mathbb{Z}^+ \text{ for some } t > 1. \tag{2}$$

It is rejected deterministically in all branches, otherwise. The quantum register is set to $(1 \ \ 0)^T$ at the beginning. $\mathcal{A}$ makes an existential move for each a but

[1] Each leaf of an accepting subtree is an accepting leaf, a leaf in which the decision of "accepting" is given.

uses different superoperators before and after the first b. Before reading the first b, $\mathcal{A}$ applies the following superoperator for each a:

$$\left\{E_c = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, E_{r_1} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix}, E_{r_2} = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}\right\}.$$

After the first b, $\mathcal{A}$ applies the following superoperator for each a:

$$\left\{E_c = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, E_{r_1} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, E_{r_2} = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}\right\}.$$

In both cases, the computation continues whenever outcome "c" is observed and it is halted with decision of "rejection", otherwise. $\mathcal{A}$ does not apply any operator to the quantum register when reading the first b. Thus, after reading the first b, the state of the quantum register becomes $\left(\frac{1}{2}\right)^x \begin{pmatrix} 1 \\ x \end{pmatrix}$ in the only non-halting branch of $\mathcal{A}$. This quantum state changes for each new a as follows:

$$\left(\frac{1}{2}\right)^i \begin{pmatrix} 1 \\ j \end{pmatrix} \rightarrow \left(\frac{1}{2}\right)^{i+1} \begin{pmatrix} 1 \\ j-1 \end{pmatrix}, \text{ where } i \in \mathbf{Z}^+ \text{ and } j \in \mathbf{Z}.$$

On each b except the first one, $\mathcal{A}$ firstly makes an existential move based on the following superoperator

$$\left\{E_c = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, E_{r_1} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, E_{r_2} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, E_u = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right\}$$

such that the computation continues as usual if the outcome "c" is observed, it is halted with the decision of "rejection" if the outcome "r_1" or "r_2" is observed, and a universal transition is implemented based on the following superoperator if the outcome "u" is observed:

$$\left\{E_a = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, E_r = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\right\}$$

such that the computation is halted with the decision of "accepting" ("rejection") if the outcome "a" ("r") is observed. (*Note that the branch leading to "accepting" continues until to the right end-marker and the input is accepted only if the input is of the form as given in Equation (2).*) If the second component of the quantum state is zero, then this universal transition has only one child which leads to "accepting". The other branch is not implemented since the resulting quantum state is zero vector. Therefore, if the number of a's before the first b equals to the number of other a's until the current b, this universal transition returns "accept", and it returns "reject", otherwise. That is, for the members, one of the existential path ends with the decision of "accepting" and so the input is accepted by $\mathcal{A}$; and, for the non-members, each existential path before the right end-marker ends with the decision of "rejection", and so the input can be rejected by $\mathcal{A}$ by also giving the decision of "rejection" in the existential path surviving until the right end-marker. □

3.2 1AQFAs Are Turing-Equivalent

In [18], the simulation of a given DTM by a constant-space qAM proof system was given. We give our result based on this simulation after making some certain modifications. Therefore, we first review this simulation:

Let $\mathcal{D}$ be the simulated DTM and x be the given input. In an infinite loop, the verifier requests the computation of $\mathcal{D}$ on x from the prover, i.e. $w = c_1\$\$c_2\$\$\cdots$, where c_1 is the initial configuration and c_{i+1} is the legal successor of c_i for $i \geq 1$. But, against the cheating provers, the verifier checks the correctness of w. The verifier can check the first configuration by reading the input once. Let $\mathsf{next}(c_i)$ be the legal successor of c_i. To check whether c_{i+1} is the legal successor of c_i, the verifier encodes $\mathsf{next}(c_i)$ and c_{i+1} into the amplitudes of two states on the quantum register, and then compares them by subtracting these two amplitudes and rejects the input with the result amplitude. (Note that the rejecting probability is the square of the result amplitude.) So, if $\mathsf{next}(c_i) \neq c_{i+1}$, the input is rejected with some probability, and it is rejected with zero probability, otherwise. If w leads to an accepting (rejecting) computation, the verifier also accepts (rejects) the input at the end. Each encoding operation is implemented with some superoperators having more than one outcome such that the encoding is succeed with some probabilities in one branch and a new round is initiated (by returning to the beginning of the protocol) with the remaining probabilities in all the other branches. Therefore, after processing a symbol from w, the protocol continues only with some small probabilities. The analysis of a single round is as follows:

- For the members, after communicating with an honest prover, the verifier accepts the input with a non-zero probability and restart the protocol with the remaining probability; and,
- For the non-members, (i) if the prover lies about w, then the verifier rejects the input with some probability by detecting a defect on w, (ii) if the prover is honest and w is infinite, the verifier never makes a decision and a new round is initiated with probability 1, and (iii) the verifier rejects the input with some probability, otherwise.

Theorem 2. *Any Turing-recognizable language can be recognized by a 1AQFA.*

Proof. Let $\mathtt{L}$ be a Turing-recognizable language, $\mathcal{D}$ be a DTM recognizing $\mathtt{L}$, and (P, V) be the constant-space qAM system for $\mathtt{L}$ as described above, where P is an honest prover and V is the verifier. Since the input head is used only at the beginning of each round to check whether the prover sends the valid initial configuration, V never needs to move its input head to the left in a single round. We define a new one-way finite state verifier V' based on V. The only difference between V and V' is that when the outcome originally corresponding to "restarting a new round" is observed, V' terminates the computation with decision of "accepting" instead of initiating a new round. Thus, V' executes only a single round (and so V' never needs to move its input head to the left).

The analysis of (P, V') is as follows. Let x be an input string. If $x \in \mathtt{L}$, the computation is terminated in every branch, and the input is accepted by V' with probability 1 by the help of P. If $x \notin \mathtt{L}$, there are two cases depending on the prover (P^*) strategy and also the behaviour of $\mathcal{D}$ on x: (1) the computation may run forever in some branches, and, (2) the computation is terminated in every branch and the input is rejected by V' with some non-zero probability.

Now, based on V', we can easily construct a 1AQFA $\mathcal{A}$ recognizing $\mathtt{L}$. The universal states of $\mathcal{A}$ simulate V' and the existential states of $\mathcal{A}$ simulate the communications with all possible provers. If $x \in \mathtt{L}$, there exists a finite *accepting* subtree whose existential moves correspond to the communication with P. If $x \notin \mathtt{L}$, the finite subtree(s) can only be possible in the second case given above. Obviously, at least one leaf of such a subtree ends with the decision of rejection. Therefore, there is no finite accepting subtree for the nonmembers. □

Corollary 1. *For any space bound $s(n)$, 1AQFAs are strictly more powerful than any $s(n)$ space-bounded private ATM.*

Proof. This follows from Theorem 2 and the fact that the class of languages recognized by any space-bounded private ATM is a proper subset of decidable languages [15]. □

Theorem 3. *Any language recognized by a qATM is Turing-recognizable.*

Proof. Since the entries of any operation element are rational numbers, any space-bounded qATM can be simulated by a ATM (and so by a DTM) in a straightforward way. □

Corollary 2. *1QFAs are Turing-equivalent.*

3.3 Strong q-Alternation

Due to Corollary 2, we cannot mention a space hierarchy for q-alternation. Moreover, the computation of qATMs may not halt in some paths. Therefore, we define a restricted version of q-alternation: *strong q-alternation*. Any q-alternating machine is a strong one if it halts on every computational path. We will denote the related space-bounded complexity classes by $\mathsf{qASPACE}(\cdot)$: $\mathsf{qASPACE}(\mathsf{s(n)})$ is the class of languages recognized by $s(n)$ space-bounded strong qATMs. qAL and $\mathsf{qAPSPACE}$ are strong q-alternating counterparts of AL and $\mathsf{APSPACE}$, respectively. We show that strong q-alternation (similar to private alternation [15]) shifts the deterministic space hierarchy by exactly one level.

Theorem 4. *For any space-constructible $s(n) \in \Omega(\log(s(n)))$,*

$$\mathsf{DSPACE}(2^{\mathsf{O(s(n))}}) = \mathsf{qASPACE}(\mathsf{s(n)}).$$

Proof. From Lemma 3 (see below) and Savitch's theorem [16], we can deduce that

$$\mathsf{qASPACE}(\mathsf{s}(\mathsf{n})) \subseteq \mathsf{NSPACE}(2^{\mathsf{O}(\mathsf{s}(\mathsf{n}))}) \subseteq \mathsf{DSPACE}(2^{\mathsf{O}(\mathsf{s}(\mathsf{n}))}).$$

From Lemma 4 (see below) and Chandra and Stockmeyer [4], we can deduce that

$$\mathsf{DSPACE}(2^{\mathsf{O}(\mathsf{s}(\mathsf{n}))}) \subseteq \mathsf{ATIME}(2^{\mathsf{O}(\mathsf{s}(\mathsf{n}))}) \subseteq \mathsf{qASPACE}(\mathsf{s}(\mathsf{n})).$$

□

Corollary 3. $\mathsf{L} \subsetneq \mathsf{qAL} = \mathsf{PSPACE} \subsetneq \mathsf{qAPSPACE} = \mathsf{EXPSPACE}$.

In the remaining part, we give some technical lemmas used in the proof of Theorem 4. We begin with showing an upper bound on the running time of a space-bounded strong qATM.

Lemma 1. *For any $s(n) \in \Omega(\log(n))$, the running time of a $s(n)$ space-bounded strong qATM can be at most $2^{O(s(n))}$.*

Proof. See Appendix D of [19]. □

Lemma 2. *Any $t(n)$ time-bounded qATM can be exactly simulated by a $O(t^2(n))$ time-bounded ATM.*

Proof. A qATM can be exactly simulated by an ATM by tracing the content of the quantum register. The simulation of the classical part of a $t(n)$ time-bounded given qATM can be implemented in $O(t(n))$ steps. After each quantum operation, the precisions of amplitudes on the quantum register increase by a constant. And so, the time needed to trace it can also increase by a constant. Therefore, the ATM can use $O(1) + O(2) + \cdots + O(t(n)) = O(t^2(n))$ steps to trace the state of the quantum register. Therefore, the overall runtime of the ATM is $O(t^2(n))$. □

Thus, due to Lemma 1, we can also provide a deterministic space simulation of a given space-bounded strong qATM by exponential blow-up.

Lemma 3. *For any space-constructible $s(n) \in \Omega(\log(n))$, a given $s(n)$ space-bounded strong qATM can be simulated by a $2^{O(s(n))}$ space-bounded DTM.*

Proof. Due to Lemma 1, the length of any computation path of a given $s(n)$ space-bounded strong qATM can be at most $2^{c_1 s(|x|)}$ for a suitable constant c_1. Then, due to Lemma 2, it can be simulated by a $2^{O(s(n))}$ time-bounded ATM which can be simulated by a $2^{O(s(n))}$ space-bounded DTM [5]. □

Now, we show that the bound given in Lemma 3 is actually tight.

Lemma 4. *For any log-space constructible $t(n) \in \Omega(n)$, if a language is recognized by an ATM running in time $t(n)$, then there exists a $O(\log(t(n)))$ space-bounded strong qATM recognizing the same language.*

Proof. By modifying the proof of Theorem 2, we can show that 1AQFAs can also simulate the computation of an ATM on a given input. The computation of an ATM on a given input can be represented by a tree. A 1AQFA can simulate each path of such a tree as given in the proof of Theorem 2 by providing the related existential and universal choices on the tree. The 1AQFA can also relate these paths conveniently to each other.

The length of any configuration of a $t(n)$ time-bounded ATM can be at most $O(t(n))$, and so the length of the computation of a path in the tree can be at most $O(t^2(n))$. So, if the 1AQFA can have $O(log(n))$ space, then it can simulate the computation tree of a given $t(n)$ time-bounded ATM by not allowing the length of any path to exceed $O(t^2(n))$. Thus, we can obtain a $O(\log(t(n)))$ space-bounded *strong* qATM simulating the $t(n)$ time-bounded ATM. □

3.4 QFA Counterpart of Polynomial Hierarchy

We denote the class of languages recognized by rtAQFA as AQAL (alternating quantum automaton languages). We begin with an upper bound for AQAL.

Theorem 5. $\mathsf{AQAL} \subseteq \mathsf{ATIME}(\mathsf{n}^2) = \mathsf{DSPACE}(\mathsf{n}^2)$.

Proof. We can follow the results due to Lemma 2 since any rtAQFA is a linear-time qATM. □

We introduce a QFA counterpart of the polynomial hierarchy, $\mathsf{PH} = \cup_{i \geq 0} \mathsf{\Sigma_k P}$: For any $k \geq 0$, $\mathsf{\Sigma_k QAL}$ and $\mathsf{\Pi_k QAL}$ are the class of languages recognized by $\Sigma_k QFA$s and $\Pi_k QFA$s, respectively.

$\mathsf{\Sigma_1 QAL}$ has already been defined as NQAL (nondeterministic quantum automaton languages) and it was shown to be equivalent to exclusive stochastic languages ($\mathsf{S^{\neq}}$) [20], which is a proper superset of regular languages (REG) [14]. Moreover, rtΣ_0QFAs and rtΠ_0QFAs are one-way deterministic finite automata. Then we have the following relations for the initial levels:

- $\mathsf{\Sigma_0 QAL} = \mathsf{\Pi_0 QAL} = \mathsf{REG}$
- $\mathsf{\Sigma_1 QAL} \neq \mathsf{\Pi_1 QAL}$.

Since $\mathtt{NH}$ is a nonstochastic language [10] and $\mathsf{S^{\neq}(\Sigma_1 QAL)}$ is a proper subset of stochastic languages [14], we can say that $\mathtt{NH} \notin \mathsf{\Sigma_1 QAL}$ and $\overline{\mathtt{NH}} \notin \mathsf{\Pi_1 QAL}$. Therefore, we can separate the second level from the first level by Theorem 1.

Corollary 4. $\mathsf{\Sigma_1 QAL} \subsetneq \mathsf{\Sigma_2 QAL}$ *and* $\mathsf{\Pi_1 QAL} \subsetneq \mathsf{\Pi_2 QAL}$.

We continue with a rtΣ_2QFA algorithm for the well-known NP-complete language SUBSETSUM, which is the collection of all strings of the form $S\$a_1\$\cdots\$a_n\$$ such that S and the a_i's are numbers in binary ($1 \leq i \leq n$), and there exists a set $I \subseteq \{1,\ldots,n\}$ satisfying $\sum_{i\in I} a_i = S$, where $n > 0$.

Theorem 6. SUBSETSUM $\in \Sigma_2$QAL.

Proof. For SUBSETSUM, Yakaryılmaz presented a constant-space qAM system in [18]. By modifying this protocol, we can also give a rtΣ_2QFAs, say $\mathcal{A}$. We can assume the input to be of the form $S\#a_1\#\cdots\#a_n\#$, where S and the a_i's are numbers in binary ($1 \leq i \leq n$), and $n > 0$. Otherwise, the input is rejected deterministically in all branches.

The initial quantum state is $(1\ 0\ 0)^T$. $\mathcal{A}$ encodes S into the amplitude of a state on the quantum register. We can use the following two superoperators called $\mathcal{E}_0$ and $\mathcal{E}_1$ for this purpose where $\mathcal{E}_j$ is applied when $j \in \{0,1\}$ is read and the computation continues (is terminated with the decision of "rejection") when the outcome "c" ("r_1" or "r_2") is observed.

$$\mathcal{E}_0 = \left\{ E_c = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, E_{r_1} = \frac{1}{3}\begin{pmatrix} 2 & 0 & -2 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix}, E_{r_2} = \frac{1}{3}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\}$$

$$\mathcal{E}_1 = \left\{ E_c = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, E_{r_1} = \frac{1}{3}\begin{pmatrix} 2 & -1 & 0 \\ 1 & 0 & 2 \\ 1 & 0 & -2 \end{pmatrix}, E_{r_2} = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\}$$

Thus, S is encoded into the amplitudes of the second state, i.e. after reading S, the quantum state becomes $3^{-|S|}(1\ S\ 0)^T$ on the non-halting path. After that, for each block of $a_i\#$, $\mathcal{A}$ nondeterministically splits into two non-halting branches: It does nothing on the quantum register in the first branch, and it encodes a_i into the amplitudes of the third state and then subtract it from the amplitudes of the second state in the second branch. Let P be the set of all subsets of $\{1,\ldots,n\}$ and, for the subset $p \in P$, T_p be the summation of all a_j's, where $j \in p$. At the end of the computation, $\mathcal{A}$ has 2^n non-halting branches. The branch corresponding to the subset $p \in P$ has the quantum state $c_p(1\ \ (S-T_p)\ \ 0)^T$, where c_p depends on the lengths of a_j's ($j \in p$). Note that, if the input is a member, then $S - T_p$ is equal to 0 for some p, and it is nonzero for any p, otherwise. On the right end-marker, $\mathcal{A}$ applies the following superoperator when it is an universal state such that the input is accepted (rejected) if the outcome "a" ("r") is observed.

$$\mathcal{E}_\$ = \left\{ E_a = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, E_r = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\}.$$

This transition leads to "accepting" only if the second entry of the quantum state is equal to 0. Otherwise, the outcome "r" is always observed with some nonzero probability, and so it leads to "rejection". Therefore, only the members are accepted by $\mathcal{A}$. □

Now, we give a rtΣ_3QFA algorithm for a Σ_2P-complete (under log-space reduction) language [3]: GENERALIZED-SUBSETSUM $= \{u\#v\#t \mid (\exists x)(\forall y)[ux+vy \neq t]\}$, where u and v are integer vectors, t is an integer, and x and y are binary vectors of the same length as u and v, respectively.

Theorem 7. GENERALIZED-SUBSETSUM $\in \Sigma_3$QAL.

Proof. We present a rtΣ_3QFA, say $\mathcal{A}$, for GENERALIZED-SUBSETSUM similar to the rtΣ_2QFA for SUBSETSUM given above. $\mathcal{A}$ nondeterministically selects some entries of u and then universally selects some entries of v. Let x (y) represent the nondeterministic (universal) selection. For each (x, y) pair, $\mathcal{A}$ encodes $ux+vy-t$ into the amplitudes of a state on the quantum register. It is not hard to show that a quantum register with 3 states is sufficient (see the proof of Theorem 6). Thus, the quantum state of the (x, y)-branch can be as $c_{(x,y)}(1 \;\; (ux+vy-t) \;\; 0)^T$, where $c_{(x,y)}$ is the coefficient depending on (x, y). On the right end-marker, $\mathcal{A}$ applies the following superoperator when it is an existential state such that the input is rejected (accepted) if the outcome "r" ("a") is observed.

$$\mathcal{E}_\$ = \left\{ E_r = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, E_a = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\}.$$

This transition leads to "accepting" only if the second entry of the quantum state is not equal to 0. Otherwise, only the outcome "r" is observed with some nonzero probability, and so it leads to "rejection". Therefore, for the members, for some x, $(ux + vy - t)$ is non-zero for all y, and each universal branches ends with the decision of "accepting". That is, all members are accepted by $\mathcal{A}$. For the nonmembers, for any x, there exists a y such that $(ux + vy - t)$ is zero. So, for any x, some universal branches end with the decision of "rejection". That is, none of the nonmembers can be accepted. □

3.5 Log-Space q-Alternation Counterpart of Polynomial Hierarchy

We have already known that qAL = PSPACE. In this section, we focus on the classes of languages recognized by limited alternation of log-space strong qATMs. We begin with an upper bound for each level.

Theorem 8. qΣ_iL $\subseteq \Sigma_i$P *and* qΠ_iL $\subseteq \Pi_i$P *for each* $i \geq 0$.

Proof. The proof is similar to the proof of Lemma 2. The computation of a log-space qΣ_iTM can be simulated by a poly-time Σ_iTM, i.e. the precision on the quantum register can be at most polynomial since any log-space strong qATM must halt in polynomial time. □

By using Theorems 6 and 7, we can also give a lower bound for the second and third level.

Theorem 9. $\mathsf{NP} \subseteq \mathsf{q\Sigma_2L}$ *and* $\mathsf{\Sigma_2P} \subseteq \mathsf{q\Sigma_3L}$.

Proof. Since any language in NP, say L, is log-space reducible to SUBSETSUM [13], we can design a log-space qΣ_2TM for L as follows: The qΣ_2TM can implement a log-space reduction from L to SUBSETSUM and parallelly runs the rtΣ_2QFA given in the proof of Theorem 6 on the output string. The qΣ_2TM follows the decisions of the rtΣ_2QFA. The proof for the second relation is the same. □

We left open whether such a relation exists also for the other levels.

Acknowledgements. We would like to thank Andris Ambainis and A. C. Cem Say for their many helpful comments on some contents of this paper. We are grateful to the anonymous reviewers.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
2. Babai, L.: Trading group theory for randomness. In: STOC 1985: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, pp. 421–429 (1985)
3. Berman, P., Karpinski, M., Larmore, L.L., Plandowski, W., Ryttere, W.: On the complexity of pattern matching for highly compressed two-dimensional texts. Journal of Computer and System Sciences 65(2), 332–350 (2002)
4. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. Journal of the ACM 28(1), 114–133 (1981)
5. Chandra, A.K., Stockmeyer, L.J.: Alternation. In: FOCS 1976: Proceedings of the 17th IEEE Symposium on Foundations of Computer Science, pp. 98–108 (1976)
6. Condon, A.: Computational Models of Games. MIT Press (1989)
7. Fenner, S., Green, F., Homer, S., Puim, R.: Quantum NP is hard for PH. In: Sixth Italian Conference on Theoretical Computer Science, pp. 241–252. World Scientific, Singapore (1998)
8. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(1), 186–208 (1989)
9. Kozen, D.C.: On parallelism in Turing machines. In: FOCS 1976: Proceedings of the 17th IEEE Symposium on Foundations of Computer Science, pp. 89–97 (1976)
10. Nasu, M., Honda, N.: A context-free language which is not acceptable by a probabilistic automaton. Information and Control 18(3), 233–236 (1971)
11. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
12. Papadimitriou, C.H.: Games against nature. Journal of Computer and System Sciences 31(2), 288–301 (1985)

13. Papadimitriou, C.H.: Computational Complexity. Addison Wesley (1994)
14. Paz, A.: Introduction to Probabilistic Automata. Academic Press, New York (1971)
15. Reif, J.H.: The complexity of two-player games of incomplete information. Journal of Computer and System Sciences 29(2), 274–301 (1984)
16. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4(2), 177–192 (1970)
17. Watrous, J.: Space-bounded quantum complexity. Journal of Computer and System Sciences 59(2), 281–326 (1999)
18. Yakaryılmaz, A.: Public-qubits versus private-coins. Tech. Rep. ECCC:TR12–130 (2012), http://eccc.hpi-web.de/report/2012/130/
19. Yakaryılmaz, A.: Turing-equivalent automata using a fixed-size quantum memory. Tech. Rep. arXiv:1205.5395v1 (2012)
20. Yakaryılmaz, A., Say, A.C.C.: Languages recognized by nondeterministic quantum finite automata. Quantum Information and Computation 10(9&10), 747–770 (2010)
21. Yamakami, T., Yao, A.C.C.: $\mathrm{NQP}_{\mathbb{C}} = \text{co-}\mathrm{C}_{=}\mathrm{P}$. Information Processing Letters 71(2), 63–69 (1999)

Real Numbers, Chaos, and the Principle of a Bounded Density of Information

Gilles Dowek

INRIA, 23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France
gilles.dowek@inria.fr

Abstract. Two definitions of the notion of a chaotic transformation are compared: sensitivity to initial conditions and sensitivity to perturbations. Only the later is compatible with the idea that information has a finite density.

1 The Notion of Information in Physics

Information is not a new notion in Physics, as Ludwig Boltzmann already defined the entropy of a system as the logarithm of the number of microscopic states corresponding to its macroscopic state, that is, in modern terms, as the amount of information necessary to describe its microscopic state when its macroscopic state is known.

This definition presupposes that the number of microscopic states corresponding to a macroscopic state is finite, an hypothesis that would only be clarified later by quantum theory, and still in a very partial way.

After Bolzmann, this idea of a bound on the number of possible states of a given system, that is on the amount of information contained in such a system, or equivalently on the density of information in the Universe has slowly emerged. It is, for instance, one of the hypotheses assumed by Robin Gandy [5] in his "proof" of the physical Church-Turing thesis. It is also a thesis proposed by Jacob Bekenstein [2] in his investigation of back hole entropy. Bekenstein even proposes a bound, that is, unfortunately, not a bound on the amount of information contained in a system, but on the ratio between this amount of information and the energy of the system.

2 Physics without Real Numbers

This hypothesis of a bound on the density of information in Universe, is however inconsistent with the most common formulations of Physics, for instance Newtonian theory.

In Newtonian theory, just like information travels at an infinite velocity because the motion of a mass induces an instantaneous modification of the gravitational field in the whole Universe, an object as simple as a pencil contains an infinite amount of information, because its length is a real number, containing an infinite number of digits.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 347–353, 2013.

This idea that a magnitude, such as the length of a pencil, is a real number comes from an idealization of the process of measurement. The measure of the length of a segment, for instance the length of a pencil, is defined as the number of times a yardstick fits in this segment. More precisely, this natural number is a lower approximation of the length of the segment, with an accuracy which is the length of the yardstick—or twice this length, if the last fit is uncertain. A more precise measure is obtained by dividing this yardstick in ten equal parts, and counting the number of tenths of yardsticks that fit in the segment. Dividing again this tenth of yardstick in ten parts, an even more precise measure is obtained, and so on. The result of each individual measurement is thus a rational number, and only the hypothetical possibility to repeat this process indefinitely leads to the idea that the measured magnitude, per se, is the limit of an bounded increasing sequence of rational numbers, that is a real number.

In theory, the fact that the length of a pencil is a real number permits to record an infinite amount of information by sharpening the pencil to give it definite length. However, this idea is inconsistent with a principle, I will call the *caliper principle*, that, although it does not exactly have the status of a fundamental principle of Physics, is used as if it were one: the principle that a measuring instrument yields only an approximation of the measured magnitude, and that it is therefore impossible, except according to this idealization, to measure more than the first digits of a physical magnitude. Historically, this caliper principle might be one of the first formulations of the idea of a bounded density of information in the Universe, even if it only prevents the access to an infinite amount of information and not the existence of this infinite amount of information itself. According to this principle, this idealization of the process of measurement is a fiction. This suggests the idea, reminiscent of Pythagoras' views, that Physics could be formulated with rational numbers only.

We can therefore wonder why real numbers have been invented and, moreover, used in Physics. A hypothesis is that the invention of real numbers is one of the many situations, where the complexity of an object is increased, so that it can be apprehended more easily [3,4]. Let us illustrate this idea with an example.

If we restrict to rational numbers, the parabola of equation $y = x^2 - 2$ does not intersect the x-axis, because there exists no rational number whose square is equal to 2. But, because the continuous function $x \mapsto x^2 - 2$ takes a negative value at 1.4 and a positive one at 1.5, it can be proved that, for all positive rational numbers ε, there exists a rational number x such that the absolute value of this function at x is smaller than ε. It is even possible to build a bounded and increasing sequence of rational numbers 1.4, 1.41, 1.414, ... such that the image of this sequence by the function $x \mapsto x^2 - 2$ goes to 0 at infinity. We have here two relatively complex formulations of the intermediate value theorem. But postulating a limit $\sqrt{2}$ to this sequence permits to give a much simpler formulation to this theorem: there exists a number whose image by this function is 0.

Thus, according to the caliper principle, real numbers area fictions that permit to apprehend the Universe more easily, but there is no reason to think that physical magnitudes, per se, are real numbers.

3 The Status of the Principle of a Bounded Density of Information

If we have very few reasons to believe that physical magnitudes, per se, are real numbers, that is that they contain an infinite amount of information, we must admit that we have also few reasons to believe that, on the opposite, the density of information in the Universe is bounded.

This thesis is a hypothesis.

However this thesis is not a metaphysical hypothesis, that would only depend on the way we decide to describe the Universe—that we could decide to describe in a discrete or continuous way—and not of the properties of the Universe itself. Indeed, once a bound is fixed, the principle of a bounded density of information is a falsifiable statement: it is sufficient to record $b + 1$ bits of information in a system included in a sphere of radius 1m and to read it back to refute the thesis that the amount of information contained in a sphere of radius 1m is bounded by b, that is that the number of states of such a system is bounded by 2^b.

In the same way, it would be sufficient to transmit some information faster than light to refute the thesis that the velocity of propagation of information is bounded by the speed of light.

These two theses have a similar status. The only difference is that the bound is known in one case and not in the other.

4 Sensitivity to Initial Conditions

The introduction of real numbers in Physics has had the advantage to simplify the way we apprehend the Universe. But postulating that real numbers are more than just fictions, and that physical magnitudes are, per se, real numbers, has a remarkable consequence: it becomes possible, for a physical transformation, to be sensitive to initial conditions. It becomes possible, for instance, for the flap of a butterfly's wings in Brazil to set off a tornado in Texas.

An example of a process that is sensitive to initial conditions is the baker's transformation b, that maps each real number x, between 0 and 1, to the real number, also between 0 and 1, $2x$ when x is between 0 and $1/2$ and $2 - 2x$ when it is between $1/2$ and 1. This transformation is sensitive to initial conditions, because its iteration magnifies, step by step, small differences between two initial values x and x'. For instance, this transformation iterated on the two initial values 0 and $a/2^N$, that can be made as close as desired by taking N large enough, will lead in N steps to 0 and a respectively. Iterating this transformation progressively unfolds an infinite amount of information present in the initial state of the system.

The definition of the notion of sensitivity to initial conditions assumes that the initial values x and x' are slightly different, but that the processes applied to these values are rigorously identical: the flap of a butterfly's wings can modify the "initial" state of the atmosphere, but the butterflies must stop flapping their wings during the later evolution of the atmosphere.

The existence of transformations that are sensitive to initial conditions and unfold, step by step, an infinite amount of information present in the initial state of the system seems to be inconsistent with the principle of a bounded density of information. But more annoying is that the existence of non perturbed evolutions assumed by the definition of sensitivity to initial conditions is inconsistent with the much weaker and more consensual caliper principle, that no measurement can give, say, more that twenty relevant digits.

Indeed, is we assume physically possible to apply a perfect baker's transformation to a physical magnitude, and we have a measuring instrument I_1 that permits to measure this magnitude with an accuracy 2^{-10}, that is three decimal digits, it becomes possible to build an other measuring instrument I_2 that permits to measure some magnitudes with an accuracy 2^{-100}, that is thirty decimal digits.

This instrument just iterates the baker's transformation 100 times on the magnitude x to be measured and measures with the first instrument the 101 results $x = s_0, ..., s_{100}$ of these iterations. If one of the 101 measures yields a result that it between $1/2 - 2^{-10}$ and $1/2 + 2^{-10}$, so that we cannot decide if the measured magnitude is smaller or larger of $1/2$, the global measurement with the instrument I_2 fails. Otherwise, it is possible to determine whether each of these magnitudes is smaller or larger than $1/2$ and, in this case, the magnitude x is can be determined with an accuracy 2^{-102}. Indeed, it is easy to prove, by induction on i, that knowing if each of the terms $s_0, ..., s_{100}$ is smaller or greater than $1/2$ is sufficient to determine s_i with an accuracy $2^{-(102-i)}$. For $i = 100$, if s_{100} is smaller than $1/2$, then $1/4$ is an approximation of s_i, and if it is larger, $3/4$ is an approximation. In both cases, the accuracy is $1/4 = 2^{-2}$. Otherwise, by induction hypothesis, we know s_{i+1} with an accuracy $2^{-(102-(i+1))}$. If s_i is smaller than $1/2$ then $s_{i+1} = 2s_i$, that is $s_i = s_{i+1}/2$, and if it is larger, $s_{i+1} = 2 - 2s_i$, that is $s_i = 1 - s_{i+1}/2$. In both cases we obtain s_i with an accuracy $2^{-(102-(i+1))}/2 = 2^{-(102-i)}$. Thus, at the end, we obtain x with an accuracy 2^{-102}.

The success rate of the instrument I_2 is larger than eighty percent. To prove this, we prove that the measurement always succeeds when the number x does not have a sequence of 9 identical digits among the 110 first digits of its binary development. Indeed, the baker's transformation acts on the binary development of a number z in the following way: if the first digit of the number z is a zero, that is if $z \leq 1/2$, then it shifts all digits to the left, that is multiplies it by 2, if its is a one, that is if $z \geq 1/2$, then it replaces each one by a zero and each zero by a one, that is takes the opposite and adds 1, and shifts all the digits to the left, that is multiplies it by 2. Thus, applying the baker's transformation

100 times to the initial state x, that does not contain a sequence of 9 identical digits in the first 110 digits of its binary development, yields a sequence of 101 numbers s_0, ..., s_{100}, such that no element of this sequence contains a sequence of 9 identical digits in the first 10 digits of its binary development. Note that a number z such that $1/2 - 2^{-10} = 0.0111111111 < z \leq 1/2 = 0.011111111...$ has a sequence of 9 ones in its first 10 digits and that a number z such that $1/2 = 0.1000000000... \leq z < 1/2 + 2^{-10} = 0.1000000001$ has a sequence of 9 zeros in its first 10 digits. Thus, as none of the s_0, ..., s_{100} has a sequence of 9 identical digits in its first 10 digits, none is in the grey area of numbers between $1/2 - 2^{-10}$ and $1/2 + 2^{-10}$. Finally, the probability for a real number not to have a sequence 9 identical digits among the 110 first digits of its binary development is 0.815..., as the number k_l of sequences of l digits not containing a sequence of 9 identical digits verifies the induction relation: $k_0 = 1$, for $l+1 \leq 8$: $k_{l+1} = 2k_l$, $k_9 = 2k_8 - 2$, and for $l + 1 \geq 10$: $k_{l+1} = 2k_l - k_{l-8}$, from which we get $k_{110} = 1.058... \ 10^{33}$ and $k_{110}/2^{110} = 0.815...$

Thus, the success rate of I_2 is larger than or equal to 0.815... and the mere existence of a magnitude x for which this measurement can be performed is sufficient to contradict the caliper principle.

5 Sensitivity to Perturbations

Thus, a consequence of the caliper principle is that the a physical dynamical system must always be slightly perturbed. The course of its evolution is defined neither by

$$s_0 = x$$

$$s_{i+1} = f(s_i)$$

nor by

$$s_0 = x + p_0$$

$$s_{i+1} = f(s_i)$$

where p_0 would be a modification of the initial state x, but by

$$s_0 = x + p_0$$

$$s_{i+1} = f(s_i) + p_{i+1}$$

where $p_0, p_1, ...$ is a *perturbation sequence*.

It thus becomes difficult to distinguish, in the causes of a tornado in Texas, the role of a modification of the "initial" state of the atmosphere, due to a flap of a butterfly's wings, from later perturbations due, for instance, to other flaps of butterflies' wings.

A transformation f that is sensitive to perturbations is sensitive to initial conditions, as the sequence $p_0, 0, 0, ...$ is a perturbation sequence. *Shadowing* lemmas show that for many dynamical systems, the converse also holds: if a

system is sensitive to initial conditions, then it is also sensitive to perturbations: the information brought by perturbations during the evolution of the system can be aggregated in its initial state, to produce almost the same evolution.

For instance, consider N iterations $s_0, ..., s_N$ of the baker's transformation, perturbed by a sequence $p_0, p_1, ...$, such that for all i, $|p_i| \leq \varepsilon$ and s_i is between 0 and 1. Then it is easy to prove, by decreasing induction on i, that for all i, there exists a element s'_i, such that $|s'_i - s_i| \leq \varepsilon$ and the non perturbed evolution starting at step i with s'_i yields at s_N at step N. For $i = N$ just take $s'_N = s_N$. Assume the property holds at $i+1$. From $|s_{i+1} - s'_{i+1}| \leq \varepsilon$, $s_{i+1} = b(s_i) + p_i$, and $|p_i| \leq \varepsilon$, we get $|s'_{i+1} - b(s_i)| \leq 2\varepsilon$. If s_i is smaller than $1/2$, we let $s'_i = s'_{i+1}/2$. As s'_{i+1} is smaller than 1, s'_i is smaller than $1/2$ and $b(s'_i) = 2s'_i = s'_{i+1}$. Then, from $|s'_{i+1} - b(s_i)| \leq 2\varepsilon$, we get $|2s'_i - 2s_i| \leq 2\varepsilon$ and $|s'_i - s_i| \leq \varepsilon$. And if s_i is larger than $1/2$, we let $s'_i = 1 - s'_{i+1}/2$. As s'_{i+1} is smaller than 1, s'_i is larger than $1/2$ and $b(s'_i) = 2 - 2s'_i = s'_{i+1}$. Then, from $|s'_{i+1} - b(s_i)| \leq 2\varepsilon$, we get $|(2 - 2s'_i) - (2 - 2s_i)| \leq 2\varepsilon$ and $|s'_i - s_i| \leq \varepsilon$. Thus, in both cases we have $b(s'_i) = s'_{i+1}$ and $|s'_i - s_i| \leq \varepsilon$.

Thus, there exists an initial value s'_0 such that $|s'_0 - x| \leq 2\varepsilon$ and the non perturbed evolution from s'_0 yields s_N after N steps.

This equivalence between sensitivity to initial conditions and sensitivity to perturbations explains why the definition of a chaotic transformation speaks only about sensitivity to initial conditions and not about sensitivity to perturbations brought during the evolution of the system.

But this aggregation, in the initial state, of an unbounded amount of information, brought during the evolution of the system is not possible for systems where the amount of information is bounded. For example, the baker's transformation on the finite set $\{0, 0.1, ..., 0.9, 1\}$ is not sensitive to initial conditions: by modifying the initial value 0 of a quantity less than or equal to 0.1, it is possible to reach the values 0, 0.1, 0.2, 0.4 and 0.8, but not, unlike in the continuous case, the values 0.9 and 1, for instance—of course this example has only an didactic value, the size of a cell of a discrete physical system would be rather on the order of magnitude of Planck's length, and the number of states rather on the order of magnitude 10^{35} than 10.

In contrast, this transformation is sensitive to perturbations: it is possible to reach all the values a in the set $\{0, 0.1, ..., 0.9, 1\}$ starting from 0 and perturbing the system of a quantity at most 0.1 at each step—of course, a smaller perturbation would not mean anything. For instance, the value 0.9 is obtained by the sequence: 0, 0.1, 0.2, 0.4, 0.9. More generally, if $a = p\ 0.1$ where p is a natural number, we let N be a natural number such that $p/2^N = 0$, where / is the integer division, and $s_i = (p/2^{N-i})\ 0.1$. We have $s_0 = 0$, $s_N = a$ and for all i between 0 and $n-1$, $s_i \leq 1/2$, $s_{i+1} = 2s_i$ or $s_{i+1} = 2s_i + 0.1$. Thus $s_{i+1} = b(s_i) + p_i$, with $|p_i| \leq 0.1$.

This transformation is sensitive to perturbations but not to initial conditions, which shows that these two conditions are not equivalent in this case.

6 The Definition of the Notion of a Chaotic Transformation

Thus, there are at least two different reasons for the perturbed baker's transformation to produce the value 0.9 after four iterations, starting from 0. One is that the initial value was not 0, but $0.9/16 = 0.05625$, an other is that at the second and fourth iteration, a perturbation of 0.1 has been brought.

The first explanation—sensitivity to initial conditions—postulates a non perturbed evolution that is inconsistent with the caliper principle and the existence of a punctual cornucopia that provides an infinite amount of information not accessible to measurement, but appearing during the evolution of the system. And it provides no explanation why this evolution cannot itself be considered as a measurement.

In contrast, the second—sensitivity to perturbations—does not assume the existence of a non perturbed transformation, remains possible even if we assume that information has a bounded density, and locates the source of the information that appears during the evolution in the environment of the system, with which it always interacts. Sensitivity to perturbations seems therefore to be a good alternative to sensitivity to initial conditions, when defining the notion of a chaotic transformation.

This clarification of the definition of the notion of a chaotic transformation is one of the benefits of using a notion of information, and a principle of a bounded density of information in Sciences, such as Physics, besides its obvious use in Computer Science.

Acknowledgements. Thanks to Pablo Arrighi, Ali Assaf, Raphaël Cauderlier, Simon Cruanes, Guiseppe Longo, Jean-Baptiste Joinet, Thierry Paul, and Maël Pegny.

References

1. Arrighi, P., Dowek, G.: The principle of a finite density of information. In: Zenil, H. (ed.) Irreducibility and Computational Equivalence: Wolfram Science 10 Years After the Publication of A New Kind of Science (2012)
2. Bekenstein, J.D.: Universal upper bound to entropy-to-energy ratio for bounded systems. Phys. Rev. D 23, 287–298 (1981)
3. Berthoz, A., Simplexité, L., Jacob, O. (2009)
4. Dowek, G.: La notion de nombre réel : une solution simplexe? (to appear)
5. Gandy, R.: Church's thesis and principles for mechanisms. In: The Kleene Symposium. North-Holland (1980)

Random Selection in Few Rounds

Timofey Stepanov

Lomonosov Moscow State University, Leninskie Gory 1, Moscow 119991,
Yandex, Leo Tolstoy St. 16, Moscow 119021

Abstract. We consider protocols for two parties to select a random string of a given length n. We are interested in protocols in which the resulting distribution is close to the uniform one even if one party deviates from the protocol. For 2 and 3 round protocols we prove tight upper bounds for the Shannon entropy that such protocols can guarantee for the honest party. We also prove some upper bound for r round protocols for every r.

Keywords: Cryptography, distributed computing, random selection protocols.

1 Introduction

We study the following problem. Two players, Alice and Bob, want to agree on a random string. They stay in different places, so none of them can see what the other does. To this end they send messages to each other using some probabilistic algorithms. Afterwards they compute some function of sent messages and return its value as a result.

Each message depends only on previous messages and random bits of the respective player (each player has its own private randomness source). A player cannot check whether another player follows the algorithm they have agreed on. So each player can abandon the protocol to change the distribution of the resulting string. We study protocols that ensure high Shannon entropy of the resulting distribution if at least one player is honest.

In this model it is impossible to generate a random bit, as for every finite game with 0,1 outcomes either Alice has a strategy forcing the outcome 0, or Bob has a strategy that forces the outcome 1. Thus we consider protocols that output random strings of length $n > 1$. The quality of the output distribution is usually measured by either its Shannon entropy, or by min-entropy, or by statistical distance from the uniform distribution, or by resiliency.

The protocol is called (μ, ε)-*resilient* if for every set $S \subset \{0,1\}^n$ with density μ, the output of the protocol falls into S with probability at most ε if at least one player is honest.

In [1], Goldreich et al. constructed a protocol that is $(\mu, \sqrt{\mu})$-resilient for all $\mu > 0$. The protocol runs in $O(n)$ rounds, communicates $O(n^2)$ bits and guarantees Shannon entropy $n - O(1)$.

In [2], Sanghvi and Vadhan for each $\delta > 0$ have presented a protocol running in $2\log^* n + O(1)$ rounds that is $(\mu, \sqrt{\mu + \delta})$-resilient for all μ. They also showed

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 354–365, 2013.

a lower bound $\log^* n - \log^* \log^* n - O(1)$ on the number of rounds of any protocol that achieves constant positive statistical distance from the uniform distribution. By $\log^* n$ we denote the iterated logarithm, which is the minimal number k, such that $\log^{(k)} n = 1$, where $\log^{(i)} n = \underbrace{\log \ldots \log}_{i} n$ and $\log n = \lceil \log_2 n \rceil$.

In [3], Buhrman et al. have presented a protocol which runs in r rounds, where $r \geq 3$ is an arbitrary odd integer, and guarantees Shannon entropy $n - \log^{((r-1)/2)} n$. In particular, for $r = 2\log^* n + 1$ rounds their protocol guarantees Shannon entropy $n - O(1)$. They also showed that any protocol guaranteeing Shannon entropy $n - O(1)$ must run in $\Omega(\log^* n)$ rounds.

The natural question that arises in connection with the last two results is the following. Let n and r be given. What is the largest Shannon entropy $H_r(n)$ which protocols with r rounds can guarantee? In this paper we present tight bounds for $H_r(n)$ for $r = 2, 3$ and an upper bound for any r, n. More specifically,

- we show that $H_2(n) = n/2 + O(\log \log n)$,
- we prove that the bound $H_3(n) > n - \log n - O(1)$ from [3] is tight up to $O(\log \log n)$, that is, $H_3(n) = n - \log n + O(\log \log n)$.
- we show the bound $H_r(n) < n - \frac{1}{8} \log^{(\log^* \log^* n + r)} n + O(1)$ for all n and all r.

Notice that $\log^* \log^* n \leq 3$ for all n that the mankind will ever use in practice (the maximal number n with $\log^* \log^* n \leq 3$ is the tower of 2s of height 16). The question whether the number of iterations $r + \log^* \log^* n$ can be reduced to $(r-1)/2$ to meet the lower bound of [3] remains open.

In the proofs we use the techniques and the results from [2] and [3].

2 Definitions

Definition 1. *An r round protocol Π for generating n-bit strings is specified by integers $n_1, \ldots, n_r$, probabilistic algorithms A and B, the indication who starts the communication and a function $f : \mathbb{B}^{n_1} \times \ldots \times \mathbb{B}^{n_r} \to \mathbb{B}^n$, where $\mathbb{B} = \{0, 1\}$. The random variable $f(M_1, \ldots, M_r)$, where $M_1, \ldots, M_r$ stand for the messages sent by communicating algorithms A, B is called the protocol's result (or output) and is denoted by $\Pi(A, B)$. If a player, say Alice, deviates from the protocol and uses an algorithm A^* in place of A, the resulting random variable is denoted by $\Pi(A^*, B)$.*

In this definition we assume that the first message M_1 is computed by applying the algorithm A (if Alice starts the communication) to the empty string, the second message M_2 is computed by applying B to M_1 and so on: M_i is computed by applying A or B (depending on i mod 2) to the concatenation the of previous messages. We also assume that algorithms A and B halt in finite time on all inputs and all outcomes of their randomness generators.

In other words, we use the regular model of randomized communication protocols with fixed lengths of messages and private randomness. As we do not care

about the communication length, we can also use the variable length model. The crucial distance between the model used in communication complexity and ours is this: we will allow one player to cheat.

Definition 2. *The Shannon entropy of a random variable with N outcomes with probabilities $p_1, \ldots, p_N$ is defined by the formula*

$$H = -\sum_{i=1}^{N} p_i \log_2 p_i,$$

(if $p_i = 0$ then the corresponding term in the sum is considered to be zero).

Definition 3. *We say that the protocol Π guarantees Shannon entropy h if in the case when at least one player is honest the entropy of the output is at least h, that is, Shannon entropy of both random variables $\Pi(A^*, B)$ and $\Pi(A, B^*)$ is at least h for all algorithms A^*, B^*.*

Recall that Shannon entropy of any random variable with values in the set of strings of length n is at most n (and the value n is attained for the random variable uniformly distributed over this set). Thus the guaranteed entropy cannot exceed n. Moreover, the following simple observation shows that it even cannot exceed $n - 1$. Indeed, Zermelo's theorem [4] for finite games of two players with two outcomes states that for every such game either the first or the second player has a winning strategy. For every r-round protocol Π consider the r-moves game where moves are strings of length $n_1, \ldots, n_r$ and the first player wins if the first bit of $f(M_1, \ldots, M_r)$ is zero. By Zermelo's theorem either Alice has a deterministic strategy forcing the first bit of the result be 0, or Bob has a deterministic strategy forcing the first bit of the result be 1. In both cases Shannon entropy of the outcome is at most $n - 1$.

Another randomness measure used in this context is called *resiliency*:

Definition 4. *A distribution on a finite set U is called (μ, ε)-resilient if probability of the result to fall into any subset of U of density not greater than μ is not greater than ε. We call a protocol (μ, ε)-resilient, if in case at least one player is honest the result distribution is (μ, ε)-resilient.*

3 Upper Bounds for 2- and 3-Round Protocols

3.1 2-Round Protocols

There is a 2-round protocol which guarantees entropy $\frac{1}{2}n$: the players send random strings of length $\frac{1}{2}n$ to each other and the output is their concatenation. Next theorem shows, that this is essentially the best protocol.

Theorem 1. *For any 2-round protocols generating n-bit strings the guaranteed Shannon entropy is at most $\frac{1}{2}n + O(\log \log n)$.*

In the proof we will use the following simple lemma.

Lemma 1. *Let μ be a probability distribution over a family of at most s-element subsets of a "universe" U of cardinality N. For any b, there is a subset A of U of size at most $\frac{Nb}{s}$, such that a random set S in the family chosen according to distribution μ intersects A with probability at least $1 - e^{-b}$.*

Proof. Select at random $\frac{Nb}{s}$ elements from U in A (all elements of U are equiprobable and the choices are independent). The probability that the resulting set A does not intersect a fixed set of size $\leq s$ is at most $(1 - \frac{s}{N})^{\frac{Nb}{s}} < e^{-b}$. Therefore, the average probability over A of the probability of the event "S does not intersect A" is less than e^{-b}. Hence, there is A for which that probability is less than e^{-b}. □

The following lemma is a well known property of Shannon entropy.

Lemma 2. *Assume X is a random variable in the set U and A is a subset of U. Then*

$$H(X) \leq p \log |A| + (1-p) \log(|U| - |A|) + 1,$$

where $p = \Pr[X \in A]$.

Proof (of Theorem 1). Assume Bob starts the protocol. He sends a message β. Let S_β denote the set of all n-bit strings x such that $f(\beta, \alpha) = x$ for some α. Sending a message, Alice selects a string from S_β. We will show that if all S_β are large, then Alice is able to force the small output entropy. On the other hand, if at least one S_β is small then Bob can do so.

Fix a natural number s (in this proof we will use $s = 2^{n/2}$) and distinguish the following two cases: (a) there is β_0 with $|S_{\beta_0}| \leq s$ and (b) for all β, $|S_\beta| > s$.

In case (a) Bob, sending the message β_0, can force the output to belong to S_{β_0}. Thus Bob has a strategy such that the entropy of the outcome is at most $\log_2 s$.

In case (b) Alice also can force the protocol output to belong to a small set, with probability close to 1. To show this, we use Lemma 1 for the family of subsets S_β (where β ranges over all Bob's messages) and $b = \ln n$. By the lemma there is a set R of size $\frac{2^n \ln n}{s}$ which intersects with $1 - e^{-\ln n} = 1 - \frac{1}{n}$ subsets from the family. Whatever message Bob sends, Alice can choose a message such that the outcome of the protocol is in R. By Lemma 2, Alice has a strategy that ensures the entropy of the outcome be at most

$$\left(1 - \frac{1}{n}\right) \log_2 \frac{2^n \ln n}{s} + \frac{1}{n} \log_2 2^n + 1.$$

The second term here is equal to 1 and in the first term we drop the $-1/n$. In this way we obtain the bound $\log_2(2^n \ln n/s) + 2$ and

$$H_2(n) \leq \max\{\log_2 s,\ n - \log_2 s + \log_2 \ln n + 2\}$$

for every s. For $s = 2^{n/2}$ we get the bound we need. □

3.2 3-Round Protocols

A 3-round protocol that guarantees the entropy $n - \log_2 n$ was presented in [3]. In this section we show that one cannot do better.

Theorem 2. $H_3(n) < n - \log_2 n + O(\log \log n)$.

In the proof we will use the following lemma from [2].

Lemma 3 ([2]). *Assume we are given a collection of disjoint subsets $S_1, \ldots, S_m$ of a finite "universe" U of cardinality N. Assume further that for all i, $|S_i| \leq s$. Then the probability that a set $R \subset U$ chosen at random among all sets of density μ includes at least one set S_i from the collection is at least $1 - \frac{1}{m}\left(\frac{e}{\mu}\right)^s$.*

Proof (of Theorem 2). Assume Alice starts the communication. Let γ denote her fist message, β the first message of Bob, and α the second message of Alice. Sending α Alice selects the output from the set $S_{\gamma\beta}$ of all x such that there is α with $f(\gamma, \beta, \alpha) = x$. Again we distinguish two cases:

(a) For all γ there are m pairwise disjoint sets of the form $S_{\gamma\beta}$, each of size at most s.
(b) There is γ_0, for which such m sets $S_{\gamma\beta}$ do not exist. Here m, s stand for natural numbers to be chosen later.

In case (a) we claim that there is a set R of n-bit strings of density $\frac{1}{n}$ that has the following property. With probability at least $p = 1 - \frac{(en)^s}{m}$ Alice's message γ is *good for* R, which means that there is Bob's message β with $S_{\gamma\beta} \subset R$. The claim is proved by probabilistic arguments. Choose both R and Alice's message at random. The set R is chosen uniformly among all sets of density $1/n$ and Alice's message is chosen independently of R according to her randomized algorithm. As we are in the case (a), any Alice's message γ, by Lemma 3 is good for R with probability at least p. Thus, the overall probability that γ is good for R is at least p. Hence, there is a set R with probability at least p.

Bob's strategy is the following: he picks R of density $1/n$ satisfying the claim and for every good Alice's message forces the outcome to fall into R (if Alice's message is bad, he sends any message). By Lemma 2, this Bob's strategy ensures that the entropy of the outcome is at most

$$\begin{aligned} H \;&\leq\; p \log_2 \frac{2^n}{n} + (1-p)n + 1 \;=\; n - p \log_2 n + 1 \\ &=\; n - \log_2 n + \frac{(en)^s}{m} \log_2 n + 1 \end{aligned}$$

In case (b) Alice first sends message γ_0. As we are in case (b), there is a set X of size ms that intersects every small $S_{\gamma_0\beta}$ ("small" means "of cardinality at most s). Indeed, let X be the union of any maximal disjoint family of small sets.

By Lemma 1 there is a set Y of size $\frac{2^n \ln n}{s}$ which intersects a randomly chosen large $S_{\gamma_0\beta}$ (of size more than s) with probability at least $1 - e^{-\ln n} = 1 - 1/n$. Here we mean the probability under condition that $S_{\gamma_0\beta}$ is large. Thus, with (unconditional) probability at least $1-1/n$ a randomly chosen set $S_{\gamma_0\beta}$ intersects $X \cup Y$. Thus, Alice can force the output to fall into $X \cup Y$ with that probability. By Lemma 2 this implies that the entropy of the outcome is at most:

$$\begin{aligned} H &\le \left(1 - \frac{1}{n}\right) \log_2 \left(\frac{2^n \ln n}{s} + sm\right) + \frac{1}{n} n + 1 \\ &\le \log_2 \left(\frac{2^n \ln n}{s} + sm\right) + 2 \end{aligned}$$

Combining cases (a) and (b) we see that $H_3(n)$ does not exceed

$$\max\{n - \log_2 n + (en)^s (\log_2 n)/m,\ \log_2(2^n \ln n/s),\ \log_2(sm)\} + O(1)$$

(we have replaced the logarithm of the sum by the maximum of logarithms, as up to adding 1 they are equal). Now we need to adjust s and m so that this maximum is minimal. It is hard to find optimal m and s. However, it is not hard to see that whatever m, s we choose, the maximum is at least $n - \log_2 n + \log_2 \ln n$. Indeed, if $m \le (en)^s$, then the first term is larger than n. If $s \le n$, then $\log_2(2^n \ln n/s)$ is greater than $n - \log_2 n + \log_2 \ln n$. And otherwise $\log_2(sm)$ is greater than $\log_2 m > s \log_2 n > n \log_2 n$.

Guided by these hints we let $m = (en)^s \log_2 n$, say, so that the first term becomes $n - \log_2 n + 1$. Then let $s = n/(2 \log_2 n)$. Then

$$\log_2(2^n \ln n/s) = n - \log_2 n + O(\log \log n)$$

and

$$\log_2(sm) = \log_2 s + s \log_2(en) = n/2 + o(n),$$

and we are done. □

4 Protocols with More Than 3 Rounds

Definition 5. *The iterated logarithm of a number $n \in \mathbb{R}, n \ge 1$ to base d, $\log_d^* n$, is defined as minimal natural number k, such that*

$$\underbrace{\lceil \log_d \ldots \lceil \log_d}_{k} n \rceil \ldots \rceil = 1.$$

In other words, $\log_d^* n \le k$ iff n is not larger than the tower

$$d^{d^{\cdots^d}}$$

of d's of height k. If $d = 2$ we will skip the index d and write $\log^* n$.

In [3], for all odd $r \geq 3$ a protocol was constructed that runs in r rounds and guarantees the entropy at least $n - \log^{(\frac{r-1}{2})} n$. Here $\log^{(i)} n = \underbrace{\log \ldots \log}_{i} n$ and $\log n = \lceil \log_2 n \rceil$. In this paper we establish an upper bound for the entropy guaranteed by r-round protocols.

Theorem 3. *For all r and all n, $H_r(n) < n - \frac{1}{8} \log^{(\log^* \log^* n + r)} n + O(1)$.*

In the proof of this theorem we will use lower bounds from [2] for the number of rounds of resilient protocols. First, we remind the definition.

Definition 6. *The distribution on a finite set U is (μ, ε)-resilient if the probability of the result to fall in any subset of U of density not greater than μ is at most ε. We call a protocol (μ, ε)-resilient, if in case at least one player is honest the output distribution is (μ, ε)-resilient.*

Theorem 4 (Sanghvi and Vadhan, [2]). *For any $\nu, \varepsilon > 0$, there exists a constant c such that any $(\nu, 1-\varepsilon)$-resilient protocol runs in more than $\log^* n - \log^* \log^* n - c$ rounds.*

To prove our result, we need to understand how the constant c depends on ν and ε. To this end we will sketch the proof of this theorem from [2].

Proof (A scetch of the proof of Theorem 4). Given any positive μ, ε and r, define sequence of reals $s_0, \ldots, s_r$ by induction:

$$s_0 = 1 \quad \text{and} \quad s_k = \frac{r}{\varepsilon} \left(\frac{re}{\mu} \right)^{s_{k-1}} s_{k-1}.$$

The result from [2] (Theorem 4.5 from that paper) states that for any r-round protocol there is a set X of density at most $\mu + s_r/2^n$ such that either Alice or Bob can force the outcome of the protocol to fall into X with probability more than $1 - \varepsilon$. Hence, there is no $(\mu + s_r/2^n, 1-\varepsilon)$-resilient protocol running in r rounds.

We apply this result to $\mu = \nu/2$ and show that if r is less than $\log^* n - \log^* \log^* n - c$ (for some $c = c(\nu, \varepsilon)$), then $s_r/2^n$ is less than $\mu = \nu/2$, thus, there is no r-round $(\nu, 1-\varepsilon)$-resilient protocol either. □

To find the value of c we need to upper bound s_k.

Lemma 4. *If $\mu \leq 1/2$, then s_k can be upper bounded by the tower of d's of height k where $d = \frac{r^2 e}{\varepsilon \mu}$.*

Proof. We have to show that $s_k \leq d^{s_{k-1}}$. For $k = 1$, the left hand side and the right hand side of this inequality coincide. For all $x, y \geq 2$ we have $xy \leq x^y$ and by assumption $\mu \leq 1/2$. It follows that

$$s_k = \left(\frac{re}{\mu} \right)^{s_{k-1}} \frac{r}{\varepsilon} s_{k-1} \leq \left(\frac{re}{\mu} \right)^{s_{k-1}} \left(\frac{r}{\varepsilon} \right)^{s_{k-1}} = \left(\frac{r^2 e}{\varepsilon \mu} \right)^{s_{k-1}}$$

for all $k \geq 2$. □

Theorem 5 (Refined version of Theorem 4). *For any $\varepsilon > 0$ and $\nu \geq 2n/2^n$, every $(\nu, 1-\varepsilon)$-resilient protocol runs in more than $\log^* n - \log^* \log^* n - \log^* \frac{4e^2}{\varepsilon^2 \nu^2}$ rounds.*

Proof. We start with the following

Lemma 5. *If $r \leq \log^* n - \log^* \log^* n - \log^* \frac{e^2}{\varepsilon^2 \mu^2}$ then $s_r \leq n$.*

In the proof of this lemma we will use the following facts about the iterated logarithm:

Lemma 6. *If $a, b > 1$ then $\log^*(a+b) \leq \log^* a + \log^* b$.*

Proof. Obvious. □

Lemma 7. *If $a, b > 2$ then $\log^*(ab) \leq \log^* a + \log^* b - 1$.*

Proof.

$$\begin{aligned} \log^*(ab) &= \log^* \lceil \log(ab) \rceil + 1 \leq \log^*(\lceil \log a \rceil + \lceil \log b \rceil) + 1 \\ \text{(Using Lemma 6)} \quad &\leq \log^* \lceil \log a \rceil + \log^* \lceil \log b \rceil + 1 = \log^* a + \log^* b - 1 \end{aligned}$$

□

Lemma 8. *For all integer $a > 2$, $\log^*(a^4) \leq \log^* a + 2$.*

Proof. For $a = 3$ and $a = 4$ this can be verified manually. For $a \geq 5$ we obtain a slightly stronger bound:

$$\log^*(a^4) = \log^* \lceil 4 \log a \rceil + 1 \overset{\text{(Lemma 7)}}{\leq} \log^* 4 + \log^* \lceil \log a \rceil = \log^* a + 1$$

□

The next two lemmas were proved in [2]. For the sake of completeness we prove them here.

Lemma 9 ([2]). *If $d \geq 4$ and $k \leq \log_d^* a$, then $\log^{(k)} a \leq (2 \log d) \log_d^{(k)} a$.*

Proof. The base case $k = 0$ is clear. Assume, then, that $\log^{(k-1)} a \leq (2 \log d) \log_d^{(k-1)} a$. Applying log to both sides, we have:

$$\begin{aligned} \log^{(k)} a &\leq \log(2 \log d) + \log(\log_d^{(k-1)} a) = \log(2 \log d) + (\log_d^{(k)} a)(\log d) \\ &\leq (2 \log d)(\log_d^{(k)} a) \end{aligned}$$

where the last line follows because for $d \geq 4$, $d \geq 2 \log d$ and for $k \leq \log^* a$, $\log^{(k)} a \geq 1$. □

Lemma 10 ([2]). *If $d \geq 4$ then $\log^*_d a \geq \log^* a - \log^*(2\log d)$.*

Proof. By Lemma 9 with $k = \log^*_d a$, we have $\log^{(\log^*_d a)} a \leq 2\log d$. Applying $\log^*(2\log d)$ logarithms to both sides, we have $\log^{(\log^* da + \log^*(2\log d))} a \leq 1$. Since $\log^* a$ is defined to be the least k such that $\log^{(k)} a \leq 1$, it follows that $\log^* a \leq \log^*_d a + \log^*(2\log d)$. □

Proof (of Lemma 5). By Lemma 4 $s_r \leq n$ provided $r \leq \log^*_d n$. It remains to lower bound the right hand side of this inequality using Lemma 10:

$$\begin{aligned}
\log^*_d n &\geq \log^* n - \log^*(2\log d) = \log^* n - \log^*\left(2\log\frac{r^2 e}{\varepsilon\mu}\right) \\
&= \log^* n - \log^*\log\left(\frac{r^2 e}{\varepsilon\mu}\right)^2 = \log^* n - \log^*\left(\frac{r^2 e}{\varepsilon\mu}\right)^2 + 1 \\
\text{(Using } r \leq \log^* n\text{)} \quad &\geq \log^* n - \log^*\left((\log^* n)^4 \frac{e^2}{\varepsilon^2\mu^2}\right) + 1 \\
\text{(Using Lemma 7)} \quad &\geq \log^* n - \log^*(\log^* n)^4 - \log^*\frac{e^2}{\varepsilon^2\mu^2} + 2 \\
\text{(Using Lemma 8)} \quad &\geq \log^* n - \log^*\log^* n - \log^*\frac{e^2}{\varepsilon^2\mu^2}.
\end{aligned}$$

Lemma 5 is proved. □

Let us continue the proof of Theorem 5. Let Π be an r-round $(\nu, 1-\varepsilon)$-resilient protocol. Let $\mu = \nu/2$. By assumption $\mu \geq n/2^n$. We get a contradiction, if r is so small that $s_r \leq n$ (and hence $\mu + s_r/2^n \leq \nu$). And by Lemma 5 this happens whenever

$$r \leq \log^* n - \log^*\log^* n - \log^*\frac{e^2}{\varepsilon^2\mu^2}.$$

Theorem 5 is proved. □

To use the lower bound of Theorem 5, we have to show that protocol producing high entropy is resilient for appropriate parameters. This can be done, following [3], by means of the notion of statistical distance.

Definition 7. *The statistical distance between random variables X and Y with values in the same set U is defined as*

$$\max_{A \subset U} |\Pr[X \in A] - \Pr[Y \in A]|$$

The following theorem was proved in [3]. For the sake of completeness we prove it.

Theorem 6 ([3]). *If the entropy of a distribution is less than $n - c$, then the statistical distance between that distribution and the uniform one is not greater than $1 - 2^{-2c+O(1)}$.*

Proof. Fix c. Let ξ be a random variable with range $\{0,1\}^n$ such that $H(\xi) \geq n - c$.

For $x \in \{0,1\}^n$ let $p_x = \Pr[\xi = x]$. For all integer $i \leq n$ let $N_i$denote the number of x with

$$2^{-n+i-1} < p_x \leq 2^{-n+i} \tag{1}$$

and let w_i denote their total probability.

The statistical distance between ξ and the uniformly distributed random variable is equal to

$$\sum_{x:p_x>2^{-n}} (p_x - 2^{-n}) = \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} N_i 2^{-n} \leq \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} 2^{-i} w_i.$$

Here the last inequality holds, as $w_i \leq N_i 2^{i-n}$.

Thus it suffices to prove the inequality

$$\sum_{i=1}^{n} (1 - 2^{-i}) w_i \leq 1 - 2^{-2c-7}$$

provided $H(\xi) \geq n - c$.

The contribution $-p_x \log p_x$ to the entropy of ξ of each x satisfying (1) is

$$-p_x \log p_x < -p_x \log 2^{-n+i-1} = p_x(n + 1 - i).$$

Therefore we can estimate the entropy of ξ as

$$H(\xi) \leq \sum_{i \leq n} w_i (n + 1 - i) = n + 1 - \sum_{i \leq n} i w_i$$

hence

$$\sum_{i \leq n} i w_i \leq c + 1 \tag{2}$$

Here i ranges over all integers $i \leq n$, including negative ones. However, the contribution of negative i is bounded by a constant. Indeed, as $2^{n-i} w_i \leq N_i \leq 2^n$ we can conclude that $w_i \leq 2^i$ hence

$$0 \geq \sum_{i<0} i w_i \geq \sum_{i<0} i 2^i = 2.$$

Thus, inequality (2) implies that the sum of the iw_i over positive i is bounded by a constant:

$$\sum_{i=1}^{n} i w_i \leq c + 3 \leftrightharpoons d.$$

Divide the sum $\sum_{i=1}^{n}(1-2^{-i})w_i$ into two groups: the sum over all $i \geq 2d$ and the rest. The first sum is small by the last inequality. Indeed, the factors i in the iw_i are at least $2d$, hence the sum of all terms with $i \geq 2d$ is at least $\sum_{i\geq 2d} 2dw_i$ and at most d. This implies $\sum_{i=2d}^{n} w_i \leq 1/2$. In the second sum the coefficient $(1-2^{-i})$ is small:

$$1-2^{-i} \leq 1-2^{-2d}.$$

Thus the total sum can be upper bounded by

$$1-2^{-2d-1} = 1-2^{-2c-7}.$$

Theorem 6 is proved. □

Now we can finish the proof of the main result.

Proof (of Theorem 3). If the statistical distance between a random variable ξ and the random variable uniformly distributed over the same set is δ, then ξ is $(\mu, \mu+\delta)$-resilient for every μ. Thus, Theorem 6 implies the following: if a protocol guarantees entropy $H \geq n-d$, then it is $(\mu, 1-2^{-2d+O(1)}+\mu)$-resilient for all μ. According to Theorem 4 it runs in more than

$$\log^* n - \log^* \log^* n - \log^* \frac{4e^2}{\mu^2(2^{-2d+O(1)}-\mu)^2}$$

rounds. It remains to choose μ such that this value is maximal. To this end let $\mu = 2^{-2d+O(1)}/2$. We may apply Theorem 4 provided $\mu \geq 2n/2^n$. It will certainly be the case if $d \leq n/2$.

Thus, we obtain the following lower bound for the number of rounds r of protocols guaranteeing the entropy $n-d$:

$$\begin{aligned} r &> \log^* n - \log^* \log^* n - \log^* \frac{4e^2}{2^{-8d+O(1)}} \\ &= \log^* n - \log^* \log^* n - \log^* 2^{8d+O(1)} \\ &= \log^* n - \log^* \log^* n - \log^*(8d+O(1)) - 1. \end{aligned}$$

It remains to derive an upper bound for d as a function of r from this inequality. We have

$$\log^*(8d+O(1)) \geq \log^* n - \log^* \log^* n - r$$

hence

$$8d+O(1) \geq \log^{(\log^* \log^* n + r)} n,$$

which implies

$$H_r(n) \leq n - \frac{1}{8}\log^{(\log^* \log^* n + r)} n + O(1)$$

□

References

1. Goldreich, O., Goldwasser, S., Linial, N.: Fault-tolerant computation in the full information model. SIAM Journ. on Computing 27(2), 506–544 (1998)
2. Sanghvi, S., Vadhan, S.: The Round Complexity of Two-Party Random Selection. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing. Baltimore, MD, USA, pp. 338–347
3. Buhrman, H., Christandl, M., Koucký, M., Lotker, Z., Patt-Shamir, B., Vereshchagin, N.K.: High Entropy Random Selection Protocols. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 366–379. Springer, Heidelberg (2007)
4. Schwalbe, U., Walker, P.: Zermelo and the Early History of Game Theory. Games and Economic Behavior 34(1), 123–137 (2001)

One-Counter Verifiers for Decidable Languages*

Abuzer Yakaryılmaz

University of Latvia, Faculty of Computing, Raina bulv. 19, Rīga, LV-1586, Latvia
abuzer@lu.lv

Abstract. Condon and Lipton (FOCS 1989) showed that the class of languages having a space-bounded interactive proof system (IPS) is a proper subset of decidable languages, where the verifier is a probabilistic Turing machine. In this paper, we show that if we use architecturally restricted verifiers instead of restricting the working memory, i.e. replacing the working tape(s) with a single counter, we can define some IPS's for each decidable language. Such verifiers are called two-way probabilistic one-counter automata (2pca's). Then, we show that by adding a fixed-size quantum memory to a 2pca, called a two-way one-counter automaton with quantum and classical states (2qcca), the protocol can be space efficient. As a further result, if the 2qcca uses a quantum counter, then the protocol can even be public, also known as Arthur-Merlin games.

We also investigate the computational power of 2pca's and 2qcca's as language recognizers. We show that bounded-error 2pca's can be more powerful than their deterministic counterparts by giving a bounded-error simulation of their nondeterministic counterparts. Then, we present a new programming technique for bounded-error 2qcca's and show that they can recognize a language which seems not to be recognized by any bounded-error 2pca. We also obtain some interesting results for bounded-error 1-pebble quantum finite automata based on this new technique. Lastly, we prove a conjecture posed by Ravikumar (FSTTCS 1992) regarding 1-pebble probabilistic finite automata, i.e. they can recognize some nonstochastic languages with bounded error.

1 Introduction

The only known interactive proof systems (IPS) having a restricted verifier for decidable languages were given by Feige and Shamir [8] (and independently by Condon and Lipton [4]). Although the verifier is a one-way probabilistic finite automaton, the protocols require to communicate with two provers. Therefore, the question remains open for IPS with one prover. In fact, Condon and Lipton [4] showed that the class of languages having a space-bounded interactive proof system (IPS) is a proper subset of decidable languages, where the verifier is a probabilistic Turing machine (PTM). In this paper, we show that if we use architecturally restricted verifiers instead of restricting the working memory, i.e. replacing the working tape(s) with a single counter, we can define some IPS's for each decidable language.

* This work was partially supported by FP7 FET-Open project QCS.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 366–377, 2013.

We present four new protocols for decidable languages. In the first protocol, the verifier is a two-way probabilistic one-counter automaton (2pca). By relaxing the requirement of having arbitrary small error bound, we obtain another protocol, in which the verifier does not need to move its input head to the left. In the third protocol, we show that if the verifier uses a fixed-size quantum register, called a two-way one-counter automaton with quantum and classical states (2qcca), then the protocol can also be space efficient. In all of these protocols, the verifiers hide some information from the prover. In the fourth protocol, we show that if we replace the classical counter of the verifier with a quantum counter, called a two-way quantum one-counter automaton (2qca), then the third protocol turns out to be public (Arthur-Merlin games), i.e. the prover always has a complete information about the verifier. The techniques behind these protocols are inspired from the previous weak-protocols, i.e. the nonmembers does not need to be rejected with high probability by the verifier, given for Turing recognizable languages by Condon and Lipton [4] and Yakaryılmaz [20].

We also examine 2pca's and 2qcca's as recognizers. We show that 2pca's form a bigger class than two-way deterministic one-counter automata (2dca's). We obtain this result by giving a bounded-error simulation of two-way nondeterministic one-counter automata. Then, we present a new programming technique for bounded-error 2qcca's and show that they can recognize a language which seems not to be recognized by any bounded-error 2pca. We also obtain some interesting results for bounded-error 1-pebble quantum finite automata based on this new technique. Moreover, we prove a conjecture posed by Ravikumar [16] regarding 1-pebble probabilistic finite automata, i.e. they can recognize some nonstochastic languages with bounded error.

To our knowledge, 2pca's have never been investigated. The only related work that we know is [9], in which Hromkovic and Schnitger examined two-way probabilistic multi-counter machines that are restricted to polynomial time, but, they did not present any result related to 2pca's. 2qca's, on the other hand, are examined only by Yamasaki et. al. [25] as language recognizers, in which the authors presented some bounded-error 2qca algorithms for some languages. However, these languages are known to be recognized by 2dca's and bounded-error 2qcca's without a counter [14]. Therefore, our results seem the first interesting results on 2qca's.

We provide the necessary background in Section 2. The four protocols for decidable languages are given in Section 3. The language recognition powers of probabilistic and quantum counter automata are investigated in Section 4. Lastly, the results on probabilistic and quantum finite automata with 1-pebble are given in Section 5.

2 Background

Throughout the paper, Σ not containing ¢ and \$ denotes the input alphabet and $\tilde{\Sigma} = \Sigma \cup \{¢, \$\}$. For a given string x, $|x|$ is the length of x and x_i is the i^{th}

symbol of x, where $1 \leq i \leq |x|$. The string $\text{¢}x\$$ is represented by $\tilde{x}$. $\mathbf{P}(\cdot)$ denotes all subsets of a given set.

Each model defined in the paper has a two-way infinite read-only input tape whose squares are indexed by integers. Any given input string, say $x \in \Sigma^*$, is placed on the tape as $\tilde{x}$ between the squares indexed by 1 and $|\tilde{x}|$. The tape has a single head, and it can stay in the same position ($\downarrow$) or move to one square to the left ($\leftarrow$) or to the right ($\rightarrow$) in one step. It must always be guaranteed that the input head never leaves $\tilde{x}$. Some models in the paper have also a counter, an infinite storage having two status, i.e. *zero* (0) or *nonzero* ($\pm$), and being updated by a value from $\{-1, 0, +1\}$ in one step. We assume that the reader familiar with the modes of language recognition with errors (see [19]).

Classical Models. A two-way deterministic one-counter automaton (2dca) is a two-way deterministic finite automaton with a counter. Formally, a 2dca $\mathcal{D}$ is a 6-tuple $\mathcal{D} = (S, \Sigma, \delta, s_1, s_a, s_r)$, where S is the set of states, $s_1 \in Q$ is the initial state, $s_a \in S$ and $s_r \in S$ ($s_a \neq s_r$) are the accepting and rejecting states, respectively, and δ is the transition function governing the behaviour of $\mathcal{D}$ in each step, i.e. $\delta : S \times \tilde{\Sigma} \times \{0, \pm\} \rightarrow S \times \{\leftarrow, \downarrow, \rightarrow\} \times \{-1, 0, +1\}$. Specifically, $\delta(s, \sigma, \theta) \rightarrow (s', d_i, c)$ means that when $\mathcal{D}$ is in state $s \in S$, reads symbol $\sigma \in \tilde{\Sigma}$, and the status of its counter is $\theta \in \{0, \pm\}$, then it updates its state to $s' \in S$, the position of the input head with respect to $d_i \in \{\leftarrow, \downarrow, \rightarrow\}$, and the value of the counter by $c \in \{-1, 0, +1\}$.

At the beginning of the computation, $\mathcal{D}$ is in state s_1, the input head is placed on symbol ¢, and the value of the counter is set to zero. A configuration of $\mathcal{D}$ on a given input string is represented by a triple (s, i, v), where s is the state, i is the position of the input head, and v is the value of the counter. The computation is terminated and the input is accepted (rejected) by $\mathcal{D}$ when it enters to s_a (s_r). The class of languages recognized by 2dca's is denoted 2DCA.

We will use *the same terminology* also for the other models unless otherwise is specified. A two-way nondeterministic one-counter automaton (2nca), say $\mathcal{N}$, is a 2dca having capability of making nondeterministic choices in each step. The transition function of $\mathcal{N}$ is extended as $\delta : S \times \tilde{\Sigma} \times \{0, \pm\} \rightarrow \mathbf{P}(S \times \{\leftarrow, \downarrow, \rightarrow\} \times \{-1, 0, +1\})$. In other words, for each triple (s, σ, θ) (see above), there may be more than one transition. Thus, $\mathcal{N}$ can follow more than one path during the computation, and if any path ends with a decision of acceptance, then the input is accepted. The class of languages recognized by 2nca's is denoted 2NCA.

A two-way probabilistic one-counter automaton (2pca), say $\mathcal{P}$, is a 2dca having capability of making probabilistic choices in each step. In order to explicitly represent the probabilistic part the machine, each step is divided into two transitions. Formally, $\delta = (\delta_p, \delta_d)$. For each triple (s, σ, θ) (see above), there are some predefined outcomes, i.e. $\Delta_{(s,\sigma,\theta)} = \{1, \ldots, k_{(s,\sigma,\theta)}\}$. Each outcome is selected with some (rational) probability: $\delta_p(s, \sigma, \theta, \tau) = p_\tau \in \mathbb{Q}$, where $\tau \in \Delta$ and $\sum_{\tau \in \Delta} p_\tau = 1$. After observing the outcome (τ), the deterministic transition is implemented as follows: $\delta_d(s, \sigma, \theta) \xrightarrow{\tau} (s', d_i, c)$ (see above). Note that δ_d must be defined for each possible τ. The class of languages recognized by 2pca's

with bounded error is denoted 2PCA. If we remove the counter, then we obtain a 2pfa (two-way probabilistic finite automaton). If the input head of a 2pca is not allowed to move to left, then we obtain a one-way probabilistic one-counter automaton (1pca).

A one-way deterministic two-counter automaton (1d2ca) is a one-way deterministic finite automaton with two counters. It was shown that any deterministic Turing machine (DTM) can be simulated by a 1d2ca[12]. We denote the class of decidable languages DECIDABLE.

Quantum Models. A two-way finite state automaton with quantum and classical states [1,24] (2qcfa) is a 2pfa using a finite quantum register instead of a classical random generator. Note that the quantum register can keep some information by its (pure) quantum state as well as making probabilistic choices.[1]

Formally, a 2qcfa[2] $\mathcal{Q}$ is a 8 tuple $(S, Q, \Sigma, \delta, s_1, s_a, s_r, q_1)$, where, apart from a classical model, there are two different components: Q is the state set of quantum register and q_1 is its initial state. Similar to probabilistic models, $\delta = (\delta_q, \delta_d)$, where δ_q governing the quantum part. In each step, firstly, δ_q determines a superoperator (see Figure 1 for the details) depending on the current classical state ($s \in S$) and scanning symbol ($\sigma \in \tilde{\Sigma}$), i.e. $\mathcal{E}_{s,\sigma}$, and then it is applied to the quantum register and one outcome, say τ, is observed. Secondly, the classical part of $\mathcal{Q}$ is updated depending on s, σ, and τ, which is formally represented as $\delta_d(s,\sigma) \xrightarrow{\tau} (s', d_i)$, where $s' \in S$ is the new classical state and $d_i \in \{\leftarrow, \downarrow, \rightarrow\}$ is the update of the position of input tape. Note that δ_d must be defined for each possible τ.

A two-way one-counter automaton with quantum and classical states (2qcca) is a 2pca using a finite quantum register instead of a classical random generator. The formal definition of a 2qcca is exactly the same as a 2qcfa. In fact, a 2qcca is a 2qcfa with a classical counter. So, the transition functions of a 2qcfa (δ_q and δ_d) can be extended for a 2qcca with the following modifications:

- The superoperator is determined by also the status of the counter ($\theta \in \{0, \pm\}$), i.e. $\mathcal{E}_{s,\sigma,\theta}$.
- The classical part of $\mathcal{Q}$ is updated depending on s, σ, θ, and τ, which is formally represented as $\delta_d(s,\sigma,\theta) \xrightarrow{\tau} (s', d_i, c)$, where $s' \in S$ is the new classical state, d_i is the update of the position of input tape, and $c \in \{-1, 0, +1\}$ is the update on the counter.

A generalization of 2qcca is a two-way quantum counter automaton (2qca) that uses a quantum counter instead of a classical one. (Note that this model is still not the most general one, but it is sufficiently general for our purpose.) We can see 2qca as the combination of a 2qcfa and a realtime quantum one-counter

[1] It was shown that 2qcfa's are more powerful than 2pfa's by Ambainis and Watrous [1]. Moreover, Yakaryılmaz and Say [22,23] showed that they can also recognize many interesting languages.

[2] Although the formal definition of 2qcfa given here is a bit different than the ones given in [1,24], all the models are equivalent.

The most general quantum operator is a superoperator, which generalizes stochastic and unitary operators and also includes measurement. Formally, a superoperator $\mathcal{E}$ is composed by a finite number of operation elements, $\mathcal{E} = \{E_1, \ldots, E_k\}$, satisfying that

$$\sum_{i=1}^{k} E_i^\dagger E_i = I, \tag{1}$$

where $k \in \mathbb{Z}^+$ and the indices are the measurement outcomes. When a superoperator, say $\mathcal{E}$, is applied to the quantum register in state $|\psi\rangle$, i.e. $\mathcal{E}(|\psi\rangle)$, we obtain the measurement outcome i with probability $p_i = \langle\widetilde{\psi}_i|\widetilde{\psi}_i\rangle$, where $|\widetilde{\psi}_i\rangle$, *the unconditional state vector*, is calculated as $|\widetilde{\psi}_i\rangle = E_i|\psi\rangle$ and $1 \leq i \leq k$. (Note that using unconditional state vector simplifies calculations in many cases.) If the outcome i is observed ($p_i > 0$), the new state of the system is obtained by normalizing $|\widetilde{\psi}_i\rangle$, which is $|\psi_i\rangle = \frac{|\widetilde{\psi}_i\rangle}{\sqrt{p_i}}$. Moreover, as a special operator, the quantum register can be initialized to a predefined quantum state. This initialize operator, which has only one outcome, is denoted $\acute{\mathcal{E}}$. In this paper, the entries of quantum operators are defined by rational numbers. Thus the probabilities of the outcomes are always rational numbers.

Fig. 1. The details of superoperators [20]

automaton (rt-qca) [18]: The 2qcfa part governs the computation, and access the counter through the rt-qca by feeding some input to it and also observing the outcomes. We will use this model in one of our results (Theorem 3), and our simple definition will also simplify the proof.

Interactive Proof Systems. In this part, we provide the necessary background, based on [7,3], for the proof systems. An interactive proof system (IPS) consists of a prover (P) and a verifier (V). The verifier is a restricted/resource-bounded machine. The classical states of the verifier are partitioned into reading, communication, and halting (accepting or rejecting) states, and it has a special communication cell for communicating with the prover, where the capacity of the cell is finite.

The one-step transitions of the verifier can be described as follows. When in a reading state, the verifier implements its standard transition. When in a communication symbol, the verifiers writes a symbol on the communication cell with respect to the current state. Then, in response, the prover writes a symbol in the cell. Based on the state and the symbol written by prover, the verifier defines the next state of the verifier. Note that the communication is always classical even though the verifier can use some quantum memory.

The prover P is specified by a prover transition function, which determines the response of the prover to the verifier based on the input and the verifier's communication history until then. Note that this function does not need to be *computable*.

The prover-verifier pair (P, V) is an IPS for language $\mathtt{L}$ with error probability $\epsilon < \frac{1}{2}$ if (i) for all $x \in \mathtt{L}$, the probability that (P, V) accepts x is greater than $1-\epsilon$, (ii) for all $x \notin \mathtt{L}$, and all provers P^*, the probability that (P^*, V) rejects x is greater than $1-\epsilon$. These conditions are known as completeness and soundness, respectively.

An Arthur-Merlin (AM) proof system is a special case of IPS such that after each probabilistic or quantum operation, the outcome is automatically written on the communication cell, and so the prover can have complete information about the computation of the verifier.[3] We also refer them as public proof systems.

$\mathsf{IP}(\mathsf{v})$ represents the class of languages having an IPS with some error probability $\epsilon < \frac{1}{2}$, where the verifier is v-type. Moreover, $\mathsf{IP}^*(\mathsf{v})$ is a subset of $\mathsf{IP}(\mathsf{v})$ providing that each language in $\mathsf{IP}^*(\mathsf{v})$ has an IPS for any error bound. $\mathsf{AM}(\mathsf{v})$ and $\mathsf{AM}^*(\mathsf{v})$ are defined similarly.

3 Counter Automata Verifiers for Decidable Languages

In this section, we will present four different protocols for decidable languages. We begin with the classical verifiers.

Theorem 1. $\mathsf{IP}^*(\mathsf{2pca}) = \mathsf{DECIDABLE}$.

Proof. [4] The relation $\mathsf{IP}(\mathsf{2pca}) \subseteq \mathsf{DECIDABLE}$ is trivial. We will give the proof for the other direction. The proof idea is inspired from the protocol given by Condon and Lipton [4].

Let $\mathtt{L}$ be a decidable language. Then there exists a 1d2ca $\mathcal{D}$, which halts on every input, recognizing $\mathtt{L}$ [13]. Any configuration of $\mathcal{D}$ on an input, say $x \in \Sigma^*$, can be represented by (s, i, u, v), where s is the state, i is the head position, and u and v are contents of the counters.

We will describe an IPS (P, V) for $\mathtt{L}$ by giving a simulation of $\mathcal{D}$ on the given input, say x, where V is a $2pca$. If V accesses the status of both counters in each step, then it can easily simulate $\mathcal{D}$ on x by tracing the state and the head position updates of $\mathcal{D}$. The prover can provide the contents of the counters for each step. But, the verifier should be careful about the cheating provers. For this purpose, V can use its counter. That is, in each step, the verifier can determine the changes on the counters, and so can compare the current value and the next value of a counter. Therefore, before starting the simulation, V equiprobably selects a counter of $\mathcal{D}$ to test the changes on it. Moreover, V should also compare the contents of the selected counter not only for $(2i-1)^{th}$ and $(2i)^{th}$ steps but also for $(2i)^{th}$ and $(2i+1)^{th}$ steps, where $i \geq 1$. Thus, V can start the comparisons from either the first step or the second step, which can also be decided equiprobably at the beginning of the simulation. Therefore,

[3] Note that all the verifiers defined in the paper are allowed to use only rational number transitions.

[4] One of the anonymous reviewers proposed a simpler proof which will appear in the full version of the paper.

we can identify four comparison strategies, i.e. C_i^j (V selects the i^{th} counter of $\mathcal{D}$ and starts to compare from the step-j), where $1 \leq i, j \leq 2$. This is the base strategy of V. However, as described below, it is not sufficient to define a protocol for any error bound.

The simulation of $\mathcal{D}$ on x by (P, V) is executed in an infinite loop. In each round, a new simulation is started. V requests the contents of the counters for each step from the prover. Let w be the string obtained from the prover in a single round. The verifier expects w as $a^{u_1}b^{v_1}\#a^{u_2}b^{v_2}\#\cdots\#a^{u_t}b^{v_t}\#$ such that u_j (v_j) is the content of the first (the second) counter after j^{th} step, where $1 \leq j \leq t$ and $t \geq 1$. On the other hand, there are four disjoint cases for w as listed below:

- (C1) w is of the form $\boxed{(a^*b^*\#)^+}$,
- (C2) there is an a after b in w,
- (C3) w is infinite and of the form $\boxed{(a^*b^*\#)^+aaa\cdots}$ or $\boxed{(a^*b^*\#)^+a^*bbb\cdots}$, or
- (C4) w is infinite and of the form $\boxed{(a^*b^*\#)(a^*b^*\#)(a^*b^*\#)\cdots}$.

It is obvious that V can check C2 deterministically, and reject the input if there exists an a after b in w. In other words, such a round is certainly terminated with the decision of rejection. Therefore, in the remaining part, we assume that w satisfies one of the other cases.

If w is valid (correct), then V can exactly simulate $\mathcal{D}$ on x. Otherwise, V may give a wrong decision. Moreover, V may also enter an infinite loop. Note that since P is honest and provides the valid w, we specifically focus on the strategies of cheating provers on the nonmembers: In each round, V should deal with infinite loops and should also guarantee that, for the nonmembers, the probability of accepting the input, which can only be given based on the simulation, is sufficiently smaller than the probability of rejecting the input due to detecting the defects on w. The aforementioned (base) strategy of V is quite strong, and so any invalid w is detected by at least one of C_i^j. But the prover can still mislead the verifier in the other choices. Thus, the defect can be detected with a probability at least $\frac{1}{4}$, and the verifier can follow an invalid w with a probability at most $\frac{3}{4}$. Therefore, when V is convinced to accept the input, it gives the decision of acceptance with probability $\frac{1}{k}$, and terminates the current round with the remaining probability $1 - \frac{1}{k}$. So, the total accepting probability of an invalid computation ($\frac{3}{4k}$) can be sufficiently small compared to the rejecting probability due to the defect ($\frac{1}{4}$) by setting k to an appropriate value. However, there is still the problem of infinite loop. We can solve this problem by terminating the round with probability $\frac{1}{2}$ after obtaining a symbol w from the prover. Thus, any infinite loop can be terminated with probability 1. Although the probability of making decisions is dramatically decreased due to this new restart strategy, the ratio of accepting and rejecting probabilities for the nonmembers can still be preserved since any decision of acceptance can only be given after a defect.

Now, we can analyse the overall protocols. Let l be the length of the valid w. If $x \in \mathtt{L}$, then V accepts x with probability $\frac{1}{k2^t}$ in each round, and so it is

accepted with probability 1. If $x \notin \mathtt{L}$, if there is no defect in a round, then it is rejected with probability $\frac{1}{2^t}$. If there is a defect in a round, then, it is detected by V after obtaining $(l_1)^{th}$ symbol of w, where $l_1 \leq l$. Moreover, the input can be accepted by V after obtaining $l_2 \geq l_1$ symbol of w. Then, the input is rejected with a probability at least $\frac{1}{4 \cdot 2^{l_1}}$, and it is accepted with a probability at most $\frac{3}{4k \cdot 2^{l_2}}$. Thus, the input is rejected with high probability depending on the value of k. Moreover, the protocol is always terminated with probability 1. □

In the protocol above, if we allow the infinite loops, we can still obtain an IPS for any decidable language by using the base strategy. Besides, it is sufficient to simulate $\mathcal{D}$ once. Thus, the verifier does not need to move its input head to the left.

Corollary 1. $\mathsf{IP}(\mathsf{1pca}) = \mathsf{DECIDABLE}$.

Proof. The input is rejected with probability $\frac{3}{7}$ by the verifier at the beginning of the computation. Then the verifier follows its base strategy once. Therefore, the members are accepted with probability $\frac{4}{7}$ by the help of a honest prover, and the non-members are rejected with a probability at least $\frac{3}{7} + \frac{4}{7}\left(\frac{1}{4}\right) = \frac{4}{7}$. The error bound is $\frac{3}{7} < \frac{1}{2}$. □

We continue with the quantum verifiers. Recently, Yakaryılmaz [20] showed that for each Turing-recognizable language, say L, there exists an AM proof systems with a 2qcfa verifier, say (P, V), such that each $x \in \mathtt{L}$ is accepted by V exactly and each $x \notin \mathtt{L}$ is accepted with a small probability. Such proof systems are also known as weak-IPS [7,3].

By combining the protocol given in [20] with the first protocol given above (given in the proof of Theorem 1), we present two more protocols for decidable languages. A review of the protocol given in [20] is as follows. Let L be a decidable language, and $\mathcal{D}$ be a DTM, which halts on every input, recognizing L. In this protocol (P, V) simulates the computation of $\mathcal{D}$ on a given input, say x. In an infinite loop, V requests the computation of $\mathcal{D}$ on x, as $w = c_1\$\$c_2\$\$c_3\$\$\cdots$, where $c_{i>0}$'s are some configurations of $\mathcal{D}$ on x and c_1 is the initial one. If $x \in \mathtt{L}$, P provides the valid w, and V accepts the input with some probability in each round, then it is accepted exactly. If $x \notin \mathtt{L}$, then the input is always rejected with a bigger probability than the accepting probability in a single round as long as the prover sends \$\$ symbols. If the prover does not send \$\$ after some point, the round is still terminated with probability 1, but probably with no decision. This is why the system is "weak". From a given configuration, the length of the next valid configuration can be easily determined, which can be differ at most one. So, if the verifier uses a classical counter, it can detect when the prover does not send \$\$ with some probability, i.e. instead of terminating the round with "no decision", the round is terminated with some nonzero "rejecting" probability. Similar to the first protocol given above, the verifier equiprobably

decides to compare the lengths of which configurations, i.e. $(2i-1)^{th}$ and $(2i)^{th}$ configurations or $(2i)^{th}$ and $(2i+1)^{th}$ configurations, at the beginning of each round, where $i \geq 1$. Although the protocol given in [20] is a public one, the computations on the classical counter must be hidden from the prover in the new protocol. We can formalize this result as follows.

Theorem 2. $\mathsf{IP}^*(\mathsf{2qcca}) = \mathsf{DECIDABLE}$.

It is a well-known fact that the simulation of a DTM by a 1d2ca is space (and time) inefficient [11]. Therefore, we can say that for the same language, the latter protocol (Theorem 2) can be more space efficient than the former protocol (Theorem 1) for the members of the language since the latter protocol directly simulates a DTM. This can be seen as an advantage of using a few quantum states.

Corollary 2. *For any language recognized by a $s(n)$-space DTM, there exists an IPS with a 2qcca verifier such that the verifier uses $s(n)$-space on its counter for the members.*

Our fourth result is to make the third protocol (given for 2qcca) *public*.

Theorem 3. $\mathsf{AM}^*(\mathsf{2qca}) = \mathsf{DECIDABLE}$. *(See [19] for the proof)*

4 Counter Machines as Recognizer

In this section, we examine the bounded-error computational powers of 2pca's and 2qcca's as language recognizers. We begin with a useful lemma and a lower bound to 2PCA.

Lemma 1. *Let $\mathcal{N} = (S, \Sigma, \delta, s_1, s_a, s_r)$ be a 2nca, x be an input, and $M = |S||\tilde{x}|$. If $s \in S$ is reachable from s_1 by $\mathcal{N}$ on x, then there is a path of length no more than M^2 from $(s_1, 1, 0)$ to (s, i, u) for some $1 \leq i \leq |\tilde{x}|$ and $u \leq M$ such that the value of counter never exceeds M. (See [19] for the proof)*

Theorem 4. *Let* L *be a language recognized by a 2nca $\mathcal{N}$, then there exists a 2pca $\mathcal{P}$ recognizing* L *with one-sided bounded-error. (See [19] for the proof)*

Remark that if a language is recognized by a 2nca, then it is recognized by a 2pca with one-sided unbounded error, vice versa. Therefore, in case of one-sided error, the language recognition power of 2pca's remain the same.

Corollary 3. *A language is recognized by a 2nca if and only if it is recognized by a 2pca with one-sided bounded-error.*

Moreover, due to the fact that $\mathsf{2DCA} \subsetneq \mathsf{2NCA}$ [2] and Theorem 4, we can say that bounded-error 2pca's are more powerful than 2dca's.

Corollary 4. $\mathsf{2DCA} \subsetneq \mathsf{2NCA} \subseteq \mathsf{2PCA}$.

Now, we turn our attention to the language recognition power of bounded-error 2qcca's. We begin with the definitions of two languages: $\texttt{TWIN} = \{u\#u \mid u \in \{a,b\}^*\}$ and $\texttt{EXIST-TWIN} = \{u\#v_1\#\cdots\#v_k \mid k \geq 1, u \in \{a,b\}^*, v_i \in \{a,b\}^* (1 \leq i \leq k), \text{and } \exists i \in \{1,\ldots,k\}(u = v_i)\}$. Ďuriš and Galil [5,6] showed that EXIST-TWIN cannot be recognized by any 2dca. Moreover, Chrobak stated [2] that EXIST-TWIN does not seem to be in 2NCA. We show that 2qcca's can recognize EXIST-TWIN for any error bound by using a new technique which uses some 2qcfa's as black boxes. In the next section, we will also show that this programming technique can also be used by 2qcfa's having a pebble.

Theorem 5. EXIST-TWIN *can be recognized by a 2qcca $\mathcal{Q}$ with bounded error.*

Proof. Recently, Yakaryılmaz and Say [22,23] showed that TWIN can be recognized by any 2qcfa for any negative one-sided error bound. Let $\mathcal{Q}_{\texttt{TWIN}}$ be such a 2qcfa for error bound $\frac{1}{5}$. (We also refer the reader to [19] for the details of $\mathcal{Q}_{\texttt{TWIN}}$.) We will use $\mathcal{Q}_{\texttt{TWIN}}$ as a black box. As a special remark, $\mathcal{Q}_{\texttt{TWIN}}$ reads the input from left to right in an infinite loop.

Let x be an input. We assume that x is of the form $u\#v_1\#\cdots\#v_k$ for some $k \geq 1$, where $u, v_i \in \{a,b\}^*$ $(1 \leq i \leq k)$. Otherwise, it is deterministically rejected. The idea behind the algorithm is that $\mathcal{Q}$ selects v_1, and then simulates $\mathcal{Q}_{\texttt{TWIN}}$ by feeding $u\#v_1$ as the input. $\mathcal{Q}_{\texttt{TWIN}}$ gives the decision of rejection only if $u \neq v_1$. So, whenever $\mathcal{Q}_{\texttt{TWIN}}$ gives the decision of rejection, then $\mathcal{Q}$ continues by selecting v_2, and so on. We call each such selection, in which $\mathcal{Q}$ gives the decision of rejection, *completed*. If $x \notin$ EXIST-TWIN, then $\mathcal{Q}$ can obtain k completed-selection with some nonzero probability. Otherwise, this probability becomes zero since $\mathcal{Q}$ can obtain at most $(k-1)$ completed-selection. So, by accepting the input with some carefully tuned probability, we can obtain the desired machine. Note that $\mathcal{Q}$ always remembers its selection by using its counter. The pseudo code of the algorithm is given below.

FOR $i = 1$ TO k (v_i is selected)
  RUN $\mathcal{Q}_{\texttt{TWIN}}$ on $x' = u\#v_i$
    IF $\mathcal{Q}_{\texttt{TWIN}}$ accepts x' THEN **TERMINATE** FOR-LOOP
    IF $\mathcal{Q}_{\texttt{TWIN}}$ rejects x' AND $i = k$ THEN **REJECT** the input
END FOR
ACCEPT x with probability $\left(\frac{1}{5}\right)^k$
RESTART the algorithm

As can be seen from the pseudo code, the algorithm is actually executed in an infinite loop. We begin with analysing a single round of the algorithm. It is straightforward that if $x \in$ EXIST-TWIN, the input is accepted with probability $\left(\frac{1}{5}\right)^k$, and it is rejected with zero probability. If $x \notin$ EXIST-TWIN, $\mathcal{Q}_{\texttt{TWIN}}$ halts with the decision of rejection with a probability at least $\frac{4}{5}$ in each iteration of the for-loop, and so the input is rejected with a probability at least $\left(\frac{4}{5}\right)^k$, and it is accepted with a probability no more than $\left(\frac{1}{5}\right)^k$. Therefore, the members are accepted exactly. Since the rejecting probability is at least 4^k times bigger than the accepting probability in a single round, the input is rejected with a probability at least $\frac{4}{5}$. The error bound can be reduced to any desired value. □

Note that since PTM's cannot recognize TWIN in sublogarithmic space [21], we cannot use the same idea for 2pca's. In fact, we believe that EXIST-TWIN $\notin$ 2PCA.

Another interesting language is a unary one: USQUARE $= \{b^{n^2} \mid n \geq 1\}$. It is still not known whether USQUARE can be recognized by 2dca's [14,15]. Since 2qcfa's can recognize SQUARE $= \{a^n b^{n^2} \mid n \geq 1\}$ with negative one-sided bounded error [23], we can also obtain the following result.

Theorem 6. USQUARE *can be recognized by a 2qcca with bounded error. (See [19] for the proof)*

5 Pebble Automata

A 1-pebble finite automaton has the capability of placing a pebble to at most one tape square, of sensing whether a tape square has a pebble or not, and removing the pebble from the marked tape square. The algorithms given for 2qcca's in the previous section can also be implemented by 1-pebble 2qcfa. In these algorithms, the counter is actually used to remember some positions on the input. A pebble can also be used in the same way by marking those positions. Therefore, we can conclude that EXIST-TWIN and USQUARE can be recognized by some 1-pebble 2qcfa's with bounded error.

Similarly, we can also show that SIAM-TWINS $= \{uu \mid u \in \{a,b\}^*\}$ can be recognized by 1-pebble 2qcfa's. This is an interesting language since it cannot be recognized by any 1-pebble NTM using sublogarithmic space [10].

Theorem 7. EXIST-TWIN *can be recognized by a 1-pebble 2qcfa $\mathcal{Q}$ with bounded error. (See [19] for the proof)*

Ravikumar [17] showed that 1-pebble 2pfa's are more powerful than 2pfa's in the unbounded error case by giving an unbounded-error 1-pebble 2pfa for non-stochastic language CENTER $= \{ubv \mid u,v \in \{a,b\}^*$ and $|u| = |v|\}$. We show the same separation for the bounded-error machines as conjectured by him [16].

Theorem 8. *1-pebble 2pfa's are more powerful than 2pfa's in the bounded error case. (See [19] for the proof)*

Acknowledgements. We would like to thank A. C. Cem Say not only for his useful comments on a draft of this paper but also for many helpful discussions on the subject matter of this paper; Juraj Hromkovič and Holger Petersen for kindly answering our questions; and, Holger Petersen for giving an idea used in the proof of Lemma 1. We are grateful to the anonymous reviewers.

References

1. Ambainis, A., Watrous, J.: Two–way finite automata with quantum and classical states. Theoretical Computer Science 287(1), 299–311 (2002)
2. Chrobak, M.: Nondeterminism is essential for two-way counter machines. In: Chytil, M.P., Koubek, V. (eds.) MFCS 1984. LNCS, vol. 176, pp. 240–244. Springer, Heidelberg (1984)

3. Condon, A.: Complexity Theory: Current Research, chap. The complexity of space bounded interactive proof systems, pp. 147–190. Cambridge University Press (1993)
4. Condon, A., Lipton, R.J.: On the complexity of space bounded interactive proofs (extended abstract). In: FOCS 1989: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, pp. 462–467 (1989)
5. Ďuriš, P., Galil, Z.: Fooling a two-way automaton or one pushdown store is better than one counter for two way machines (preliminary version). In: STOC 1981: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, pp. 177–188 (1981)
6. Duriš, P., Galil, Z.: Fooling a two way automaton or one pushdown store is better than one counter for two way machines. Theoretical Computer Science 21, 39–53 (1982)
7. Dwork, C., Stockmeyer, L.: Finite state verifiers I: The power of interaction. Journal of the ACM 39(4), 800–828 (1992)
8. Feige, U., Shamir, A.: Multi-oracle interactive protocols with space bounded verifiers. In: Structure in Complexity Theory Conference, pp. 158–164 (1989)
9. Hromkovic, J., Schnitger, G.: On the power of randomized multicounter machines. Theoretical Computer Science 330(1), 135–144 (2005)
10. Inoue, A., Ito, A., Inoue, K., Okazaki, T.: Some properties of one-pebble Turing machines with sublogarithmic space. Theoretical Computer Science 341(1-3), 138–149 (2005)
11. van Emde Boas, P.: Machine models and simulations. In: Handbook of Theoretical Computer Science, vol. A, pp. 1–66 (1990)
12. Minsky, M.: Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. Annals of Mathematics 74(3), 437–455 (1961)
13. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
14. Petersen, H.: Two-way one-counter automata accepting bounded languages. SIGACT News 25(3), 102–105 (1994)
15. Petersen, H.: Private communication (June 2012)
16. Ravikumar, B.: Some observations on 2-way probabilistic finite automata. In: Shyamasundar, R.K. (ed.) FSTTCS 1992. LNCS, vol. 652, pp. 392–403. Springer, Heidelberg (1992)
17. Ravikumar, B.: On some variations of two-way probabilistic finite automata models. Theoretical Computer Science 376(1-2), 127–136 (2007)
18. Say, A.C.C., Yakaryılmaz, A.: Quantum counter automata. International Journal of Foundations of Computer Science 23(5), 1099–1116 (2012)
19. Yakaryılmaz, A.: One-counter verifiers for decidable languages. Tech. Rep. ECCC: TR12-091 (2012)
20. Yakaryılmaz, A.: Public-qubits versus private-coins. Tech. Rep. ECCC: TR12-130 (2012)
21. Yakaryılmaz, A., Freivalds, R., Say, A.C.C., Agadzanyan, R.: Quantum computation with write-only memory. Natural Computing 11(1), 81–94 (2012)
22. Yakaryılmaz, A., Say, A.C.C.: Languages recognized by nondeterministic quantum finite automata. Quantum Information and Computation 10(9&10), 747–770 (2010)
23. Yakaryılmaz, A., Say, A.C.C.: Succinctness of two-way probabilistic and quantum finite automata. Discrete Mathematics and Theoretical Computer Science 12(2), 19–40 (2010)
24. Yakaryılmaz, A., Say, A.C.C.: Unbounded-error quantum computation with small space bounds. Information and Computation 279(6), 873–892 (2011)
25. Yamasaki, T., Kobayashi, H., Imai, H.: Quantum versus deterministic counter automata. Theoretical Computer Science 334(1-3), 275–297 (2005)

More on the Complexity of Quantifier-Free Fixed-Size Bit-Vector Logics with Binary Encoding*

Andreas Fröhlich, Gergely Kovásznai, and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria

Abstract. Bit-precise reasoning is important for many practical applications of Satisfiability Modulo Theories (SMT). In recent years, efficient approaches for solving fixed-size bit-vector formulas have been developed. From the theoretical point of view, only few results on the complexity of fixed-size bit-vector logics have been published. Most of these results only hold if unary encoding on the bit-width of bit-vectors is used.

In previous work [1], we showed that binary encoding adds more expressiveness to bit-vector logics, e.g. it makes fixed-size bit-vector logic without uninterpreted functions nor quantification NExpTime-complete.

In this paper, we look at the quantifier-free case again and propose two new results. While it is enough to consider logics with *bitwise operations*, *equality*, and *shift by constant* to derive NExpTime-completeness, we show that the logic becomes PSpace-complete if, instead of *shift by constant*, only *shift by 1* is permitted, and even NP-complete if *no shifts* are allowed at all.

1 Introduction

Bit-precise reasoning over bit-vector logics is important for many practical applications of Satisfiability Modulo Theories (SMT), particularly for hardware and software verification. Examples of state-of-the-art SMT solvers with support for bit-precise reasoning are Boolector, MathSAT, STP, Z3, and Yices.

Syntax and semantics of *fixed-size bit-vector logics* do not differ much in the literature [2–6]. Concrete formats for specifying bit-vector problems also exist, e.g. the SMT-LIB format [7] or the BTOR format [8].

Working with *non-fixed-size* bit-vectors has been considered for instance in [4, 9], and more recently in [10], but is not the focus of this paper. Most industrial applications (and examples in the SMT-LIB) have fixed bit-width.

We investigate the *complexity* of solving *fixed-size bit-vector formulas*. Some papers propose such complexity results, e.g. in [3] the authors consider quantifier-free bit-vector logic and give an argument for the NP*-hardness* of its satisfiability problem. In [5], a sublogic of the previous one is claimed to be NP*-complete*. Interestingly, in [11] there is a claim about the full quantifier-free bit-vector

* Supported by FWF, NFN Grant S11408-N23 (RiSE).

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 378–390, 2013.

logic without uninterpreted functions (QF_BV) being NP-complete, however, the proposed decision procedure confirms this claim only if the bit-widths of the bit-vectors in the input formula are written/encoded in *unary* form. In [12, 13], the *quantified* case is addressed, and the satisfiability problem of this logic with uninterpreted functions (UFBV) is proved to be NExpTime-*complete*. Again, the proof only holds if we assume unary encoded bit-widths. In practice, a more natural and exponentially more succinct *logarithmic* encoding is used, such as in the SMT-LIB, the BTOR, and the Z3 format.

In previous work [1], we already investigated how complexity varies if we consider either a unary or a logarithmic, actually without loss of generality, *binary encoding*. Apart from this, we are not aware of any work that investigates how the particular encoding of the bit-widths in the input affects complexity (as an exception, see [14, Page 239, Footnote 3]). Tab. 1 summarizes the completeness results we obtained in [1].

Table 1. Completeness results of [1] for various bit-vector logics and encodings

		quantifiers			
		no		*yes*	
		uninterpreted functions		uninterpreted functions	
		no	*yes*	*no*	*yes*
encoding	*unary*	NP	NP	PSpace	NExpTime
	binary	NExpTime	NExpTime	?	2-NExpTime

In this paper, we revisit QF_BV2, the quantifier-free case with binary encoding and without uninterpreted functions. We then put certain restrictions on the operations we use (in particular on the *shift* operation). As a result, we obtain two new sublogics which we show to be PSpace-complete resp. NP-complete.

2 Motivation

In practice, state-of-the-art bit-vector solvers rely on rewriting and bit-blasting. The latter is defined as the process of translating a bit-vector resp. word-level description into a bit-level circuit, as in hardware synthesis. The result can then be checked by a (propositional) SAT solver. In [1], we gave the following example (in SMT2 syntax) to point out that bit-blasting is not polynomial in general. It checks commutativity of adding two bit-vectors of bit-width 1000000:

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(assert (distinct (bvadd x y) (bvadd y x)))
```

Bit-blasting such formulas generates huge circuits, which shows that checking bit-vector logics through bit-blasting cannot be considered to be a polynomial reduction. This also disqualifies bit-blasting as a sound way to argue that the decision problem for (quantifier-free) bit-vector logics is in NP. We actually proved in [1], that deciding bit-vector logics, even without quantifiers, is much harder. It turned out to be NExpTime-complete in the general case.

However, in [1] we then also defined a class of *bit-width bounded problems* and showed that under certain restrictions on the bit-widths this growth in complexity can be avoided and the problem remains in NP.

In this paper, we give a more detailed classification of quantifier-free fixed-size bit-vector logics by investigating how complexity varies when we restrict the operations that can be used in a bit-vector formula. We establish two new complexity results for restricted bit-vector logics and bring together our previous results in [1] with work on linear arithmetic on non-fixed-size bit-vectors [10, 15] and work on the reduction of bit-widths [16, 17]. The formula in the given example only contains bitwise operations, equality, and addition. Solving this kind of formulas turns out to be PSpace-complete.

3 Definitions

We assume the usual syntax for (quantifier-free) bit-vector logics, with a restricted set of bit-vector operations: bitwise operations, equality, and (left) shift by constant.

Definition 1 (Term). *A bit-vector term t of bit-width n ($n \in \mathbb{N}$, $n \geqslant 1$) is denoted by $t^{[n]}$. A term is defined inductively as follows:*

	term	condition	bit-width
bit-vector constant:	$c^{[n]}$	$c \in \mathbb{N}$, $0 \leqslant c < 2^n$	n
bit-vector variable:	$x^{[n]}$	x *is an identifier*	n
bitwise negation:	$\sim t^{[n]}$	$t^{[n]}$ *is a term*	n
bitwise and/or/xor: $\bullet \in \{\&, \mid, \oplus\}$	$\left(t_1{}^{[n]} \bullet t_2{}^{[n]}\right)$	$t_1{}^{[n]}$ *and* $t_2{}^{[n]}$ *are terms*	n
equality:	$\left(t_1{}^{[n]} = t_2{}^{[n]}\right)$	$t_1{}^{[n]}$ *and* $t_2{}^{[n]}$ *are terms*	1
shift by constant:	$\left(t^{[n]} \ll c^{[n]}\right)$	$t^{[n]}$ *is a term,* $c^{[n]}$ *is a constant*	n

We also define how to measure the size of bit-vector expressions:

Definition 2 (Size). *The size of a bit-vector term $t^{[n]}$ is denoted by $|t^{[n]}|$ and is defined inductively as follows:*

	term	size
natural number:	$enc(n)$	$\lceil \log_2 (n+1) \rceil + 1$
bit-vector constant:	$\lvert c^{[n]} \rvert$	$enc(c) + enc(n)$
bit-vector variable:	$\lvert x^{[n]} \rvert$	$1 + enc(n)$
bitwise negation:	$\lvert \sim t^{[n]} \rvert$	$1 + \lvert t^{[n]} \rvert$
binary operations: $\bullet \in \{\&, \lvert, \oplus, =, \ll\}$	$\lvert \left(t_1{}^{[n]} \bullet t_2{}^{[n]}\right) \rvert$	$1 + \lvert t_1{}^{[n]} \rvert + \lvert t_2{}^{[n]} \rvert$

A bit-vector term $t^{[1]}$ is also called a *bit-vector formula*. We say that a bit-vector formula is in *flat form* if it does not contain nested equalities. It is easy to see that any bit-vector formula can be translated to this form with only linear growth in the number of variables. In the rest of the paper, we may omit parentheses in a formula for the sake of readability.

Let Φ be a bit-vector formula and α an assignment to the variables in Φ. We use the notation $\alpha(\Phi)$ to denote the evaluation of Φ under α, with $\alpha(\Phi) \in \{0, 1\}$. α satisfies Φ if and only if $\alpha(\Phi) = 1$. We define three different bit-vector logics:

- QF_BV2$_{\ll c}$: bitwise operations, equality, and shift by any constant are allowed
- QF_BV2$_{\ll 1}$: bitwise operations, equality, and shift by only $c = 1$ are allowed
- QF_BV2$_{bw}$: only bitwise operations and equality are allowed

Obviously, QF_BV2$_{bw} \subseteq$ QF_BV2$_{\ll 1} \subseteq$ QF_BV2$_{\ll c}$. In Sec. 4, we investigate the complexity of the satisfiability problem for these logics:

- QF_BV2$_{\ll c}$ is NExpTime-complete.
- QF_BV2$_{\ll 1}$ is PSpace-complete.
- QF_BV2$_{bw}$ is NP-complete.

Adding uninterpreted functions does not change expressiveness of these logics, since in the quantifier-free case, uninterpreted functions can always be replaced by new variables. To guarantee functional consistency, Ackermann constraints have to be added to the formula. However, even in the worst case, the number of Ackermann constraints is only quadratic in the number of function instances. Without loss of generality, we therefore do not explicitly deal with uninterpreted functions.

4 Complexity Results

Theorem 1. QF_BV2$_{\ll c}$ *is* NExpTime*-complete.*

Proof. The claim directly follows from our previous work in [1]. We informally defined QF_BV2 as the quantifier-free bit-vector logic that uses the common

bit-vector operations as defined for example in SMT-LIB, including bitwise operations, equality, shifts, addition, multiplication, concatenation, slicing, etc., and then showed that QF_BV2 is NExpTime-complete.

Obviously, $\text{QF\_BV2}_{\ll c} \subseteq \text{QF\_BV2}$ and therefore, $\text{QF\_BV2}_{\ll c} \in \text{NExpTime}$. To show the NExpTime-hardness of QF_BV2, we gave a (polynomial) reduction from DQBF (which is NExpTime-complete [18]) to QF_BV2. Since we only used *bitwise operations*, *equality*, and *shift*[1] *by constant* in our reduction, we also immediately get the NExpTime-hardness of $\text{QF\_BV2}_{\ll c}$.

Theorem 2. $\text{QF\_BV2}_{\ll 1}$ *is* PSpace*-complete.*

Proof. In Lemma 1, we give a (polynomial) reduction from QBF (which is PSpace-complete) to $\text{QF\_BV2}_{\ll 1}$. This shows the PSpace-hardness of $\text{QF\_BV2}_{\ll 1}$. In Lemma 2, we then prove that $\text{QF\_BV2}_{\ll 1} \in \text{PSpace}$ by giving a translation from $\text{QF\_BV2}_{\ll 1}$ to (polynomial sized) Sequential Circuits. As pointed out for example in [19], the symbolic reachability problem is PSpace-complete as well.

Lemma 1. QBF *can be (polynomially) reduced to* $\text{QF\_BV2}_{\ll 1}$.

Proof. To show the PSpace-hardness of $\text{QF\_BV2}_{\ll 1}$, we give a polynomial reduction from QBF similar to the one from DQBF to QF_BV2 that we proposed in [1]. For our reduction, we again use the so-called *binary magic numbers* (or magic masks in [20, p. 141]).

Given $m, n \in \mathbb{N}$ with $0 \leqslant m < n$, a binary magic number can be written in the following form:

$$\mathit{binmagic}\,(2^m,\ 2^n) = \overbrace{\underbrace{0\ldots0}_{2^m}\,\underbrace{1\ldots1}_{2^m}\ldots\underbrace{0\ldots0}_{2^m}\,\underbrace{1\ldots1}_{2^m}}^{2^n}$$

Note that in [1], we used *shift by constant* to construct the binary magic numbers, as done in the literature [20]. This is not permitted in $\text{QF\_BV2}_{\ll 1}$. We therefore give an alternative construction using only *bitwise operations*, *equality*, and *shift by 1*:

Given $n > 0$, for all m, $0 \leqslant m < n$, add the following equation to the formula:

$$b'_m{}^{[2^n]} = \left(\bigwedge_{0 \leqslant i < m} b_i{}^{[2^n]} \right) \oplus b_m{}^{[2^n]}$$

Consider all the bit-vector variables $b_0{}^{[2^n]}, \ldots, b_{n-1}{}^{[2^n]}$ as column vectors in a matrix $B^{[2^n \times n]}$ and all the bit-vector variables $b'_0{}^{[2^n]}, \ldots, b'_{n-1}{}^{[2^n]}$ as column vectors in a matrix $B'^{[2^n \times n]}$. If each row of B is interpreted as a number $0 \leqslant c < 2^n$ in binary representation, the corresponding row of B' is equal to $c + 1$.

[1] Note, logical right shifts were used in the proof in [1]. However, by applying negated bit masks throughout the proof, all right shifts can be rewritten as left shifts.

Now, again for all m, $0 \leqslant m < n$, add another constraint:

$$b'_m{}^{[2^n]} = b_m{}^{[2^n]} \ll 1^{[2^n]}$$

Together with the previous n equations, those n constraints force the rows of B to represent an enumeration of all binary numbers $0 \leqslant c < 2^n$. Therefore, the columns of B, i.e. the individual bit-vectors $b_0{}^{[2^n]}, \ldots, b_{n-1}{}^{[2^n]}$, exactly define the binary magic numbers: $\mathit{binmagic}\,(2^m,\ 2^n) := b_m{}^{[2^n]}$.

Of course, all b'_m, for $0 \leqslant m < n$, can be eliminated and the two sets of constraints can be replaced by a single set of constraints:

$$\left(\bigwedge_{0 \leqslant i < m} b_i{}^{[2^n]}\right) \oplus b_m{}^{[2^n]} = b_m{}^{[2^n]} \ll 1^{[2^n]}$$

Now let $\phi := Q.M$ denote a QBF formula with quantifier prefix Q and matrix M. Since ϕ is a QBF formula (in contrast to DQBF in [1]), we know that Q defines a total order on the universal variables. We now assume the universal variables $u_0, \ldots, u_{n-1}$ of ϕ are ordered according to their appearance in Q, with u_0 (resp. u_{n-1}) being the innermost (resp. outermost) variable.

Translate ϕ to a QF_BV2$_{\ll 1}$ formula Φ by eliminating the quantifier prefix and translating the matrix as follows:

Step 1. Replace Boolean constants 0 and 1 with $0^{[2^n]}$ resp. $\sim 0^{[2^n]}$ and logical connectives with corresponding bitwise bit-vector operations (e.g. $\wedge$ with &). Let Φ' denote the formula generated so far. Extend it to the formula $\left(\Phi' = \sim 0^{[2^k]}\right)$.

Step 2. For each universal variable $u_m \in \{u_0, \ldots, u_{n-1}\}$,

1. translate (all the occurrences of) u_m to a new bit-vector variable $U_m{}^{[2^n]}$;
2. in order to assign a binary magic number to $U_m{}^{[2^n]}$, add the following equation (i.e., conjunct it with the current formula):

$$U_m{}^{[2^n]} = \mathit{binmagic}\,(2^m,\ 2^n)$$

Step 3. For an existential variable e depending on $\mathit{Deps}(e) = \{u_m, \ldots, u_{n-1}\}$, with u_m being the innermost universal variable that e depends on,

1. translate (all the occurrences of) e to a new bit-vector variable $E^{[2^n]}$;
2. if $\mathit{Deps}(e) = \varnothing$ add the following equation:

$$(E\ \&\ \sim 1) = (E \ll 1) \tag{1}$$

otherwise, if $m \neq 0$ add the two equations:

$$U'_m = \sim\left((U_m \ll 1) \oplus U_m\right) \tag{2}$$

$$(E\ \&\ U'_m) = \left((E \ll 1)\ \&\ U'_m\right) \tag{3}$$

Note that we omitted the bit-widths in the last equations to improve readability. Each bit position of Φ corresponds to the evaluation of ϕ under a specific assignment to the universal variables $u_0, \ldots, u_{n-1}$, and, by construction of ${U_0}^{[2^n]}, \ldots, {U_{n-1}}^{[2^n]}$, all possible assignments are considered. Eqn. (2) creates a bit-vector ${U'_m}^{[2^n]}$ for which each bit equals to 1 if and only if the corresponding universal variable changes its value from one universal assignment to the next.

Of course, Eqn. (2) does not have to be added multiple times, if several existential variables depend on the same universal variable. Eqn. (3) (resp. Eqn. (1)) ensures that the corresponding bits of $E^{[2^n]}$ satisfy the dependency scheme of ϕ by only allowing the value of e to change if an outer universal variable takes a different value. If $m = 0$, i.e. if e depends on all universal variables, Eqn. (2) evaluates to ${U'_0}^{[2^n]} = 0$, and as a consequence Eqn. (3) simplifies to *true*. Because of this no constraints need to be added for $m = 0$. A similar approach used for translating QBF to Symbolic Model Verification (SMV) can be found in [21]. See also [19] for a translation from QBF to Sequential Circuits.

Lemma 2. $\mathrm{QF\_BV2}_{\ll 1}$ *can be (polynomially) reduced to Sequential Circuits.*

Proof. In [10, 15], the authors give a translation from quantifier-free Presburger arithmetic with bitwise operations (QFPAbit) to Sequential Circuits. We can adopt their approach in order to construct a translation for $\mathrm{QF\_BV2}_{\ll 1}$. The main difference between QFPAbit and $\mathrm{QF\_BV2}_{\ll 1}$ is the fact that bit-vectors of arbitrary, non-fixed, size are allowed in QFPAbit while all bit-vectors contained in $\mathrm{QF\_BV2}_{\ll 1}$ have a fixed bit-width.

Given $\Phi \in \mathrm{QF\_BV2}_{\ll 1}$ in flat form. Let $x^{[n]}, y^{[n]}$ denote bit-vector variables, $c^{[n]}$ a bit-vector constant, and ${t_1}^{[n]}, {t_2}^{[n]}$ bit-vector terms only containing bit-vector variables and bitwise operations. Following [10, 15] we further assume w.l.o.g that Φ only consists of three types of expressions: ${t_1}^{[n]} = {t_2}^{[n]}$, $x^{[n]} = c^{[n]}$, and $x^{[n]} = y^{[n]} \ll 1^{[n]}$, since any $\mathrm{QF\_BV2}_{\ll 1}$ formula can be written like this with only a linear growth in the number of original variables.

We encode each equality in Φ separately into an atomic Sequential Circuit. Compared to [10, 15], two modifications are needed. First, we need to give a translation for $x = y \ll 1$ to Sequential Circuits. This can be done for example by using the Sequential Circuit for $x = 2 \cdot y$ in QFPAbit. However, a direct translation can also easily be constructed.

The second modification relates to dealing with *fixed-size* bit-vectors. Let n be the bit-width of all bit-vectors in a given equality. We extend each atomic Sequential Circuit to include a counter (circuit). The counter initially is set to 0 and is incremented by 1 in each clock cycle up to a value of n.

When the counter reaches a value of n, it does not change anymore and the output of the atomic Sequential Circuit is set to the same value as the output in the previous cycle. A counter like this can be realized with $\lceil \log_2(n) \rceil$ gates, i.e. polynomially in the size of Φ. In contrast to the implementation described in [15], we assume that the input streams for all variables start with the least significant bit. However, as already pointed out by the authors in [15], their

choice was arbitrary and it is no more complicated to construct the circuits the other way round.

Finally, after constructing atomic circuits, their outputs are combined by logical gates following the Boolean structure of Φ, in the same way as for unbounded bit-width in [10, 15]. Due to adding counters, we ensure that for every input stream x_i, only the first n_i bits of x_i influence the result of the whole circuit.

For the proof of Thm. 3, we need the following definition and lemma from [1]:

Definition 3 (Bit-Width Bounded Formula Set [1]). *Given a formula Φ, we denote the* maximal bit-width *in Φ with $max_{bw}(\Phi)$. An infinite set S of bit-vector formulas is* (polynomially) bit-width bounded*, if there exists a polynomial function $p : \mathbb{N} \mapsto \mathbb{N}$ such that $\forall \Phi \in S.\ max_{bw}(\Phi) \leqslant p(|\Phi|)$.*

Lemma 3 ([1]). *$S \in$ NP for any bit-width bounded formula set $S \subseteq$ QF_BV2.*

Theorem 3. *QF_BV2$_{bw}$ is NP-complete.*

Proof. Since *Boolean Formulas* are a subset of QF_BV2$_{bw}$, NP-hardness follows directly. To show that QF_BV2$_{bw} \in$ NP, we give a reduction from QF_BV2$_{bw}$ to a *bit-width bounded* set of formulas. The claim then follows from Lemma 3.

Given a formula $\Phi \in$ QF_BV2$_{bw}$ in flat form. If Φ contains any constants $c^{[n]} \neq 0^{[n]}$, we remove those constants in a (polynomial) pre-processing step. Let ${c_{max}}^{[n]} = b_{k-1} \ldots b_1 b_0$ be the largest constant in Φ denoted in binary representation with $b_{k-1} = 1$ and arbitrary bits $b_{k-2}, \ldots, b_0$. We now replace each equality ${t_1}^{[m]} = {t_2}^{[m]}$ in Φ with

$$({t_{1,k'-1}}^{[1]} = {t_{2,k'-1}}^{[1]}) \ \&\ \ldots\ \&\ ({t_{1,0}}^{[1]} = {t_{2,0}}^{[1]})$$

where $k' = min\{m, k\}$, and, if $m > k$, we additionally add

$$\&\ ({t_{1,hi}}^{[m-k]} = {t_{2,hi}}^{[m-k]})$$

For $0 \leqslant i < k$, we use $({t_{1,i}}^{[1]} = {t_{2,i}}^{[1]})$ to express the ith row of the original equality. All occurrences of a variable $x^{[m]}$ are replaced with a new variable ${x_i}^{[1]}$. All occurrences of a constant $c^{[m]}$ are replaced with $0^{[1]}$ if the ith bit of the constant is 0, and by $\sim 0^{[1]}$ otherwise.

In a similar way, if $m > k$, $({t_{1,hi}}^{[m-k]} = {t_{2,hi}}^{[m-k]})$ represents the remaining $(m-k)$ rows of the original equality corresponding to the most significant bits. All occurrences of a variable $x^{[m]}$ are replaced with a new variable ${x_{hi}}^{[m-k]}$ and all occurrences of a constant $c^{[m]}$ are replaced with $0^{[m-k]}$. Since this pre-processing step is logarithmic in the value of c_{max}, it is polynomial in $|\Phi|$. Without loss of generality, we now assume that Φ does not contain any bit-vector constants different from $0^{[n]}$.

We now construct a formula Φ' by reducing the bit-widths of all bit-vector terms in Φ. Each term $t^{[n]}$ in Φ with bit-width n is replaced with a term $t^{[n']}$, with $n' := min\{n, |\Phi|\}$. Apart from this, Φ' is exactly the same as Φ. As a consequence, $max_{bw}(\Phi') \leqslant |\Phi|$. The set of formulas constructed in this way is bit-width bounded according to Def. 3.

To complete our proof, we now have to show that the proposed reduction is sound, i.e. out of every satisfying assignment to the bit-vector variables $x_1{}^{[n_1]}, \ldots, x_k{}^{[n_k]}$ for Φ we can also construct a satisfying assignment to $x_1{}^{[n'_1]}, \ldots, x_k{}^{[n'_k]}$ for Φ' and vice versa.

It is easy to see that whenever we have a satisfying assignment α' for Φ', we can construct a satisfying assignment α for Φ. This can be done by simply setting all additional bits of all bit-vector variables to the same value as the most significant bit of the corresponding original vector, i.e. by performing a signed extension. Since all equalities still evaluate to the same value under the extended assignment, $\alpha(F) = \alpha'(F')$ for all equalities F (resp. F') of Φ (resp. Φ'). As a direct consequence, $\alpha(\Phi) = \alpha'(\Phi) = 1$.

The other direction needs slightly more reasoning. Given α, with $\alpha(\Phi) = 1$, we need to construct α', with $\alpha'(\Phi') = 1$. Again, we want to ensure that $\alpha'(F') = \alpha(F)$ for all equalities F (resp. F') in Φ (resp. Φ').

In each variable $x_i{}^{[n_i]}$, $i \in \{1, \ldots, k\}$, we are going to select some of the bits. For each equality F with $\alpha(F) = 0$, we select a bit-index as a witness for its evaluation. If $\alpha(F) = 1$, we select an arbitrary bit-index. We then mark the selected bit-index in all bit-vector variables contained in F, as well as in all other bit-vector variables of the same bit-width. Having done this for all equalities, we end up with sets M_i of selected bit-indices, for all $i \in \{1, \ldots, k\}$, where

$$|M_i| \leqslant min\{n_i, |\Phi|\}$$
$$M_i = M_j \qquad \forall j \in \{1, \ldots, k\} \text{ with } n_i = n_j$$

The selected indices contain a witness for the evaluation of each equality. We now add arbitrary further bit-indices, again selecting the same indices in bit-vector variables of the same bit-width, until $|M_i| = min\{n_i, |\Phi|\}\ \forall i \in \{1, \ldots, k\}$.

Finally, we can directly construct α' using the selected indices and get $\alpha'(\Phi') = \alpha(\Phi) = 1$ because of the fact that we included a witness for every equality in our index-selection process. Note, that we only had to choose a specific witness for the case that $\alpha(F) = 0$. For $\alpha(F) = 1$, we were able to choose an arbitrary bit-index because every satisfied equality will trivially still be satisfied when only a subset of all bit-indices is considered.

Remark 1. A similar proof can be found in [16, 17]. While the focus of [16, 17] lies on improving the practical efficiency of SMT-solvers by reducing the bit-width of a given formula before bit-blasting, the author does not investigate its influence on the complexity of a given problem class. In fact, the author claims that bit-vector theories with common operators are NP-complete. As we have already shown in [1], this only holds if unary encoding on the bit-widths is used. However, unary encoding leads to the fact that the given class of formulas remains NP-complete, independent of whether a reduction of the bit-width is possible. While the arguments on bit-width reduction given in [16, 17] still hold for binary encoded bit-vector formulas when only bitwise operators are used, our proof considers the complexity of the problem class.

5 Discussion

The complexity results given in Sec. 4 provide some insight in where the expressiveness of bit-vector logics with binary encoding comes from. While we assume bitwise operations and equality naturally being part of a bit-vector logic, if and to what extent we allow shifts directly determines its complexity. Shifts, in a certain way, allow different bits of a bit-vector to interact with each other. Whether we allow no interaction, interaction between neighbouring bits, or interaction between arbitrary bits is crucial to the expressiveness of bit-vector logics and the complexity of their decision problem.

Additionally, we directly get classifications for various other bit-vector operations: for example, we still remain in PSPACE if we add *linear modular arithmetic* to QF_BV2$_{\ll 1}$. This can be seen by replacing expressions $x^{[n]} = y^{[n]} + z^{[n]}$ by

$$\left(x^{[n]} = y^{[n]} \oplus z^{[n]} \oplus {c_{in}}^{[n]}\right) \;\&\; \left({c_{in}}^{[n]} = {c_{out}}^{[n]} \ll 1^{[n]}\right) \;\&\; \left({c_{out}}^{[n]} = \left(x^{[n]} \;\&\; y^{[n]}\right) \mid \left({c_{in}}^{[n]} \;\&\; y^{[n]}\right) \mid \left(x^{[n]} \;\&\; {c_{in}}^{[n]}\right)\right)$$

with new variables ${c_{in}}^{[n]}, {c_{out}}^{[n]}$, and by splitting multiplication by constant into several multiplications by 2 (resp. shift by 1), similar to [10, 15]. However, this is not surprising since QFPABIT is already known to be PSPACE-complete [15].

More interestingly, we can also extend QF_BV2$_{\ll 1}$ (resp. QFPABIT) by *indexing* (denoted by $x^{[n]}[i]$) without growth in complexity. The counter we introduced in our translation from QF_BV2$_{\ll 1}$ to Sequential Circuits can be used to return the value at a specific bit-index of a bit-vector. Extending QF_BV2$_{\ll 1}$ with additional relational operators like e.g. *unsigned less than* (denoted by $x^{[n]} <_{\mathbf{u}} y^{[n]}$) does not increase complexity, either. For instance, the above expression can be replaced by checking whether $x - y < 0$ holds, which can simply be done by constructing an adder for $x^{[n]} + \left(\sim y^{[n]} + 1^{[n]}\right)$, as shown above, and then check whether overflow occurs, i.e., $\left(y^{[n]} \neq 0^{[n]}\right) \;\&\; \left({c_{out}}^{[n]}[n-1] = 0^{[1]}\right)$.

On the other hand, *slicing* (denoted by $x^{[n]}\,[i:j]$) cannot be added without growth in complexity. To prove this, consider

$$\left(x^{[n]}\,[n-1:c] = y^{[n]}\,[n-c-1:0]\right) \;\&\; \left(x^{[n]}\,[c-1:0] = 0^{[c]}\right)$$

which is equivalent to

$$x^{[n]} = (y^{[n]} \ll c^{[n]})$$

and shows that *slicing* can be used to express shift by constant. Therefore, the resulting logic becomes NEXPTIME-complete. The same result holds for general *multiplication*. We can use

$$x^{[n]} = (y^{[n]} \cdot 2^{c^{[n]}})$$

to replace shift by constant and use exponentiation by squaring to calculate $2^{c^{[n]}}$ with $\lceil \log_2(n) \rceil$ multiplications.

Note that those results only hold for fixed-size bit-vector logics. For example, allowing *multiplication* (in combination with *addition*) makes non-fixed-size bit-vector logics undecidable [22]. We are not aware of any complexity results concerning non-fixed-size bit-vector logics with *slicing* or *shift by constant*.

6 Conclusion

In this paper, we discussed the complexity of fixed-size bit-vector logics with binary encoding on numbers. In contrast to existing literature, except for [1], where usually it is not distinguished between unary or binary encoding, we argued that it is important to make this distinction. Our results apply to the actually much more natural binary encoding as it is also used in standard formats, e.g. in the SMT-LIB format. In previous work [1], we already showed the quantifier-free case of those bit-vector logics to be NExpTime-complete. We now extended our previous work by analyzing the quantifier-free case in more detail and gave two new complexity results.

In particular, we showed that the complexity of deciding quantifier-free bit-vector logics with bitwise operations and equality depends on whether we allow *shift by constant* (QF_BV2$_{\ll c}$), *shift by 1* (QF_BV2$_{\ll 1}$), or *no shifts at all* (QF_BV2$_{bw}$). While deciding QF_BV2$_{\ll c}$ remains NExpTime-complete, we proved that QF_BV2$_{\ll 1}$ is PSpace-complete, and QF_BV2$_{bw}$ even becomes NP-complete.

In addition to the already previously proposed concept of bit-width boundedness, this gives an alternative way to avoid the increase in complexity that comes with binary encoding in the general case. To be more specific for practical logics, we then looked at the effect some other common operations have on this complexity results. We discussed why logics with *addition*, *multiplication by constant*, *indexing*, and *relational operations* still can be decided in PSpace, and showed that allowing *general multiplication* or *slicing* already leads to NExpTime-completeness.

On the one hand, our theoretical results give an argument for using more powerful solving techniques when dealing with bit-vector logics. Currently the most common approach used in state-of-the-art SMT solvers for bit-vectors is based on simple rewriting, bit-blasting, and SAT solving. We have shown this can possibly produce exponentially larger formulas when a logarithmic encoding is used in the input. As already argued in [1], possible candidates for the general case are techniques used in EPR and/or DQBF solvers (see e.g. [23, 24]).

On the other hand, we described various logics that remain in lower complexity classes. For QF_BV2$_{bw}$ this shows the importance of bit-width reduction as proposed in [16, 17] before bit-blasting. For formulas in QF_BV2$_{\ll 1}$ or one of the related classes, only using *shift by 1*, *addition*, *multiplication by constant*, and *indexing*, techniques used in state-of-the-art QBF solvers [25] or symbolic model checking on Sequential Circuits [19] might be of interest.

References

1. Kovásznai, G., Fröhlich, A., Biere, A.: On the complexity of fixed-size bit-vector logics with binary encoded bit-width. In: Proc. SMT 2012, pp. 44–55 (2012)
2. Cyrluk, D., Möller, O., Rueß, H.: An efficient decision procedure for the theory of fixed-sized bit-vectors. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 60–71. Springer, Heidelberg (1997)
3. Barrett, C.W., Dill, D.L., Levitt, J.R.: A decision procedure for bit-vector arithmetic. In: Proc. DAC 1998, pp. 522–527 (1998)
4. Bjørner, N., Pichora, M.C.: Deciding fixed and non-fixed size bit-vectors. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 376–392. Springer, Heidelberg (1998)
5. Bruttomesso, R., Sharygina, N.: A scalable decision procedure for fixed-width bit-vectors. In: Proc. ICCAD 2009, pp. 13–20. IEEE (2009)
6. Franzén, A.: Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT. PhD thesis, University of Trento (2010)
7. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: Version 2.0. In: Proc. SMT 2010, Edinburgh, UK (2010)
8. Brummayer, R., Biere, A., Lonsing, F.: BTOR: bit-precise modelling of word-level problems for model checking. In: Proc. BPR 2008, pp. 33–38. ACM, New York (2008)
9. Ayari, A., Basin, D.A., Klaedtke, F.: Decision procedures for inductive boolean functions based on alternating automata. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 170–186. Springer, Heidelberg (2000)
10. Spielmann, A., Kuncak, V.: Synthesis for unbounded bit-vector arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 499–513. Springer, Heidelberg (2012)
11. Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 358–372. Springer, Heidelberg (2007)
12. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: Efficiently solving quantified bit-vector formulas. In: Proc. FMCAD 2010, pp. 239–246. IEEE (2010)
13. Wintersteiger, C.M.: Termination Analysis for Bit-Vector Programs. PhD thesis, ETH Zurich, Switzerland (2011)
14. Cook, B., Kroening, D., Rümmer, P., Wintersteiger, C.M.: Ranking function synthesis for bit-vector relations. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 236–250. Springer, Heidelberg (2010)
15. Spielmann, A., Kuncak, V.: On synthesis for unbounded bit-vector arithmetic. Technical report, EPFL, Lausanne, Switzerland (February 2012)
16. Johannsen, P.: Reducing bitvector satisfiability problems to scale down design sizes for RTL property checking. In: Proc. HLDVT 2001, 123–128 (2001)
17. Johannsen, P.: Speeding Up Hardware Verification by Automated Data Path Scaling. PhD thesis, CAU Kiel, Germany (2002)
18. Peterson, G.L., Reif, J.H.: Multiple-person alternation. In: Proc. FOCS 1979, pp. 348–363 (1979)
19. Prasad, M.R., Biere, A., Gupta, A.: A survey of recent advances in SAT-based formal verification. STTT 7(2), 156–173 (2005)
20. Knuth, D.E.: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms. Addison-Wesley (2011)

21. Donini, F.M., Liberatore, P., Massacci, F., Schaerf, M.: Solving QBF with SMV. In: Proc. KR 2002, pp. 578–589 (2002)
22. Davis, M., Matijasevich, Y., Robinson, J.: Hilbert's tenth problem: Diophantine equations: positive aspects of a negative solution. In: Proc. Sympos. Pure Mathematics, vol. 28, pp. 323–378 (1976)
23. Fröhlich, A., Kovásznai, G., Biere, A.: A DPLL algorithm for solving DQBF. In: Proc. POS 2012 (2012)
24. Korovin, K.: iProver – an instantiation-based theorem prover for first-order logic (System description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 292–298. Springer, Heidelberg (2008)
25. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 158–171. Springer, Heidelberg (2010)

Composition with Algebra at the Background*

On a Question by Gurevich and Rabinovich on the Monadic Theory of Linear Orderings

Thomas Colcombet

Liafa/Cnrs/Université Denis Diderot, Paris 7
thomas.colcombet@liafa.univ-paris-diderot.fr

Abstract. Gurevich and Rabinovich raised the following question: given a property of a set of rational numbers definable in the real line by a monadic second-order formula, is it possible to define it directly in the rational line? In other words, is it true that the presence of reals at the background do not increase the expressive power of monadic second-order logic?

In this paper, we answer positively this question. The proof in itself is a simple application of classical and more recent techniques. This question will guide us in a tour of results and ideas related to monadic theories of linear orderings.

1 In Which the Legacy Is Acknowledged

Büchi, Elgot, Kleene, Rabin, Scott, Shelah, Schützenberger, Trakhtenbrot and others shaped the notion of regular languages as we know it. In less than two decades, a beautiful theory involving computability, logic, algebra and topology has emerged. Today's researcher still walk on this path and are far from its end.

Most of the attention in this theory is put on **monadic second-order logic**. Monadic logic (we will drop the second-order from now) is the extension of first-order logic with the possibility to quantify over sets of elements. For comparison, full second-order logic would allow to quantify over relations of all arities, while in monadic logic, only 1-ary relations are allowed, and these can be interpreted as sets. 1-ary relations are called **monadic**.

Monadic logic allows for instance to express properties of directed graph. In this case, we assume that the elements are vertices of the graph and the only available symbol $\texttt{edge}(x, y)$ expresses the existence of an edge from vertex x to vertex y. The existence of a path starting from a node i and reaching a node f can be expressed in monadic logic, as follows:

> All sets of vertices that contain i and are closed under the edge relation also contain f.

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n 259454. The author also acknowledges support from the project ANR 2010 BLAN 0202 02 FREC.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 391–404, 2013.

Of course, this is to be translated in mathematical symbols, as follows:

$$\texttt{path}(i,f) \quad \stackrel{\texttt{def}}{=} \quad \forall Z (i \in Z \wedge \forall x \forall y \, (x \in Z \wedge \texttt{edge}(x,y) \to y \in Z)) \\ \to f \in Z \ .$$

The important convention here is that lower case letter (here i, f, x, y) implicitly range over elements, while upper case letters (here Z) range over sets of such elements. We also see that a membership relation is available, such as in $x \in Z$, with an obvious meaning. In some works, emphasizing on the fact that sets are 1-ary relations, this is denoted as $Z(x)$.

(In this example, the power of monadic logic compared to first-order logic, is already transparent. Indeed, it is a classical fact that reachability, i.e., the existence of a path, is not expressible in first-order logic. This is usually the first application that is given to Ehrenfeucht-Fraïssé games.)

The view we have on monadic logic is the one of **algorithmic model theory**, *i.e.*, the aim is to develop algorithms that can "solve" the logic. Solving has several meaning depending on the situation. The first is **satisfiability** which means, given a formula, to determine the existence of a structure for which it holds (a model). The second is **model-checking** which means, given a formula and some description of a structure (possibly infinite), to determine if the formula holds on this specific structure. Grand expectations have to be immediately lowered: satisfiability of monadic logic is undecidable as such. This is inherited from first-order logic, that it is extending, which is already undecidable [11].

The interest of monadic logic appears when it is considered on structures of specific shapes, namely **word**s and trees or resemblant models. We will not develop the tree-like structures in this paper. By words, we mean structures that are linearly ordered, and in which each element is decorated by some finite local information. These are also referred to as **chain**s. These two terminologies describe properties of exactly the same nature, but differ concerning many choices of notations. In particular the notations for chains are consistent with the view of logic, while the notations for words are consistent with language theory. We adopt in this paper the *word terminology* and stick to it, even if it is rightfully arguable.

Let us explain how a word can be encoded as a relational structure: a **word** can be seen as a linearly ordered set of positions enriched with unary predicates that describe what letter is carried by each position. Formally, a word over an alphabet $\mathbb{A}$ is a structure $(L, \leq, (a)_{a \in \mathbb{A}})$ where $\leq$ equips L of a total order – we call $(L, \leq)$ the **domain** of the word, and refer to $(L, \leq)$ as a **linear ordering** – and for all elements x of L there exists one and exactly one letter $a \in \mathbb{A}$ such that $a(x)$ holds. Remark here that we did not make any assumption concerning the finiteness of the linear ordering. This is important since we will eventually be entering the realm of infinite structures. We will refer to **countable words** or **ω-words** if the underlying linear ordering is countable or isomorphic to ω respectively.

A language of finite words over the alphabet $\mathbb{A}^*$ is **monadic definable** if there is a monadic sentence φ such that $L = \{u \in \mathbb{A}^* \ : \ u \models \varphi\}$ (where, as

is usual $u \models \varphi$ is pronounced "u **models** φ" and means that φ holds over the relational structure encoding the word u). If $\varphi(X_1, \ldots, X_k)$ is a monadic formula of free monadic variables $X_1, \ldots, X_k$, and $A_1, \ldots, A_k$ are subsets of the domain of a word u, then $u \models \varphi(A_1, \ldots, A_k)$ is true if φ is satisfied on u when its free variables $X_1, \ldots, X_k$ take $A_1, \ldots, A_k$ as respective value.

For instance, over finite words, a property $\varphi(X)$ expressing that "between any two occurrences of the letter a, at least one point belongs to X" can be written as $\forall x \forall y\, (x < y \wedge a(x) \wedge a(y)) \rightarrow \exists z (z \in X \wedge x < z < y)$.

The starting point in the description of monadic logic is its equivalence over finite words with regular languages, as it has been found independently by Trakhtenbrot on the one side, and Elgot and Büchi on the other side:

Theorem 1 (Büchi, Elgot and Trakhtenbrot [1,4,10]). *A language of finite words is definable in monadic logic if and only if it is regular*[1]*.*
Furthermore, the translations are effective, and as a consequence, satisfiability of monadic logic over finite words is decidable.

The decidability result means that we can symbolically test the formula toward an infinite number of potential inputs, here words. But Büchi made a further step by showing that in such results, even the input can be infinite. Here, an ω-word is a word such that the underlying linear ordering is isomorphic to ω, or equivalently $(\mathbb{N}, \leq)$.

Theorem 2 (Büchi [2]). *A language of ω-words is definable in monadic logic if and only if it is recognized by a Büchi-automaton*[2]*.*
Furthermore, the translations are effective, and as a consequence, satisfiability of monadic logic over ω-words is decidable.

This result was the first success in the decidability of the monadic theory of some infinite models.

Remark 1. The original result of Büchi establishes the decidability of the monadic theory of $(\mathbb{N}, \leq)$. Of course, $(\mathbb{N}, \leq)$ can be seen as an instance of a word over a unary alphabet. Thus, from Theorem 2, we can deduce the decidability of $(\mathbb{N}, \leq)$. The converse also holds. Indeed, a word over the finite alphabet $\mathbb{A}$ can be encoded, *e.g.,* as the linear ordering of its domain together with sets $(X_a)_{a \in \mathbb{A}}$ such that a position i belongs to X_a if and only if i carries the letter a. Using this encoding, a monadic formula over ω can guess on ω-word using existential quantifiers over sufficiently many monadic variables. Using this technique, the decidability of the satisfiability of monadic logic over ω-words can be deduced from the decidability of the theory of ω. These kind of encodings are doable for all linear orderings.

Of course, we can go further. The two landmark linear orderings are the **rational line** $(\mathbb{Q}, \leq)$ (sometimes denoted η of $\mathcal{Q}$), and the **real line** $(\mathbb{R}, \leq)$ (sometimes denoted λ or $\mathcal{R}$).

[1] Say, recognized by a finite state automaton.
[2] We do not present this very classical model here.

Theorem 3 (Rabin [8]). *The monadic theory of the rational line is decidable, or equivalently the satisfiability of monadic logic over words of domain the rational line is decidable.*

In fact, the proof of Rabin concerns the infinite binary tree, which is a richer structure than the rational line. The proof of Rabin establishes that, *over infinite trees*, monadic logic is equivalent to a certain form of automata. The decidability of the rational line is then obtained by interpreting the rational line inside the infinite binary tree. This allows to decide the monadic theory of the rational line, but it is not at all informative concerning the expressive power of monadic logic over the rational line. In particular, one cannot deduce, say, a model of automata that would have the expressive power of monadic logic over the rational line.

Example 1. In order to illustrate the above theorem, let us try to give some intuition of what can be defined using monadic logic over linear orderings in general. We do not try to be exhaustive, and merely list some classical constructions and examples.

Relativisation. Given a monadic formula Ψ and a set X, the **relativisation** of Ψ to X is the formula Ψ^X in which all first-order quantifiers are required to range over X and all monadic quantifiers are required to range over subsets of X. This can be done by a simple syntactic transformation of Ψ. The formula Ψ^X holds over a structure if the formula Ψ holds over the structure restricted to the set X.

As a consequence, Theorem 3 can be used to decide monadic logic over the class of all countable linear orderings. Indeed, remark that any countable linear ordering is isomorphic to a sub-ordering of the rational line. As a consequence, Theorem 3 can be used to decide the existence of a countable linear ordering that satisfy a formula Ψ: such a linear ordering exists if and only if $(\mathbb{Q}, \leq) \models \exists X\, \Psi^X$.

Finiteness. Remark that a non-empty linear ordering that has no maximal point is infinite. Furthermore, this property is expressible in monadic logic (in fact in first-order logic). Now, it is easy to see that conversely, if a linear ordering is infinite, then either it has a non-empty sub-ordering that has no maximal point, or a non-empty sub-ordering that has no minimal point. This is expressible in monadic logic.

Using relativisation, this allows to express properties such as "X is finite" for X a monadic variable. As a consequence we can express properties such as "every finite sub-words belong to L", where L is a regular language of finite words.

Digression. Let us recall that finiteness is not first-order definable in linear orderings. This is a straightforward consequence of compactness, as follows. For the sake of contradiction, assume that the property "the linear ordering is finite" is expressible in first order, then the property $P_n =$ "the linear ordering is finite and contains at least n distinct elements" would be expressible in first-order logic too. But any finite sets of properties P_n has a model (the finite linear ordering of length m where m is the maximal of the n's involved in the P_n's under consideration). Thus by compactness, there exists a linear ordering that satisfies all P_n's simultaneously. This is obviously impossible since this would be a linear ordering that is at the same be finite and contains more than n points for all n.

Density. A linear ordering is said **dense** if for any two points $x < y$ there exists another point $z \in (x, y)$. The rational line is dense, while ω (and more generally any ordinal) is not dense. As an extra example, the integer line $(\mathbb{Z}, \leq)$ is also not dense, and it is not isomorphic to an ordinal either.

In particular, assuming that a linear ordering is countable, then being isomorphic to $(\mathbb{Q}, \leq)$ is expressible in monadic logic (in fact first-order). Indeed, up to isomorphism, $(\mathbb{Q}, \leq)$ is the only countable linear ordering that is dense, contains at least two points, and has no minimal nor maximal point.

Scatteredness. A linear ordering is said **scattered** if none of its suborderings are dense. This is definable directly in monadic logic using relativisation.

Remark that it may happen that a linear ordering is neither dense nor scattered: imagine for instance a copy of the rational line in which every point is replaced by a copy of $(\mathbb{Z}, \leq)$.

In his seminal paper [9], Shelah uses another technique – the composition method – and use it to prove the decidability of the monadic theory of the rational line. However, the most impressive result he obtains is the undecidability of the monadic theory of the real line.

Theorem 4 (Shelah [9]). *The monadic theory of the real line is undecidable*[3].

This is a intriguing result that in some sense contradicts the intuition.

At this point, we have seen the key results concerning the decidability of the monadic theory of linear orderings. Though both the result of decidability and of undecidability can be improved, these improvements do not help understanding the picture better.

2 In Which the Problem Is Exposed and Some of Its Intriguing Characteristics Appear

As we have seen, the central problems of decidability are solved for most of them. The questions we are really interested in this paper are more related to expressivity.

Question 1. Given a monadic formula $\varphi(X_1, \ldots, X_k)$, does there exist another formula $\varphi^*(X_1, \ldots, X_k)$ such that for all sets of rationals $A_1, \ldots, A_k \subseteq \mathbb{Q}$,

$$(\mathbb{R}, \leq) \models \varphi(A_1, \ldots, A_k) \quad \text{if and only if} \quad (\mathbb{Q}, \leq) \models \varphi^*(A_1, \ldots, A_k) \ .$$

In other words, the question is whether the ability to use all points of the real line does give more expressive power for stating properties of predicates over the rational line. Notice here that implicitely, we use the fact that there is a fixed embedding of $(\mathbb{Q}, \leq)$ into $(\mathbb{R}, \leq)$ (the usual one).

Gurevich and Rabinovich use the nice and suggestive terminology that the formula φ has access to the reals **at the background** [6]. The open above question is thus a rephrasing of the following open question in [6]:

[3] Originally under a weak version of the continuum hypothesis, which has then be removed in collaboration with Gurevich [7].

> "Is it true that a family of point-sets is definable in the chain $(\mathbb{Q}, \leq)$ of rationals if and only if it is definable in $(\mathbb{Q}, \leq)$ with the chain of reals at the background?"

Gurevich and Rabinovich solved this question for the integers.

Theorem 5 (Theorem 1 in [6]). *For all monadic formula $\varphi(X_1, \ldots, X_k)$, there exists a formula $\varphi^*(X_1, \ldots, X_k)$ such that for all $A_1, \ldots, A_k \subseteq \mathbb{N}$,*

$$(\mathbb{R}, \leq) \models \varphi(A_1, \ldots, A_k) \quad \textit{if and only if} \quad (\mathbb{N}, \leq) \models \varphi^*(A_1, \ldots, A_k) \ .$$

We will not go into this proof which is superseded by what follows. However, already in this case a very interesting phenomenon occurs: the existence of the formula is inherently non-effective. Even if one allows the produced formula to use extra predicates of decidable monadic theory.

Theorem 6 (variant of Theorem 2 in [6]). *Let $Q_1, \cdots \subseteq \mathbb{N}$ be monadic predicates such that $(\mathbb{N}, \leq, \bar{Q})$ has a decidable monadic theory. There exists no algorithm which, given a monadic formula φ, constructs φ^* such that:*

$$(\mathbb{R}, \leq) \models \varphi(X_1, \ldots, X_k) \quad \textit{if and only if} \quad (\mathbb{N}, \leq, \bar{Q}) \models \varphi^*(X_1, \ldots, X_k) \ .$$

Proof. Assume that such an algorithm exists, and consider some monadic sentence φ. We apply the algorithm to φ, and obtain a sentence φ^* such that $(\mathbb{R}, \leq) \models \varphi$ if and only if $(\mathbb{N}, \leq, \bar{Q}) \models \varphi^*$. Since the monadic theory of $(\mathbb{N}, \leq, \bar{Q})$ is decidable, we could decide $(\mathbb{R}, \leq) \models \varphi$. This contradicts Theorem 4. □

Despite this inherent difficulty due to non-effectiveness of the construction, we shall provide a positive answer to Question 1.

3 In Which the Composition Theorem Is Introduced

In the same seminal paper as the one showing the undecidability of the monadic theory of the real line [9], Shelah develops a technique referred to as the **composition method**. This is a variant for monadic logic of techniques developed originally by Feferman and Vaught for first-order logic [5]. It becomes particularly relevant in the context of monadic logic. The versatility of this approach lies in its central theorem (known as the composition theorem, Theorem 7 below) which is not in itself a decidability result, but provides the skeleton for a decision procedure.

We need some notations first. Let α be a linear ordering and u_i for all $i \in \alpha$ be words. Then, we denote:

$$\prod_{i \in \alpha} u_i$$

the word consisting of copies of the u_i's arranged according to the linear ordering α. Formally, assume that $\alpha = (L, \leq)$ and that the domain of u_i is $(K_i, \leq_i)$

for all $i \in \alpha$, then the domain of the word $\prod_{i\in\alpha} u_i$ is the set of pairs (i, x) where $i \in L$ and $x \in K_i$, ordered by $(i, x) \leq (j, y)$ if $i < j$ or $i = j$ and $x \leq_i y$. Furthermore, the letter at position (i, x) is the letter at position x in u_i. In the terminology of Shelah, this is denoted as a sum. The product notation reflects the fact that it extends the concatenation product used for words.

Example 2. Before stating it, let us try to give an intuitive meaning to the composition theorem. Consider that you are interested in the following property of a word u: "there is an even number of occurrences of the letter a". For simplicity, we denote from now by $|u|_a$ the number of occurrences of the letter a in the word u. Let us distinguish four formulae:

- `none` holds over u if $|u|_a = 0$,
- `even` holds over u if $|u|_a > 0$ is even,
- `odd` holds over u if $|u|_a > 0$ is odd,
- `infinite` holds over u if $|u|_a$ is infinite.

Over any word, exactly one of these four formulae holds, and the property we are interested in is a disjunction of such formulae, namely `empty` $\vee$ `even`.

Now consider a word $u = \prod_{i\in\alpha} u_i$, where the u_i's are themselves words. It is easy to check that:

- $u \models$ `none` if and only if $u_i \models$ `none` for all $i \in \alpha$,
- $u \models$ `even` if and only if
  1. $u_i \not\models$ `infinity` for all $i \in \alpha$,
  2. either $u_i \models$ `even` or $u_i \models$ `odd` for some $i \in \alpha$, and finitely many of them, and;
  3. there is a finite even number of $i \in \alpha$ such that $u_i \models$ `odd`,
- $u \models$ `odd` if and only if [...as above, replacing even by odd in 3...].
- $u \models$ `infinite` if and only if either $u_i \models$ `infinite` for some $i \in \alpha$ or there are infinitely many $i \in \alpha$ such that $u_i \models$ `even` or there are infinitely many $I \in \alpha$ such that $u_i \models$ `odd`.

For all words u there is one and only one formula φ among `none`, `even`, `odd` and `infinity` such that $u \models \varphi$: let us call this formula the type of u, and denote it `type`(u). What we have seen in this example is that in order to know the type of $\prod_{i\in\alpha} u_i$, it is sufficient to know the type of each of the u_i's. What is crucial is that there are only finitely many types that are sufficient. The composition theorem (Theorem 7) generalizes this example. It states that in order to know the truth value of any monadic formula it is sufficient to consider only finitely many types that have properties similar to this example.

Let us fix now a constant $k \in \mathbb{N}$. A monadic formula has **quantifier rank** k if there are at most k nested quantifiers.

Proposition 1. *Over a fixed finite signature with only relational symbols (the case of words over a fixed finite alphabet), there are only finitely many formulas*

up to syntactic equivalence. Here, the syntactic equivalence involves the usual associativity, commutativity, idempotency, and distributivity of conjunctions and disjunctions, the renaming of bound variables, and

$$\exists s(\varphi \vee \psi) \equiv (\exists s\, \varphi) \vee (\exists s\, \psi) \qquad \textit{and} \qquad \forall s(\varphi \wedge \psi) \equiv (\forall s\, \varphi) \wedge (\forall s\, \psi)\ ,$$

for t a first-order variable or a monadic variable.

From now, all formulae will be considered modulo this syntactic equivalence, and since we fix k, by the above fact, we only have to consider finitely many formulae.

Now, given a word u, its **(k-)type** is the set of sentences φ of quantifier rank at most k such that $u \models \varphi$.

Theorem 7 (composition [9]). *If $\mathtt{type}_k(u_i) = \mathtt{type}_k(v_i)$ for all $i \in \alpha$, then*

$$\mathtt{type}_k\left(\prod_{i\in\alpha} u_i\right) = \mathtt{type}_k\left(\prod_{i\in\alpha} v_i\right)\ .$$

This completely reflects the intuition of Example 2, in which only four possible types were distinguished for simplicity.

Remark 2. In fact the real composition theorem is more precise in that it expresses how $\mathtt{type}_k(\prod_{i\in\alpha} u_i)$ can be defined from $\prod_{i\in\alpha} \mathtt{type}_k(u_i)$. Formally, it states that for all type t of quantifier rank k, there exists a monadic formula t^* such that:

$$\mathtt{type}_k\left(\prod_{i\in\alpha} u_i\right) = t \qquad \text{iff} \qquad \prod_{i\in\alpha} \mathtt{type}_k(u_i) \models t^*\ .$$

This implies Theorem 7 as if $\mathtt{type}_k(u_i) = \mathtt{type}_k(v_i)$ for all $i \in \alpha$, this means that $\prod_{i\in\alpha} \mathtt{type}_k(u_i) = \prod_{i\in\alpha} \mathtt{type}_k(v_i)$. However, this more complete presentation is quite misleading since the quantifier rank of t^* may be (much) higher than k. In practice, the decision procedures using the composition method do not make use of this formula t^*.

4 In Which Algebraic Recognizability for Countable Words Is Defined

In this section, we introduce a notion of recognizability that is suitable for capturing monadic logic over countable words [3]. It is highly related to the composition method as shall be shown below.

Let us denote by M° the set of words of countable length over the alphabet M. We call them **M°-words** from now. Consider an application $\otimes : M^\circ \to M$. We will often use, for α a countable linear ordering, and $a_i \in M$ for all $i \in \alpha$, the notation

$$\bigotimes_{i\in\alpha} a_i$$

to denote $\otimes u$ where u is the word of domain α that carries at position i the letter a_i for all $i \in \alpha$. This operation $\otimes$ is **associative**, or **a product**, if

- $\otimes(a) = a$ for all $a \in M$, and;
- for all countable linear orderings α and all families of M°-words $(u_i)_{i\in\alpha}$,

$$\otimes\left(\prod_{i\in\alpha} u_i\right) = \bigotimes_{i\in\alpha} \otimes(u_i) \ .$$

A **$\circ$-monoid $\mathbf{M} = (M, \otimes)$** is a set M equipped with a product $\otimes$.

Remark 3. A consequence of the above definition is that for a, b, c in M:

$$\otimes(a \otimes (bc)) = \otimes(\otimes(a) \otimes (bc)) = \otimes(abc) = \otimes(\otimes(ab) \otimes (c)) = \otimes(\otimes(ab)c) \ .$$

Thus, if you denote $\otimes(ab)$ as $a \cdot b$, this means $a \cdot (b \cdot c) = (a \cdot b) \cdot c$. Hence this generalizes the usual notion of associativity. Also, notice that the empty-word ε belonging to M°, it has a value $\otimes(\varepsilon)$ under $\otimes$. Let us denote by 1 this value. Still using associativity, we get for all $a \in M$,

$$a \cdot 1 = \otimes(a1) = \otimes(\otimes(a) \otimes (\varepsilon)) = \otimes(a\varepsilon) = \otimes(a) = a \ ,$$

and similarly $1 \cdot a = a$. Thus, every $\circ$-monoid is in particular a monoid.

Of course, the **free $\circ$-monoid** generated by a set M is simply $(M^\circ, \prod)$. The notion of a morphism of $\circ$-monoids is also natural. Given two $\circ$-monoids $\mathbf{M} = (M, \otimes)$ and $\mathbf{M}' = (M', \otimes')$, a **morphism** from $\mathbf{M}$ to $\mathbf{M}'$ is a function f from M to M' such that for all countable linear orderings α and $(a_i)_{i\in\alpha}$ elements of M,

$$f\left(\bigotimes_{i\in\alpha} a_i\right) = \bigotimes_{i\in\alpha}{}' f(a_i) \ .$$

We are now ready to define the notion of a recognizable set of countable words. A set of words $L \subseteq \mathbb{A}^\circ$ is **recognizable** if there exist a finite $\circ$-monoid $\mathbf{M} = (M, \otimes)$, a morphism f from $\mathbb{A}^\circ$ to $\mathbf{M}$ and a set $F \subseteq M$ such that for all words $u \in \mathbb{A}^\circ$,

$$u \in L \qquad \text{if and only if} \qquad f(u) \in F \ .$$

Remark here that clearly, the finiteness refers to the carrier M of $\mathbf{M}$ (even if M is finite, $\otimes$ is a mapping of infinite domain). The fact that the product $\otimes$ is "infinite" means that, *as such*, a finite $\circ$-monoid cannot be represented in a computer.

In fact, the very strong properties of associativity of the product have as effect that only a finite quantity of information is sufficient for representing a finite $\circ$-monoid. Concretely, it is sufficient to know the value of $\otimes$ over a finite number of words for being able to reconstruct $\otimes$ uniquely over all countable words [9,3]. This is very similar to the fact that once the product of two elements is known in a monoid, the product can be extended uniquely to arbitrarily long finite sequences of elements. Decidability results and effective constructions are done manipulating this representation.

Example 3. Let us consider the set of $\{a,b\}^\circ$-words that have a finite and even number of occurrences of the letter a (this corresponds to Example 2), and show that it is recognizable.

We consider as carrier of the monoid the set $M = \{1, e, o, 0\}$. We start by giving the morphism f that sends $\{a,b\}^\circ$ to M:

$$f(u) = \begin{cases} 1 & \text{if } |u|_a = 0, \\ e & \text{if } |u|_a > 0 \text{ is finite and even}, \\ o & \text{if } |u|_a > 0 \text{ is finite and odd}, \\ 0 & \text{otherwise.} \end{cases}$$

With this morphism in mind, it is simple to uniquely define the product over M. Consider for instance the M°-word oo, since $o = f(a)$, this means $\otimes(oo) = \otimes(f(a)f(a)) = \otimes(f(aa)) = e$. Using similar arguments, we can complete the product as:

- $\otimes(u) = 1$ if u contains only 1's,
- $\otimes(u) = e$ if u does not contain the letter 0, it contains at least one non-1 letter, finitely many e letters, and a finite even number of o letters,
- $\otimes(u) = e$ if u does not contain the letter 0, it contains at least one non-1 letter, finitely many e letters, and a finite odd number of o letters,
- $\otimes(u) = 0$ otherwise.

The intuition behind the previous example generalizes. A direct consequence of the composition theorem is the recognizability of all monadic definable languages.

Theorem 8. *All monadic definable languages of $\circ$-words are recognizable.*

Proof. Let φ be a monadic formula defining $L \subseteq \mathbb{A}^\circ$. Let k be the quantifier rank of φ.

We shall construct a $\circ$-monoid $\mathbf{M}$ and a morphism f from $\mathbb{A}^\circ$ to $\mathbf{M}$. Define

$$M = \{\mathtt{type}_k(v) \ : \ v \in \mathbb{A}^\circ\} \ .$$

Since $\mathtt{type}_k$ is surjective from $\mathbb{A}^\circ$ onto M, there exists a mapping $g : M \to \mathbb{A}^\circ$ such that $\mathtt{type}_k \circ g$ is the identity over M. We extend it to M°-words letter by letter, yielding a mapping $\tilde{g}$ from M° to $\mathbb{A}^\circ$ (formally $\tilde{g}(\prod_{i\in\alpha} a_i) = \prod_{i\in\alpha} g(a_i)$). We now equip M with an operation $\otimes$ as follows. Let u be a word in M°, define

$$\otimes(u) = \mathtt{type}_k(\tilde{g}(u)) \ .$$

We have for all $\mathbb{A}^\circ$-words $(v_i)_{i\in\alpha}$ indexed by a countable linear ordering α,

$$\begin{aligned}\mathtt{type}_k\left(\prod_{i\in\alpha} v_i\right) &= \mathtt{type}_k\left(\prod_{i\in\alpha} g(\mathtt{type}_k(v_i))\right) && \text{(Theorem 7)}\\ &= \mathtt{type}_k\left(\tilde{g}\left(\prod_{i\in\alpha}\mathtt{type}_k(v_i)\right)\right)\\ &= \bigotimes_{i\in\alpha}\mathtt{type}_k(v_i)\ ,\end{aligned}$$

in which the first equality is by the composition theorem (Theorem 7) since by construction of g, $\mathtt{type}_k(v_i) = \mathtt{type}_k(g(\mathtt{type}_k(v_i)))$.

We do not know yet that $\otimes$ is a product. However, since $\mathtt{type}_k$ is surjective from $\mathbb{A}^\circ$ onto M and satisfies the properties of a morphism, it follows that the fact that $(\mathbb{A}^\circ, \prod)$ is a $\circ$-monoid is automatically transferred to $(M, \otimes)$. Thus $(M, \otimes)$ is also a $\circ$-monoid, and $\mathtt{type}_k$ is a morphism from $(\mathbb{A}^\circ, \prod)$ to $(M, \otimes)$.

Let now $F = \{\mathtt{type}_k(u)\ :\ u \in L\}$. Let us show that $\mathbf{M}, \mathtt{type}_k, F$ recognize L. Let $u \in \mathbb{A}^\circ$ be a word. We have that $u \in L$ if and only if $u \models \varphi$ if and only if $\varphi \in \mathtt{type}_k(u)$ if and only if $\mathtt{type}_k(u) \in F$. Hence $\mathbf{M}, \mathtt{type}_k, F$ recognize L. □

One of the main contributions in [3] is to provide a form of converse to Theorem 8.

Theorem 9. *All recognizable languages of $\circ$-words are monadic definable.*

This direction relies on completely different techniques. It involves in particular the theory of ideals of the monoid underlying the $\circ$-monoid and special forms of factorizations of the words. In particular, it heavily relies on the fact that the $\circ$-monoids used to recognize languages are finite, an assumption that was not used so far (in fact, it is already crucial in Shelah's work, but for different reasons, for decidability). We will use this result as a black-box.

5 In Which the Question Is Answered

Let us consider now Question 1 again. We will see that the situation is not much different from the previous section.

In order to deal with the real line, we need to describe a bit more precisely the relationship between the rational line and the real line. For technical reasons it is not very convenient to work directly with the real line, but rather on the expansion of the rational line with all "Dedekind cuts". The real line itself is slightly different: it is obtained from the rational line using a similar expansion, but keeping only the so-called "natural cuts". Nevertheless, as far as logic is concerned, this difference is very minor.

Given a linear ordering $(E, \leq)$, a (Dedekind) **cut** is an ordered pair (A, B) of sets $A, B \subseteq E$ such that $A \cup B = E$ and $x < y$ for all $x \in A$ and $y \in B$. Cuts are ordered by $(A, B) \leq (A', B')$ if $A \subseteq A'$. Cuts can also be compared with the

elements of E by $x < (A, B)$ if $x \in A$ and $(A, B) < x$ if $x \in B$. Equipped with this relation, the (disjoint) union of the elements of E with the cuts form a linear ordering. A cut (A, B) is **extremal** if either $A = \emptyset$ or $B = \emptyset$. Given a linear ordering $\alpha = (L, \leq)$, denote by $\hat{\alpha}$ the **completion** of α, which is obtained from α by adding to it all non-extremal cuts. Remark that to every element in $x \in E$ corresponds three copies $x^- < x < x^+$ in the the completed linear orderings, where $x^- = ((-\infty, x), [x, \infty))$ and $x^+ = ((-\infty, x], (x, \infty))$. Cuts that are not of the form x^- or x^+ are called **natural**. As mentioned above, the real line is obtained from the rational line by adding to it the non-extremal natural cuts only.

The completion of a word is done as follows. We fix ourselves a dummy letter ι that is intended to label cuts. The **(cut) completion** $\mathtt{comp}(u)$ of a word u over the alphabet $\mathbb{A}$ is a word over the alphabet $\mathbb{A} \cup \{\iota\}$ defined as $\prod_{i \in \hat{\alpha}} b_i$ where $b_i = a_i$ for all $i \in \alpha$ and $b_i = \iota$ otherwise (*i.e.*, if i is a cut).

A simple, yet important, point is the relationship between the completion and the product $\prod$. The following lemma discloses this point. It essentially states that the completion of the product is equivalent to a variant product of the completion, where the variant product "fills the missing cuts".

Lemma 1. *For all linear orderings α and words $(v_i)_{i \in \alpha}$,*

$$\mathtt{comp}\left(\prod_{i \in \alpha} v_i\right) \equiv \hat{\prod_{i \in \alpha}}^{\iota} \mathtt{comp}(v_i) \ ,$$

where for all words (w_i) indexed by a linear ordering α we set

$$\hat{\prod_{i \in \alpha}}^{\iota} w_i = \prod_{i \in \hat{\alpha}} w'_i$$

with for all $i \in \hat{\alpha}$, $w'_i = \begin{cases} w_i & \text{if } i \in \alpha, \\ \iota & \text{otherwise, i.e., if } i \text{ is a cut.} \end{cases}$

A language L of countable words is called **monadic definable with cuts at the background** if there exists a monadic formula φ such that $u \in L$ if and only if $\hat{u} \models \varphi$.

The following key proposition follows exactly the same proof scheme as the one of Theorem 8.

Proposition 2. *Languages of countable words that are monadic definable with cuts at the background are $\circ$-recognizable.*

Proof. Let φ be a monadic formula defining with cuts at the background a language of countable words $L \subseteq \mathbb{A}^\circ$. Let k be the quantifier rank of φ.

We shall construct a $\circ$-monoid $\mathbf{M}$ and a morphism f from $\mathbb{A}^\circ$ to $\mathbf{M}$. Let first set $\mathtt{typec}_k(v)$ to be $\mathtt{type}_k \circ \mathtt{comp}(v)$ for all $v \in \mathbb{A}^\circ$. Define

$$M = \{\mathtt{typec}_k(v) \ : \ v \in \mathbb{A}^\circ\} \ .$$

Since $\mathtt{typec}_k$ is surjective from $\mathbb{A}^\circ$ onto M, there exists a mapping $g : M \to \mathbb{A}^\circ$ such that $\mathtt{typec}_k \circ g$ is the identity over M. We extend it to M°-words letter by letter, yielding a mapping $\tilde{g}$ from M° to $\mathbb{A}^\circ$ (formally $\tilde{g}(\prod_{i\in\alpha} a_i) = \prod_{i\in\alpha} g(a_i)$). We now equip M with an operation $\hat{\otimes}$ as follows. Let u be a word in M°, define

$$\hat{\otimes}(u) = \mathtt{typec}_k(\tilde{g}(u)) \ .$$

We show now that $\mathtt{typec}_k$ has the properties of a morphism from $\mathbb{A}^\circ$ to M (though we do not know yet that $\hat{\otimes}$ is a product). Consider a family of $\mathbb{A}^\circ$-words $(v_i)_{i\in\alpha}$ indexed by a countable linear ordering α, we have:

$$\begin{aligned}
\mathtt{typec}_k\left(\prod_{i\in\alpha} v_i\right) &= \mathtt{type}_k\left(\hat{\prod_{i\in\alpha}}^{\iota} \mathtt{comp}(v_i)\right) && \text{(Lemma 1)}\\
&= \mathtt{type}_k\left(\hat{\prod_{i\in\alpha}}^{\iota} \mathtt{comp}(g(\mathtt{typec}_k(v_i)))\right) && \text{(Theorem 7)}\\
&= \mathtt{typec}_k\left(\prod_{i\in\alpha} g(\mathtt{typec}_k(v_i))\right) && \text{(Lemma 1)}\\
&= \mathtt{typec}_k\left(\tilde{g}\left(\prod_{i\in\alpha} \mathtt{typec}_k(v_i)\right)\right)\\
&= \hat{\bigotimes_{i\in\alpha}} \mathtt{typec}_k(v_i) \ .
\end{aligned}$$

in which the equality between the first and second line is by the composition theorem using the fact that $\mathtt{typec}_k(v_i) = \mathtt{typec}_k(g(\mathtt{typec}_k(v_i))$.

We do not know yet that $\hat{\otimes}$ is a product. However, since $\mathtt{typec}_k$ is surjective from $\mathbb{A}^\circ$ onto M and satisfies the properties of a morphism, it follows that the fact that $(\mathbb{A}^\circ, \prod)$ is a $\circ$-monoid is automatically transferred to $(M, \hat{\otimes})$. Thus $(M, \hat{\otimes})$ is also a $\circ$-monoid, and $\mathtt{typec}_k$ is a morphism from $(\mathbb{A}^\circ, \prod)$ to $(M, \hat{\otimes})$.

Let now $F = \{\mathtt{typec}_k(u) \ : \ u \in L\}$. Let us show that $\mathbf{M}, \mathtt{typec}_k, F$ recognize L. Let $u \in \mathbb{A}^\circ$ be a word. We have that $u \in L$ if and only if $\mathtt{comp}(u) \models \varphi$ if and only if $\varphi \in \mathtt{typec}_k(u)$ if and only if $\mathtt{typec}_k(u) \in F$. Hence $\mathbf{M}, \mathtt{type}_k, F$ recognize L. □

Thus, in combination with Theorem 9, we get the following corollary.

Corollary 1. *Every language of countable words monadic definable with cuts at the background is monadic definable.*

If we restate this corollary in terms of relational structures, we get:

Corollary 2. *Given a monadic formula $\varphi(X_1, \ldots, X_k)$, there exists a formula $\varphi^*(X_1, \ldots, X_k)$ such that for all countable linear orderings α and all sets of rationals $A_1, \ldots, A_k \subseteq \alpha$,*

$$\hat{\alpha} \models \varphi(A_1, \ldots, A_k) \quad \textit{if and only if} \quad \alpha \models \varphi^*(A_1, \ldots, A_k) \ .$$

Indeed, a countable linear ordering α labeled with $A_1, \ldots, A_k$ can be seen as a countable word over the alphabet 2^k.

Now, recall that the reals are the completion of the rationals with natural cuts. The only reason that the above theorem does not exactly solve Question 1 as it stands is that $\hat{\alpha}$ also contains cuts that are not natural. Thus, it is sufficient to remark that given a linear ordering α of domain A, there is a first-order formula $\varphi(X, x)$ such that $\hat{\alpha} \models \varphi(A, a)$ if and only if a is a non-natural cut (recall that the non-natural cuts are the ones that are predecessors or successors of elements of A; a property that makes them easily definable). Thus, using Corollary 2 together with a relativisation to the natural cuts and the original ordering, we finally answer positively Question 1.

Theorem 10. *Given a monadic formula $\varphi(X_1, \ldots, X_k)$, there exists a formula $\varphi^*(X_1, \ldots, X_k)$ such that for all $A_1, \ldots, A_k \subseteq \mathbb{Q}$,*

$$(\mathbb{R}, \leq) \models \varphi(X_1, \ldots, X_k) \quad \textit{if and only if} \quad (\mathbb{Q}, \leq) \models \varphi^*(X_1, \ldots, X_k) \ .$$

Acknowledgments. I am grateful to Alexander Rabinovich who introduced me to the question of reals at the background, and for all the discussions we had on the subject. I also thank Olivier Carton and Gabriele Puppis for many discussions we had together.

References

1. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik Grundlagen Math. 6, 66–92 (1960)
2. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.), pp. 1–11. Stanford Univ. Press, Stanford (1962)
3. Carton, O., Colcombet, T., Puppis, G.: Regular languages of words over countable linear orderings. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 125–136. Springer, Heidelberg (2011)
4. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Amer. Math. Soc. 98, 21–51 (1961)
5. Feferman, S., Vaught, R.L.: The first order properties of products of algebraic systems. Fund. Math. 47, 57–103 (1959)
6. Gurevich, Y., Rabinovich, A.M.: Definability and undefinability with real order at the background. J. Symb. Log. 65(2), 946–958 (2000)
7. Gurevich, Y., Shelah, S.: Monadic theory of order and topology in zfc. In: Ann. of Math. Logic, vol. 23, pp. 179–198. North-Holland Publishing Compagny (1982)
8. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
9. Shelah, S.: The monadic theory of order. Ann. of Math. (2) 102(3), 379–419 (1975)
10. Trakhtenbrot, B.A.: Finite automata and monadic second order logic (Russian). Siberian Math. J 3, 103–131 (1962)
11. Trakthenbrot, B.A.: The impossibility of an algorithm for the decision problem for finite domains (Russian). Doklady Academii Nauk SSSR 70, 569–572 (1950)

Model-Checking Bounded Multi-Pushdown Systems*

Kshitij Bansal[1] and Stéphane Demri[1,2]

[1] New York University, USA
[2] LSV, CNRS, France

Abstract. We provide complexity characterizations of model checking multi-pushdown systems. We consider three standard notions for boundedness: context boundedness, phase boundedness and stack ordering. The logical formalism is a linear-time temporal logic extending well-known logic `CaRet` but dedicated to multi-pushdown systems in which abstract operators are parameterized by stacks. We show that the problem is ExpTime-complete for context-bounded runs and unary encoding of the number of context switches; we also prove that the problem is 2ExpTime-complete for phase-bounded runs and unary encoding of the number of phase switches. In both cases, the value k is given as an input, which makes a substantial difference in the complexity.[1]

1 Introduction

Verification problems for pushdown systems, systems with a finite automaton and an unbounded stack, have been extensively studied and decidability can be obtained as in the case for finite-state systems. For instance, computing $\mathrm{pre}^\star(X)$ (set of configurations *reaching a* regular set X), $\mathrm{post}^\star(X)$ (correspondingly, configurations *accessible from* X), reachability and LTL model checking have been shown to be decidable [8,18]. These have also been implemented, for instance in the model-checker Moped [18]. It can be argued that they are natural models for modeling recursive programs. Two limitations though of the model are the inability to model programs with infinite domains (like integers) and modeling concurrency. Having an infinite automaton to handle the former limitation leads to undecidability. An approach to tackle this has been to abstract infinite-state programs to Boolean programs using, for instance, predicate abstraction. The model is repeatedly refined, as needed, like in tools SLAM, SatAbs etc. For concurrency, a natural way to extend this model would be to consider pushdown automata with multiple stacks, which has seen significant interest in the recent past [2,6,9,10]. This is the main object of study in this paper which we call *multi-pushdown systems (MPDS)*.

The difficulty of model-checking MPDS is that a pushdown system with even two stacks and with a singleton stack alphabet is sufficient to model a Turing

* Work partially supported by projects *ARCUS IdF/Inde* and *EU Seventh Framework Programme* under grant agreement No. PIOF-GA-2011-301166 (DATAVERIF).

[1] Omitted proofs and additional material can be found in the technical report [5].

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 405–417, 2013.

machine, hence making the problem of even testing reachability undecidable. This is not a unique situation and similar issues exists with other abstractions, like model-checking problems on counter systems; other models of multithreaded programs are also known to admit undecidable verification problems. That is why subclasses of runs have been introduced as well as problems related to the search for 'bounded runs' that may satisfy a desirable or undesirable property. For instance, context-bounded model-checking (bound on the number of context switches) [17] allows to regain decidability.

This paper focuses on the study of model-checking problems for MPDS based on LTL-like dialects, naturally allowing to express liveness properties, when some bounds are fixed. Though decidability of these problems has been established in some recent works we aim to provide optimal computational complexity analysis for LTL-like properties. In particular, we consider a LTL-like specification language based on `CaRet` [1], which strikes to us as fitting given the interest of the model in program verification. As in [14], `CaRet` generalized to multiple stacks and called Multi-`CaRet` is considered. Under this logic, we show model-checking problem of MPDS restricted to k-**context bounded runs** is in ExpTime, when k is encoded in unary. Since this problem is a generalization of LTL model checking pushdown systems which is known to be ExpTime-hard, this is an optimal result. Viewed as an extension of [8], we consider both a more general model and a more general logic, while still preserving the complexity bounds. At a technical level, we focus on combining several approaches in order to achieve optimal complexity bounds. In particular, we combine the approach taken in `CaRet` model-checking of recursive state machines machines, ideas from reachability analysis of multi-pushdown systems [18] and the techniques introduced in [8,18]. We also consider less restrictive notions, showing optimal 2-ExpTime for k-**phase bounded runs** [12] when k is in unary. Note that in all restrictions we consider, k is given as an input and not as a parameter of the problem, which makes a substantial difference when complexity analysis is provided. When k is encoded in binary, the bounds are 2-ExpTime and 3-ExpTime for context and phase boundedness respectively. For a third notion of **ordered multi-pushdown systems** [3], model-checking is in 2ExpTime.

Related Work. In [16], decidability results are found for classes of automata with auxiliary storage based on MSO property, see also [15]. This includes MPDS with bounded context and ordered MPDS. Unlike our ExpTime bound, the complexity is non-elementary in the size of the formula. This stems from the use of celebrated Courcelle's Theorem, which has parameterized complexity non-elementary, the parameter being the size of formula plus the tree-width.

More closely related to our approach of generalizing the automata-based approach for LTL are two recent works [4,14]; indeed model-checking of linear-time properties for MPDS under several boundedness hypothesis has been the subject of several recent studies. In [4], LTL model-checking on multi-pushdown systems when runs are k-scope-bounded is shown ExpTime-complete. Scope-boundedness strictly extends context-boundedness and therefore Corollary 2(I) and [4, Theorem 7] are closely related even though Corollary 2(I) deals with the

richer Multi-CaRet and it takes into account specifically context-boundedness. By contrast, [14] introduces an extension of CaRet that is expressively identical to the variant we consider in our paper (models are multiply nested words). Again, it deals with scope-boundedness and Corollary 2(I) and [14, Theorem 6] are closely related even though Corollary 2(I) takes into account context-boundedness specifically, which leads to a slightly different result. Similarly, upper bounds [14, Theorem 7] about ordered multiply nested words, is related to upper bound we provide in Corollary 3 for OBMC. Nevertheless, as technical contributions, we first deal with context-boundedness, phase-boundedness and ordered MPDS uniformly independent of the notion of boundedness by following an automata-based approach reducing to the corresponding repeated reachability problem. In second step, we provide optimal complexity bounds by building on analysis for context-boundedness on [8,18] whereas for ordered MPDS it relies on [2]. Finally, our construction allows us to add regularity constraints on stack contents, extending notions from [11], that are known to go beyond first-order language, by an adaptation of the case for Multi-CaRet.

2 Preliminaries

We write $[N]$ to denote the set $\{1, 2, \ldots, N\}$. We also use a boldface as a shorthand for elements indexed by $[N]$, for e.g., $\boldsymbol{a} = \{a_i \mid i \in [N]\}$. For a finite word $w = a_1 \ldots a_k$ over the alphabet Σ, we write $|w|$ to denote its *length* k. For $0 \leq i < |w|$, $w(i)$ represents the $(i+1)$-th letter of the word, here a_{i+1}.

Pushdown systems provide a natural execution model for programs with recursion. A generalization with multiple stacks allows us to model threads, formally defined next. A *multi-pushdown system* (MPDS) is a tuple of the form $P = (G, N, \Gamma, \Delta_1, \ldots, \Delta_N)$, for some $N \geq 1$ such that G is a non-empty finite set of *global states*, Γ is the finite *stack alphabet* containing the distinguished letter $\bot$, for every $s \in [N]$, Δ_s is the *transition relation* acting on the s-th stack where Δ_s is a relation included in $G \times \Gamma \times G \times \mathfrak{A}(\Gamma)$ with $\mathfrak{A}(\Gamma)$ defined as $\mathfrak{A}(\Gamma) \stackrel{\text{def}}{=} \bigcup_{a \in \Gamma} \{\mathsf{call}(a), \mathsf{return}(a), \mathsf{internal}(a)\}$. Elements of the set $\mathfrak{A}(\Gamma)$ are to be thought of as *actions* modifying the stack with alphabet Γ. A *configuration* c of P is the global state along with contents of the N stacks, i.e. c belongs to $G \times (\Gamma^*)^N$. For every $s \in [N]$, we write $\rightarrow_s$ to denote the *one-step relation* w.r.t. the s-th stack. Given two configurations $c = (g, w_1, \ldots, w_s a, \ldots w_N)$ and $c' = (g', w_1, \ldots, w'_s, \ldots, w_N)$, $c \rightarrow_s c'$ iff $(g, a, g', \mathfrak{a}(b)) \in \Delta_s$ where $\mathfrak{a}(b)$ reflects the change in the stack enforcing one of the conditions below: $w_s = w'_s$, $\mathfrak{a} = \mathsf{return}$ and $a = b$, or $w'_s = w_s b$ and $\mathfrak{a} = \mathsf{internal}$, or $w'_s = w_s ab$ and $\mathfrak{a} = \mathsf{call}$. The letter $\bot$ from the stack alphabet plays a special role; indeed the initial content of each stack is precisely $\bot$. Moreover, $\bot$ cannot be pushed, popped or replaced by any other symbol. This is a standard way to constrain the transition relations and to check for 'emptiness' of the stack. We write $\rightarrow_P$ to denote the relation $(\bigcup_{s \in [N]} \rightarrow_s)$. Given a configuration c, there may exist c_1, c_2 and $i_1 \neq i_2 \in [N]$ such that $c \rightarrow_{i_1} c_1$ and $c \rightarrow_{i_2} c_2$, which is the fundamental property to consider such models as adequate for modeling concurrency. An infinite

run is an ω-sequence of configurations $c_0, c_1, c_2, \ldots$ s.t. for every $i \geq 0$, we have $c_i \rightarrow_P c_{i+1}$. If $c_i \rightarrow_s c_{i+1}$, then we say that for that step, the s-th stack is *active*. Similar notions can be defined for finite runs. A standard problem on MPDS is the state reachability problem: given a MPDS P, a configuration c and a global state g, is there a run from c to some configuration c' s.t. the state of c' is g?

An *enhanced* multi-pushdown system is a multi-pushdown system of the form $P = (G \times [N], N, \Gamma, \Delta_1, \ldots, \Delta_N)$ s.t. for every $s \in [N]$, $\Delta_s \subseteq (G \times \{s\}) \times \Gamma \times (G \times [N]) \times \mathfrak{A}(\Gamma)$. In such systems, the global state contains enough information to determine the next *active* stack. Observe that the way the one-step relation is defined, we do not necessarily need to carry this information as part of the finite control (see Lemma 1). We do that in order to enable us to assert about active stack in our logic (see Section 3), and for technical convenience.

Lemma 1. *Given $P = (G, N, \Gamma, \boldsymbol{\Delta})$, one can construct in polynomial time an enhanced $P' = (G \times [N], N, \Gamma, \boldsymbol{\Delta'})$ such that (I) for every infinite run of P of the form $c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ there is an infinite run $c'_0 \rightarrow_{s_0} c'_1 \rightarrow_{s_1} \cdots c'_t \rightarrow_{s_t} c'_{t+1} \cdots$ of P' such that $(\star)$ for $t \geq 0$, if $c_t = (g_t, \{w^t_s\}_s)$, then $c'_t = ((g_t, s_t), \{w^t_s\}_s)$ and (II) similarly, for every infinite run of P' of the form $c'_0 \rightarrow_{s_0} c'_1 \rightarrow_{s_1} \cdots c'_t \rightarrow_{s_t} c'_{t+1} \cdots$ there is an infinite run $c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ of P such that $(\star)$.*

The proof is by an easy verification. In the sequel, w.l.o.g., we consider enhanced MPDS only since all the properties that can be expressed in our logical languages are linear-time properties. For instance, there is a logspace reduction from the state reachability problem to its restriction to enhanced MPDS.

State reachability problem is known to be undecidable by a simple reduction from the non-emptiness problem for intersection of context-free grammars. This has motivated works on restrictions on runs so that decidability can be regained (for state reachability problem and for model-checking problems). We recall below standard notions for boundedness; other notions can be found in [13,9]. Definitions are provided for infinite runs but they can be adapted to finite runs.

For the notion of k-boundedness, a phase is understood as a sub-run such that a single stack is active (see e.g. [17]). Let $\rho = c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ be an infinite run and $k \geq 0$. We say that ρ is *k-bounded* if there exist positions $i_1 \leq i_2 \leq \ldots \leq i_{k-1}$ such that $s_t = s_{t+1}$ for all $t \in \mathbb{N} \setminus \{i_1 \ldots i_{k-1}\}$. In the notion of k-phase-boundedness defined below, a phase is understood as a sub-run such that return actions are performed on a single stack, see e.g. [12]. Let $\rho = c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ be an infinite run and $k \geq 0$. We say that ρ is *k-phase-bounded* if there is a partition $Y_1, \ldots, Y_\alpha$ of $\mathbb{N}$ with $\alpha \leq k$ such that for every $j \in [1, \alpha]$ there is $s \in [N]$ s.t. for every $i \in Y_j$, if a return action is performed from c_i to c_{i+1}, then it is done on the sth stack. Finally, in the notion of order-boundedness defined below, the stacks are linearly ordered and a return action on a stack can only be performed if the smallest stacks are empty, see e.g. [3]. Let P be a multi-pushdown system and $\preceq = ([N], \leq)$ be a total ordering. Let $\rho = c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ be an infinite run. We say that ρ is *$\preceq$-bounded* if for every $t \in \mathbb{N}$ that a return is performed on the s-th stack, all the stacks strictly smaller than s w.r.t. $\preceq$ are empty.

3 Specification Language Multi-CaRet

Below, we introduce Multi-CaRet, an extension of the logic CaRet proposed in [1], and dedicated to runs of MPDS (instead of for runs of recursive state machines as done in [1]). The logic below can be seen as a fragment of MSO and therefore the decidability results from [6,16] apply to the forthcoming model-checking problems. However, our definition makes a compromise between a language of linear properties that extends the logic from [1] and the most expressive logic for which our model-checking problems are known to be decidable. The logic below is expressively identical as well as syntactically and semantically similar to one in [14], except for the presence of regular constraints.

Models of Multi-CaRet are infinite runs of multi-pushdown systems. For each (enhanced) multi-pushdown system $P = (G \times [N], N, \Gamma, \Delta_1, \ldots, \Delta_N)$, the fragment Multi-CaRet(P) of CaRet that uses syntactic resources from P (namely G and $[N]$). Multi-CaRet is defined as the union of all the sub-languages Multi-CaRet(P). The grammar $\phi := g \mid s \mid \mathsf{call} \mid \mathsf{return} \mid \mathsf{internal} \mid \phi \vee \phi \mid \neg\phi \mid \mathsf{X}\phi \mid \phi\ \mathsf{U}\phi \mid \mathsf{X}^{\mathsf{a}}_s\phi \mid \phi\ \mathsf{U}^{\mathsf{a}}_s\phi \mid \mathsf{X}^{\mathsf{c}}_s\phi \mid \phi\ \mathsf{U}^{\mathsf{c}}_s\phi$, defines formulas of Multi-CaRet(P), with $s \in [N]$, $g \in G$. Models of Multi-CaRet(P) formulae are ω-sequences in $\left(G \times [N] \times (\Gamma^*)^N\right)^\omega$, which can be obviously understood as infinite runs of P.

Semantics. Given an infinite run $\rho = c_0 c_1 \ldots c_t \ldots$ with $c_t = (g_t, s_t, w_1^t, \ldots, w_N^t)$ for every position $t \in \mathbb{N}$, the satisfaction relation $\rho, t \models \phi$ with ϕ in Multi-CaRet(P) is defined inductively as follows (successor relations are defined just below and obvious clauses are omitted):

$\rho, t \models g$ iff $g_t = g$ and $\rho, t \models s$ iff $s_t = s$

$\rho, t \models \mathfrak{a}$ iff $\left(\mathfrak{a}, \left|w_{s_t}^{t+1}\right| - \left|w_{s_t}^{t}\right|\right) \in \{(\mathsf{call}, 1), (\mathsf{internal}, 0), (\mathsf{return}, -1)\}$

$\rho, t \models \phi_1\ \mathsf{U}\phi_2$ iff there is a sequence of positions $i_0 = t$, $i_1 \ldots$, i_k, s.t. for $j < k$, $i_{j+1} = \mathrm{succ}_\rho(i_j)$, $\rho, i_j \models \phi_1$ and $\rho, i_k \models \phi_2$

For $b \in \{\mathsf{a}, \mathsf{c}\}$ and $s \in [N]$:

$\rho, t \models \mathsf{X}^b_s\phi$ iff $\mathrm{succ}^{b,s}_\rho(t)$ is defined and $\rho, \mathrm{succ}^{b,s}_\rho(t) \models \phi$

$\rho, t \models \phi_1\ \mathsf{U}^a_s\phi_2$ iff there exists a sequence of positions $t \leq i_0 < i_1 \cdots < i_k$, where i_0 smallest such with $s_{i_0} = s$, for $j < k$, $i_{j+1} = \mathrm{succ}^{a,s}_\rho(i_j)$, $\rho, i_j \models \phi_1$ and $\rho, i_k \models \phi_2$

$\rho, t \models \phi_1\ \mathsf{U}^c_s\phi_2$ iff there exists a sequence of positions $t \geq i_0 > i_1 \cdots > i_k$, where i_0 greatest such with $s_{i_0} = s$, for $j < k$, $i_{j+1} = \mathrm{succ}^{c,s}_\rho(i_j)$, $\rho, i_j \models \phi_1$ and $\rho, i_k \models \phi_2$

Definition for $\models$ uses three successor relations: *global* successor relation, *abstract* successor relation that jumps to the first future position after a return action at the same level, if any, and the *caller* successor relation that jumps to the latest past position before a call action at the same level, if any. Here are the definitions: $\mathrm{succ}_\rho(t) \stackrel{\text{def}}{=} t + 1$ for every $t \in \mathbb{N}$; $\mathrm{succ}^{\mathsf{c},s}_\rho(t)$ (caller of s-th stack):

largest $t' < t$ s.t. $s_{t'} = s$ and $\left|w_s^{t'}\right| = |w_s^t| - 1$. If such a t' does not exist, then $\mathrm{succ}_\rho^{\mathtt{c},s}(t)$ is undefined; and $\mathrm{succ}_\rho^{\mathtt{a},s}(t)$ is defined when s is active at position t:

1. If $\left|w_s^{t+1}\right| = |w_s^t| + 1$ (call), then $\mathrm{succ}_\rho^{\mathtt{a},s}(t)$ is the smallest $t' > t$ such that $s_{t'} = s$ and $\left|w_s^{t'}\right| = |w_s^t|$. If there is no such t' then $\mathrm{succ}_\rho^{\mathtt{a},s}(t)$ is undefined.
2. If $\left|w_s^{t+1}\right| = |w_s^t|$ (internal), then $\mathrm{succ}_\rho^{\mathtt{a},s}(t)$ is the smallest $t' > t$ such that $s_{t'} = s$ (first position when sth stack is active).
3. If $\left|w_s^{t+1}\right| = |w_s^t| - 1$ (return), then $\mathrm{succ}_\rho^{\mathtt{a},s}(t)$ is undefined.

In the sequel, we write $\rho \models \phi$ whenever $\rho, 0 \models \phi$.

Adding Regularity Constraints. We define Multi-$\mathtt{CaRet}^{reg}$ as the extension of Multi-$\mathtt{CaRet}$ in which regularity constraints on stack contents can be expressed. Logic Multi-$\mathtt{CaRet}^{reg}$ is defined from Multi-$\mathtt{CaRet}$ by adding atomic formulae of the form $\mathtt{in}(s, \mathcal{A})$ where s is a stack identifier and $\mathcal{A}$ is a finite-state automaton over the stack alphabet Γ. The satisfaction relation $\models$ is extended accordingly: $\rho, t \models \mathtt{in}(s, \mathcal{A})$ iff $w_s^t \in \mathrm{L}(\mathcal{A})$ where $\mathrm{L}(\mathcal{A})$ is the set of finite words accepted by $\mathcal{A}$. Note that regularity constraints can be expressed on each stack.

Let us introduce the model-checking problems considered herein. The model-checking problem for MPDS (MC) is defined s.t. it takes as inputs a MPDS P, a configuration $\left(g, (\bot)^N\right)$ and a formula ϕ in Multi-$\mathtt{CaRet}(P)$ and asks whether there is an infinite run ρ from $\left(g, (\bot)^N\right)$ such that $\rho \models \phi$. We know that the model-checking problem for MPDS is undecidable whereas its restriction to a single stack is ExpTime-complete [1]. Now, let us turn to bounded model-checking problems. Bounded model-checking problem for MPDS (BMC) is defined such that it takes as inputs P, a configuration $\left(g, (\bot)^N\right)$, a formula ϕ in Multi-$\mathtt{CaRet}(P)$ and a bound $k \in \mathbb{N}$ and it asks whether there is an infinite k-bounded run ρ from $\left(g, (\bot)^N\right)$ such that $\rho \models \phi$. Note that $k \in \mathbb{N}$ is an input and not a parameter of BMC. This makes a significant difference for complexity since usually complexity can increase when passing from being a constant to being an input. Phase-bounded model-checking problem (PBMC) is defined similarly by replacing in the above definition 'k-bounded run' by 'k-phase-bounded run'. Similarly, we can obtain a definition with order-boundedness. Order-bounded model-checking problem for multi-pushdown systems (OBMC) is defined such that it takes as inputs P, a configuration $\left(g, (\bot)^N\right)$, a formula ϕ in Multi-$\mathtt{CaRet}(P)$ and a total ordering $\preceq = ([N], \leq)$ and it asks whether there is an infinite $\preceq$-bounded run ρ from $\left(g, (\bot)^N\right)$ such that $\rho \models \phi$.

The problem of repeated reachability of MPDS, written REP, is defined in the expected way with a generalized Büchi acceptance condition related to states. We refer to the problem restricted to k-bounded runs by BREP. Obviously, the variants with other notions of boundedness can be defined too. Now, the simplified version of Multi-$\mathtt{CaRet}$ consists of the restriction of Multi-$\mathtt{CaRet}$ in which atomic formulae are of the form (g, s) when enhanced MPDS are involved. For every $\mathcal{P}$ in {MC,BMC,PBMC,OBMC}, there is a logspace reduction to $\mathcal{P}$ restricted to formulae from the simplified language. The proof idea consists in adding to global states information about the next active stack and about the

type of action. In the sequel, w.l.o.g., we restrict ourselves to the simplified languages. By [16], we conclude that BMC, PBMC and OBMC are decidable (use of Courcelle's Theorem). However, it provides non-elementary upper bounds. As a main result of our paper, we show that BMC is ExpTime-complete when k is encoded in unary even in presence of regular constraints.

4 From Model-Checking to Repeated Reachability

Herein, we reduce the problem of model checking (MC) to the problem of repeated reachability (REP) while noting complexity features that are helpful later on (Theorem 1). This generalizes Vardi-Wolper reduction from LTL model-checking into non-emptiness for generalized Büchi automata, similarly to the approach followed in [14]; not only we have to tailor the reduction to Multi-`CaRet` and to MPDS but also we aim at getting tight complexity bounds afterwards. The instance of MC that we have is a MPDS P, a formula ϕ and initial state (g_0, i_0). For the instance of REP we will reduce to, we will denote the MPDS by $\widehat{P}$, the set of acceptance sets by $\mathcal{F}$ and the set of initial states by I_0.

Augmented Runs. Let ρ be a run of the multi-pushdown system $P = (G \times [N], N, \Gamma, \boldsymbol{\Delta})$ with $\rho \in (G \times [N] \times (\Gamma^*)^N)^\omega$. The multi-pushdown system $\widehat{P}$ is built in such a way that its runs correspond exactly to runs from P but augmented with pieces of information related to the satisfaction of subformulas (taken from the closure set $\mathrm{Cl}(\phi)$ elaborated on shortly), whether a stack is dead or not (using a tag from $\{\mathsf{alive}, \mathsf{dead}\}$) and whether the current call will ever be returned or not (using a tag from $\{\mathsf{noreturn}, \mathsf{willreturn}\}$). These additional tags will suffice to reduce the existence of a run satisfying ϕ to the existence of a run satisfying a generalized Büchi condition. First, we define from ρ an "augmented run" $\gamma(\rho)$ which is an infinite sequence from $(\widehat{G} \times [N] \times (\widehat{\Gamma}^*)^N)^\omega$ where $\widehat{G} = G \times \mathcal{P}(\mathrm{Cl}(\phi))^N \times \{\mathsf{noreturn}, \mathsf{willreturn}\}^N \times \{\mathsf{alive}, \mathsf{dead}\}^N$ and $\widehat{\Gamma} = \Gamma \times \mathcal{P}(\mathrm{Cl}(\phi)) \times \{\mathsf{noreturn}, \mathsf{willreturn}\}$. By definition, an augmented run is simply an ω-sequence but it remains to check that indeed, it will be also a run of the new system. We will see that $\widehat{G} \times [N]$ is the set of global states of $\widehat{P}$ and $\widehat{\Gamma}$ is the stack alphabet of $\widehat{P}$. Before defining $\gamma(\cdot)$ which maps runs to augmented runs, let us introduce the standard notion for *closure* but slightly tailored to our needs. Each global state is partially made of sets of formulas that can be viewed as future obligations. This is similar to what is done for LTL and is just a variant of Fischer-Ladner closure. An obligation for a stack is a set of subformulas that is locally consistent; such consistent sets are called atoms and they are defined below as well as the notion of closure. Given a formula ϕ, its *closure*, denoted $\mathrm{Cl}(\phi)$, is the smallest set that contains ϕ, the elements of $G \times [N]$, and satisfies the following properties ($b \in \{a, c\}$ and $s \in [N]$): (i) if $\neg\phi' \in \mathrm{Cl}(\phi)$ or $\mathsf{X}\phi' \in \mathrm{Cl}(\phi)$ or $\mathsf{X}^b_s\phi' \in \mathrm{Cl}(\phi)$ then $\phi' \in \mathrm{Cl}(\phi)$; (ii) if $\phi' \vee \phi'' \in \mathrm{Cl}(\phi)$, then $\phi', \phi'' \in \mathrm{Cl}(\phi)$; (iii) if $\phi' \,\mathsf{U}\, \phi'' \in \mathrm{Cl}(\phi)$, then ϕ', ϕ'', and $\mathsf{X}(\phi' \,\mathsf{U}\, \phi'') \in \mathrm{Cl}(\phi)$; (iv) if $\phi' \,\mathsf{U}^b_s\, \phi'' \in \mathrm{Cl}(\phi)$, then ϕ', ϕ'', and $\mathsf{X}^b_s(\phi' \,\mathsf{U}^b_s\, \phi'') \in \mathrm{Cl}(\phi)$; (v) if $\phi' \in \mathrm{Cl}(\phi)$ and ϕ' in not of the form $\neg\phi''$, then $\neg\phi' \in \mathrm{Cl}(\phi)$. The number of formulas in $\mathrm{Cl}(\phi)$ is linear in the

size of ϕ and P. An *atom* of ϕ, is a set $A \subseteq \mathrm{Cl}(\phi)$ that satisfies the following properties: (a) for $\neg\phi' \in \mathrm{Cl}(\phi)$, $\phi' \in A$ iff $\neg\phi' \notin A$; (b) or $\phi' \vee \phi'' \in \mathrm{Cl}(\phi)$, $\phi' \vee \phi'' \in A$ iff ($\phi' \in A$ or $\phi'' \in A$); (c) or $\phi'\ \mathsf{U}\phi'' \in \mathrm{Cl}(\phi)$, $\phi'\ \mathsf{U}\phi'' \in A$ iff $\phi'' \in A$ or ($\phi' \in A$ and $\mathsf{X}(\phi'\ \mathsf{U}\phi'') \in A$); (d) A contains exactly one element from $G \times [N]$. Let $\mathrm{Atoms}(\phi)$ denote the set of atoms of ϕ, along with empty set (used as special atom, use will become clear later). Note that there are $2^{\mathcal{O}(|\phi|)}$ atoms of ϕ.

We write $((g^t, s^t), \boldsymbol{w}^t)$ to denote the t-th configuration of ρ. We define the augmented run $\gamma(\rho)$ so that its t-th configuration is of the form $((\widehat{g^t}, s^t), \widehat{\boldsymbol{w}^t})$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{r}^t, \boldsymbol{d}^t)$ and $\widehat{w_j^t} = (w_j^t, v_j^t, u_j^t)$ for every j in $[N]$. We say that the stack j is *active* at time t if $s^t = j$. Then, we define dead-alive tag to be dead if and only if the stack is not active at or after the corresponding position. The idea of the closure as we discussed is to maintain the set of subformulas that hold true at each step. We will expect it to be the empty set if the stack is dead. As for willreturn-noreturn tag, it reflects whether a call action has a "matching" return. This is similar to the $\{\infty, ret\}$ tags in [1]. This may be done by defining tag to be noreturn if stack will never become smaller than what it is now. Finally, the formulas and willreturn-noreturn tag on the stack are defined to be what they were in the global state at the time when the corresponding letter was pushed.

$$\forall t \geq 0,\ j \in [N] : (d_j^t = \mathsf{dead}) \ \stackrel{\text{def}}{\Leftrightarrow}\ (\forall t' \geq t,\ s^{t'} \neq j). \tag{1}$$

$$\forall t \geq 0,\ j \in [N] \ \text{ with } d_j^t = \mathsf{alive},\ \psi \in \mathrm{Cl}(\phi):$$
$$\psi \in A_j^t \ \stackrel{\text{def}}{\Leftrightarrow}\ \rho, t' \models \psi \text{ where } t' \text{ is the least } t' \geq t \text{ with } s^{t'} = j. \tag{2}$$

$$\forall t \geq 0,\ j \in [N] \ \text{ with } d_j^t = \mathsf{dead}\text{: } A_j^t \stackrel{\text{def}}{=} \emptyset. \tag{3}$$

$$\forall t \geq 0,\ j \in [N] : (r_j^t = \mathsf{noreturn}) \ \stackrel{\text{def}}{\Leftrightarrow}\ (\forall t' \geq t,\ \left|w_j^{t'}\right| \geq \left|w_j^t\right|). \tag{4}$$

$$\forall t \geq 0,\ j \in [N] : v_j^t \stackrel{\text{def}}{=} A_j^{t_1} A_j^{t_2} \ldots A_j^{t_l} \text{ and } u_j^t \stackrel{\text{def}}{=} d_j^{t_1} d_j^{t_2} \ldots d_j^{t_l},$$
$$\text{where for } k \text{ in } [l]\text{: } t_k \text{ is largest } t_k \leq t \text{ such that } \left|w_j^{t_k}\right| = k - 1. \tag{5}$$

Construction. We construct a system which simulates the original system with accepting runs having augmentations faithful to the semantics described above in (1)-(5). We define the multi-pushdown system $\widehat{P}$ as $(\widehat{G} \times [N], N, \widehat{\Gamma}, \widehat{\boldsymbol{\Delta}})$ with the states and alphabet as defined earlier, and each transition relation $\widehat{\Delta_s}$ is defined s.t. $(\widehat{g}, s, \widehat{a}, \widehat{g'}, s', \mathfrak{a}(\widehat{a'}))$ is in $\widehat{\Delta_s}$ $\stackrel{\text{def}}{\Leftrightarrow}$ conditions from Fig. 1 are satisfied. The set $\mathcal{F}$ is defined by the following sets of accepting states:

(a) For each $\psi = \phi_1\ \mathsf{U}\phi_2 \in \mathrm{Cl}(\phi)$, we define $F_\psi^1 \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid \phi_2 \in A_s \text{ or } \psi \notin A_s\}$.
(b) For each abstract-until formula $\psi = \phi_1\ \mathsf{U}_s^{\mathsf{a}}\phi_2 \in \mathrm{Cl}(\phi)$, we define $F_\psi^2 \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid r_s = \mathsf{noreturn} \text{ and } (\phi_2 \in A_s \text{ or } \psi \notin A_s)\}$.
(c) For each $j \in [N]$, we define $F_j^3 \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid j = s\} \cup \{(\widehat{g}, s) \mid d_j = \mathsf{dead}\}$.
(d) For each $j \in [N]$, $F_j^4 \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid d_j = \mathsf{dead}\} \cup \{(\widehat{g}, s) \mid j = s, d_s = \mathsf{noreturn}\}$.

Lemma 2. *Let ρ be a run of P. Then, $\gamma(\rho)$ is a run of $\widehat{P}$ such that for every $F \in \mathcal{F}$, there is a global state in F that is repeated infinitely often.*

1. $((g,s),a_S,(g',s'),\mathfrak{a}(a'_S)) \in \Delta_s$
2. $d_s = \mathsf{alive}$
3. $\forall j \neq s,\ d_j = d'_j$
4. If $\mathfrak{a} = \mathsf{call}$, then $r_s = \mathsf{willreturn} \Rightarrow r'_s = \mathsf{willreturn}$ and $a'_r = r_s$
5. If $\mathfrak{a} = \mathsf{internal}$, then $r'_s = r_s$ and $a'_r = a_r$
6. If $\mathfrak{a} = \mathsf{return}$, then $r_s = \mathsf{willreturn}$ and $r'_s = a_r$
7. $\forall j \neq s,\ r_j = r'_j$
8. $(g,s) \in A_s$
9. $\forall j \neq s,\ A_j = A'_j$
10. $\mathsf{X}A_s \subseteq A'_{s'}$ $(= A_{s'})$
11. If $\mathfrak{a} = \mathsf{call}$, then $a'_A = A_s$
12. If $\mathfrak{a} = \mathsf{internal}$, then $\mathsf{X}^{\mathsf{a}}_s A_s \subseteq A'_s$ and $a'_A = a_A$.
13. If $\mathfrak{a} = \mathsf{return}$, then $\mathsf{X}^{\mathsf{a}}_s a_A \subseteq A'_s$
14. Further, $\mathsf{X}^{\mathsf{a}}_s A_s = \emptyset$ if
    (a) $\mathfrak{a} = \mathsf{call}$ and $r_s = \mathsf{noreturn}$, or
    (b) $\mathfrak{a} = \mathsf{return}$, or
    (c) $d'_s = \mathsf{dead}$
15. If $\mathfrak{a} = \mathsf{call}$, $\mathsf{X}^{\mathsf{c}}_s A'_{s'} = (\mathsf{X}^{\mathsf{c}}_s \mathrm{Atoms}(\phi)) \cap A_s$
16. If $\mathfrak{a} = \mathsf{internal}$, then $\mathsf{X}^{\mathsf{c}}_s A'_{s'} = \mathsf{X}^{\mathsf{c}}_s A_s$.
17. Let $b \in \{\mathsf{a},\mathsf{c}\}$. Let $\psi \in \mathrm{Cl}(\phi)$, $\psi = \phi_1\ \mathsf{U}^b_s \phi_2$. Then, $\psi \in A_s$ iff either $\phi_2 \in A_s$ or ($\phi_1 \in A_s$ and $\mathsf{X}^{\mathsf{a}}_s \psi \in A_s$).
18. Let $b \in \{\mathsf{a},\mathsf{c}\}$. Let $\psi \in \mathrm{Cl}(\phi)$, $\psi = \phi_1\ \mathsf{U}^b_j \phi_2$ with $j \neq s$. Then $\psi \in A_s$ iff $\psi \in A'_s$.
19. $\forall j$: If $d_j = \mathsf{dead}$, then $A_j = \emptyset$ and $r_j = \mathsf{noreturn}$.

Fig. 1. Conditions for $\widehat{\Delta_s}$. $\mathsf{X}A = \{\psi \mid \mathsf{X}\psi \in A\}$, $\mathsf{X}^{\mathsf{a}}_1 A = \{\psi \mid \mathsf{X}^{\mathsf{a}}_1 \psi \in A\}$ and $\widehat{a} = (a_S, a_A, a_r)$ (similarly, $\widehat{a'}$).

Lemma 3. *Let $\widehat{\rho}$ be a run of $\widehat{P}$ satisfying the acceptance condition $\mathcal{F}$. Then, $\widehat{\rho}$ projected over states of P, denoted $\Pi(\widehat{\rho})$, is a run of P and $\gamma(\Pi(\widehat{\rho})) = \widehat{\rho}$.*

From Lemmas 2 and 3 the soundness and completeness of the reduction follow if we define the set of new initial states I_0 for the REP problem as states with initial state (g_0, i_0) for the MC problem and ϕ present in the part tracking formulas that hold true: $I_0 = \{((g_0, \boldsymbol{A}, \boldsymbol{d}, \boldsymbol{r}), i_0) \mid \phi \in A_{i_0}\}$. This gives an exponential-time reduction from MC to REP as well as their bounded variants. Theorem 1 below can be viewed as a counterpart of [14, Theorem 3].

Theorem 1. *Let P be a MPDS with initial configuration $(g, (\bot)^N)$ and ϕ be a Multi-*`CaRet` *formula. Let $\widehat{P}$ be the system built from P, g and ϕ, I_0 be the associated set of initial states and $\mathcal{F}$ be the acceptance condition.* ***(I)*** *If ρ_1 is a run of P from $(g, (\bot)^N)$ then $\rho_2 = \gamma(\rho_1)$ is a run of $\widehat{P}$ satisfying $\mathcal{F}$ and (A)-(C) hold true.* ***(II)*** *If ρ_2 is a run of $\widehat{P}$ from some configuration with global state in I_0 and satisfying $\mathcal{F}$, then $\Pi(\rho_2)$ is a run of P and (A)-(C) hold true too.*

Conditions (A)–(C) are defined as follows: ***(A)*** *ρ_1 is k-bounded iff ρ_2 is k-bounded, for all $k \geq 0$;* ***(B)*** *ρ_1 is k-phase-bounded iff ρ_2 is k-phase-bounded, for all $k \geq 0$;* ***(C)*** *ρ_1 is $\preceq$-bounded iff ρ_2 is $\preceq$-bounded, for all total orderings of the stacks $\preceq = ([N], \leq)$.*

Note that at each position, ρ_1 and ρ_2 work on the same stack and perform the same type of action (call, return, internal move), possibly with slightly different letters. This is sufficient to guarantee the satisfaction of the conditions (A)–(C).

5 Complexity Analysis with Bounded Runs

Bounded Repeated Global State Reachability Problem. We evaluate the complexity of BREP as well as its variant restricted to a single accepting global state, written $\text{BREP}_{\text{single}}$. There is a logspace reduction from BREP to $\text{BREP}_{\text{single}}$ by copying the MPDS as many times as the cardinality of $\mathcal{F}$ (as done to reduce non-emptiness problem for generalized Büchi automata to non-emptiness problem for standard Büchi automata). This allows us to conclude about the complexity upper bound for BMC itself but it is worth noting that the MPDS obtained by synchronization has an exponential number of global states and therefore a refined complexity analysis is required to get optimal upper bounds. In order to analyze the complexity for $\text{BREP}_{\text{single}}$, we take advantage of proof techniques that are introduced earlier and for which we provide a complexity analysis that will suit our final goal. Namely, existence of an infinite k-bounded run s.t. a final global state (g_f, i_f) is repeated infinitely often is checked: (1) by first guessing a sequence of intermediate global states witnessing context switches of length at most $k+1$, (2) by computing the (regular) set of reachable configurations following that sequence and then (3) by verifying whether there is a reachable configuration leading to an infinite run s.t. (g_f, i_f) is repeated infinitely often and no stack switch is performed. The principle behind (2) is best explained in [17] but we provide a complexity analysis using the computation of $\text{post}^\star(X)$ along the lines of [18]. Sets $\text{post}^\star(X)$ need to be computed at most k times, which might cause an exponential blow-up (for instance if at each step the number of states were multiplied by a constant). Actually, computing $\text{post}^\star$ adds an additive factor at each step, which is essential for our analysis. Let us define $\text{BREP}_{\text{single}}$: it takes as inputs P, a configuration $\left((g,i),(\bot)^N\right)$, a global state (g_f, i_f) and $k \in \mathbb{N}$ and it asks whether there is an infinite k-bounded run ρ from $\left((g,i),(\bot)^N\right)$ s.t. (g_f, i_f) is repeated infinitely often.

Proposition 1. *$\text{BREP}_{\text{single}}$ can be solved in time $\mathcal{O}(|P|^{k+1} \times p(k,|P|))$ for some polynomial $p(\cdot,\cdot)$.*

The proof of Proposition 1 is at the heart of our complexity analysis and it relies on constructions from [8,18]. We take advantage of it with the input system $\widehat{P}$.

Corollary 1. *(I) BMC with k encoded with a unary representation is* ExpTime*-complete. (II) BMC with k in binary encoding is in 2*ExpTime*.*

Note that [6, Theorem 15] would lead to an ExpTime upper bound for BMC if k is not part of the input, see the ExpTime upper bound for the problem NESTED-TRACE-SAT($\mathcal{L}^-, k$) from [6]; in our case k *is* indeed part of the input and in that case, the developments in [6] will lead to a 2ExpTime bound by using the method used for NESTED-TRACE-SAT($\mathcal{L}^-, k$) even if k is encoded in unary. Indeed, somewhere in the proof, the path expression $succ_{\leq k}$ is exponential in the value k. Hence, Corollary 1(I) is the best we can hope for when k is part of the input of the model-checking problem. We write BMC^{reg} to denote the extension of BMC in which Multi-`CaRet` is replaced by

Corollary 2. *(I) BMC^{reg} with k encoded with an unary representation is* ExpTime*-complete. (II) BMC^{reg} with k in binary encoding is in* 2ExpTime*.*

Complexity Results for Other Boundedness Notions. We focus on the complexity analysis for OBMC and PBMC. Let $\text{OREP}_{\text{single}}$ be the variant of $\text{BREP}_{\text{single}}$ with ordered MPDS: it takes as inputs an ordered multi-pushdown system P, a configuration $\left((g,i),(\bot)^N\right)$, a global state (g_f,i_f) and it asks whether there is an infinite run ρ from $\left((g,i),(\bot)^N\right)$ such that (g_f,i_f) is repeated infinitely often. According to [2, Theorem 11], $\text{OREP}_{\text{single}}$ restricted to ordered multi-pushdown systems with k stacks can be checked in time $\mathcal{O}(|P|^{2^{d\ k}})$ where d is a constant. Our synchronized product $\widehat{P}$ is exponential in the size of formulas (see Section 4), whence OBMC is in 2ExpTime too (k is linear in the size of our initial P). Condition (C) from Theorem 1 needs to be used here.

Corollary 3. *OBMC is in* 2ExpTime*.*

The same complexity upper bound can be shown with regularity constraints.

Now, let us consider k-bounded-phase runs. Let us define $\text{PBREP}_{\text{single}}$ in a similar way: it takes as inputs a MPDS P, a configuration $\left((g,i),(\bot)^N\right)$, a global state (g_f,i_f) and $k\in\mathbb{N}$ and it asks whether there is an infinite k-phase-bounded run ρ from $\left((g,i),(\bot)^N\right)$ such that (g_f,i_f) is repeated infinitely often. In [3, Section 5], it is shown that non-emptiness for k-phase MPDS can be reduced to non-emptiness for ordered MPDS with $2k$ stacks. By inspecting the proof, we can conclude: a similar reduction can be performed for reducing the repeated reachability of a global state, and non-emptiness of k-phase P with N stacks is reduced to non-emptiness of one of N^k instances of P' with $2k$ stacks and each P' is in polynomial-size in $k+|P|$. Therefore, $\text{PBREP}_{\text{single}}$ is in 2ExpTime. Indeed, there is an exponential number of instances and checking non-emptiness for one of them can be done in double exponential time. By combining the different complexity measures above, checking an instance of $\text{PBREP}_{\text{single}}$ with $\widehat{P}$ requires time in $\mathcal{O}(N^k\times\left|\widehat{P}\right|^{2^{d\ 2k}})$ which is double-exponential in the size of P. Consequently, bounded model-checking with bounded-phase MPDS is in 2ExpTime too if the number of phases is encoded in unary.

Corollary 4. *(I) PBMC where k is encoded in unary is in* 2ExpTime*. (II) PBMC where k is encoded in binary is in* 3ExpTime*.*

Again, the same complexity upper bounds apply when regularity constraints are added. Note that an alternative proof of Corollary 4(I) can be found in the recent paper [7] where fragments of MSO are taken into account.

6 Conclusion

We showed that model-checking over MPDS with k-bounded runs is ExpTime-complete when k is an input bound encoded in unary, otherwise the problem is in 2ExpTime with a binary encoding. The logical language is a version of

CaRet in which abstract temporal operators are related to calls and returns and parameterized by the stacks, and regularity constraints on stack contents are present too. A 2ExpTime upper bound is also established with ordered MPDS or with k-phase bounded runs.

Acknowledgments. We thank the reviewers for their time and helpful comments. K. Bansal also thanks LSV & ENS Cachan (France), and Clark Barrett in New York, for making the internship in Summer 2011 and hence, this work, possible.

References

1. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
2. Atig, M.: Global model checking of ordered multi-pushdown systems. In: FST&TCS 2010. LIPICS, pp. 216–227 (2010)
3. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
4. Atig, M.F., Bouajjani, A., Narayan Kumar, K., Saivasan, P.: Linear-time model-checking for multithreaded programs under scope-bounding. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 152–166. Springer, Heidelberg (2012)
5. Bansal, K., Demri, S.: A note on the complexity of model-checking bounded multi-pushdown systems. Technical Report TR2012-949, NYU (December 2012)
6. Bollig, B., Cyriac, A., Gastin, P., Zeitoun, M.: Temporal logics for concurrent recursive programs: Satisfiability and model checking. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 132–144. Springer, Heidelberg (2011)
7. Bollig, B., Kuske, D., Mennicke, R.: The complexity of model-checking multi-stack systems (2012) (submitted)
8. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
9. Cyriac, A., Gastin, P., Kumar, K.N.: MSO decidability of multi-pushdown systems via split-width. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 547–561. Springer, Heidelberg (2012)
10. Esparza, J., Ganty, P.: Complexity of pattern-based verification for multithreaded programs. In: POPL 2011, pp. 499–510. ACM (2011)
11. Esparza, J., Kučera, A., Schwoon, S.: Model-checking LTL with regular valuations for pushdown systems. In: Kobayashi, N., Babu, C. S. (eds.) TACS 2001. LNCS, vol. 2215, pp. 316–339. Springer, Heidelberg (2001)
12. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS 2007, pp. 161–170. IEEE (2007)
13. La Torre, S., Napoli, M.: Reachability of multistack pushdown systems with scope-bounded matching relations. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 203–218. Springer, Heidelberg (2011)

14. La Torre, S., Napoli, M.: A temporal logic for multi-threaded programs. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 225–239. Springer, Heidelberg (2012)
15. La Torre, S., Parlato, G.: Scope-bounded multistack pushdown systems: fixed-point, sequentialization and tree-width. In: FSTTCS 2012. LIPICS, pp. 173–184 (2012)
16. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: POPL 2011, pp. 283–294. ACM (2011)
17. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
18. Schwoon, S.: Model-checking pushdown systems. PhD thesis, TUM (2002)

Multi-weighted Automata and MSO Logic

Manfred Droste and Vitaly Perevoshchikov*

Universität Leipzig, Institut für Informatik,
04109 Leipzig, Germany
{droste,perev}@informatik.uni-leipzig.de

Abstract. Weighted automata are non-deterministic automata where the transitions are equipped with weights. They can model quantitative aspects of systems like costs or energy consumption. The value of a run can be computed, for example, as the maximum, limit average, or discounted sum of transition weights. In multi-weighted automata, transitions carry several weights and can model, for example, the ratio between rewards and costs, or the efficiency of use of a primary resource under some upper bound constraint on a secondary resource. Here, we introduce a general model for multi-weighted automata as well as a multi-weighted MSO logic. In our main results, we show that this multi-weighted MSO logic and multi-weighted automata are expressively equivalent both for finite and infinite words. The translation process is effective, leading to decidability results for our multi-weighted MSO logic.

Keywords: Multi-priced automata, quantitative logic, average behavior, power series.

1 Introduction

Recently, multi-priced timed automata [6,5,17,20] have received much attention for real-time systems. These automata extend priced timed automata by featuring several price parameters. This permits to compute objectives like the optimal ratio between rewards and costs [6,5], or the optimal consumption of several resources where more than one resource must be restricted [20]. Arising from the model of timed automata, the multi-weighted setting has also attracted much notice for classical non-deterministic automata [1,3,16,18].

The goal of the present paper is to develop a multi-weighted monadic second order (MSO) logic and to show that it is expressively equivalent to multi-weighted automata.

Büchi's and Elgot's fundamental theorems [7,15] established the expressive equivalence of finite automata and MSO logic. Weighted MSO logic with weights taken from an arbitrary semiring was introduced in [12,10] and it was shown that a fragment of this weighted logic and semiring-weighted automata on finite and infinite words have the same expressive power [10]. Chatterjee, Doyen, and Henzinger [8,9] investigated weighted automata modeling the average and long-time

* Partially supported by DFG Graduiertenkolleg 1763 (QuantLA).

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 418–430, 2013.

behavior of systems. The behavior of such automata cannot be described by semiring-weighted automata. In [13,14], valuation monoids were presented to model the quantitative behaviors of these automata. Their logical characterization was given in [14]. In this paper, we establish, both for finite and infinite words, the Büchi-type result for multi-weighted automata; these do not fit into the framework of other weighted automata like semiring automata [2,11,19,22], or even valuation monoid automata [13,14].

First, we develop a general model for multi-weighted automata which incorporates several multi-weighted settings from the literature. Next, we define a multi-weighted MSO logic by extending the classical MSO logic with constants which could be tuples of weights. The semantics of formulas should be single weights (not tuples of weights). Different from weighted MSO logics over semirings or valuation monoids, this makes it impossible to define the semantics inductively on the structure of an MSO formula. Instead, for finite words, we introduce an intermediate semantics which maps each word to a finite multiset containing tuples of weights. The semantics of a formula is then defined by applying to the multiset semantics an operator which evaluates a multiset to a single value. Our Büchi-type result for multi-weighted automata on finite words is established by reducing it to the corresponding result of [14] for the product valuation monoid of finite multisets.

In the case of infinite words, it is usually not possible to collect all the information about weights of paths in finite multisets. Therefore, we cannot directly reduce the desired result to the proof given in [14] for infinite words. But we can use the result of [14] to translate each multi-weighted formula of our logic into an automaton over the product ω-valuation monoid of multisets, and we show that the weights of transitions in this automaton satisfy certain properties which allow us to translate it into a multi-weighted automaton.

All our automata constructions are effective. Thus, decision problems for multi-weighted logic can be reduced to decision problems of multi-weighted automata. Some of these problems for automata can be solved whereas for others the details still have to be explored.

2 Multi-weighted Automata on Finite Words

The model of *multi-weighted* (or *multi-priced*) automata is an extension of the model of weighted automata over semirings [2,11,19,22] and valuation monoids [13,14] by featuring several price parameters. In the literature, different situations of the behaviors of multi-weighted automata were considered (cf. [1,3,6,5,16,17,18,20]) to model the consumption of several resources. For instance, the model of multi-priced timed automata introduced in [6] permits to describe the optimal ratio between accumulated rewards and accumulated costs of transitions. In this section, we introduce a general model to describe the behaviors of multi-weighted automata on finite words.

Consider an automaton in which every transition carries a reward and a cost. For paths of transitions, we are interested in the ratio between accumulated rewards and accumulated costs. The automaton should assign to each word the maximal reward-cost ratio of accepting paths on w. The idea is to model the weights by elements of the set $M = \mathbb{R} \times \mathbb{R}_{\geq 0}$. We use a valuation function $\mathrm{val} : M^+ \to M$ to associate to each sequence of such weights a single weight in M. Since our automata are nondeterministic and a word may have several accepting paths, we obtain a multiset of weights of these paths, hence a multiset of elements from M. We use an evaluator function Φ which associates to each multiset of M a single value. The mapping Φ can be considered as a general summation operator. Now we turn to formal definitions.

To cover also the later case of infinite words, we let $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$. Let M be a set. A *multiset* over M is a mapping $r : M \to \overline{\mathbb{N}}$. For each $m \in M$, $r(m)$ is the number of copies of m in r. We let $\mathrm{supp}(r) = \{m \in M \mid r(m) \neq 0\}$, the *support* of r. We say that a multiset r is *finite* if $\mathrm{supp}(r)$ is finite and $\infty \notin r(M)$. We denote the collection of all multisets by $\overline{\mathbb{N}}\langle\langle M \rangle\rangle$ and the collection of all finite multisets by $\mathbb{N}\langle M \rangle$.

Definition 1. *Let K be a set. A* K-valuation structure (M, val, Φ) *consists of a set M, a* valuation function $\mathrm{val} : M^+ \to M$ *with* $\mathrm{val}(m) = m$ *for all $m \in M$, and an* evaluator function $\Phi : \mathbb{N}\langle M \rangle \to K$.

A *nondeterministic automaton* over an alphabet Σ is a tuple $\mathcal{A} = (Q, I, T, F)$ where Q is a set of *states*, $I, F \subseteq Q$ are sets of *initial* resp. *final states* and $T \subseteq Q \times \Sigma \times Q$ is a *transition relation. Finite paths* $\pi = (t_i)_{0 \leq i \leq n}$ of $\mathcal{A}$ are defined as usual as finite sequences of matching transitions, say $t_i = (q_i, a_i, q_{i+1})$. Then we call the word $w = a_0 a_1 ... a_n \in \Sigma^+$ the *label* of the path π and π a path on w. A path is *accepting* if it starts in I and ends in F. We denote the set of all accepting paths of $\mathcal{A}$ on $w \in \Sigma^+$ by $\mathrm{Acc}_{\mathcal{A}}(w)$.

Definition 2. *Let Σ be an alphabet, K a set and $\mathcal{M} = (M, \mathrm{val}, \Phi)$ a K-valuation structure. A* multi-weighted automaton *over Σ and $\mathcal{M}$ is a tuple (Q, I, T, F, γ) where (Q, I, T, F) is a nondeterministic automaton and $\gamma : T \to M$.*

Let $\mathcal{A}$ be a multi-weighted automaton over Σ and $\mathcal{M}$, $w \in \Sigma^+$ and $\pi = t_0 ... t_n$ a path on w. The *weight* of π is defined by $\mathrm{Weight}_{\mathcal{A}}(w) = \mathrm{val}(\gamma(t_i))_{0 \leq i \leq n}$. Let $|\mathcal{A}|(w) \in \mathbb{N}\langle M \rangle$ be the finite multiset containing the weights of all accepting paths in $\mathrm{Acc}_{\mathcal{A}}(w)$. Formally, $|\mathcal{A}|(w)(m) = |\{\pi \in \mathrm{Acc}_{\mathcal{A}}(w) \mid \mathrm{Weight}_{\mathcal{A}}(\pi) = m\}|$ for all $m \in M$. The *behavior* $||\mathcal{A}|| : \Sigma^+ \to K$ of $\mathcal{A}$ is defined for all $w \in \Sigma^+$ by $||\mathcal{A}||(w) = \Phi(|\mathcal{A}|(w))$.

Note that every weighted automaton over a valuation monoid $(M, +, \mathrm{val}, \mathbb{0})$ (cf. [13,14]) can be considered as a multi-weighted automaton over the K-valuation structure (M, val, Φ) with $K = M$ and $\Phi : \mathbb{N}\langle M \rangle \to M$ defined by $\Phi(r) = \sum(m \mid m \in \mathrm{supp}(r) \text{ and } 1 \leq i \leq r(m))$ (as usual, $\sum \emptyset = \mathbb{0}$). Moreover, multi-weighted automata extend the model of weighted automata over valuation monoids in two directions. First, whereas the weights of transitions in multi-weighted automata are taken from M, the behavior is a mapping with the

codomain K where K and M do not necessarily coincide. Second, we resolve the nondeterminism in multi-weighted automata using an evaluator function Φ defined on finite multisets.

Next, we consider several examples how to describe the behavior of multi-weighted automata known from the literature using valuation structures. In each of the three examples below, let Σ be an alphabet, $\mathcal{M} = (M, \mathrm{val}, \Phi)$ a K-valuation structure, and $\mathcal{A}$ a multi-weighted automaton over Σ and $\mathcal{M}$.

Example 1. Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$. Let $M = \mathbb{R} \times \mathbb{R}_{\geq 0}$, $K = \overline{\mathbb{R}}$, $\mathrm{val}((x_1, y_1), ..., (x_k, y_k)) = \left(\sum_{i=1}^{k} x_i, \sum_{i=1}^{k} y_i\right)$ be the componentwise sum, and Φ defined by $\Phi(r) = \max_{(x,y) \in \mathrm{supp}(r)} \frac{x}{y}$ where we put $\frac{x}{0} = \infty$ and $\max(\emptyset) = -\infty$. For instance, for every transition weight $(x, y) \in M$, x might mean the reward and y the cost of the transition. Then $||\mathcal{A}||(w)$ is the maximal ratio between accumulated rewards and costs of accepting paths on w. The ratio setting was considered first for multi-priced timed automata [6,5] and also for nondeterministic automata [3,18].

Example 2. Let $M = \mathbb{R} \times \mathbb{R}$, $K = \mathbb{R} \cup \{\infty\}$ and $p \in \mathbb{R}$. Let val be as in the previous example and $\Phi(r) = \min\{x \mid (x, y) \in \mathrm{supp}(r) \text{ and } y \leq p\}$, for $r \in \mathbb{N}\langle M \rangle$, with $\min(\emptyset) = \infty$. Let t be a transition and $\gamma(t) = (x, y)$. We call x the primary and y the secondary cost. Then $||\mathcal{A}||(w)$ is the cheapest primary cost of reaching with w some final state under the given upper bound constraint $p \in \mathbb{R}$ on the secondary cost. The optimal conditional reachability problem for multi-priced timed automata was studied in [20].

Example 3. Let $M = \mathbb{R}^n$ for some $n \geq 1$, $K = \mathbb{R}$, and val be the componentwise sum of vectors. We define $\Phi : \mathbb{N}\langle M \rangle \to \mathbb{R}$ as follows. Let $r \in \mathbb{N}\langle M \rangle$ and $S = \mathrm{supp}(r)$. Then $\Phi(r) = 0$ if $S = \emptyset$ and $\Phi(r) = \frac{\sum_{v \in S} r(v) \cdot ||v||}{\sum_{v \in S} r(v)}$ otherwise. Here, for $v = (v_1, ..., v_n)$, $||v|| = \sqrt{v_1^2 + ... + v_n^2}$ is the length of v. Suppose that $\mathcal{A}$ controls the movement of some object in $\mathbb{R}^n$ and each transition t carries the coordinates of the displacement vector of this object. Then, $||\mathcal{A}||(w)$ is the value of the average displacement of the object after executing w.

3 Multi-weighted MSO Logic on Finite Words

In this section, we wish to develop a multi-weighted MSO logic where the weight constants are elements of a set M. Again, if weight constants are *pairs* of a reward and a cost, the semantics of formulas must reflect the maximal reward-cost ratio setting, so the weights of formulas should be *single weights*. Then, there arises a problem to define the semantics function inductively on the structure of a formula as in [10,14]. We solve this problem in the following way. We associate to each word a multiset of elements of M. Here, for disjunction and existential quantification, we use the multiset union. For conjunction, we extend a product operation given on the set M to the Cauchy product of multisets. Similarly, for

universal quantification, we extend the valuation function on M^+ to $\mathbb{N}\langle M\rangle^+$. Then, we use an evaluator function Φ which associates to each multiset of elements a single value (e.g. the maximal reward-cost ratio of pairs contained in a multiset).

As in the case of weighted MSO logics over product valuation monoids [14], we extend a valuation structure (cf. Definition 1) with a unit element and a binary operation in order to define the semantics of atomic formulas and of the conjunction.

Definition 3. *Let K be a set. A* product K-valuation structure (K-pv-structure) $(M, \mathrm{val}, \diamond, \mathbb{1}, \Phi)$ *consists of a K-valuation structure (M, val, Φ), a constant $\mathbb{1} \in M$ with $\mathrm{val}(m\mathbb{1}...\mathbb{1}) = m$ for $m \in M$, and a multiplication $\diamond : M \times M \to M$ such that $m \diamond \mathbb{1} = \mathbb{1} \diamond m = m$ for all $m \in M$.*

For the rest of this section, we fix an alphabet Σ and a K-pv-structure $\mathcal{M} = (M, \mathrm{val}, \diamond, \mathbb{1}, \Phi)$. Let V be a countable set of first and second order variables. Lower-case letters like x, y denote first order variables whereas capital letters like X, Y etc. denote second order variables. The syntax of *multi-weighted MSO logic* over Σ and $\mathcal{M}$ is defined as in [4] by the grammar:

$$\beta ::= P_a(x) \mid x \le y \mid x \in X \mid \neg\beta \mid \beta \wedge \beta \mid \forall x\beta \mid \forall X\beta$$
$$\varphi ::= m \mid \beta \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x\varphi \mid \forall x\varphi \mid \exists X\varphi$$

where $a \in \Sigma$, $m \in M$, $x, y, X \in V$. The formulas β are called *boolean* formulas and the formulas φ *multi-weighted MSO*-formulas. Note that negation and universal second order quantification are allowed in boolean formulas only. Note also that the boolean formulas have the same expressive power as (unweighted) MSO logic.

The class of *almost boolean* formulas over Σ and M is the smallest class containing all constants $m \in M$ and all boolean formulas and which is closed under $\wedge$ and $\vee$. A multi-weighted MSO formula φ is *syntactically restricted* if whenever it contains a sub-formula $\forall x\psi$, then ψ is almost boolean, and if for every subformula $\varphi_1 \wedge \varphi_2$ of φ either both φ_1 and φ_2 are almost boolean, or φ_1 or φ_2 is boolean.

The set $\mathrm{Free}(\varphi)$ of free variables in φ is defined as usual. For $w \in \Sigma^+$, let $\mathrm{dom}(w) = \{0, ..., |w|\text{-}1\}$. Let $\mathcal{V}$ be a finite set of variables with $\mathrm{Free}(\varphi) \subseteq \mathcal{V}$. A $(\mathcal{V}, w)$-*assignment* is a mapping $\sigma : \mathcal{V} \to \mathrm{dom}(w) \cup 2^{\mathrm{dom}(w)}$ where every first order variable is mapped to an element of $\mathrm{dom}(w)$ and every second order variable to a subset of $\mathrm{dom}(w)$. The update $\sigma[x/i]$ for $i \in \mathrm{dom}(w)$ is defined as: $\sigma[x/i](x) = i$ and $\sigma[x/i]|_{\mathcal{V}\setminus\{x\}} = \sigma|_{\mathcal{V}\setminus\{x\}}$. The update for second order variables can be defined similarly. Each pair (w, σ) of a word and $(\mathcal{V}, w)$-assignment can be encoded as a word over the extended alphabet $\Sigma_{\mathcal{V}} = \Sigma \times \{0, 1\}^{\mathcal{V}}$. Note that a word $(w, \sigma) \in \Sigma_{\mathcal{V}}^+$ represents an assignment if and only if, for every first order variable in $\mathcal{V}$, the corresponding row in the extended word contains exactly one 1; then (w, σ) is called *valid*. The set of all valid words in $\Sigma_{\mathcal{V}}^+$ is denoted by $\mathcal{N}_{\mathcal{V}}$. We also denote by Σ_φ the alphabet $\Sigma_{\mathrm{Free}(\varphi)}$.

Consider again the collection $\mathbb{N}\langle M\rangle$ of all finite multisets over M. Here, we consider the set of natural numbers as the semiring $(\mathbb{N}, +, \cdot, 0, 1)$ where $+$ and $\cdot$ are usual addition and multiplication. The *union* $(r_1 \oplus r_2) \in \mathbb{N}\langle M\rangle$ of finite multisets $r_1, r_2 \in \mathbb{N}\langle M\rangle$ is defined by $(r_1 \oplus r_2)(m) = r_1(m) + r_2(m)$ for all $m \in M$. We define the *Cauchy product* $(r_1 \cdot r_2) \in \mathbb{N}\langle M\rangle$ of two finite multisets $r_1, r_2 \in \mathbb{N}\langle M\rangle$ by

$$(r_1 \cdot r_2)(m) = \sum (r_1(m_1) \cdot r_2(m_2) \mid m_1, m_2 \in M, m_1 \diamond m_2 = m)\,.$$

Note that in the equation above there are finitely many non-zero summands, because the multisets r_1 and r_2 are finite. Let $n \geq 1$ and $r_1, ..., r_n \in \mathbb{N}\langle M\rangle$. We also define the *valuation* $\mathrm{val}(r_1, ..., r_n) \in \mathbb{N}\langle M\rangle$ by

$$\mathrm{val}(r_1, ..., r_n)(m) = \sum \Big(\prod\nolimits_{i=1}^{n} r_i(m_i) \mid m_1, ..., m_n \in M, \mathrm{val}(m_1, ..., m_n) = m\Big)\,.$$

Note that the right side of the equation above also contains only finitely many non-zero summands. The *empty multiset* ε is the finite multiset whose support is empty. A *simple multiset* over M is a finite multiset $r \in \mathbb{N}\langle M\rangle$ such that $\mathrm{supp}(r) = \{m_r\}$ and $r(m_r) = 1$, so $r(m) = 0$ for all $m \neq m_r$. We denote such a simple multiset r by $[m_r]$. The collection of all simple multisets over M is denoted by $\mathrm{Mon}(M)$.

As opposed to the case of pv-monoids [14], the pv-structure $\mathcal{M}$ does not contain a commutative and associative sum operation to define the semantics of the disjunction and the existential quantification. For this, we employ the sum of multisets. Let φ be a multi-weighted formula over Σ and $\mathcal{M}$, and $\mathcal{V} \supseteq \mathrm{Free}(\varphi)$. We define the auxiliary multiset semantics function $\langle\varphi\rangle_{\mathcal{V}} : \Sigma_{\mathcal{V}}^{+} \to \mathbb{N}\langle M\rangle$ relying on the ideas of [10] (cf. also [14]) as follows: for all $(w, \sigma) \notin \mathcal{N}_{\mathcal{V}}$, $\langle\varphi\rangle_{\mathcal{V}}(w, \sigma) = \varepsilon$ and, for all $(w, \sigma) \in \mathcal{N}_{\mathcal{V}}$, $\langle\varphi\rangle_{\mathcal{V}}(w, \sigma)$ is defined inductively as shown in Table 1.

Here, $x, y, X \in \mathcal{V}, a \in \Sigma$, $m \in M$, β is a boolean formula and $\varphi, \varphi_1, \varphi_2$ are multi-weighted formulas. In Table 1, for the semantics of $\forall X\varphi$ the subsets $I \subseteq \mathrm{dom}(w)$ are enumerated in some fixed order, e.g. lexicographically. For a formula φ, we put $\langle\varphi\rangle = \langle\varphi\rangle_{\mathrm{Free}(\varphi)}$. Then, we define the *semantics* $\langle\!\langle\varphi\rangle\!\rangle : \Sigma_{\varphi}^{+} \to K$ as the composition $\langle\!\langle\varphi\rangle\!\rangle = \Phi \circ \langle\varphi\rangle$.

Example 4. Let A be an object on the plane whose displacement is managed by two types of commands: $\leftrightarrow$ and $\updownarrow$. After receiving the command $\leftrightarrow$ the object moves one step to the left or to the right; after receiving $\updownarrow$ one step up or down. Consider the $\mathbb{R}$-valuation structure $(\mathbb{R}^2, \mathrm{val}, \Phi)$ from Example 3. We define $\diamond$ as the componentwise sum of vectors and put $\mathbb{1} = (0, 0)$. Then, $\mathcal{M} = (\mathbb{R}^2, \mathrm{val}, \diamond, \mathbb{1}, \Phi)$ is an $\mathbb{R}$-pv-structure. Consider the following multi-weighted MSO sentence over the alphabet $\Sigma = \{\leftrightarrow, \updownarrow\}$ and the $\mathbb{R}$-pv-structure $\mathcal{M}$:

$$\varphi = \forall x((P_{\leftrightarrow}(x) \to ((-1, 0) \vee (1, 0))) \wedge (P_{\updownarrow}(x) \to ((0, -1) \vee (0, 1))))$$

where, for a boolean formula φ and a multi-weighted formula ψ, $\beta \to \psi$ is an abbreviation for $(\beta \wedge \psi) \vee \neg\beta$. For every sequence of commands $w \in \Sigma^{+}$, the multiset $\langle\varphi\rangle(w)$ contains all possible displacement vectors of the object.

Table 1. The auxiliary multiset semantics of multi-weighted MSO formulas over a pv-structure

$$\langle m\rangle_{\mathcal{V}}(w,\sigma) = [m]$$

$$\langle P_a(x)\rangle_{\mathcal{V}}(w,\sigma) = \begin{cases} [\mathbb{1}], & \text{if } w_{\sigma(x)} = a, \\ \varepsilon, & \text{otherwise} \end{cases}$$

$$\langle x \le y\rangle_{\mathcal{V}}(w,\sigma) = \begin{cases} [\mathbb{1}], & \text{if } \sigma(x) \le \sigma(y), \\ \varepsilon, & \text{otherwise} \end{cases}$$

$$\langle x \in X\rangle_{\mathcal{V}}(w,\sigma) = \begin{cases} [\mathbb{1}], & \text{if } \sigma(x) \in \sigma(X), \\ \varepsilon, & \text{otherwise} \end{cases}$$

$$\langle \neg\beta\rangle_{\mathcal{V}}(w,\sigma) = \begin{cases} [\mathbb{1}], & \text{if } \langle\beta\rangle_{\mathcal{V}}(w,\sigma) = \varepsilon, \\ \varepsilon, & \text{otherwise} \end{cases}$$

$$\langle \varphi_1 \vee \varphi_2\rangle_{\mathcal{V}}(w,\sigma) = \langle\varphi_1\rangle_{\mathcal{V}}(w,\sigma) \oplus \langle\varphi_2\rangle_{\mathcal{V}}(w,\sigma)$$

$$\langle \varphi_1 \wedge \varphi_2\rangle_{\mathcal{V}}(w,\sigma) = \langle\varphi_1\rangle_{\mathcal{V}}(w,\sigma) \cdot \langle\varphi_2\rangle_{\mathcal{V}}(w,\sigma)$$

$$\langle \exists x\varphi\rangle_{\mathcal{V}}(w,\sigma) = \bigoplus_{i \in \mathrm{dom}(w)} \langle\varphi\rangle_{\mathcal{V}\cup\{x\}}(w, \sigma[x/i])$$

$$\langle \exists X\varphi\rangle_{\mathcal{V}}(w,\sigma) = \bigoplus_{I \subseteq \mathrm{dom}(w)} \langle\varphi\rangle_{\mathcal{V}\cup\{X\}}(w, \sigma[X/I])$$

$$\langle \forall x\varphi\rangle_{\mathcal{V}}(w,\sigma) = \mathrm{val}\big(\langle\varphi\rangle_{\mathcal{V}\cup\{x\}}(w,\sigma[x/i])\big)_{i \in \mathrm{dom}(w)}$$

$$\langle \forall X\beta\rangle_{\mathcal{V}}(w,\sigma) = \mathrm{val}\big(\langle\beta\rangle_{\mathcal{V}\cup\{X\}}(w,\sigma[X/I])\big)_{I \subseteq \mathrm{dom}(w)}$$

For example, let $w = {\leftrightarrow}{\leftrightarrow}$. The object has 4 possibilities to move: 1) two steps to the right; 2) one step to the right and then to the home position; 3) one step to the left and then to the home position; 4) two steps to the left. Then $\langle\varphi\rangle(w) = [(2,0),(0,0),(0,0),(-2,0)]$. The average displacement of the object is given by $\langle\!\langle\varphi\rangle\!\rangle$ for each sequence of commands w. For example, $\langle\!\langle\varphi\rangle\!\rangle({\leftrightarrow}{\leftrightarrow}) = 1$, $\langle\!\langle\varphi\rangle\!\rangle({\leftrightarrow}{\updownarrow}) = \sqrt{2}$.

Note that the multi-weighted MSO logic over K-pv-structures contains the case of weighted MSO logic over semirings (cf. [12,10]). Hence, in general, multi-weighted MSO logic is expressively more powerful than multi-weighted automata.

Our main result for finite words is the following theorem.

Theorem 1. *Let Σ be an alphabet, K a set, $\mathcal{M} = (M, \mathrm{val}, \diamond, \mathbb{1}, \Phi)$ a K-pv-structure and $s\colon \Sigma^+ \to K$. Then $s = ||\mathcal{A}||$ for some multi-weighted automaton $\mathcal{A}$ over Σ and $\mathcal{M}$ iff $s = \langle\!\langle\varphi\rangle\!\rangle$ for a syntactically restricted multi-weighted MSO sentence φ over Σ and $\mathcal{M}$.*

The proof is similar to the proof of the corresponding Theorem 2 for infinite words. For lack of space, we skip it.

We consider examples of decision problems for multi-weighted MSO logic.

Example 5. Let Σ be an alphabet and $\mathcal{M} = (\mathbb{Q} \times \mathbb{Q}_{\ge 0}, \mathrm{val}, \diamond, (0,0), \Phi)$ the $\overline{\mathbb{R}}$-pv-structure where $\diamond$ is the componentwise sum, and val and Φ are defined as in Example 1. Let φ be a multi-weighted MSO sentence over Σ and $\mathcal{M}$, and $\nu \in \mathbb{Q}$ a threshold. The *$\ge\nu$-emptiness problem* is whether there exists a word $w \in \Sigma^+$ such that $\langle\!\langle\varphi\rangle\!\rangle(w) \ge \nu$. If φ is syntactically restricted, then, using our Theorem 1, we can effectively translate φ into a multi-weighted automaton over Σ and $\mathcal{M}$. Then $\ge\nu$-emptiness for these multi-weighted automata can be decided in the following way. First, we use a shortest path algorithm to decide whether

there exists a path with cost 0, i.e. $||\mathcal{A}||(w) = \infty \geq \nu$ for some w. If this is not the case (i.e. the costs of all accepting paths in $\mathcal{A}$ are strictly positive), we use the same technique as for the $\geq\nu$-emptiness problem for ratio automata with strictly positive costs (cf. [18], Theorem 3). We replace the weight (r, c) of every transition by the single value $r - \nu c$ and obtain a weighted automaton $\mathcal{A}'$ over the max-plus semiring $\mathbb{Q} \cup \{-\infty\}$. Then, $||\mathcal{A}||(w) \geq \nu$ iff the semiring-behavior of $\mathcal{A}'$ on w is not less than zero. Then, the decidability of our problem follows from the decidability of the $\geq$0-emptiness problem for max-plus automata.

Example 6. Let Σ be an alphabet and $\mathcal{M} = (\mathbb{Q}^2, \mathrm{val}, \diamond, (0,0), \Phi)$ where $\diamond$ is the componentwise sum, and val and Φ are as in Example 2. Again, using our Theorem 1, we can reduce the $\leq\nu$-emptiness problem (defined similarly as in Example 5) for syntactically restricted multi-weighted MSO logic over Σ and $\mathcal{M}$ to the emptiness problem for multi-weighted automata. This problem is decidable, since the optimal conditional reachability for multi-priced timed automata is decidable [20].

4 Multi-weighted Automata and MSO Logic on Infinite Words

In this section, we develop a general model for both multi-weighted automata and MSO logic on infinite words. Recall that, for a set M, $\overline{\mathbb{N}}\langle\langle M\rangle\rangle$ is the collection of all multisets over M. Let M^ω denote the set of all ω-infinite words over M.

Definition 4. *Let K be a set. A* product K-ω-valuation structure (K-ω-pv structure) *is a tuple $(M, \mathrm{val}^\omega, \diamond, \mathbb{1}, \Phi)$ where*

- *M is a set, $\mathbb{1} \in M$ and $\Phi : \overline{\mathbb{N}}\langle\langle M\rangle\rangle \to K$;*
- *$\mathrm{val}^\omega : M^\omega \to M$ with $\mathrm{val}^\omega(m\mathbb{1}^\omega) = m$ for all $m \in M$;*
- *$\diamond : M \times M \to M$ such that $m \diamond \mathbb{1} = \mathbb{1} \diamond m = m$ for all $m \in M$.*

A *Muller automaton* over an alphabet Σ is a tuple $\mathcal{A} = (Q, I, T, \mathcal{F})$ where Q is a set of states, $I \subseteq Q$ is a set of initial states, $T \subseteq Q \times \Sigma \times Q$ is a transition relation and $\mathcal{F} \subseteq 2^Q$ is a Muller acceptance condition. *Infinite paths* $\pi = (t_i)_{i\in\omega}$ of $\mathcal{A}$ are defined as infinite sequences of matching transitions, say $t_i = (q_i, a_i, q_{i+1})$. Then we call the word $w = (a_i)_{i\in\omega}$ the *label* of the path π and π a path on w. We say that a path $\pi = (q_i, a_i, q_{i+1})_{i\in\omega}$ is *accepting* if $q_0 \in I$ and $\{q \in Q \mid q = q_i \text{ for infinitely many } i \in \omega\} \in \mathcal{F}$. Let $\mathrm{Acc}_\mathcal{A}(w)$ denote the set of all accepting paths of $\mathcal{A}$ on w.

For the rest of this section, we fix an alphabet Σ and a K-ω-pv structure $\mathcal{M} = (M, \mathrm{val}^\omega, \diamond, \mathbb{1}, \Phi)$.

Definition 5. *A* multi-weighted Muller automaton *over Σ and $\mathcal{M}$ is a tuple $\mathcal{A} = (Q, I, T, \mathcal{F}, \gamma)$ where $(Q, I, T, \mathcal{F})$ is a Muller automaton and $\gamma : T \to M$.*

Let $\mathcal{A}$ be a multi-weighted Muller automaton over Σ and $\mathcal{M}$, $w \in \Sigma^\omega$ and $\pi = (t_i)_{i\in\omega}$ an accepting path on w. The *weight* of π is defined by $\mathrm{Weight}_\mathcal{A}(\pi) =$

$\mathrm{val}^\omega(\gamma(t_i))_{i\in\omega}$. Let $|\mathcal{A}|(w) \in \overline{\mathbb{N}}\langle\langle M\rangle\rangle$ be the multiset containing the weights of paths in $\mathrm{Acc}_\mathcal{A}(w)$. Formally, $|\mathcal{A}|(w)(m) = |\{\pi \in \mathrm{Acc}_\mathcal{A}(w) \mid \mathrm{Weight}_\mathcal{A}(w) = m\}|$ where, for an infinite set X, we put $|X| = \infty$. The *behavior* of $\mathcal{A}$ is the ω-series $||\mathcal{A}|| : \Sigma^\omega \to K$ defined by $||\mathcal{A}||(w) = \Phi(|\mathcal{A}|(w))$.

Remark 1. The multiplication $\diamond$, the unital element $\mathbb{1}$ and the condition $\mathrm{val}^\omega(m\mathbb{1}^\omega) = m$ are irrelevant for the definition of the behaviors of multi-weighted automata. However, they will be used to describe the semantics of multi-weighted MSO formulas.

We consider several examples of multi-weighted automata $\mathcal{A}$ over Σ and $\mathcal{M}$, and their behaviors.

Example 7. Consider the reward-cost ratio setting of Example 1 for infinite words. For a sequence $(r_i, c_i)_{i\in\omega} \in (\mathbb{R} \times \mathbb{R}_{\geq 0})^\omega$ of reward-cost pairs, the *supremum ratio* (cf. [6]) is defined by $\limsup_{n\to\infty} \frac{\sum_{i=0}^n r_i}{\sum_{i=0}^n c_i} \in \overline{\mathbb{R}}$ where $\frac{r}{0} = \infty$. Unfortunately, since $\sum_{i=0}^\infty r_i$ and $\sum_{i=0}^\infty c_i$ may not exist or may be infinite, we cannot proceed as for finite words by considering pairs of accumulated rewards and costs and their ratios. Instead, we can define $\mathcal{M}$ as follows. Let $M = \overline{\mathbb{R}} \times \mathbb{R}_{\geq 0}$, $K = \overline{\mathbb{R}}$ and $\mathbb{1} = (0, 0)$. Let $\mu = (r_i, c_i)_{i\in\omega} \in (\mathbb{R} \times \mathbb{R}_{\geq 0})^\omega$. If $\sum_{i=0}^\infty r_i$ and $\sum_{i=0}^\infty c_i$ are finite, then we put $\mathrm{val}^\omega(\mu) = (\sum_{i=0}^\infty r_i, \sum_{i=0}^\infty c_i)$. Otherwise, we put $\mathrm{val}^\omega(\mu) = \left(\limsup_{n\to\infty} \frac{\sum_{i=0}^n r_i}{\sum_{i=0}^n c_i}, 1\right)$. For sequences $\mu \in M^\omega \setminus (\mathbb{R} \times \mathbb{R}_{\geq 0})^\omega$, we define $\mathrm{val}^\omega(\mu)$ arbitrarily keeping $\mathrm{val}^\omega(m\mathbb{1}^\omega) = m$. Let also $\diamond$ be the componentwise sum where ∞+(-∞) is defined arbitrarily. The evaluator function Φ is defined by $\Phi(r) = \sup_{(x,y)\in\mathrm{supp}(r)} \frac{x}{y}$. Then, $||\mathcal{A}||(w)$ is the maximal supremum ratio of accepting paths of w. The corresponding model for timed automata was considered in [6,5].

Example 8. Let $E_{\max} = (E^1_{\max}, ..., E^n_{\max}) \in \mathbb{Z}^n$ where $E^i_{\max} > 0$ for all i, and $M = [-E_{\max}, E_{\max}] \subseteq \mathbb{Z}^n$, i.e. M consists of all vectors $(v^1, ..., v^n) \in \mathbb{Z}^n$ such that $-E^i_{\max} \leq v^i \leq E^i_{\max}$ for each $i \in \{1, ..., n\}$. Let $K = \mathbb{B} = \{\mathrm{false}, \mathrm{true}\}$, the boolean semiring and $\mathbb{1} = (0, ..., 0)$. For $u_1 = (u^1_1, ..., u^n_1)$ and $u_2 = (u^1_2, ..., u^n_2) \in M$, we put $u_1 \diamond u_2 = (v^1, ..., v^n)$ where $v^i = \max\{\min\{u^i_1 + u^i_2, E^i_{\max}\}, -E^i_{\max}\}$. For $(m_i)_{i\in\omega} \in M^\omega$ we define the sequence $(v_i)_{i\in\omega}$ in M as follows. We put $v_0 = (0, ..., 0)$ and $v_{i+1} = v_i \diamond m_i$ for all $i \in \omega$. Then, let $\mathrm{val}^\omega((m_i)_{i\in\omega}) = (x^1, ..., x^n) \in M$ where $x^j = \inf\{v^j_i \mid i \in \omega\}$ for all $1 \leq j \leq n$. Let Φ be defined by $\Phi(r) = \mathrm{true}$ iff there exists $(m^1, ..., m^n) \in \mathrm{supp}(r)$ with $m^j \geq 0$ for all $1 \leq j \leq n$. This model corresponds to the one-player energy games considered in [16].

The syntax of the *multi-weighted MSO logic* over Σ and $\mathcal{M}$ is defined exactly as for finite words (cf. Section 3). To define the semantics of this logic, we proceed similarly as for finite words, i.e. by means of the auxiliary multiset semantics. For this, we consider $\overline{\mathbb{N}}$ as the totally complete semiring $(\overline{\mathbb{N}}, +, \cdot, 0, 1)$ (cf. [10]) where $0 \cdot \infty = \infty \cdot 0 = 0$. The sum $\oplus$ and the Cauchy product $\cdot$ for infinite multisets

from $\overline{\mathbb{N}}\langle\langle M\rangle\rangle$ are defined as for finite words. The ω-valuation $\mathrm{val}^\omega(r_i)_{i\in\omega}$ for $r_i \in \overline{\mathbb{N}}\langle\langle M\rangle\rangle$ is defined for all $m \in M$ by

$$\mathrm{val}^\omega((r_i)_{i\in\omega})(m) = \sum \Big(\prod\nolimits_{i\in\omega} r_i(m_i) \mid (m_i)_{i\in\omega} \in M^\omega \text{ and } \mathrm{val}^\omega(m_i)_{i\in\omega} = m\Big).$$

The *empty multiset* $\varepsilon \in \overline{\mathbb{N}}\langle\langle M\rangle\rangle$ and *simple multisets* $[m] \in \overline{\mathbb{N}}\langle\langle M\rangle\rangle$ (for $m \in M$) are defined in the same way as for finite words. Let $\mathrm{Mon}(M) = \{[m] \mid m \in M\}$.

Let φ be a multi-weighted MSO formula over Σ and $\mathcal{M}$, and $\mathcal{V} \supseteq \mathrm{Free}(\varphi)$. We define the auxiliary multiset semantics $\langle\varphi\rangle_\mathcal{V} : \Sigma^\omega_\mathcal{V} \to \overline{\mathbb{N}}\langle\langle M\rangle\rangle$ inductively on the structure of φ as in Table 1 where we have to replace val by val^ω. For $w \in \Sigma^\omega$, we let $\mathrm{dom}(w) = \omega$. To define the semantics $\langle\forall X\varphi\rangle$, we have to extend val^ω for multisets to index sets of size continuum such that $\mathrm{val}^\omega((r_i)_{i\in I}) = \varepsilon$ whenever $r_i = \varepsilon$ for some $i \in I$, and $\mathrm{val}^\omega(([\mathbb{1}])_{i\in I}) = [\mathbb{1}]$. The *semantics* of φ is defined by $\langle\langle\varphi\rangle\rangle = \Phi \circ \langle\varphi\rangle$.

Example 9. Assume that a bus can operate two routes A and B which start and end at the same place. The route R lasts t_R time units and profits p_R money units on the average per trip, for $\mathrm{R} \in \{\mathrm{A}, \mathrm{B}\}$. We may be interested in making an infinite schedule for this bus which is represented as an infinite sequence from $\{\mathrm{A}, \mathrm{B}\}^\omega$. This schedule must be fair in the sense that both routes A and B must occur infinitely often in this timetable (even if the route A or B is unprofitable). The optimality of the schedule is also preferred (we wish to profit per time unit as much as possible). We consider the K-ω-pv structure $\mathcal{M}$ from Example 7 and a one-element alphabet $\Sigma = \{\tau\}$ which is irrelevant here. Now we construct a weighted MSO sentence φ over Σ and $\mathcal{M}$ to define the optimal income of the bus per time unit (supremum ratio between rewards and time):

$$\varphi = \exists X\Big(\overset{\infty}{\exists} x(x \in X) \wedge \overset{\infty}{\exists} x(x \notin X) \wedge \forall x((x \in X \to (p_\mathrm{A}, t_\mathrm{A})) \wedge (x \notin X \to (p_\mathrm{B}, t_\mathrm{B}))\Big)$$

where $\overset{\infty}{\exists} x\psi$ is an abbreviation for a boolean formula $\forall y(\neg\forall x(\neg(y \leq x \wedge \psi)))$. Here, the second order variable X corresponds to the set of positions in an infinite schedule which can be assigned to the route A. Then,

$$|\varphi|(\tau^\omega) = \sup\left\{\limsup_{n\to\infty} \frac{p_\mathrm{A}\cdot|I\cap\overline{n}| + p_\mathrm{B}\cdot|I^c\cap\overline{n}|}{t_\mathrm{A}\cdot|I\cap\overline{n}| + t_\mathrm{B}\cdot|I^c\cap\overline{n}|} \mid I \subseteq \mathbb{N} \text{ with } I, I^c \text{ infinite}\right\}$$

where $\overline{n} = \{0, ..., n\}$ and $I^c = \mathbb{N} \setminus I$.

Now we state our main result for infinite words.

Theorem 2. *Let Σ be an alphabet, K a set and $\mathcal{M} = (M, \mathrm{val}^\omega, \diamond, \mathbb{1}, \Phi)$ a K-ω-pv structure. Let $s : \Sigma^\omega \to K$ be an ω-series. Then $s = ||\mathcal{A}||$ for some multi-weighted Muller automaton $\mathcal{A}$ over Σ and $\mathcal{M}$ iff $s = \langle\langle\varphi\rangle\rangle$ for some syntactically restricted multi-weighted MSO sentence φ over Σ and $\mathcal{M}$.*

In the rest of this section, we give the proof idea of this theorem. Let $\mathbf{mwMA}(\Sigma, \mathcal{M})$ denote the collection of all multi-weighted Muller automata

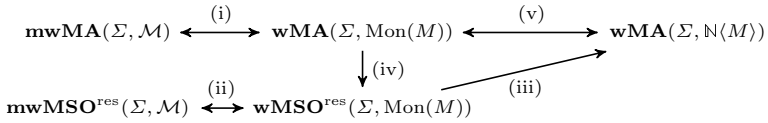


Fig. 1. The proof scheme of Theorem 2

over Σ and $\mathcal{M}$. Let $\mathcal{A} \in \mathbf{mwMA}(\Sigma, \mathcal{M})$. We can consider $|\mathcal{A}|$ as an ω-series $|\mathcal{A}| : \Sigma^\omega \to \overline{\mathbb{N}}\langle\langle M \rangle\rangle$. We call $|\mathcal{A}|$ the *multiset-behavior* of $\mathcal{A}$. Then $||\mathcal{A}|| = \Phi \circ |\mathcal{A}|$. Let $\mathbf{mwMSO}^{\mathrm{res}}(\Sigma, \mathcal{M})$ denote the set of all syntactically restricted multi-weighted MSO sentences over Σ and $\mathcal{M}$. Since, for any multi-weighted formula φ, $\langle\langle \varphi \rangle\rangle = \Phi \circ \langle \varphi \rangle$, it suffices to show that $\mathbf{mwMA}(\Sigma, \mathcal{M})$ with the multiset-behavior and $\mathbf{mwMSO}^{\mathrm{res}}(\Sigma, \mathcal{M})$ with the multiset-semantics are expressively equivalent.

For this, we can show that $(\overline{\mathbb{N}}\langle\langle M \rangle\rangle, \oplus, \mathrm{val}^\omega, \cdot, \varepsilon, [\mathbb{1}])$ is an ω-pv monoid as defined in [14]. Let $D \subseteq \overline{\mathbb{N}}\langle\langle M \rangle\rangle$. We denote by $\mathbf{wMA}(\Sigma, D)$ the collection of weighted automata over Σ and the ω-pv monoid $\overline{\mathbb{N}}\langle\langle M \rangle\rangle$ where the weights of transitions are taken from D. Let $\mathbf{wMSO}^{\mathrm{res}}(\Sigma, D)$ denote the set of syntactically restricted sentences over Σ and the ω-pv monoid $\overline{\mathbb{N}}\langle\langle M \rangle\rangle$ with constants from D. Let $[\![\varphi]\!]$ denote the semantics of $\varphi \in \mathbf{wMSO}^{\mathrm{res}}(\Sigma, \mathcal{M})$ as defined in [14]. The proof scheme of our result is depicted in Fig. 1. Here, $\leftrightarrow$ means the expressive equivalence and $\to$ the expressive inclusion.

(i) If we replace the weight $m \in M$ of every transition of a multi-weighted automaton $\mathcal{A}$ by the simple multiset $[m] \in \mathrm{Mon}(M)$, we obtain a weighted automaton $\mathcal{A}'$ over the pv monoid $\mathbb{N}\langle M \rangle$ such that the pv-monoid behavior of $\mathcal{A}'$ is equal to $|\mathcal{A}|$. Conversely, we can replace the weights $[m]$ in $\mathcal{A}'$ by m to obtain a multi-weighted automaton with the same behavior.

(ii) Similarly to (i), we replace the constants m occurring in MSO formulas by simple multisets $[m]$ and vice versa.

(iii) The proof is based on the proof of Theorem 6.2 (a) of Droste and Meinecke [14]. We proceed inductively on the structure of $\varphi \in \mathbf{wMSO}^{\mathrm{res}}(\Sigma, \mathrm{Mon}(M))$. Using the property $\mathrm{val}^\omega(m\mathbb{1}^\omega) = m$ for $m \in M$, we show that every almost boolean formula is equivalent to a weighted Muller automaton with weights from $\mathrm{Mon}(M) \subseteq \mathbb{N}\langle M \rangle$. Let φ, φ_1 and φ_2 be weighted MSO formulas with constants from $\mathrm{Mon}(M)$ such that $[\![\varphi]\!]$, $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$ are recognizable by weighted Muller automata with weights from $D \subseteq \overline{\mathbb{N}}\langle\langle M \rangle\rangle$. Let β be any boolean formula. It can be shown that $[\![\varphi_1 \vee \varphi_2]\!]$, $[\![\exists x \varphi]\!]$, $[\![\exists X \varphi]\!]$ and $[\![\varphi \wedge \beta]\!] = [\![\beta \wedge \varphi]\!]$ are also recognizable by weighted Muller automata with weights from D. If φ is almost boolean, then $[\![\varphi]\!]$ is an ω-recognizable step function with coefficients from $\mathbb{N}\langle M \rangle$. Using the construction of Lemma 8.11 of [10], cf. Theorem 6.2 of [14], we establish that $[\![\forall x \varphi]\!]$ is recognizable by a weighted automaton with weights from $\mathbb{N}\langle M \rangle$.

(iv) The proof follows from Theorem 6.2 of [14] where a weighted automaton with weights in $D \subseteq \overline{\mathbb{N}}\langle\langle M \rangle\rangle$ was translated into an MSO sentence with weights in D.

(v) Let $\mathcal{A} = (Q, I, T, \mathcal{F}, \gamma) \in \mathbf{wMA}(\Sigma, \mathbb{N}\langle M \rangle)$. We construct an automaton $\mathcal{A}' = (Q', I', T', \mathcal{F}', \gamma') \in \mathbf{wMA}(\Sigma, \mathrm{Mon}(M))$ with the same behavior by unfolding each single transition of $\mathcal{A}$ labeled by a finite multiset into several transitions labeled by elements of this multiset as follows.
- $Q' = I \cup \{(q, m, i) : t = (p, a, q) \in T, m \in \mathrm{supp}(\gamma(t)), 1 \leq i \leq \gamma(t)(m)\}$
- $I' = I$, $\mathcal{F}' = \{\{(q_1, m_1, k_1), ..., (q_n, m_n, k_n)\} \subseteq Q' \setminus I \mid \{q_1, ..., q_n\} \in \mathcal{F}\}$.
- $T' = T_1 \cup T_2$, where T_1 consists of all transitions $(p, a, (q, m, i))$ from $I \times \Sigma \times (Q' \setminus I)$ with $(p, a, q) \in T$; T_2 consists of all transitions $((q_1, m_1, i_1), a, (q_2, m_2, i_2))$ from $(Q' \setminus I) \times \Sigma \times (Q' \setminus I)$ with $(q_1, a, q_2) \in T$.
- For all $t = (q', a, (q, m, i)) \in T'$, let $\gamma'(t) = [m]$.

5 Conclusion

We have extended the use of weighted MSO logic to a new class of multi-weighted settings. We just note that, as in [14], for K-pv-structures and K-ω-pv structures with additional properties there are larger fragments of multi-weighted MSO logic which are still expressively equivalent to multi-weighted automata. Since our translations from formulas to automata are effective, we can reduce the decidability problems for multi-weighted logics to the corresponding problems for multi-weighted automata. Decidability results of, e.g., [6,16,18,20] lead to decidability results for multi-weighted nondeterministic automata. However, for infinite words, the authors did not consider Muller acceptance condition for automata. Therefore, our future work will investigate decision problems for multi-weighted Muller automata. Also, weighted MSO logic for weighted timed automata was investigated in [21]. In our further work, we wish to combine the ideas of [21] and the current work to obtain a Büchi theorem for multi-weighted timed automata.

References

1. Bauer, S., Juhl, L., Larsen, K., Legay, A., Srba, J.: A logic for accumulated-weight reasoning on multiweighted modal automata. In: Proc. of the 6th Int. Symp. on Theoret. Aspects of Software Engineering (TASE 2012), pp. 77–84. IEEE Computer Society Press (2012)
2. Berstel, J., Reutenauer, C.: Rational Series and Their Languages. EATCS Monographs on Theoretical Computer Science, vol. 12. Springer (1988)
3. Bloem, R., Greimel, K., Henzinger, T.A., Jobstmann, B.: Synthesizing robust systems. In: Proc. of 9th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD 2009), pp. 85–92. IEEE (2009)
4. Bollig, B., Gastin, P.: Weighted versus probabilistic logics. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 18–38. Springer, Heidelberg (2009)
5. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design 32(1), 3–23 (2008)

6. Bouyer, P., Brinksma, E., Larsen, K.G.: Staying alive as cheaply as possible. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 203–218. Springer, Heidelberg (2004)
7. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik und Grundl. Math. 6, 66–92 (1960)
8. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
9. Chatterjee, K., Doyen, L., Henzinger, T.A.: Expressiveness and closure properties for quantitative languages. Logical Methods in Comp. Sci. 6(3) (2010)
10. Droste, M., Gastin, P.: Weighted automata and weighted logics. In: Droste et.al.: [11], ch. 5
11. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. EATCS Monographs on Theoretical Computer Science. Springer (2009)
12. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theoret. Comp. Sci. 380(1-2), 69–86 (2007)
13. Droste, M., Meinecke, I.: Weighted automata and regular expressions over valuation monoids. Int. J. Found. Comput. Sci. 22(8), 1829–1844 (2011)
14. Droste, M., Meinecke, I.: Weighted automata and weighted MSO logics for average and long-time behaviors. Inf. Comput. 221, 44–59 (2012)
15. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Amer. Math. Soc. 98, 21–51 (1961)
16. Fahrenberg, U., Juhl, L., Larsen, K.G., Srba, J.: Energy games in multiweighted automata. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 95–115. Springer, Heidelberg (2011)
17. Fahrenberg, U., Larsen, K.G., Thrane, C.R.: Model-based verification and analysis for real-time systems. In: Software and Systems Safety - Specification and Verification, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 30, pp. 231–259. IOS Press (2011)
18. Filiot, E., Gentilini, R., Raskin, J.F.: Quantitative languages defined by functional automata. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 132–146. Springer, Heidelberg (2012)
19. Kuich, W., Salomaa, A.: Semirings, Automata and Languages. EATCS Monographs on Theoretical Computer Science, vol. 5. Springer (1986)
20. Larsen, K.G., Rasmussen, J.I.: Optimal conditional reachability for multi-priced timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 234–249. Springer, Heidelberg (2005)
21. Quaas, K.: MSO logics for weighted timed automata. Form. Methods Syst. Des. 38(3), 193–222 (2011)
22. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science. Springer (1978)

Overlapping Tile Automata

David Janin*

Université de Bordeaux, LaBRI UMR 5800,
351, cours de la Libération, F-33405 Talence, France
janin@labri.fr

Abstract. Premorphisms are monotonic mappings between partially ordered monoids where the morphism condition $\varphi(xy) = \varphi(x)\varphi(y)$ is relaxed into the condition $\varphi(xy) \leq \varphi(x)\varphi(y)$. Their use in place of morphisms has recently been advocated in situations where classical algebraic recognizability collapses. With languages of overlapping tiles, an extension of classical recognizability by morphisms, called quasi-recognizability, has already proved both its effectiveness and its power. In this paper, we complete the theory of such tile languages by providing a notion of (finite state) non deterministic tile automata that capture quasi-recognizability in the sense that quasi-recognizable languages correspond to finite boolean combinations of languages recognizable by finite state non deterministic tile automata. As a consequence, it is also shown that quasi-recognizable languages of tiles correspond to finite boolean combination of upward closed (in the natural order) languages of tiles definable in Monadic Second Order logic.

Introduction

Motivations and Background. There are many ways to describe one-dimensional overlapping tiles : the objects which are studied in this paper. Arising in inverse semigroup theory, they can be defined as (representations of) elements of a McAlister monoid [15], i.e. linear and unidirectional birooted trees. Then they are used in studies [11,12] of the structure of tiling (in the usual sense with no overlaps) of the d-dimensional Euclidian space $\mathbb{R}^d$.

Overlapping tiles can also be seen as two way string objects extended by extra history recording capacities that prevent a new letter from being placed in a position where another distinct letter has already been positioned in the past.

For instance, starting from a given string object, say ab with two distinct letters a and b, one can remove letter b from the right of that object. The resulting object is denoted by abb^{-1}. If these objects are treated just as standard string objects, b^{-1} acts as the *group inverse* of b, and thus $bb^{-1} = 1$ henceforth $abb^{-1} = a$. If these string objects are treated as strings extended with recording capacity as mentioned above, then $abb^{-1} \neq a$. In that case, b^{-1} acts as the *pseudo-inverse* of b and bb^{-1} is seen rather as sort of a footprint of letter b that is kept on the right side of letter a.

* Partially funded by the project CONTINT 2012 - ANR 12 CORD 009 02 - INEDIT.

A.A. Bulatov and A.M. Shur (Eds.): CSR 2013, LNCS 7913, pp. 431–443, 2013.

Now if one adds that same letter b to the right of abb^{-1} again, this leads to build again the object $abb^{-1}b$ which, in both cases, equals the string object ab. Indeed, adding letter b to the right of its footprint bb^{-1} merely amounts to rebuilding b, i.e. we have $bb^{-1}b = b$.

On the contrary, if one tries to add letter a - distinct from b - to the right of abb^{-1} then the resulting object $abb^{-1}a$ equals 0: the undefined tile. Indeed, with standard string objects the resulting value would be aa but with overlapping tiles, no other letter than the original can be positioned on the right footprint bb^{-1} of b hence $bb^{-1}a = 0$ and thus $abb^{-1}a = 0$.

Surprisingly, this extension of the string data type turns out to be a mathematically well-founded and structurally rich extension of that type. Adding and removing letters to the right or to the left of an extended string induces an associative product: the underlying algebraic structure is the monoid of overlapping tiles. The history recording mechanism induces a partial order relation: the natural order defined on tiles.

Recent modeling experiments in computational music [1,10], conducted with variants of overlapping tiles, further illustrate how the structural richness of the monoid of tiles can be used to great benefit. Indeed many derived operators, e.g. sequential or parallel compositions, can be defined from the inverse monoid structure.

There is thus a need to develop a language theory for overlapping tiles. Such a study has been initiated in [9].

Doing so, an immediate difficulty is that, as already observed for inverse monoids [16,19], classical language theory tools are not directly applicable. Indeed, on birooted trees [19] or on positive overlapping tiles [9], the notion of recognizability defined via morphisms into finite monoids or, equivalently, defined by via classical finite state (two way) automata, collapses in terms of expressive power.

As a remedy, the use of premorphisms instead of morphisms has been successfully proposed [7]. Indeed, this variant of algebraic recognizability, called quasi-recognizability, is shown to *essentially* captures the expressive power of Monadic Second Order Logic (MSO) over tiles [9]. The purpose of the present paper is to extend and strengthen such an emerging algebraic theory by providing it with an automata theoretical counterpart.

Outline. Overlapping tile automata are non deterministic finite state word automata with a semantics (acceptance condition) that is now defined in terms of overlapping tiles.

We first show that languages of tiles recognized by such finite state automata correspond to upward closed (in some natural order) languages definable in MSO. Then, we prove that they capture the notion of quasi-recognizable languages of tiles [7] in the sense that quasi-recognizable languages of tiles correspond to finite boolean combination of languages of tiles definable by finite state tile automata.

It must be mentioned that our former definition of recognizability by premorphisms was only defined for languages of positive tiles. This new automata theoretical approach induces a refined definition that, equivalent to our former proposal on positive tiles, is now applicable to languages of arbitrary positive and negative tiles.

The paper is organized as follows. Monoids of overlapping tiles and the related notion of tile automata are presented in Section 1. They are shown to capture upward closed (in the natural order) MSO definable languages of tiles (Theorem 1).

Special classes of premorphisms and partially ordered monoids, referred to as adequate, are defined and studied in Section 2. They provide the appropriate concepts for defining an effective notion of quasi-recognizability (Lemma 2).

Tile automata and quasi-recognizable languages are then related in Section 3. It is shown that quasi-recognizable languages of tiles exactly correspond to finite boolean combinations of languages recognizable by finite tile automata (Theorem 3) or, equivalently, finite boolean combinations of upward MSO definable languages of tiles (Corollary 1).

1 Overlapping Tiles and Their Automata

Here we briefly give a description of the McAlister monoid [15] on the alphabet A, or, as presented and studied in [9], the monoid of one-dimensional overlapping tiles. Then we define and study the notion of overlapping tile automata.

1.1 Preliminaries

Let A be a finite alphabet A and let A^* be the free monoid generated by A with neutral denoted by 1. The concatenation of two words u and v is denoted by $u \cdot v$ or simply uv. The monoid A_0^* is defined as the extension of the free monoid A^* with a zero with $0 \cdot u = u \cdot 0 = 0$ for every $u \in A_0^*$.

Let $\leq_p$ (resp. $\leq_s$) be the prefix (resp. the suffix) order over A_0^*, that is, for every u and $v \in A_0^*$, $u \leq_p v$ (resp. $u \leq_s v$) when there exists $w \in A_0^*$ such that $uw = v$ (resp. $wu = v$). Observe that for every $u \in A_0^*$, we have $u \leq_p 0$ (resp. $v \leq_s 0$). Let then $\vee_p$ be the *(prefix) join* operator defined, for every u and $v \in A_0^*$, by $u \vee_p v = v$ when $u \leq_p v$, by $u \vee_p v = u$ when $v \leq_p u$ and by $u \vee_p v = 0$ otherwise. One can check that $\vee_p$ is indeed the join for the set A_0^* ordered by the prefix order. The *(suffix) join* operator $\vee_s$ is defined symmetrically.

Given $\bar{A}$ a disjoint copy of A, let $u \mapsto \bar{u}$ be the syntactic dual mapping defined by $\bar{1} = 1$ and, for every every letter $a \in A$ and every $u \in (A + \bar{A})^*$, by $\overline{ua} = \bar{u} \cdot \bar{a}$ and $\overline{u\bar{a}} = \bar{u} \cdot a$. The *free group* $FG(A)$ generated by A is defined as the quotient of $(A + \bar{A})^*$ by the least congruence $\simeq$ such that, for every letter $a \in A$, $a\bar{a} \simeq 1$ and $\bar{a}a \simeq 1$. As usual, every element $[u] \in FG(A)$ is represented by the unique word $v \in [u]$ that contains no factors of the form $a\bar{a}$ or $\bar{a}a$.

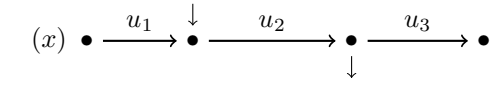
1.2 Overlapping Tiles

A *one-dimensional overlapping tile*, or just *tile*, over the alphabet A is a triple of words $x = (u_1, u_2, u_3) \in A^* \times (A^* + \bar{A}^*) \times A^*$ such that, if $u_2 \in \bar{A}^*$, the syntactic inverse $\overline{u_2} \in A^*$ is a suffix of the word u_1 and a prefix of the word u_3.

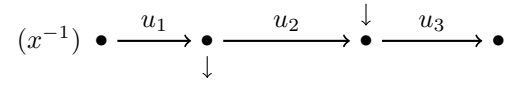
When $u_2 \in A^*$ we say that x is a *positive tile*. When $u_2 \in \bar{A}^*$ we say that x is a *negative tile*. When $u_2 = 1$, i.e. when x is both positive and negative, we say that x is a *context tile*. Sets T_A, T_A^+, T_A^- and C_A will respectively denote the set of tiles, the set of positive tiles, the set of negative tiles and the set of context tiles over A.

The *(syntactic) inverse* of a tile $x \in T_A$ is defined to be the tile $x^{-1} = (u_1 \cdot u_2, \bar{u}_2, u_2 \cdot u_3)$, with the product defined in the free group $FG(A)$. One can easily check that the induced mapping $x \mapsto x^{-1}$ is an involution that maps positive (resp. negative) tiles to negative (resp. positive) tiles and that, restricted to context tiles, is the identity mapping.

The *word domain* of a tile $x = (u_1, u_2, u_3)$ is the word $u_1 \cdot u_2 \cdot u_3 \in A^*$ with product performed in $FG(A)$. The *directed root* of the tile x is the word $u_2 \in A^* + \bar{A}^*$. A positive tile $x = (u_1, u_2, u_3)$ is conveniently drawn as a (linear, unidirectional and left to right) Munn's birooted word tree [17] with dangling arrows to identify the input vertex and the output vertex that marks the extremities of the directed root of x.

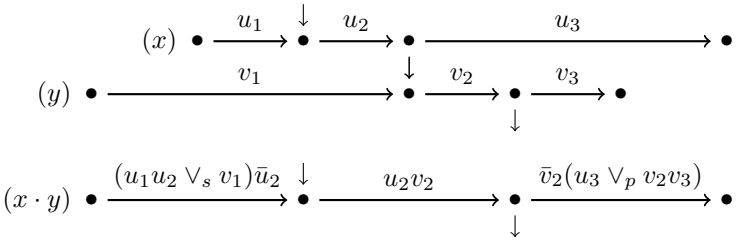


The negative tile $x^{-1} = (u_1 u_2, \overline{u_2}, u_2 u_3)$ is obtained from x by just inverting the input and the output vertices.



The product $x \cdot y$ of two tiles $x = (u_1, u_2, u_3)$ and $y = (v_1, v_2, v_3) \in T_A$ is defined in two steps: the output vertex of x is first positioned (or synchronized) with the input vertex of y, then, if possible, the word domains of x and y are merged letter by letter. The input vertex (resp. output vertex) of the resulting product $x \cdot y$ is then defined to be the input vertex of x (resp. the output vertex of y).

In the case $x = (u_1, u_2, u_3) \in T_A^+$ and $y = (v_1, v_2, v_3) \in T_A^+$, these two phases can be depicted as follows:



Formally, the (partial) *product* of two non-zero tiles $x = (u_1, u_2, u_3)$ and $y = (v_1, v_2, v_3)$ is defined as

$$x \cdot y = ((u_1 u_2 \vee_s v_1)\overline{u_2}, u_2 v_2, \overline{v_2}(u_3 \vee_p v_2 v_3))$$

when both pattern matching conditions, $u_1 u_2 \vee_s v_1 \neq 0$ and $u_3 \vee_p v_2 v_3 \neq 0$, when evaluated in $FG(A)$, are satisfied.

Adding an undefined tile denoted by 0, this product is completed by $x \cdot y = 0$ when any of the pattern matching condition is not satisfied with $x \cdot 0 = 0 = 0 \cdot x$ for every $x \in T_A$.

For instance, with $A = \{a, b, c\}$ we have $(a, b, a) \cdot (b, a, c) = (a, ba, c)$ while $(a, b, a) \cdot (a, a, c) = 0$ since $a \neq b$. Another example is given by $(1, a, 1) \cdot (ba, c, ab) = (b, ac, ab)$. It illustrates the fact that the left and the right parts of the resulting product can arbitrarily come from any of the two product components.

As a special case of product, when $x = (u_1, u_2, 1)$ and $y = (1, v_2, v_2)$ the product $x \cdot y$ is always non zero. In that case, we say that the product $x \cdot y$ is a *disjoint product*.

The resulting set T_A^0 equipped with the above product is a *monoid* with unit $1 = (1, 1, 1)$ that is shown [9,4] to be isomorphic to the McAlister monoid [15] generated by A. Extending the inverse mapping to 0 by taking $0^{-1} = 0$, for every tile $x \in T_A^0$, the tile x^{-1} is the unique tile such that both $xx^{-1}x = x$ and $x^{-1}xx^{-1} = x^{-1}$. The monoid T_A^0 is thus an *inverse* monoid [14].

As such, the idempotent elements are the elements of the form xx^{-1} (equivalently $x^{-1}x$) and the *natural order* associated with the inverse monoid T_A^0 is defined by $x \leq y$ when $x = xx^{-1}y$ (or, equivalently $x = yx^{-1}x$). One can easily check that for every non zero tile $x = (u_1, u_2, u_3)$ and $y = (v_1, v_2, v_3)$, we have $x \leq y$ if and only if $u_1 \geq_s v_1$, $u_2 = v_2$ and $u_3 \geq_p v_3$. One can also check that for every $x \in T_A^0$, $x \leq 1$ if and only if $x \cdot x = x$. In the sequel, idempotent tiles are thus also called *subunits*.

For every non zero tile $x = (u_1, u_2, u_3) \in T_A$, we define the *left projection* $x^L = x^{-1}x = (u_1 u_2, 1, u_3)$ and the *right projection* $x^R = xx^{-1} = (u_1, 1, u_2 u_3)$. One can check that $x^R \cdot x = x = x \cdot x^L$. Even more, the tile x^R (resp. x^L) is the least (in the natural order) idempotent tile $z \in T_A^0$ such that $z \cdot x = x$ (resp. $x = x \cdot z$). As an immediate consequence of the definition, for every x and $y \in T_A^0$, the product $x \cdot y$ is a disjoint product if and only if $x \cdot y \neq 0$ and 1 is the unique idempotent element that is both above x^L and y^R in the natural order. These notions of projections and disjoint products play a central role in the notion of adequately ordered monoid and adequate premorphism that are presented below.

Last, one can observe that the mapping $u \mapsto (1, u, 1)$ from A^* to T_A is a one-to-one morphism. In other words, the free monoid A^* is embedded into the McAslister monoid T_A^0 of overlapping tiles. In the remainder of the text we may use the same notation for words of A^* and their images in T_A^0.

1.3 Tile Automata

Definition 1. *A non deterministic (finite)* overlapping tile automaton *is a triple* $\mathcal{A} = \langle Q, \delta, K \rangle$ *with a (finite) set of states* Q*, a non deterministic transition function* $\delta : A \to \mathcal{P}(Q \times Q)$ *and an accepting set* $K \subseteq Q \times Q$*.*

An run of the automaton $\mathcal{A}$ *on a word* $u = a_1 \cdots a_n \in A^*$ *from state* p *to state* q*, which is denoted by* $p \xrightarrow{u} q$*, is a sequence of* $n+1$ *states* $q_0 = p$*,* q_1*,* q_2*,* $\ldots$*,* $q_n = q \in Q$ *such that for every* $1 \leq i \leq n$*, we have* $(q_{i-1}, q_i) \in \delta(a_i)$*.*

A run of the automaton $\mathcal{A}$ *on a positive tile* $x = (u, v, w) \in T_A^+$ *(resp. a negative tile* $x = (uv, \bar{v}, vw) \in T_A^-$*) is quadruple of states* $(s, p, q, e) \in Q \times Q \times Q \times Q$*: a start state* s*, an input state* p*, an output state* q *and an end state* e*, such that* $s \xrightarrow{u} p$*,* $p \xrightarrow{v} q$ *and* $q \xrightarrow{w} e$ *(resp.* $s \xrightarrow{u} q$*,* $q \xrightarrow{v} p$ *and* $p \xrightarrow{w} e$*).*

Such a run is an accepting run when $(p, q) \in K$*. The set of tiles over which there is an accepting run of the automaton* $\mathcal{A}$ *is denoted by* $L(\mathcal{A}) \subseteq T_A$*. It is the language of tiles* recognized *by the automaton* $\mathcal{A}$*.*

Examples. Let us consider the tile automaton (graph) defined with the set of states $Q = \{1, 2, 3\}$ and transitions $\delta(a) = \{(1,1), (1,2), (2,3), (3,3)\}$ for every $a \in A$. With $K^+ = \{(1,2)\}$ or $K^+ = \{(2,3)\}$ we recognize the language of all strictly positive tiles. With $K^- = \{(2,1)\}$ or $K^- = \{(3,2)\}$ we recognize the language of all strictly negative tiles. With $K^0 = \{(2,2)\}$ we recognize the language of all idempotent tiles.

As an immediate consequence of tile automata semantics, every language of tiles recognized by a tile automaton is upward closed in the natural order. The following theorem shows that this is actually the characteristic property of these languages.

Theorem 1. *For every language of tiles* $L \subseteq T_A$*, the language* L *is recognizable by a finite state overlapping tile automaton if and only if the language* L *is upward closed (in the natural order) and definable in monadic second order logic.*

Proof. This essentially follows from the characterization of MSO definable languages that is provided by Theorem 4 in [9].

As far as complexity issues are concerned, one can easily check that, as for a non deterministic word automaton, deciding if a tile $x \in T_A$ belongs to the language $L(\mathcal{A})$ for some finite state automaton can be done in time $2n \cdot 2^p$ with n the size of x and p the number of states in $\mathcal{A}$. Similarly, the language emptyness problem can be solved in linear time in the size of automaton $\mathcal{A}$. Indeed, this just amounts to check that there is a path in the automaton graph from a state p to a state q with $(p, q) \in K$.

2 Quasi-Recognizable Languages of Tiles

We define in this section a notion of quasi-recognizability extending the one proposed in [7]. The major difference is that in [7] our proposal was only defined for languages of positive tiles.

2.1 Adequately Ordered Monoids

Let S be a monoid partially ordered by a relation $\leq_S$ (or just $\leq$ when there is no ambiguity). We always assume that the order relation $\leq$ is stable under product, i.e. if $x \leq y$ then $xz \leq yz$ and $zx \leq zy$ for every x, y and $z \in S$. The set $U(S)$ of *subunits* of the partially ordered monoid S is defined by $U(S) = \{y \in S : y \leq 1\}$.

Definition 2 (Adequately ordered monoid). *A partially ordered monoid S is an* adequately ordered monoid *when all subunits of S are idempotents, and, for every $x \in S$, both $x^L = \bigwedge\{y \in U(S) : xy = x\}$ and $x^R = \bigwedge\{y \in U(S) : yx = x\}$ exist in $U(S)$ with $x^R x = x x^L = x$. The subunit x^L (resp. x^R) is called the* left projection *(resp. the* right projection*) of the element x.*

Since subunits are assumed to be idempotents, one can check that they commute and thus, ordered by the monoid order, form a meet semilattice with the product as the meet operator. It follows that when x is itself a subunit, we have $x = x^L = x^R$, i.e. both left and right projection mappings are indeed projection mappings from S onto $U(S)$.

Examples. Every monoid S extended with the trivial order $x \leq y$ only when $x = y$ is a adequately ordered monoid with $x^L = x^R = 1$ for every $x \in S$. Every inverse monoid S ordered by the natural order relation defined by $x \leq y$ when $x = xx^{-1}y$ (or equivalently $y = yx^{-1}x$) for every x and $y \in S$ is a adequately ordered monoid with $x^L = x^{-1}x$ and $x^R = xx^{-1}$. Especially, the monoid T_A^0 is an adequately ordered monoid. As shown in the next section, the relation monoid $\mathcal{P}(Q \times Q)$ ordered by inclusion is also an adequately ordered monoid.

Remark 1. For the reader familiar with the work initiated by Fountain [5], an ordered monoid S is adequately ordered exactly when it is $U(S)$-semiadequate in the sense of [13]. This suggests that, conversely, a U-semiadequate can be called adequately ordered when its *natural order* defined by $x \preceq y$ when $x = x^R y x^L$ can be *extended* into a partial order $\leq$ that is stable under product with $U = U(S)$. This is not necessarily the case. However, when such a extension exists, both orders $\preceq$ and $\leq$ coincide on subunits.

In [7], only U-semiadequate monoids with stable natural order were considered and shown to suffice for languages of positive tiles. Of course, every such monoid is also an adequately ordered monoid. The more relaxed definition proposed here copes with the fact that the natural order on the relation monoid $\mathcal{P}(Q \times Q)$ is not stable under product while the inclusion order, that extends the natural order, is stable.

Lemma 1. *Let S be an adequately ordered monoid. For every x and $y \in S$, if x and y are $\mathcal{R}$-equivalent (i.e. if $x \cdot S = y \cdot S$) then $x^R = y^R$ and, symmetrically, if x and y are $\mathcal{L}$-equivalent (i.e. if $S \cdot x = S \cdot y$) then $x^L = y^L$.*

In other words, left and right canonical local identities of a given element can be seen as some approximation of its left and right Green's classes.

Remark 2. We prove in [7] that every monoid S can be embedded into an adequately ordered monoid $\mathcal{Q}(S)$: the *quasi-inverse expansion* of S, in such a way that, for every (images of) two elements x and $y \in S$, we have $x^L = y^L$ (resp. $x^R = y^R$) in $\mathcal{Q}(S)$ if and only if x and y are $\mathcal{L}$-equivalent (resp. $\mathcal{R}$-equivalent). We will use a similar idea in the proof of Theorem 3 below.

2.2 Premorphisms and Adequate Premorphisms

A mapping $\varphi : S \to T$ between two adequately ordered monoids is a *premorphism* (or $\vee$*-premorphism* in [6]) when $\varphi(1) = 1$ and, for every x and $y \in S$, we have $\varphi(xy) \leq_T \varphi(x)\varphi(y)$ and if $x \leq_S y$ then $\varphi(x) \leq_T \varphi(y)$.

Definition 3 (Adequate premorphisms). *A premorphism φ is an* adequate premorphism *when, for every $x \in S$, we have $\varphi(x^L) = (\varphi(x))^L$ and $\varphi(x^R) = (\varphi(x))^R$, and for every x and $y \in S$, if $xy \neq 0$ and $x^L \vee y^R = 1$ then $\varphi(xy) = \varphi(x)\varphi(y)$. In that latter case we say that the product xy is disjoint.*

Lemma 2. *Let $\varphi : T_A^0 \to S$ be an adequate premorphism. The restriction of φ to (the overlapping tile image of) A^* is a morphism and, for every positive tile $(u, v, w) \in T_A^+$ one has $\varphi((u, v, w)) = (\varphi(u))^L \cdot \varphi(v) \cdot (\varphi(w))^R$.*

Symmetrically, the restriction of φ to the inverses of (the overlapping tile image of) A^* is also a morphism and, for every negative tile $(uv, \bar{v}, vw)$ one has $\varphi((uv, \bar{v}, vw)) = (\varphi(w))^R \cdot \varphi(v^{-1}) \cdot (\varphi(u))^L = (\varphi(w^{-1}))^L \cdot \varphi(v^{-1}) \cdot (\varphi(u^{-1}))^R$.

As a consequence, when S is finite, for every tile $x \in T_A^0$, the image $\varphi(x)$ of x by φ is effectively computable, in time linear in the size of the tile x, from the images of $\varphi(A)$, $\varphi(\bar{A})$ combined by product and right (or left) projection in S.

Proof. For every $u \in A^*$ we have $\varphi((1, u, 1))$ inductively computable by $\varphi(1) = 1$ and, for every $v \in A^*$ and $a \in A$, $\varphi((1, av, 1)) = \varphi((1, a, 1)) \cdot \varphi((1, v, 1))$ since the product $(1, av, 1) = (1, a, 1) \cdot (1, v, 1)$ is a disjoint product. By symmetry, the same holds for inverses. Indeed, for every tile x and y, if xy is a disjoint product then so is $(xy)^{-1} = y^{-1}x^{-1}$. It follows that $\varphi((u, \bar{u}, u))$ is also computable for every $u \in A^*$,

Then, one can observe that for every tile $x = (u, v, w)$ we have $x = (u, 1, 1) \cdot (1, v, 1) \cdot (1, 1, w)$ with disjoint products and $(u, 1, 1) = (1, u, 1)^L$ and $(1, 1, w) = (1, w, 1)^R$. The adequacy assumption enables us to conclude. that $\varphi((u, v, w)) = (\varphi((1, u, 1)))^L \cdot \varphi((1, v, 1)) \cdot (\varphi((1, w, 1)))^R$ wich is thus computable. By symmetry, we also have $x^{-1} = (1, 1, w) \cdot (v, \bar{v}, v) \cdot (u, 1, 1)$ with disjoint products and $(1, 1, w) = (w, \bar{w}, w)^L$ and $(u, 1, 1) = (u, \bar{u}, u)^R$ hence we conclude similarly.

In these computations, right projections (or left projections) suffice since $(u, \bar{u}, u)^L = (1, u, 1)^R$ and $(u, \bar{u}, u)^R = (1, u, 1)^L$ for every $u \in A^*$.

2.3 Quasi-Recognizable Languages

Definition 4 (Quasi-recognizable languages). *A language of tiles $L \subseteq T_A$ is* quasi-recognizable *when there exists an adequate premorphism $\varphi : T_A^0 \to S$ in a finite adequately ordered monoid S such that $L = \varphi^{-1}(\varphi(L))$.*

As far as computability and complexity issues are concerned, the Lemma 2 ensures that this notion of quasi-recognizability is effective in the sense that membership in a quasi-recognizable language L of tiles is computable. Deciding if a tile $x \in T_A$ belongs to the language L can even be done in bilinear time in the size of the tile x and in the size of the ordered monoid that quasi-recognizes the language L.

It is also quite an immediate consequence of Lemma 2 above that the quasi-recognizable languages of tiles are definable in MSO (see [9] for a definition of MSO logic over tiles). What about the converse? It is a consequence of [7] that quasi-recognizability *essentially* capture MSO in the following sense. Given a new letter $\# \notin A$, given $\#(L) \subseteq T_{A+\#}$ defined for every language $L \subseteq T_A$ by $\#(L) = \{(\#u, v, w\#) : (u, v, w) \in L\}$, we have: language $\#(L)$ is quasi-recognizable if and only if L is definable in MSO. Corollary 1, proved in the next section, provides a complete logical characterization of quasi-recognizable languages of tiles.

3 Tile Automata vs Quasi-Recognizability

In this section, we relate the notions of finite state tile automata and the notion of quasi-recognizable languages of tiles.

3.1 From Tile Automata to Quasi-Recognizability

Let $\mathcal{A} = \langle Q, \delta, K\rangle$ be a finite non deterministic tile automaton. The run image of the positive tile $x = (u, v, w) \in T_A^+$ is defined as the set of pairs of states $\varphi_{\mathcal{A}}((u, v, w)) = \{(p, q) \in Q \times Q : \exists s, e \in Q, s \xrightarrow{u} p, p \xrightarrow{v} q, q \xrightarrow{w} e\}$, i.e. the set of all runs of the tile automaton $\mathcal{A}$ over u.

Theorem 2. *Every language of tiles definable by a finite state tile automaton is quasi-recognizable.*

Proof. Let $\mathcal{A} = \langle Q, \delta, K \subseteq Q \times Q\rangle$ be a tile automaton and let $\varphi_{\mathcal{A}} : T_A \to \mathcal{P}(Q \times Q)$ be the run mapping induced by $\mathcal{A}$. We essentially have to prove that $\mathcal{P}(Q \times Q)$ equipped with the product

$$X \cdot Y = \{(p, q) \in Q \times Q : \exists r \in Q, (p, r) \in X, (r, q) \in Y\}$$

and ordered by inclusion is an adequately ordered monoid and that $\varphi_{\mathcal{A}}$ extended to 0 by taking $\varphi_{\mathcal{A}}(0) = \emptyset$ is an adequate premorphism.

The fact that $\mathcal{P}(Q \times Q)$ equipped with the relation product is a stable ordered monoid with neutral element $I_Q = \{(q, q) \in Q \times Q : q \in Q\}$ and inclusion ordered is a classical result [18]. Since subunits are obviously idempotents, it suffices thus to prove that canonical left and right identities exist.

Let $X \in \mathcal{P}(Q \times Q)$ and let $X^R = \{(p, p) \in Q \times Q : \exists q \in Q, (p, q) \in X\}$ and let $X^L = \{(q, q) \in Q \times Q : \exists p \in Q, (p, q) \in X\}$. One can easily check that $X = X^R \cdot X = X \cdot X^L$ for every $X \subseteq Q \times Q$. Let then $Y \subseteq I_Q$ such that

$Y \cdot X = X$ (resp. $X \cdot Y = X$). It is an immediate observation that this implies $X^R \subseteq Y$ (resp. $X^L \subseteq Y$). In other words, X^L (resp. X^R) is indeed the least right (left) local unit for X.

The fact that $\varphi_{\mathcal{A}}$ extended to zero as defined above is an premorphism raises no real difficulty. By definition, we have $\varphi_{\mathcal{A}}(1) = I_Q$ and it is rather immediate that $\varphi_{\mathcal{A}}(u) \subseteq \varphi_{\mathcal{A}}(v)$ whenever $u \leq v$ in T_A. The fact that we also have $\varphi_{\mathcal{A}}(uv) \leq \varphi_{\mathcal{A}}(u)\varphi_{\mathcal{A}}(v)$ for every u and $v \in T_A^0$ is a little more complex to check but with no special difficulty.

The fact that $\varphi_{\mathcal{A}}$ is also adequate is somehow simpler and essentially follows from the definition. In particular, the disjoint product case just mimics the classical case where $\varphi_{\mathcal{A}}$ is defined over words only.

As an immediate consequence of the definition of $\varphi_{\mathcal{A}}$, writing $X^{-1} = \{(q,p) \in Q \times Q : (p,q) \in X\}$ for every relation $X \subseteq Q \times Q$, we have $\varphi_{\mathcal{A}}(u^{-1}) = (\varphi_{\mathcal{A}}(u))^{-1}$ and thus we also have $L^-(\mathcal{A}) = \{u \in T_A^- : (\varphi_{\mathcal{A}}(u^{-1}))^{-1} \in K\}$. In general, this property is not satisfied by an arbitrary adequate premorphism. However, we prove in Theorem 3 below that every adequate premorphism can still be translated into an equivalent premorphism of the form $\varphi_{\mathcal{A}}$ for some finite state automaton $\mathcal{A}$.

3.2 From Quasi-Recognizability to Tile Automata

Theorem 3. *For every quasi-recognizable language of tiles $L \subseteq T_A$ there exists a finite state non deterministic tile automaton $\mathcal{A}$ such that $L = \varphi_{\mathcal{A}}^{-1}(\varphi_{\mathcal{A}}(L))$, i.e. the adequate premorphism induced by $\mathcal{A}$ recognizes the language L.*

Proof. Let $\psi : T_A^0 \to S$ be an adequate premorphism into a finite adequately ordered monoid S let $K_\psi \subseteq S$ and let $L = \psi^{-1}(K_\psi)$. We want to build a finite state automaton $\mathcal{A} = \langle Q, \delta, K \rangle$ such that $\varphi_{\mathcal{A}}^{-1}(K) = \psi^{-1}(K_\psi)$.

To achieve this goal it is sufficient to define an automaton $\mathcal{A}$ that, given any positive tile $x = (u, v, w)$ (resp. negative tile $x^{-1} = (uv, \bar{v}, vw)$) as input, computes, via $\varphi_{\mathcal{A}}(x)$ (resp. $\varphi_{\mathcal{A}}(x^{-1})$), the *left ideal* $S \cdot \psi(u)$ associated to $\psi(u)$, the *right ideal* $\psi(w) \cdot S$ associated to $\psi(w)$ and the *image* $\psi(v)$ of the root v of x (resp. the *image* $\psi(v^{-1})$ of the root $\bar{v}$ of x^{-1}).

Indeed, by Lemma 1, computing these left and right ideals is enough to compute the expected left and right canonical identities $\psi(u^L)$ and $\psi(w^R)$. Then, by applying Lemma 2, together with the value of $\psi(v)$ (resp. $\psi(v^{-1})$), we can compute the value $\psi(x)$ of x (resp. $\psi(x^{-1})$ of x^{-1}) by the premorphism ψ.

Extending the idea described in the automata section in order to distinguish positive from negative tiles, the automaton $\mathcal{A}$ is defined as follows.

The set of states Q is defined to be $Q = S \times S \times S \times M$ with set of modes $M = \{P, S, PR, NR, c, p_1, p_2, n_1, n_2\}$ where P, S, PR and NR respectively stand for the "stable" automaton mode prefix, suffix, positive root and negative root automaton modes, and c, p_1, p_2, n_1 and n_2 stand for the "frontier" modes that occur at most once in between stable modes.

For every $a \in A$, the set $\delta(a)$ of transitions labeled by a is defined to be the union of the following sets of transitions:

- ▷ "prefix" transitions, from states in mode P to states in mode $m \in \{P, c, p_1, n_1\}$, of the form $\{((x, y, z, P), (x \cdot \varphi(a), y, z, m) : x, y, z \in Q\}$,
- ▷ "positive root" transitions, from states in mode $m \in \{PR, p_1\}$ to states in mode $k \in \{PR, p_2\}$, of the form $\{((x, y, z, m), (x, y \cdot \varphi(a), z, k)) : x, y, z \in S\}$,
- ▷ "negative root" transitions, from states in mode $m \in \{PN, n_1\}$ to states in mode $k \in \{PN, n_2\}$, of the form $\{((x, y, z, m), (x, \varphi(a^{-1}) \cdot y, z, k)) : x, y, z \in S\}$,
- ▷ "suffix" transitions, from states in mode $m \in \{S, c, p_2, n_2\}$ to states in mode S, of the form $\{((x, y, \varphi(a) \cdot z, m), (x, y, z, S)) : x, y, z \in S\}$.

Of course, such an automaton will run freely on tiles regardless of whether it is running on the prefix, the root or the suffix part of a tile. However, by watching the states in frontier modes occurring at the extremities of the root, we can collect all the information we need.

More precisely, the next step is then to keep from the set of all runs $\varphi_{\mathcal{A}}(x)$ of $\mathcal{A}$ on any given input tile $x \in T_{\mathcal{A}}$ only the relevant data. This is achieved by the following mapping. For every $X \subseteq \mathcal{P}(Q \times Q)$, we define the relevant image $f(X) \subseteq S \times S \times S$ of X "in" S by taking:

$$f(X) = \{(x, 1, z) \in S \times S \times S : ((x, 1, z, c), (x, 1, z, c)) \in X\} \quad (1)$$
$$\cup\{(x, y, z) \in S \times S \times S : ((x, 1, z, p_1), (x, y, z, p_2)) \in X\} \quad (2)$$
$$\cup\{(x, y, z) \in S \times S \times S : ((x, 1, z, n_2), (x, y, z, n_1)) \in X\} \quad (3)$$

where line (1) treats the case of context tiles, line (2) treats the case of (strictly) positive tiles and line (3) treats the case of (strictly) negative tiles.

With this construction, one can show that for every $x = (u, v, w) \in T_A^+$ we have: $f(\varphi_{\mathcal{A}}(x)) = S \cdot \psi(u) \times \{\psi(v)\} \times \psi(w) \cdot S$ and, for every $x^{-1} = (uv, \bar{v}, vw) \in T_A^-$ we have $f(\varphi_{\mathcal{A}}(x^{-1})) = S \cdot \psi(u) \times \{\psi(v^{-1})\} \times \psi(w) \cdot S$. In other words, for every $x \in T_A$, the finite value of $f(\varphi_{\mathcal{A}}(x))$ completely characterizes $\psi(x)$ thus we conclude the proof by taking

$$K = f^{-1}\left(\{S \cdot \psi(u) \times \{\psi(v)\} \times \psi(w) \cdot S : \psi(u^L) \cdot \psi(v) \cdot \psi(w^R) \in K_\psi\}\right)$$
$$f^{-1}\left(\{S \cdot \psi(u) \times \{\psi(v^{-1})\} \times \psi(w) \cdot S : \psi(w^R) \cdot \psi(v^{-1}) \cdot \psi(u^L) \in K_\psi\}\right)$$

By construction, for every tile $x \in T_A$, we indeed have $x \in \varphi_{\mathcal{A}}^{-1}(K)$ if and only if $x \in \psi^{-1}(K_\psi)$.

Observe that if we consider a language $L \subseteq T_A^+$ of *positive* tiles that is recognizable by a premorphism ψ from the monoid of positive tiles T_A^+ into an adequately ordered monoid S then the above proof can easily be adapted so that $\varphi_{\mathcal{A}} : T_A^0 \to \mathcal{P}(Q \times Q)$ still recognizes L. In other words, a quasi-recognizable language of positive tiles is also quasi-recognizable as a language of arbitrary tiles. This proves that the work presented here indeed generalizes the results formerly obtained in [7].

Corollary 1. *For every language of tiles $L \subseteq T_A$, the language $L \subseteq T_A$ is quasi-recognizable if and only if L is a finite boolean combination of upward closed MSO-definable languages of tiles.*

Proof. Immediate consequence of Theorem 1 and Theorem 3.

Conclusion

We have shown that the emerging notion of quasi-recognizability, defined in [7] as a remedy to the collapse of classical recognizability by monoid morphisms, can be equipped with quite a simple notion of finite state automata that, applied to languages of overlapping tiles, captures its expressiveness. Extending the theory of languages of words, this new theory of languages of tiles inherits several and significantly robust features of that classical language theory.

Compared to our former proposal [7], the notion of quasi-recognizability has also been refined - especially via the notion of disjoint products - and can now be applied to more general settings. Forthcoming studies [8] even show that the notion of recognizability by adequate premorphisms and the related notion of non deterministic finite state automata can even be extended, with similar logical characterizations, to languages of labeled birooted trees generalizing thus the known algebraic characterizations of regular languages of finite trees.

Based on the notion of U-semiadequate monoids [13], the present work also provides a rather unexpected application of the study of non regular semigroups initiated by Fountain [5] in the 70's. Our proposal still need to be further related with the research lines developed in that field such as, for instance, the notion of partial actions [2,3].

References

1. Berthaut, F., Janin, D., Martin, B.: Advanced synchronization of audio or symbolic musical patterns: an algebraic approach. International Journal of Semantic Computing 6(4), 1–19 (2012)
2. Cornock, C.: Restriction Semigroups: Structure, Varieties and Presentations. PhD thesis, Departement of Mathematics University of York (2011)
3. Cornock, C., Gould, V.: Proper two-sided restriction semigroups and partial actions. Journal of Pure and Applied Algebra 216, 935–949 (2012)
4. Dicky, A., Janin, D.: Two-way automata and regular languages of overlapping tiles. Technical Report RR-1463-12, LaBRI, Université de Bordeaux (2012)
5. Fountain, J.: Adequate semigroups. Proc. Edinburgh Math. Soc. 22(2), 113–125 (1979)
6. Hollings, C.D.: The Ehresmann-Schein-Nambooripad Theorem and its successors. European Journal of Pure and Applied Mathematics 5(4), 414–450 (2012)
7. Janin, D.: Quasi-recognizable vs MSO definable languages of one-dimensional overlapping tiles. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 516–528. Springer, Heidelberg (2012)
8. Janin, D.: Algebras, automata and logic for languages of labeled birooted trees. Technical Report RR-1467-13, LaBRI, Université de Bordeaux (2013)

9. Janin, D.: On languages of one-dimensional overlapping tiles. In: van Emde Boas, P., Groen, F.C.A., Italiano, G.F., Nawrocki, J., Sack, H. (eds.) SOFSEM 2013. LNCS, vol. 7741, pp. 244–256. Springer, Heidelberg (2013)
10. Janin, D., Berthaut, F., DeSainte-Catherine, M., Orlarey, Y., Salvati, S.: The T-calculus: towards a structured programming of (musical) time and space. Technical Report RR-1466-13, LaBRI, Université de Bordeaux, (to appear, 2013)
11. Kellendonk, J.: The local structure of tilings and their integer group of coinvariants. Comm. Math. Phys. 187, 115–157 (1997)
12. Kellendonk, J., Lawson, M.V.: Universal groups for point-sets and tilings. Journal of Algebra 276, 462–492 (2004)
13. Lawson, M.V.: Semigroups and ordered categories. i. the reduced case. Journal of Algebra 141(2), 422–462 (1991)
14. Lawson, M.V.: Inverse Semigroups: The theory of partial symmetries. World Scientific (1998)
15. Lawson, M.V.: McAlister semigroups. Journal of Algebra 202(1), 276–294 (1998)
16. Margolis, S.W., Pin, J.-E.: Languages and inverse semigroups. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 337–346. Springer, Heidelberg (1984)
17. Munn, W.D.: Free inverse semigroups. Proceeedings of the London Mathematical Society 29(3), 385–404 (1974)
18. Pin, J.-E.: Syntactic semigroups. In: Handbook of formal languages, vol. I, ch. 10, pp. 679–746. Springer (1997)
19. Silva, P.V.: On free inverse monoid languages. ITA 30(4), 349–378 (1996)

Author Index